МАКСИМ КУЗНЕЦОВ
ИГОРЬ СИМДЯНОВ

bhv®

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ на PHP

КЛАССЫ И ОБЪЕКТЫ

ИСКЛЮЧЕНИЯ

ОТРАЖЕНИЯ

FRAMEWORK

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ CMS

ПОДВОДНЫЕ КАМНИ ООП

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Максим Кузнецов
Игорь Симдянов

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ на PHP

Санкт-Петербург
«БХВ-Петербург»
2008

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.1
К89

Кузнецов, М. В.

К89 Объектно-ориентированное программирование на PHP /
М. В. Кузнецов, И. В. Симдянов. — СПб.: БХВ-Петербург, 2008. —
608 с.: ил. + CD-ROM — (Профессиональное программирование)

ISBN 978-5-9775-0142-2

Книга предоставляет наиболее полное описание объектно-ориентированных возможностей PHP.

Предполагается, что читатель знаком с базовыми возможностями PHP, языком разметки HTML и приемами работы с СУБД MySQL. Даны основы объектно-ориентированного подхода: классы, специальные методы классов, инкапсуляция, наследование и полиморфизм, интерфейсы, статические, константные и final члены класса, особенности клонирования и длительного хранения объектов, обработка исключений и др. Рассмотрена практика объектно-ориентированного программирования на примерах — от построения собственного Framework (набора классов, облегчающих разработку Web-приложений) и до создания собственной объектно-ориентированной системы управления контентом (CMS). На прилагаемом компакт-диске находятся исходные коды всех Web-приложений, рассматриваемых в книге.

Для Web-разработчиков

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Ирина Артемьева*
Компьютерная верстка	*Ольги Сергиенко*
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 24.09.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 49,02.
Тираж 2500 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 978-5-9775-0142-2

Оглавление

Введение

Начинающие и опытные программисты часто задаются вопросом: что дает объектно-ориентированное программирование? Освоив его, получу ли я какое-то конкурентное преимущество по сравнению с другими разработчиками? Если для объектно-ориентированных языков программирования, таких как Java или C++, ответ однозначен, то в Web-среде, больше ориентированной на процедурный стиль, ответить на этот вопрос не так просто.

Между тем все большее количество библиотек переходит на объектно-ориентированный интерфейс, вынуждая разработчиков обращаться к объектно-ориентированным возможностям PHP. Введение в пятой версии PHP полноценной объектно-ориентированной модели еще больше подогревает интерес к этой методологии.

Книга, которую вы держите в руках, позволяет найти ответ на вопросы: зачем нужно объектно-ориентированное программирование в PHP, когда его следует применять, а когда его применение не целесообразно и даже вредно.

Зачастую использование объектно-ориентированного подхода к месту и не к месту не обязательно делает проект успешным. Программирование новичка в стиле объектно-ориентированного программирования часто напоминает передвижение по минному полю — если не знать где мины, достичь конца проекта невозможно. Само по себе объектно-ориентированное программирование не является панацеей — это рабочая технология, которая позволяет:

- увеличить процент повторно используемого кода;
- оперировать при программировании понятиями и объектами реального мира (договор, заключение договора, распечатка договора, поиск договора), а не низкоуровневыми компьютерными терминами (файлы, строка, стандартный поток вывода), что позволяет создавать более крупные проекты с меньшим количеством ошибок и в более сжатые сроки.

Предлагаемая книга рассматривает объектно-ориентированное программирование применительно к PHP, раскрывая его методологическую часть. Будет определено, в каком случае следует взять за основу объектно-ориентированный подход в PHP, а в каком — лучше от него отказаться. На примере построения большого Web-приложения (CMS) демонстрируется, как добиться повторного использования кода в реальных проектах. Рассматриваемая система управления содержимым сайта (CMS) используется сотрудниками нашей студии SoftTime (**http://www.softtime.ru**) для построения сайтов (например, на рассмотренной в данной книге объектно-ориентированной CMS построен сайт нашего нового проекта **http://www.bipsi.ru**).

Дополнительные материалы можно также найти на группе сайтов IT-студии SoftTime, сотрудниками которой являются авторы книги:

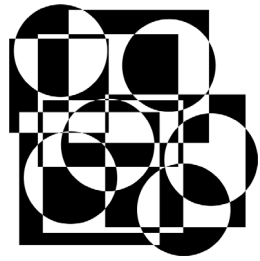
- **http://www.softtime.ru** — головной сайт нашей студии;
- **http://www.softtime.biz** — услуги, предоставляемые студией;
- **http://www.softtime.org** — различные проекты студии;
- **http://www.softtime.mobi** — мобильные версии материалов и форумов.

Обновленные версии приложений и SoftTime FrameWork можно загрузить по адресу **http://www.softtime.ru/info/downloads.php**, а обсудить вопросы, которые могут возникнуть по мере чтения материала книги, можно на форуме авторов **http://www.softtime.ru/forum/**.

Благодарности

Авторы выражают признательность сотрудникам издательства "БХВ-Петербург", благодаря которым наша рукопись увидела свет, а также посетителям форума **http://www.softtime.ru/forum/** за интересные вопросы и конструктивное обсуждение.

ГЛАВА 1



Введение в объектно-ориентированное программирование

Объектно-ориентированное программирование на сегодняшний день является одной из самых популярных методик создания программного обеспечения. Однако чтобы использовать ее по назначению и с наибольшей отдачей, необходимо четко представлять себе причины возникновения этой технологии и область ее применения.

1.1. Причины возникновения объектно-ориентированной технологии

Развитие технологий программирования, как метко заметил Дейкстра, диктуется тезисом "Разделяй и властвуй". Любые удачные технологии предполагают, что чем короче код программы, тем легче его создавать, отлаживать и сопровождать, а простая программа подвержена ошибкам в гораздо меньшей степени, чем сложная.

На заре компьютерной эры программа представляла собой один поток, который обрабатывал один массив данных (рис. 1.1).

Замечание

На рисунках в данном разделе поток обозначается стрелкой, в то время как массив данных — квадратом.

Со временем сложность программ и предъявляемых к ним требований возросли, и такой способ организации данных оказался неприемлемым. Был предложен структурный подход, при котором массив данных становился доступен из любой точки программы, однако основной поток программы разбивался на несколько процедур. Отдельную небольшую процедуру, пусть даже

использующую общие данные, разрабатывать гораздо проще, чем большой объем кода (рис. 1.2).

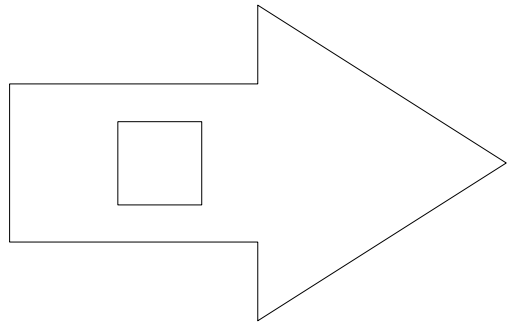


Рис. 1.1. Один поток обрабатывает один массив данных

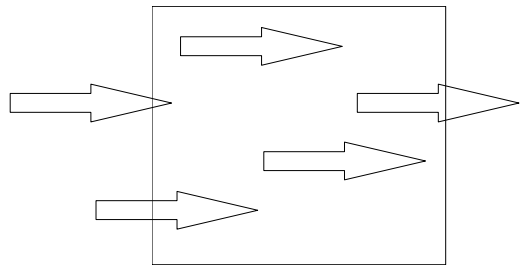


Рис. 1.2. Несколько потоков обрабатывают один массив данных

Каждая из процедур обладает локальными переменными, срок жизни которой определяется продолжительностью работы процедуры. Одни процедуры могут вызывать другие, однако массив данных в программе остается общим и доступным для всех процедур. Такой подход позволил создавать еще более значительные программные комплексы. В результате объем данных вырос еще больше, а для его обработки потребовались еще более гибкие способы масштабирования программ, учитывающие разбиение данных.

Ответом на все возрастающую сложность стало появление объектно-ориентированного подхода в программировании: программа разбивается на несколько массивов данных, каждый из которых имеет свои собственные процедуры, а также процедуры, которые взаимодействуют с другими массивами данных.

В результате сложная задача разбивается на ряд более простых подзадач, а разработчики получают более гибкий способ управления проектом: редактировать один огромный монолитный блок кода гораздо сложнее, чем совокупность небольших, слабо связанных между собой блоков.

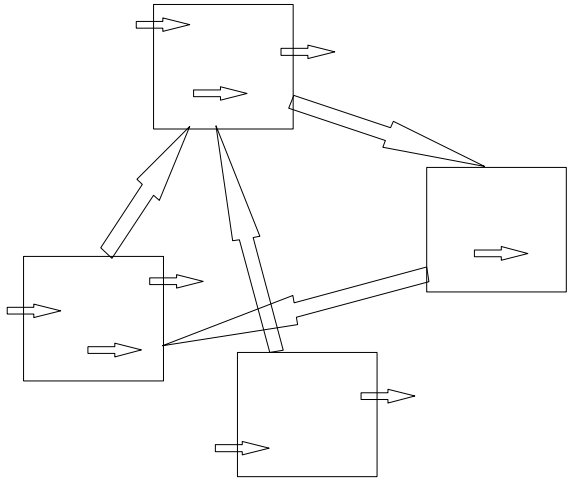


Рис. 1.3. Несколько массивов данных обрабатываются отдельным потоком

Замечание

Примечательно, что подход к программированию, использующий слабо связанные, взаимодействующие между собой блоки, был принят при создании операционной системы UNIX, в которой для решения задачи несколько программ выстраиваются в цепочку, тогда как в других операционных системах, таких как Windows, каждая программа представляет собой монолит, практически не взаимодействующий с другими программами. Именно блочный подход в UNIX объясняет его долголетие (около 40 лет): цепочки программ можно перестраивать, использовать элементы старых цепочек для построения новых, благодаря чему операционная система очень быстро приспосабливается к новому аппаратному обеспечению. Точно так же и объектно-ориентированный подход позволяет создать программное обеспечение с более гибкими и легко модифицируемыми свойствами.

1.2. Какая программа является объектно-ориентированной?

Объектно-ориентированная технология не привязана к языку программирования. Даже если язык не поддерживает ее в явном виде, с его помощью можно создать объектно-ориентированную систему. Именно так создавались первые версии операционной системы Windows: несмотря на то что язык C (не путать с C++) не содержал конструкций, поддерживающих объектно-ориентированный подход, разработчикам удалось организовать код таким образом, что он, по сути, был объектно-ориентированным.

Конечно, гораздо удобнее создавать объектно-ориентированные программы, если объектно-ориентированная методология поддерживается на уровне язы-

ка программирования. Такая поддержка избавляет от необходимости программирования рутинных операций и использования сложных нестандартных библиотек.

С другой стороны, синтаксиса объектно-ориентированного программирования недостаточно, чтобы называть программу объектно-ориентированной. Можно привести множество примеров прикладных программ (в том числе и на PHP), использующих классы, объекты, но не являющихся по своей природе объектно-ориентированными. Объект в таких приложениях выступает в качестве удобного контейнера, представляющего все приложение, организация кода в этом случае опять же сводится к одному блоку данных и одному обрабатывающему их потоку.

Независимо от привязки к языку программирования, объектно-ориентированный подход имеет ряд общих принципов, а именно:

- возможность создавать *абстрактные типы данных*, позволяющая наряду с предопределенными типами данных (таких как `integer`, `bool`, `double`, `string`) вводить свои собственные типы данных (классы) и объявлять "переменные" таких типов данных (объекты). Создавая свои собственные типы данных, программист оперирует не машинными терминами (переменная, функция), а объектами реального мира, поднимаясь тем самым на новый абстрактный уровень. Яблоки и людей нельзя умножать друг на друга, однако низкоуровневый код запросто позволит совершить такую логическую ошибку, тогда как при использовании абстрактных типов данных подобная операция становится невозможной;
- *инкапсуляция*, ограничивающая взаимодействие пользователя абстрактных типов данных только их интерфейсом и скрывающая внутреннюю реализацию объекта, не допуская влияния на его внутреннее состояние. Память человека ограничена и не может содержать все детали огромного проекта, тогда как использование инкапсуляции позволяет разработать объект и использовать его, не заботясь о внутренней реализации и прибегая только к небольшому числу интерфейсных методов;
- *наследование*, позволяющее развить существующий абстрактный тип данных — класс, создав на его основе новый класс. При этом новый класс автоматически получает возможности уже существующего абстрактного типа данных. Зачастую абстрактные типы данных слишком сложны, поэтому прибегают к их последовательной разработке, выстраивая иерархию классов от общего к частному;
- *полиморфизм*, допускающий построение целых цепочек и разветвленных деревьев наследующих друг другу абстрактных типов данных (классов). При этом весь набор классов будет иметь ряд методов с одинаковыми

названиями: любой из классов данного дерева гарантированно обладает методом с таким именем. Этот принцип помогает автоматически обрабатывать массивы данных разного типа.

Замечание

Программа, отвечающая только одному принципу, не обязательно является объектно-ориентированной. Например, абстрактные типы данных появились задолго до объектно-ориентированного подхода. Если один из приведенных выше принципов оказывается лишним в объектной схеме программы — это повод задуматься: а требуется ли вообще для решения этой задачи объектно-ориентированный подход?

Следовать этим нехитрым правилам можно без поддержки объектно-ориентированного подхода на уровне языка программирования. Доказательством служит операционная система Windows 95, разработанная на языке C, благодаря которой корпорация Microsoft заняла ведущие позиции в мире.

Замечание

Язык программирования C поддерживает абстрактные типы данных, однако инструментальная поддержка инкапсуляции, наследования и полиморфизма в нем отсутствует.

На самом деле, на каком бы языке программирования мы ни работали, мы имеем дело с абстрактными данными — программируя систему документооборота, мы в любом случае будем иметь дело с документами и волей-неволей абстрагироваться от машинных данных.

Инкапсуляция существует с момента создания первых структурных программ: модули и функции также скрывают реализацию, получая на входе данные и выдавая на выходе результат; пользующийся ими программист может не вникать в принципы их работы.

Наследование тоже в какой-то мере использовалось и до появления объектно-ориентированного подхода: программисты всегда стремились использовать ранее наработанный код в новых проектах.

Полиморфизм можно организовать в любой программе, для этого следует просто правильно организовать именование процедур и функций, так чтобы имя предполагаемой процедуры можно было "вычислить" автоматически.

Тем не менее, за последние 20 лет почти каждый "живой" язык программирования был снабжен дополнительными ключевыми словами в целях поддержки объектно-ориентированного подхода. Для чего же понадобились новые обозначения уже существующих концепций?

1.3. О важности обозначений

Без сомнения, каждому, читающему эти строки, знакомо выражение:

$$2 + 2 = 4.$$

Благодаря всеобщему образованию, таблица умножения также не вызывает затруднений:

$$6 \times 8 = 48.$$

Однако в античные времена доказательство последнего утверждения занимало множество листов и тянуло на кандидатскую диссертацию.

Благодаря логике, созданной Аристотелем, а затем развитой арабами и отточенной схоластами, на сегодняшний день у нас есть научный язык и система математических обозначений. Решение сложнейших для античности задач сейчас не является ни доблестью, ни даже правилом хорошего тона — это неотъемлемая часть культуры. При этом таблица умножения усваивается нами на подсознательном уровне — мы не вычисляем ответ, а просто знаем его и не задумываемся, откуда он берется.

Это не значит, что мыслители древности были менее способными, просто для решения задач они пользовались другим математическим аппаратом. Как известно, существует два способа решения сложной задачи:

- сложное решение задачи простыми способами;
- простое решение задачи сложными способами.

В последнем случае под сложным способом понимается не запутанный, а более высокотехнологичный и высокоэффективный способ. Как правило, на освоение такого способа требуется потратить время.

Поиск строки в тексте по сложному условию может занимать десятки и сотни строк кода с использованием строковых функций. При помощи регулярных выражений задача зачастую решается в одну строку кода. Однако для того чтобы решать задачу при помощи регулярных выражений, их следует освоить, на что уходит немало времени. Многие не без успеха используют вместо регулярных выражений строковые функции (то есть решают сложную задачу простыми способами). Однако программисты, потратившие время на изучение регулярных выражений, практически никогда больше не решают задачи поиска при помощи строковых функций.

Объектно-ориентированный подход тоже требует длительного усвоения, однако взамен предоставляет преимущества, а именно позволяет создавать простые решения для сложных задач, практически не разрешимых в рамках структурного подхода.

Рассмотрим пример PHP-программы. В листинге 1.1 перемножаются два числа: число 6, которое хранится в таблице `number`, и число 8, которое, в свою очередь, хранится в таблице `blocks`.

Листинг 1.1. Умножение двух чисел, находящихся в разных таблицах базы данных

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) {
   exit( "<P>В настоящий момент сервер базы данных не доступен,
             поэтому корректное отображение страницы невозможно.</P>" );
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) ) {
    exit( "<P>В настоящий момент база данных не доступна,
              поэтому корректное отображение страницы невозможно.</P>" );
  }

  // Извлекаем число из таблицы number
  $query = "SELECT num FROM number";
  $num = mysql_query($query);
  if(!$num) exit("Ошибка обращения к таблице number");
  if(!mysql_num_rows($num)) exit("Отсутствуют записи в таблице number");
  $number = mysql_result($num, 0);

  // Извлекаем число из таблицы blocks
  $query = "SELECT num FROM blocks";
  $blk = mysql_query($query);
  if(!$blk) exit("Ошибка обращения к таблице blocks");
  if(!mysql_num_rows($blk)) exit("Отсутствуют записи в таблице blocks");
  $blocks = mysql_result($blk, 0);

  echo $number * $blocks;
?>
```

Как можно заметить, для умножения числа 6 на 8 вместо одной строки понадобилось 38 строк кода. Нетрудно догадаться, какого размера достигнет законченное приложение, если будет реализовано подобным образом.

Задача объектно-ориентированного подхода состоит в сокращении кода, так чтобы реализация промежуточных операций не отвлекала программиста от решения поставленной задачи (листинг 1.2).

Листинг 1.2. Объектно-ориентированный подход

```
<?php
  $number = new num("number");
  $blocks = new num("blocks");
  echo $number.num * $blocks.num;
?>
```

Важно понимать, что сразу 38 строк преобразовать в пять не удастся. Если сложить все строки, которые необходимы для функционирования кода из листинга 1.1, его объем наверняка превысит код из листинга 1.2. Сэкономить можно только на очень больших приложениях и, возможно, на их комплексах.

1.4. Терминология

Ключевыми понятиями объектно-ориентированного подхода являются объект и класс, которые находятся в тесной связи с понятиями переменной и ее типа. В листинге 1.3 приведены примеры объявления целочисленной переменной `$val` и строковой переменной `$str`.

Замечание

Язык PHP не является строго типизированным, одну и ту же переменную можно использовать для хранения целого числа, строкового значения и числа с плавающей точкой. В связи с этим при объявлении переменной не требуется указывать ее тип. Однако это вовсе не означает, что тип переменной отсутствует вовсе — просто переменные принимают тип своего текущего значения. Изменить тип переменной можно при помощи функций `is_bool()`, `is_float()`, `is_int()`, `is_numeric()` и `is_string()`.

Листинг 1.3. Объявление целочисленной и строковой переменных

```
<?php
  $val = 1;
  $str = "Hello world";
?>
```

Переменная и ее тип — это низкоуровневые машинные понятия. Переменная представляет собой область памяти, в которой хранится ее значение, а тип — выделяемый для ее хранения объем памяти. Используя множество переменных и обрабатывающих их функций, можно моделировать объекты и процессы реального мира. Однако мало построить объекты, необходимо еще определить способы их взаимодействия. Связи между объектами реального мира зачастую настолько сложны, что для их эффективного моделирования необходим отдельный язык программирования. Разрабатывать специализированный язык программирования для каждой прикладной задачи — очень дорогое удовольствие. Поэтому в языки программирования вводится объектно-ориентированный подход, который позволяет разработать свой мини-язык путем создания классов и их объектов. Переменными такого мини-языка программирования выступают программные *объекты*, в качестве типа для которых выступает класс. *Класс* описывает состав объекта — переменные и функции, которые обрабатывают переменные и тем самым определяют поведение объекта. После того как класса разработан, можно объявить несколько экземпляров объекта или даже их массив.

Замечание

Элементы объектно-ориентированного программирования начали вводиться практически сразу после становления структурного подхода. Понятие классов и объектов впервые введены в языке программирования Simula 67. Считается, что окончательно объектно-ориентированный подход сформировался в языке программирования Smalltalk, который и явился прообразом всех современных объектно-ориентированных языков.

Таким образом, объект — это своеобразная макропеременная, поведение которой определяется программистом при помощи класса, выступающего в качестве типа макропеременной. В результате программист поднимается на более высокий уровень абстракции, оперируя не низкоуровневыми компьютерными терминами (переменная, функция, число, строка), а высокоуровневыми конструкциями (объект, его поведение, например, договор, клиент и т. п.).

Таким образом, не прибегая к сложным инструментам вроде создания специализированного языка высокого уровня, программист *расширяет* уже существующий объектно-ориентированный язык программирования, вводя в него новые типы данных (классы), переменные которого (объекты) обладают необходимым для решения текущей задачи поведением.

Г. Буч определяет объектно-ориентированное программирование следующим образом: *объектно-ориентированное программирование* — это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Каждый из классов разрабатывается отдельно и включает в себя переменные, которые называют *членами класса*, и функции, которые называют *методами класса*. В качестве члена класса может выступать объект другого класса, таким образом, можно создавать действительно сложные объекты.

1.5. Повторное использование кода

Классы должны обеспечивать повторное использование кода. В идеале класс, созданный в одном приложении, должен свободно извлекаться из него и встраиваться в другое приложение. Однако добиться этого на практике достаточно сложно. Без тщательного проектирования и жесткого контроля классы норовят связаться друг с другом всеми возможными способами (рис. 1.4).

Замечание

На рисунках в данном разделе класс обозначается квадратом, а его интерфейс, позволяющий ему взаимодействовать с другими классами, — стрелкой.

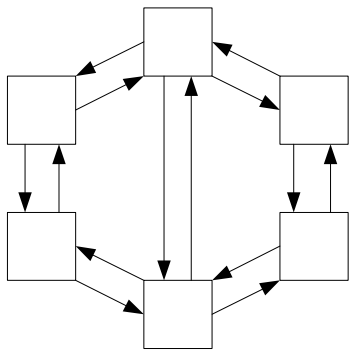


Рис. 1.4. Сильно связанная система

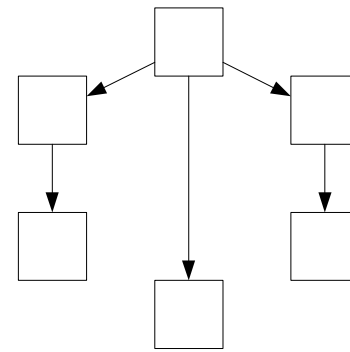


Рис. 1.5. Слабо связанная система

В результате извлечь класс для повторного использования в другом приложении можно только "с мясом" (то есть вместе с ворохом связанных с ним классов). Одна из важнейших задач разработчика состоит в проектировании системы, которая имела бы по возможности как можно меньше связей (так называемое слабое связывание) (рис. 1.5).

Дело в том, что проявить всю свою силу объектно-ориентированный подход может только в том случае, если классы слабо связаны между собой и могут разрабатываться независимо друг от друга.

Чем меньше связей между отдельными подсистемами, тем надежнее система и тем проще ее сопровождать, т. к. ее общая сложность в этом случае уменьшается. Когда каждая подсистема взаимодействует с другими подсистемами,

программисту необходимо знать особенности их работы и влияние вносимых изменений на другие части системы; когда же такое взаимодействие сведено к минимуму, программисту приходится учитывать меньшее количество связей, и вероятность совершения ошибок уменьшается.

Замечание

В 1956 г. Миллер в статье "The Magical Number Seven, Plus or Minus Two" показал, что человек способен удерживать в краткосрочной памяти семь объектов плюс-минус два. Это означает, что для эффективной работы число связей, которые должен учитывать программист при разработке того или иного блока, не должна превышать 5—9. (Со статьей "The Magical Number Seven, Plus or Minus Two" можно ознакомиться по адресу **http://www.well.com/user/smalin/miller.html**.)

Объектно-ориентированный подход предоставляет разработчику две возможности повторного использования кода:

- *композиция* — включение в класс любого количества объектов произвольного типа. Наряду с переменными встроенных типов, класс может содержать объекты других типов точно так же, как объекты реального мира могут содержать в своем составе другие объекты. Например, объект "государство" содержит объекты "области" и "края", которые, в свою очередь, включают в качестве объектов "населенные пункты". Объект "населенный пункт" может содержать объекты "улицы", в которые могут входить объекты "дома" и, наконец, последние могут включать в себя объекты "квартиры";
- *наследование* — расширение существующего класса путем наследования от него производного класса. Класс "транспортное средство" может быть расширен до класса "автомобили", который будет включать в себя все члены и методы обобщенного транспортного средства; в свою очередь "автомобиль" может быть расширен до классов "легковой автомобиль" и "грузовой автомобиль".

На рис. 1.6 схематически представлена схема композиции. В объект класса может входить набор некоторых объектов, которые также могут включать другие объекты.

Наследование позволяет расширять уже существующий класс, при этом получаемый новый класс автоматически содержит все члены и методы старого класса. Старый класс при этом называют базовым классом, а новый — производным классом. На рис. 1.7 представлена схема наследования.

Наследование позволяет избежать утомительной работы по реализации методов и членов для похожих классов, которые отличаются друг от друга только незначительными деталями.

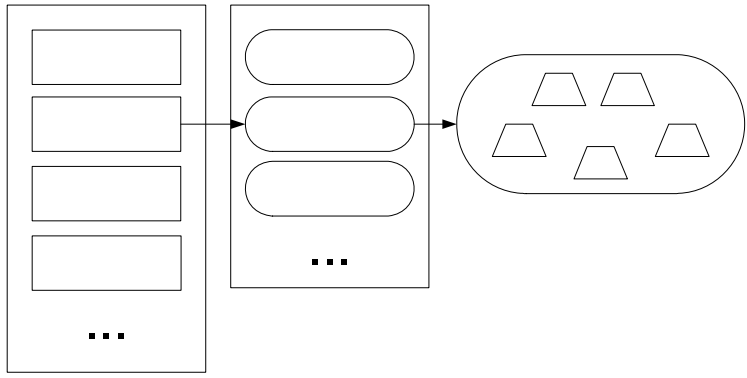


Рис. 1.6. Композиция

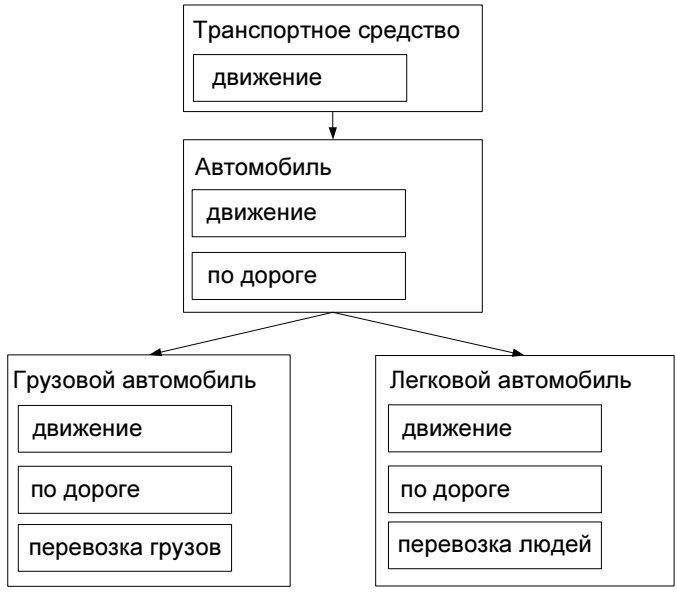


Рис. 1.7. Наследование

1.6. Недостатки объектно-ориентированного подхода

Обычно книга, посвященная какой-либо технологии, рассказывает только о ее достоинствах и сильных сторонах. Автор, негативно относящийся к использованию какого-либо подхода, обычно не берется за написание книги и даже за детальное изучение технологии, если обнаруживает в ней крити-

ческое количество недостатков. В результате вокруг технологии формируется радужный ореол, который разрушается, когда в проект уже вложено значительное количество сил, средств и времени. Поэтому следует упомянуть о подводных камнях объектно-ориентированного подхода в PHP и сразу развеять ряд устойчивых мифов.

Объектно-ориентированный подход — это не панацея. Не стоит думать, что, используя объектно-ориентированный подход, вы автоматически избавляетесь от ошибок проектирования. Запутаться в плохо спроектированных классах так же легко, как разработать неудачное структурное приложение. Если вы используете объектно-ориентированный подход просто ради его использования и не можете ответить, какие преимущества он предоставит вам по сравнению со структурным подходом, вы ничего не получите, кроме увеличения времени разработки, снижения читабельности программы и ее эффективности.

Снижение производительности. За любую функциональность следует расплачиваться либо производительностью, либо читабельностью, либо эффективностью, либо всем вместе взятым. Например, операционные системы реального времени менее эффективны своих "неточных" аналогов. Причина этого заключается в том, что для обеспечения точного момента срабатывания система вынуждена простаивать вплоть до наступления временного репера: в данном случае мы расплачиваемся производительностью за точность. Аналогично обстоит дело и с объектно-ориентированным подходом — создание и удаление объектов требует времени и памяти. Следует отметить, что только один язык не теряет эффективности при использовании объектно-ориентированного подхода — это C++. В PHP эффективность уменьшается в среднем на 30%.

Процедурные интерфейсы. За годы развития языков программирования и компьютерных технологий было разработано большое количество разнообразных интерфейсов, начиная с сетевых протоколов и заканчивая API СУБД. До настоящего времени объектно-ориентированные интерфейсы не получили широкого распространения в области Web-программирования, поскольку наибольшей популярностью пользуются реляционные СУБД, а объектно-ориентированные СУБД существуют лишь в концептуальной форме. В результате довольно часто приходится преобразовывать объект в линейную форму (например, перед помещением в базу данных или для отправки по Сети) и затем снова его восстанавливать. Это не прибавляет разработке приложений ни удобства, ни эффективности, ни скорости.

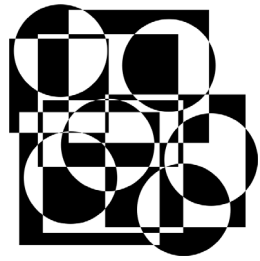
В PHP элементы объектно-ориентированного подхода не отлажены до конца. В настоящий момент существует некоторое количество ошибок, связанных с реализацией объектно-ориентированного подхода в PHP. Они не

являются критичными и их можно обойти, однако это зачастую утомительно и требует дополнительного времени и увеличивает объем кода, что снижает эффективность объектно-ориентированного подхода по сравнению со структурным, который более или менее отлажен. Ряд ошибок реализации и способы их обхода описываются в следующих главах.

1.7. Объектно-ориентированное программирование в PHP

Объектно-ориентированное программирование в PHP появилось сравнительно недавно, начиная с четвертой версии. Однако можно утверждать, что в четвертой версии PHP объектно-ориентированная модель была реализована в крайне ограниченных объемах, отсутствовал деструктор, спецификаторы доступа, исключения. Начиная с пятой версии PHP, объектно-ориентированная форма получила современное воплощение, поэтому в данной книге будет представлена объектно-ориентированная модель PHP 5.0 и 5.1.

ГЛАВА 2



Объекты и классы

Ключевыми конструкциями объектно-ориентированной технологии являются классы и объекты. В данной главе будет подробно рассмотрен процесс их создания и использования.

2.1. Объявление класса

Класс объявляется при помощи ключевого слова `class`, после которого следует уникальное имя класса и тело класса в фигурных скобках. В теле класса объявляются методы и члены. В листинге 2.1 приводится общий синтаксис объявления класса.

Листинг 2.1. Объявление класса

```
<?php
  class имя_класса
  {
    // Члены и
    // методы класса
  }
?>
```

Важной особенностью PHP является то, что PHP-скрипты могут включаться в документ при помощи тегов `<?php` и `?>`. Один документ может содержать множество включений этих тегов, однако класс должен объявляться в одном неразрывном блоке `<?php` и `?>`. Попытка разорвать объявление класса приводит к генерации интерпретатором ошибки разбора **Parse error: parse error, unexpected ';', expecting T_FUNCTION**.

Так как прерывать объявление класса недопустимо, его не удастся механически разбить и при помощи инструкций include(), include_once(), require(), require_once(). Объявление класса cls в листинге 2.2 является ошибочным.

Замечание

Напомним, что при помощи инструкций include(), include_once(), require(), require_once() можно включать в состав PHP-скриптов другие PHP-скрипты. Это позволяет разбивать объемные многострочные файлы на множество мелких файлов, которые программисту проще воспринять. В отличие от require(), при отсутствии включаемого файла инструкция include() генерирует предупреждение, однако не останавливает работу скрипта, в то время как require() аварийно завершает работу приложения. Допускается множественное включение файлов друг в друга, что может приводить к запутанным ситуациям и многократному включению файлов в приложение. Суффикс once означает, что включаемый файл будет включен лишь один раз, повторный вызов инструкции include_once() или require_once() игнорируется. Это особенно удобно для включения библиотек функций и классов, повторное объявление которых вызывает ошибку.

Листинг 2.2. Недопустимое объявление класса

```
<?php
  class cls
  {
    // Реализация класса во внешнем файле
    include "cls_include.php";
  }
?>
```

Такое поведение может обескуражить опытных программистов, привыкших иметь дело с объектно-ориентированным программированием на других языках. Язык C++, например, позволяет объявлять лишь прототипы методов, вынося реализацию в отдельный файл. Таким образом, внешний программист имеет дело только с интерфейсом класса, и у него отсутствует соблазн разбираться в его внутренней реализации, нарушая принцип инкапсуляции (сокрытия реализации).

Впрочем, имеется способ обойти подобное ограничение, т. к. в методах класса не запрещается использование конструкций include(), include_once(), require(), require_once(). Забегая вперед, представим листинг 2.3, демонстрирующий, как реализация метода test() класса cls() размещается в отдельном файле test_cls.php.

Замечание

Более подробно использование конструкций include(), include_once(), require(), require_once() обсуждается в *разделе 2.4*.

Листинг 2.3. Корректное использование в классе инструкции `include()`

```
<?php
  class cls
  {
    function test()
    {
      // Реализация метода во внешнем файле
      include "test_cls.php";
    }
  }
?>
```

Нельзя обойти стороной размещение в приложении и самих классов. Очевидно, что классы в объектно-ориентированном приложении будут использоваться множеством файлов, поэтому разумно выделить их в отдельный файл, который бы включался в остальные файлы при помощи инструкций `include_once()` или `require_once()`. Важно с самого начала использовать для включения инструкции с суффиксом `once`, чтобы предотвратить повторное включение самого файла. В небольшом прозрачном приложении практически невозможно ошибиться и допустить ошибку, связанную с повторным включением файла с объявлением класса, однако в большие порталы входит множество более мелких приложений, и не исключен случай попытки повторного ввода класса.

Файлы, описывающие классы, могут носить произвольные имена. В отличие от других языков программирования, требующих, чтобы содержащий класс файл имел то же название, что и сам класс, в PHP имя класса никак не связано с именем файла, в котором класс размещен. Объектно-ориентированное приложение включает множество классов, поэтому для простоты ориентирования лучше сразу продумать систему именования файлов. В данной книге каждый класс будет размещаться в отдельном файле, имя которого будет начинаться с ключевого слова `class`, после чего будет следовать название файла. Например, файл для класса из листинга 2.3 можно назвать class.test.php. Позже мы дадим пример механизма наследования, чтобы показать, как класс `derived` наследует класс `base`; имя файла данного класса будем называть class.base.derived.php.

Замечание

PHP предоставляет разработчику автоматическое средство для автоматической загрузки классов в момент создания объектов. Это достигается за счет функции `__autoload()`, которая рассматривается в *разделе 3.6*.

2.2. Абстрактные типы данных. Создание объекта

Согласно одной из концепций объектно-ориентированного программирования, классы выступают как настраиваемые пользователем типы данных. Программист получает возможность разрабатывать программу не при помощи компьютерных абстракций (переменных, функций, потоков и т. п.), а при помощи абстракций предметной области программы.

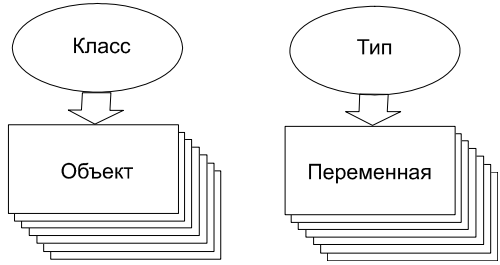


Рис. 2.1. Переменные объявляются при помощи типа, объекты — при помощи класса

Тип переменной является ключевым словом, определяющим область памяти с определенными техническими характеристиками (объем отводимой памяти, способность хранить числа с плавающей точкой, последовательность символов и т. д.). Класс также выступает в качестве своеобразного типа данных, при помощи которого можно объявлять объекты (рис. 2.1), однако в отличие от нескольких предопределенных типов, заложенных в интерпретатор PHP на этапе компиляции, функциональность класса определяет программист, причем количество классов не ограничено.

Аналогично скриптам, которые могут создавать неограниченное количество переменных одного типа, может быть создано неограниченное количество объектов одного и того же класса. В общем случае это может быть массив однотипных объектов.

Объявление объекта класса несколько отличается от объявления переменной. Следует напомнить, что благодаря слабой типизации объявлять переменные и уточнять их тип в PHP вообще не требуется: переменную можно ввести в любой момент, а впоследствии изменить ее тип (листинг 2.4).

Листинг 2.4. В PHP объявление переменной не требуется

```
<?php
  // Указывать тип переменной не обязательно
  $str = "Hello world!";
  $var = "test";
```

```
  // Создаем переменную $test с содержимым "Hello world"
  $$var = $str;
  echo $test;
  // Изменяем тип переменной на числовой
  $var = 5;
  echo "<br>$var";
?>
```

Результатом работы скрипта из листинга 2.4 являются следующие строки:

```
Hello world!
5
```

Для объявления объекта класса следует использовать ключевое слово `new`, которое выделит под него необходимую область памяти. Освобождать данную область памяти самостоятельно (как в языке программирования C++) не требуется, интерпретатор сам ее освободит. Впрочем, если скрипту не хватает памяти, ее можно освободить при помощи конструкции `unset` (см. *раздел 2.17*) в любой момент, не дожидаясь окончания работы скрипта.

Замечание

Оператор `new` является атавизмом, пришедшим в PHP из языка программирования C++. В последнем при помощи этого оператора выделяется память под объекты, которая после завершения работы должна освобождаться при помощи оператора `delete`. В PHP программист напрямую не участвует в процессе управления памятью, однако ключевое слово `new` было оставлено, т. к. PHP не знает заранее размер пользовательского объекта, и ему необходимо явно указать, что сейчас будет производиться создание объекта (а не предопределенной переменной) и требуется найти его класс.

В листинге 2.5 создается пустой класс `emp` и объявляется объект `$obj` данного класса.

Замечание

Обратите внимание, что в PHP после завершающей фигурной скобки класса не требуется точки с запятой (в отличие от C++, где она обязательна).

Листинг 2.5. Создание класса `emp` и объявление объекта `$obj` данного класса

```
<?php
  class emp {}
  $obj = new emp;
?>
```

Как видно из листинга 2.5, при помощи ключевого слова `new` переменной `$obj` присваивается в качестве значения новый объект класса `emp`. После имени

класса emp могут следовать не обязательные в данном случае круглые скобки (листинг 2.6).

Замечание

Имена классов, в отличие от имен переменных и объектов, не зависят от регистра, поэтому можно использовать для объявления классов имена emp, Emp(), EMP(). Однако использование вместо объекта $obj переменной $Obj приведет к ошибке — интерпретатор PHP будет считать, что в программу введена новая переменная.

Листинг 2.6. Альтернативный способ объявления объекта класса

```
<?php
  class emp {}
  $obj = new emp();
?>
```

Объект $obj является обычной переменной PHP, правда, со специфичными свойствами. Как и любую другую переменную, объект можно передавать в качестве параметра функции, использовать как элемент массива или присваивать ему новое значение. В листинге 2.7 приводится пример, в котором объекту $obj по мере выполнения программы присваивается некоторое числовое значение, и $obj становится переменной числового типа.

Листинг 2.7. Объект — это обычная переменная

```
<?php
  class emp {}
  $obj = new emp();
  $obj = 3;
  echo $obj; // 3
?>
```

2.3. Инкапсуляция. Спецификаторы доступа

Как было показано в предыдущем разделе, класс может использоваться без методов и членов, однако пользы в этом случае от него не больше, чем от переменной, которая не может хранить значения. Как правило, классы содержат члены и методы. Часть из них будет использоваться внешним разработчиком, часть предназначена только для внутреннего использования в рамках класса. PHP предоставляет специальные ключевые слова, позволяющие ука-

зать, какие члены и методы доступны извне, а какие нет. Это позволяет реализовать принцип инкапсуляции.

В листинге 2.8 представлен класс "сотрудник" (`employee`), который содержит три члена:

- `$surname` — фамилия сотрудника;
- `$name` — имя сотрудника;
- `$patronymic` — отчество сотрудника.

Замечание

Напомним, что при объектно-ориентированном подходе переменные, объявленные в классе, называются *членами*, а функции — *методами*. Такая избыточная терминология может поначалу показаться раздражающей, однако это необходимо для более четкого различия между переменными и функциями внешними по отношению к классу и входящими в его состав.

Листинг 2.8. Класс `employee`

```
<?php
  class employee
  {
    public $surname;
    public $name;
    public $patronymic;
  }
?>
```

Как видно из листинга 2.8, члены класса `employee` объявляются ключевым словом `public`, которое является *спецификатором доступа*. Дело в том, что все члены и методы класса делятся на закрытые и открытые. *Открытые* члены класса объявляются спецификатором доступа `public` и доступны как методам класса, так и внешнему по отношению к классу коду. *Закрытые* методы и члены класса объявляются при помощи спецификатора `private` и доступны только в рамках класса; обратиться к ним извне невозможно.

Замечание

Помимо спецификаторов `public` и `private` в PHP поддерживается спецификатор `protected`, используемый при наследовании и подробно рассматриваемый в *главе 4*.

Рассмотрим пример. Пусть метод `employee` располагается в файле class.employee.php. В листинге 2.9 объявляется объект `$emp` класса `employee`, при этом члены класса получают необходимые значения и выводятся в окно браузера.

Листинг 2.9. Обращение к членам класса `employee`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();

  // Присваиваем значения членам класса
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";

  // Выводим члены класса
  echo $emp->surname." ".$emp->name." ".$emp->patronymic."<br>"
?>
```

Как видно из листинга 2.9, при обращении к члену класса используется оператор `->`, после которого следует ввести имя этого члена. Заметьте, что при обращении к членам класса не добавляется символ `$`.

Принцип инкапсуляции — это возможность скрыть реализацию объекта. При этом мы избавляем и без того загруженное сознание программиста от лишних деталей. Если разрабатывая класс вы уверены, что внешняя программа не будет пользоваться данным членом, а доступ к нему требуется лишь для методов класса, то такой член следует объявить при помощи спецификатора доступа `private`. В листинге 2.10 приводится пример класса `employee`, в котором помимо имени, фамилии и отчества сотрудника объявляется закрытое поле `age`.

Замечание

Ключевые слова `public`, `private` и `protected`, которые называются *спецификаторами доступа*, введены, начиная с PHP 5; в более ранних версиях для объявления члена класса использовалось ключевое слово `var`. Члены, объявленные с его помощью, были открытыми. В текущих версиях PHP по-прежнему допускается использовать ключевое слово `var` для объявления членов класса, однако это не рекомендуется, а само ключевое слово признано устаревшим. С большой долей вероятности оно будет исключено из PHP 6.

Листинг 2.10. Использование спецификатора `private`

```
<?php
  class employee
```

```
  {
    public $surname;
    public $name;
    public $patronymic;
    private $age;
  }
?>
```

По-прежнему можно обращаться к членам `$surname`, `$name`, `$patronymic`, однако к члену `$age` обратиться не получится (листинг 2.11).

Листинг 2.11. Ошибочное обращение к закрытому члену класса

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");
  // Объявляем объект класса employee
  $emp = new employee();
  // Присваиваем значения членам класса
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  $emp->age = 23; // Ошибка
  // Выводим члены класса
  echo $emp->surname." ".$emp->name." ".$emp->patronymic."<br>"
?>
```

Обращение к закрытому члену класса `$age` завершится ошибкой **Fatal error: Cannot access private property employee::$age**. На рис. 2.2 приводится схематическое изображение процесса доступа к членам объекта, снабженным разными спецификаторами доступа.

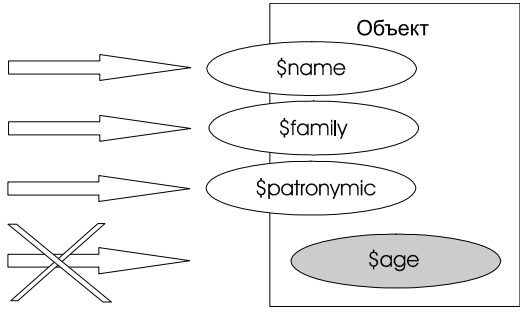


Рис. 2.2. Открытые и закрытые члены объекта

Возникает вопрос: зачем же нужны закрытые члены, если невозможно изменять и считывать их значения из внешней программы? Тому, как можно манипулировать этими членами при помощи методов классов, посвящен следующий раздел.

2.4. Методы класса. Член *$this*

Популярность объектно-ориентированного подхода заключается в том, что классы представляют собой не просто контейнеры для хранения переменных (в качестве таких контейнеров в PHP могут выступать ассоциативные массивы), но также позволяют включать методы, обрабатывающие как открытые, так и закрытые члены класса.

Метод класса представляет собой обычную функцию PHP, которую предваряет один из спецификаторов доступа. В листинге 2.12 приводится пример класса `cls_mth`, в состав которого входит метод `show_message()`. Задача метода сводится к выводу в окно браузера сообщения **Hello world**.

Замечание

В листинге 2.12 можно опустить спецификатор доступа `public` перед методом `show_message()`. В отличие от других объектно-ориентированных языков, если в PHP не уточняется спецификатор, считается, что метод объявлен со спецификатором `public`. Именно потому, что в разных языках программирования используются разные подходы, не рекомендуется опускать спецификатор доступа при объявлении метода.

Листинг 2.12. Объявление метода класса

```
<?php
  class cls_mth
  {
    public function show_message()
    {
      echo "Hello world!";
    }
  }

  $obj = new cls_mth();

  $obj->show_message();
?>
```

В листинге 2.12 метод класса не использует никаких членов, однако, как правило, методы вводят именно для обработки членов класса. Для обращения

к любому элементу класса следует использовать конструкцию `$this->`. Переменная `$this`, которая неявно присутствует в каждом классе, является ссылкой на текущий объект класса. Вернемся к классу `employee`, определенному в листинге 2.10. Класс содержит закрытую переменную `$age`, определяющую возраст сотрудника. Создадим два метода:

- `get_age()` — метод, возвращающий значение закрытого члена `$age`;
- `set_age()` — метод, возвращающий значение закрытого члена `$age` и проверяющий его принадлежность к числовому типу данных и промежутку от 18 до 65.

Замечание

Методы для получения доступа к закрытым переменным через открытые методы класса в объектно-ориентированной модели PHP рассматриваются в *разделе 3.9*.

Листинг 2.13. Использование открытых методов для доступа к закрытому члену класса

```
<?php
  class employee
  {
    // Открытые члены
    public $surname;
    public $name;
    public $patronymic;

    // Открытые методы
    public function get_age()
    {
      return $this->age;
    }
    public function set_age($val)
    {
      $val = intval($val);
      if($val >= 18 && $val <= 65)
      {
         $this->age = $val;
         return true;
      }
      else return false;
    }
```

```
    // Закрытые члены
    private $age;
  }
?>
```

Подход, представленный в листинге 2.13, позволяет удостовериться, что закрытый член `$age` получил корректное значение. Рассмотрим подробнее методы `get_age()` и `set_age()`. Метод `get_age()` не принимает ни одного параметра — единственная его задача обратиться к закрытому члену `$age` и вернуть это значение вызвавшему его коду. На первый взгляд, такой подход может показаться избыточным — вместо непосредственного обращения к члену `$age` вводится функция-посредник, на вызов которой затрачиваются лишние ресурсы. Однако лишние затраты на проектирование методов доступа к закрытым членам, а также вызов функций вместо прямого обращения с лихвой окупаются в дальнейшем. Например, для вычисления текущего возраста сотрудника на основании сведений из базы данных, актуальных на момент заполнения анкеты (вероятно, несколько лет назад), менять весь код, использующий класс `employee`, не придется: достаточно изменить внутреннюю реализацию метода `get_age()`, после чего изменения автоматически отразятся на всей системе.

При проектировании методов, которые обращаются к внутренним членам класса, необходимо следить за тем, что к членам класса следует обращаться через префикс `$this->`, а при обращении к параметрам метода данный префикс не требуется. В большинстве случаев интерпретатор PHP не сообщит о наличии ошибок, если при обращении к внутреннему члену класса не используется префикс `$this->`, а при обращении к параметру метода префикс `$this->` используется. При этом будет создана новая переменная с нулевым значением. Эта ошибка очень распространена и связана с фундаментальной особенностью PHP — отсутствием типизации: использование переменной с любым именем приводит к созданию переменной с этим именем.

Замечание

Для того чтобы заставить PHP сообщать о попытках обращения к необъявленным переменным, необходимо в конфигурационном файле php.ini добавить к директиве `error_reporting` константу `E_NOTICE` для отображения замечаний или включить отображение всех видов ошибок, предупреждений и замечаний, установив значение директивы в `E_ALL`.

Метод `set_age()` из листинга 2.13 принимает единственный параметр `$val`, который при помощи функции `intval()` приводится к числовому типу. После этого осуществляется проверка, принадлежит ли параметр `$val` интервалу от 18 до 65. Если параметр удовлетворяет этому условию, закрытый член класса `$this->age` получает новое значение, а метод возвращает значение `true`, сви-

детельствующее об успешном выполнении операции. Если новое значение не удовлетворяет условию, предъявляемому для члена `$this->age`, метод `set_age()` возвращает значение `false`, говорящее о неудачном завершении операции.

Методы могут обращаться не только к закрытым членам, но и ко всем остальным переменным и методам, в том числе и к открытым членам класса. В листинге 2.14 к рассмотренному ранее классу `employee` добавляются два новых метода:

- `get_info()` — возвращает фамилию, имя и отчество сотрудника;
- `get_full_info()` — используя метод `get_info()`, возвращает фамилию, имя, отчество сотрудника и его возраст, заключенный в скобки.

Листинг 2.14. Дополнительные методы класса `employee`

```
<?php
  class employee
  {
    // Открытые члены
    public $surname;
    public $name;
    public $patronymic;

    // Открытые методы
    public function get_age()
    {
      return $this->age;
    }
    public function set_age($val)
    {
      $val = intval($val);
      if($val >= 18 && $val <= 65)
      {
         $this->age = $val;
         return true;
      }
      else return false;
    }
    public function get_info()
    {
      return $this->surname." ".$this->name." ".$this->patronymic;
    }
```

```
    public function get_full_info()
    {
      return $this->get_info()." (".$this->age.")";
    }

    // Закрытые члены
    private $age;
  }
?>
```

Следует обратить внимание на формирование строк в методах `get_info()` и `get_full_info()`. Строки в PHP объединяются при помощи символа точки (.), при этом все типы, отличные от строкового, приводятся к нему автоматически. Однако это не единственный способ формирования строки: можно прибегнуть к так называемой *интерполяции*, при которой переменная вставляется в строку, заключенную в двойные кавычки. Например, переменная `$name = "Hello"`, подставленная в строку `"$name world!"`, приводит к формированию строки `"Hello world!"`.

Замечание

В одиночных кавычках значение переменной не интерполируется, поэтому строка `'$name world!'` отображается, как есть — `'$name world!'`.

Аналогично можно подставлять в строку значения членов класса. Однако конструкция `$this->имя_члена` достаточно сложна для интерпретации. Для того чтобы интерпретатор мог корректно различить обращение к члену класса из строки, необходимо заключить переменную в фигурные скобки. В листинге 2.15 приводится альтернативная реализация методов `get_info()` и `get_full_info()`.

Листинг 2.15. Альтернативная реализация методов `get_info()` и `get_full_info()`

```
<?php
  class employee
  {
    ...
    public function get_info()
    {
      return "{$this->surname} {$this->name} {$this->patronymic}";
    }
    public function get_full_info()
```

```
    {
      return "{$this->get_info()} ({$this->age})";
    }
    ...
  }
?>
```

Как видно из листинга 2.15, фигурные скобки позволяют интерполировать в строку не только члены класса, но и значения, возвращаемые методами.

В листинге 2.16 приводится пример использования класса `employee` из листинга 2.14.

Листинг 2.16. Использование класса `employee`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();

  // Передаем значения членам класса
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  if(!$emp->set_age(23)) exit("Ошибка вычисления возраста");

  echo $emp->get_full_info(); // Борисов Игорь Иванович (23)
?>
```

Следует обратить внимание на то, что открытые методы и члены расположены в начале класса, а закрытые — в конце. Это негласное правило позволяет сосредоточить внимание программиста в первую очередь на открытом интерфейсе.

К сожалению, в PHP практически отсутствуют механизмы для сокрытия реализации в отдельном файле или использовании прототипов методов, как это принято в C++. Программист, обращающийся к классу для ознакомления с интерфейсом, волей-неволей знакомится и с реализацией, нарушая принципы инкапсуляции. Единственным выходом из сложившейся ситуации является использование конструкции `include` и определение методов класса в отдельных файлах (листинг 2.17). Это позволяет сократить объем главного файла, определяющего класс, и сосредоточить внимание на открытом интерфейсе,

с которым предстоит работать, а не на внутренней реализации каждого из методов, редактировать которые без полного анализа всего класса нежелательно.

Листинг 2.17. Сокрытие реализации при помощи конструкции `include`

```
<?php
  class employee
  {
    // Открытые члены
    public $surname;
    public $name;
    public $patronymic;

    // Открытые методы
    public function get_age()
    {
      require_once("method/class.employee.get_age.php");
    }
    public function set_age($val)
    {
      return require_once("method/class.employee.set_age.php");
    }
    public function get_info()
    {
      return require_once("method/class.employee.get_info.php");
    }
    public function get_full_info()
    {
      return require_once("method/class.employee.get_full_info.php");
    }

    // Закрытые члены
    private $age;
  }
?>
```

Как видно из листинга 2.17, реализация каждого из методов выделена в отдельный файл из папки `method`. При этом методы, которые в предыдущих листингах самостоятельно возвращали значения, теперь возвращают значение, передаваемое им конструкцией `require_once()`.

Конструкции `include()`, `require()`, `include_once()` и `require_once()`, как и функции, позволяют возвращать значения при помощи оператора `return`.

Когда внутри включаемого файла не используется оператор `return`, эти конструкции возвращают `true`, если включаемый файл существует, и `false`, если файл не удалось найти.

В листинге 2.18 приводится содержимое файлов class.employee.get_age.php, class.employee.set_age.php, class.employee.get_info.php и class.employee.get_full_info.php, в которых скрыта реализация методов класса `employee`.

Листинг 2.18. Реализация методов класса `employee`, разбитая по отдельным файлам

```
<?php
  // class.employee.get_age.php
  return $this->age;
?>
<?php
  // class.employee.get_full_info.php
  return "{$this->get_info()} ({$this->age})";
?>
<?php
  // class.employee.get_info.php
  return "{$this->surname} {$this->name} {$this->patronymic}";
?>
<?php
  // class.employee.set_age.php
  $val = intval($val);
  if($val >= 18 && $val <= 65)
  {
     $this->age = $val;
     return true;
  }
  else return false;
?>
```

Как видно из листинга, каждый файл представляет собой фрагмент класса. К сожалению, прямое обращение к такому файлу будет приводить к возникновению ошибки **Fatal error: Using $this when not in object context in** (**Использование $this вне контекста объекта**), т. к. вырванные из контекста участки кода работоспособны лишь в классе. Чтобы избежать появления данной ошибки (по сообщениям об ошибках злоумышленники могут восстановить часть логики скрипта), можно определить в главном файле класса константу при помощи конструкции `define()`, а в начале каждого из файлов с реализацией метода проверять, определена ли константа главного файла, и

при получении отрицательного результата завершать работу скрипта. В листинге 2.19 приводится пример определения константы `CLASS_EMPLOYEE`.

Замечание

Объявить константу можно и как член класса; более подробно константные члены класса описываются в *главе 6*. Однако в данном случае необходима именно внешняя константа, не зависящая от класса, т. к. обращение к файлам реализации осуществляется в обход класса, при котором нельзя применить его штатные средства.

Листинг 2.19. Объявление константы `CLASS_EMPLOYEE`

```
<?php
  define("CLASS_EMPLOYEE", 1);
  class employee
  {
    ...
  }
?>
```

В начале каждого из файлов class.employee.get_age.php, class.employee.set_age.php, class.employee.get_info.php и class.employee.get_full_info.php, реализующих методы класса `employee`, можно при помощи конструкции `defined()` проверять, определена константа `CLASS_EMPLOYEE` или нет. Если константа определена, то нужно продолжить выполнение дальнейших инструкций файла, в противном случае завершить работу метода. В листинге 2.20 приводится возможная реализация для метода `set_age()`.

Листинг 2.20. Файл method/class.employee.set_age.php

```
<?php
  if(!defined("CLASS_EMPLOYEE")) exit("Функцию set_age() следует вызывать
                                      как метод объекта employee");
  $val = intval($val);
  if($val >= 18 && $val <= 65)
  {
     $this->age = $val;
     return true;
  }
  else return false;
?>
```

Теперь прямое обращение к файлу class.employee.set_age.php в обход класса `employee` будет приводить к выводу сообщения: **Функцию set_age() следует вызывать как метод объекта employee**.

Подход, основанный на сокрытии реализации методов в отдельных файлах, не лишен недостатков. Программист вынужден следить за правильностью путей в конструкции include, кроме того, каждый метод требует отдельного файла, в результате чего построение и сопровождение класса становится достаточно трудоемкой задачей. Главный недостаток такого подхода — сложность полного анализа класса. Прикладному программисту, использующему класс, это не требуется, однако разработчику класса (а в данном случае мы выступаем именно как разработчики класса) быстрый доступ и обзор внутренней реализации методов жизненно необходим. Было бы гораздо удобнее использовать один файл для реализации всех методов (как в языке C++), поэтому в данной книге мы практически не будем прибегать к приему разделения реализации и интерфейса класса. Впрочем, обойти данное ограничение можно при помощи динамических членов и классов, рассмотренных в следующем разделе.

2.5. Динамическое создание методов и членов

Язык программирования PHP устроен таким образом, что упоминание переменной в любой части программы приводит к ее автоматическому созданию. С одной стороны, это позволяет не задумываться о распределении памяти, а сосредоточить основное внимание на прикладных аспектах программы. С другой стороны, такое поведение при отключенном выводе замечаний может приводить к возникновению трудно определимых ошибок. Как бы то ни было, возможность создания переменных распространяется и на классы. Упоминание в любом из методов переменной $this->varname приводит к автоматическому созданию члена с именем $varname.

Замечание

Вывод замечаний (Notice) можно отключить, если добавить директиве error_reporting из конфигурационного файла php.ini значение ~E_NOTICE. Включить их вывод можно, добавив значение E_NOTICE. Замечания не являются ошибками и представляют собой своеобразные советы. В любом случае их вывод отключается на реальных серверах, работающих в Интернете.

В листинге 2.21 приводится пример класса employee, содержащий только три члена: $surname, $name и $patronymic. Однако метод set_age() обращается к члену $this->age, что приводит к созданию нового члена класса.

Листинг 2.21. Динамическое создание члена $age

```
<?php
  class employee
```

```
  {
    public $surname;
    public $name;
    public $patronymic;

    public function set_age($val)
    {
      $val = intval($val);
      if($val >= 18 && $val <= 65)
      {
         $this->age = $val;
      }
    }
  }
?>
```

Новая переменная автоматически получает спецификатор доступа `public`, т. е. становится открытой. К сожалению, изменить ее спецификатор доступа не представляется возможным. Досадно, что разработчики PHP подошли к реализации объектно-ориентированной модели достаточно шаблонно, без учета особенностей языка, в частности, полного отсутствия типизации и обязательности объявления переменных. Возможно, это будет устранено в следующих версиях языка.

Помимо динамических переменных имеется возможность создавать динамические методы. Для этого применяется один из специальных методов класса `__call()`.

Замечание

Более подробно метод `__call()` рассматривается в *разделе 3.12*.

Метод `__call()` принимает два параметра: название динамического метода и массив аргументов, переданных динамическому методу. В листинге 2.22 приводится пример использования метода `__call()` для эмуляции метода `set_age()` из листинга 2.21.

Листинг 2.22. Использование метода `__call()`

```
<?php
  class employee
  {
    public $surname;
    public $name;
    public $patronymic;
```

```
    public function __call($method, $parameters)
    {
      if($method == "set_age")
      {
        $parameters[0] = intval($parameters[0]);
        if($parameters[0] >= 18 && $parameters[0] <= 65)
        {
          $this->age = $parameters[0];
        }
      }
    }
  }
?>
```

В листинге 2.23 приводится пример использования полученного класса `employee`.

Листинг 2.23. Вызов динамического класса `set_age()`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();

  $emp->set_age(23);
  echo $emp->age; // 23
?>
```

Несмотря на то что метод `set_age()` явно не определен в классе `employee`, его вызов не приводит к возникновению ошибки **Fatal error: Call to undefined method employee::set_age() (Вызов не определенного метода set_age() класса employee**). Более того, класс, реализующий метод `__call()`, допускает вызов методов с произвольными именем, типом и количеством параметров (листинг 2.24).

Листинг 2.24. Вызов произвольного метода

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
```

```
  // Метод ничего не выполняет,
  // однако его можно вызывать
  $emp->none_method(23, "hello");
?>
```

Таким образом, можно использовать всего один метод `__call()` для эмуляции всех остальных методов класса. Поскольку метод только один, то при разделении интерфейса класса и реализации так, как описано в *разделе 2.4*, потребуется всего два файла.

Следует, однако, очень осторожно пользоваться динамическими членами и методами, т. к. любому программисту достаточно сложно воспринимать такой код. Основное преимущество объектно-ориентированного подхода заключается в четком определении интерфейса, при этом члены и методы доступны внешнему программисту, использующему класс. Для того чтобы разобраться, какие динамические члены и методы доступны, необходимо проанализировать реализацию класса, нарушая тем самым принцип инкапсуляции. При возникновении необходимости использовать динамические компоненты, лучше, если этим классом будут пользоваться другие классы, а не внешний программист.

2.6. Рекурсивные методы

Методы класса, как и обычные функции, могут быть рекурсивными, т. е. позволяют многократно вызывать сами себя. Рассмотрим класс `recurse`, который не содержит членов, а содержит единственный метод `list_struct()` (листинг 2.25).

Замечание

Классы, не содержащие членов и включающие только методы, часто используют как своеобразный контейнер-библиотеку. Это позволяет организовать некоторое подобие пространства имен и исключить конфликт между функциями, имеющими одинаковые имена, но выполняющими разные действия.

Листинг 2.25. Класс `recurse`

```
<?php
  class recurse
  {
    public function list_struct($parameters)
    {
      if(is_array($parameters))
      {
```

```
        foreach($parameters as $val) $this->list_struct($val);
      }
      else echo $parameters."<br>";
    }
  }
?>
```

Метод `list_struct()` принимает единственный параметр. Если в качестве параметра ему передается массив, то для каждого его элемента метод вновь вызывает сам себя; если же параметр является обычной переменной, то он выводится в окно браузера. В листинге 2.26 приводится пример использования объекта класса `recurse`.

Листинг 2.26. Использование объекта класса `recurse`

```
<?php
  require_once("class.recurse.php");

  $obj = new recurse();

  $arr = array("1" => array("11", "12", "13"),
               "2" => array("21", "22", "23"),
               "3" => array("31", "32", "33"));
  $obj->list_struct($arr);
?>
```

Результатом работы скрипта из листинга 2.26 будет следующий набор строк:

```
11
12
13
21
22
23
31
32
33
```

2.7. Является ли переменная объектом?

Как видно из листинга 2.25, чтобы проверить, является ли массивом параметр метода `list_struct()`, используется функция `is_array()`. Функция `is_array()` возвращает `true`, если переданный ей аргумент является массивом, и `false` —

в противном случае. Чтобы выяснить, является ли переменная объектом, также предусмотрена специальная функция — `is_object()`, которая возвращает `true`, если переданный ей аргумент является объектом, и `false` — в противном случае. Более того, члены объектов, так же как и элементы массивов, можно обходить при помощи цикла `foreach`.

Модернизируем класс `recurse` из листинга 2.25 таким образом, чтобы с его помощью можно было распечатывать структуру объекта (листинг 2.27).

Листинг 2.27. Адаптация класса `recurse` для вывода структуры объекта

```
<?php
  class recurse
  {
    public function list_struct($parameters)
    {
      if(is_array($parameters) || is_object($parameters))
      {
        foreach($parameters as $val) $this->list_struct($val);
      }
      else echo $parameters."<br>";
    }
  }
?>
```

Как видно из листинга 2.27, для вывода структуры объекта потребовалось только добавить условие `is_object($parameters)`.

В листинге 2.28 приводится пример вывода при помощи метода `list_struct()` класса `recurse` структуры объекта класса `employee`.

Листинг 2.28. Вывод структуры объекта класса `employee`

```
<?php
  require_once("class.recurse.php");
  require_once("class.employee.php");

  $emp = new employee();
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  $emp->set_age(23);

  $obj = new recurse();
```

```
  $obj->list_struct($emp);
?>
```

Результатом работы скрипта из листинга 2.28 будут следующие строки:

```
Борисов
Игорь
Иванович
```

Следует обратить внимание, что закрытая переменная `age` не выводится.

2.8. Использование методов без объектов

Класс `recurse`, создаваемый нами на протяжении *разделов 2.6* и *2.7*, не содержит членов, и, следовательно, все его объекты идентичны. Более того, для успешного функционирования метода `list_struct()` не требуется никаких членов класса.

Учитывая это, во многих объектно-ориентированных языках программирования, в том числе и в PHP, предусмотрена возможность вызова метода без создания объекта класса. Вызов метода без создания объекта осуществляется при помощи оператора разрешения области видимости `::`. Создадим класс `value`, содержащий один метод `print_value()`, выводящий единственный аргумент в окно браузера (листинг 2.29).

Листинг 2.29. Класс `value`

```
<?php
  class value
  {
    public function print_value($parameters)
    {
      echo $parameters;
    }
  }
?>
```

В листинге 2.30 приводится пример вызова метода `print_value()` класса `value` без создания объекта.

Замечание

Оператор разрешения области видимости `::` используется не только для вызова методов класса без создания объектов, но и для построения наследственных иерархий (см. *главу 4*).

Листинг 2.30. Использование оператора разрешения области видимости ::

```
<?php
  require_once("class.value.php");
  value::print_value("Hello world!");
?>
```

Однако попытка вызова метода `list_struct()` класса `recurse` заканчивается неудачей (листинг 2.31).

Листинг 2.31. Вызов метода `list_struct()` класса `recurse` без создания объекта

```
<?php
  require_once("class.recurse.php");

  $arr = array("1" => array("11", "12", "13"),
               "2" => array("21", "22", "23"),
               "3" => array("31", "32", "33"));

  recurse::list_struct($arr);
?>
```

Попытка вызвать скрипт из листинга 2.31 приведет к возникновению ошибки **Fatal error: Using $this when not in object context (Использование $this вне контекста объекта)**. Дело в том, что метод `list_struct()` вызывает сам себя с использованием ссылки на текущий объект `$this->list_struct()`, что недопустимо, если объект не создан заранее. Выходом из ситуации является вызов метода `list_struct()` с использованием оператора разрешения области видимости `::` (листинг 2.32).

Замечание

Если метод содержит хотя бы одно обращение `$this->`, его вызов без создания объекта всегда будет заканчиваться выводом сообщения об ошибке **Fatal error: Using $this when not in object context**. Следует отметить, что это ошибка исполнения, т. е. метод выполнит все операторы, предшествующие `$this->`. В других объектно-ориентированных языках программирования не допускается вызывать методы без создания объекта, если метод не был объявлен статическим — возможно, в более поздних версиях PHP это правило также будет соблюдаться. Более подробно статические методы обсуждаются в *главе 6*.

Листинг 2.32. Альтернативная реализация метода `list_struct()`

```
<?php
  class recurse
```

```
  {
    public function list_struct($parameters)
    {
      if(is_array($parameters) || is_object($parameters))
      {
        foreach($parameters as $val) recurse::list_struct($val);
      }
      else echo $parameters."<br>";
    }
  }
?>
```

2.9. Дамп объекта

Основным недостатком рассмотренного выше метода `recurse::list_struct()` является то, что он не выводит закрытые члены класса. Тем не менее при отладке Web-приложений очень часто необходимо просмотреть структуру созданного объекта, особенно если объект использует динамические члены.

Замечание

Среди PHP-программистов не распространена практика использования отладчиков (хотя на рынке существуют соответствующие решения), позволяющие пошагово выполнить программу и просмотреть состояние переменных. Это связано с тем, что зачастую Web-приложения выполняются на удаленной машине, где развернуть и использовать отладчик не представляется возможным. Поэтому Web-разработчики вынуждены осваивать методы отладки, не использующие автоматизированные средства.

Для просмотра объектов в PHP предусмотрена функция `print_r()`. В листинге 2.33 с ее помощью выводится структура объекта `$emp` класса `employee`. Для удобства восприятия вызов функции `print_r()` обрамляется HTML-тегами `<pre>` и `</pre>`, которые сохраняют структуру переносов и отступов при отображении результата в браузере.

Листинг 2.33. Получение структуры объекта класса `employee`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
```

```
  $emp->patronymic = "Иванович";
  $emp->set_age(23);

  // Выводим структуру объекта
  echo "<pre>";
  print_r($emp);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.33 являются следующие строки:

```
employee Object
(
    [surname] => Борисов
    [name] => Игорь
    [patronymic] => Иванович
    [age:private] => 23
)
```

Таким образом, метод возвращает все члены объекта, в том числе и закрытые. Закрытые члены помечаются спецификатором `private`.

Замечание

Функцию `print_r()` можно использовать для просмотра не только структуры объектов, но и структуры массивов (как ассоциативных, так и не ассоциативных), а также для вывода значения обычной переменной.

Класс может содержать в своем составе массивы, при этом дамп такого объекта приобретает более сложную структуру. В листинге 2.34 объявляется объект класса `cls`, в качестве членов которого выступает строка `$name` и массив `$arr`.

Листинг 2.34. Использование массива в качестве члена объекта

```
<?php
  class cls
  {
    public $name;
    public $arr;
  }

  $obj = new cls();
  $obj->name = "name";
  $obj->arr = array("first", "second", "third", "fourth");
```

```
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.34 являются следующие строки:

```
cls Object
(
    [name] => name
    [arr] => Array
        (
            [0] => first
            [1] => second
            [2] => third
            [3] => fourth
        )
)
```

Структура дампа объекта и многомерного массива очень похожи; определить, объект это или массив, можно по ключевым словам `Object` (объект) и `Array` (массив), которые предваряют описание структуры.

2.10. Вложенные объекты

При проектировании класса в качестве членов класса могут выступать не только переменные и массивы произвольной степени вложенности, но и другие объекты.

Создадим класс `contract`, моделирующий договор между заказчиком и исполнителем. В качестве членов такого класса могут выступать:

- `begin_date` — дата заключения договора;
- `end_date` — дата окончания действия договора;
- `description` — описание требований;
- `price` — сумма оплаты;
- `client` — заказчик;
- `employee` — сотрудник организации, принявший заказ (далее будем называть его исполнителем).

Даты удобнее всего хранить в виде количества секунд, прошедших с полуночи 1 января 1970 года. "Описание" `description` представляет собой строковую переменную, а "сумма оплаты" `price` — переменную с плавающей точкой.

В качестве "заказчика" client и "исполнителя" employee можно использовать объекты ранее разработанного класса employee. Это позволит сэкономить на разработке, повторно используя ранее созданный класс. В листинге 2.35 приводится возможная реализация класса contract.

Листинг 2.35. Класс contract

```
<?php
  require_once("class.employee.php");

  class contract
  {
    private $begin_date;
    private $end_date;
    private $description;
    private $price;
    public $client;
    public $employee;
    // Метод, инициирующий объект
    public function init_object($begin_date,
                        $end_date,
                        $description,
                        $price,
                        $client_name,
                        $client_surname,
                        $client_patronymic,
                        $employee_name,
                        $employee_surname,
                        $employee_patronymic)
    {
      $this->begin_date           = $begin_date;
      $this->end_date             = $end_date;
      $this->description          = $description;
      $this->price                = $price;
      $this->client               = new employee();
      $this->client->name         = $client_name;
      $this->client->surname      = $client_surname;
      $this->client->patronymic   = $client_patronymic;
      $this->employee             = new employee();
      $this->employee->name       = $employee_name;
      $this->employee->surname    = $employee_surname;
      $this->employee->patronymic = $employee_patronymic;
    }
```

```
    // Методы, возвращающие значения закрытых
    // переменных
    public function get_begin_date()
    {
      return $this->begin_date;
    }
    public function get_end_date()
    {
      return $this->end_date;
    }
    public function get_description()
    {
      return $this->description;
    }
    public function get_price()
    {
      return $this->price;
    }
  }
?>
```

Как можно видеть из листинга 2.35, сначала при помощи конструкции `require_once()` подключается реализация класса `employee`, т. к. в состав класса `contract` входят объекты данного типа. Метод `init_object()` класса `contract` инициализирует члены класса. Для доступа к членам вложенных объектов `client` и `employee` используется комбинация символов `->`. Например, для доступа к фамилии заказчика внутри класса используется конструкция `$this->client->surname`. Степень вложенности объектов не ограничена, поэтому цепочки могут быть сколь угодно длинными. В листинге 2.36 приводится пример объявления и инициализации объекта класса `contract`.

Листинг 2.36. Объявление класса `contract`

```
<?php
  // Подключаем объявление класса
  require_once("class.contract.php");

  // Объявляем объект класса contract
  $cnt = new contract();
  $cnt->init_object(time(), // Текущая дата
                    time() + 86400*30, // Месяц спустя текущей даты
                    "описание контракта",
                    "10000.00",
```

```
                    "Иван",
                    "Иванов",
                    "Иванович",
                    "Петр",
                    "Петров",
                    "Петрович");

  // Вывод дампа объекта
  echo "<pre>";
  print_r($cnt);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.36 являются следующие строки:

```
contract Object
(
    [begin_date:private] => 1174778074
    [end_date:private] => 1177370074
    [description:private] => описание контракта
    [price:private] => 10000.00
    [client] => employee Object
        (
            [surname] => Иванов
            [name] => Иван
            [patronymic] => Иванович
        )
    [employee] => employee Object
        (
            [surname] => Иванов
            [name] => Иван
            [patronymic] => Иванович
        )
)
```

При обращении к открытым членам вложенных классов следует последовательно через оператор -> перечислять все вложенные члены. В листинге 2.37 приводится пример скрипта, выводящего фамилии и инициалы заказчика и исполнителя.

Листинг 2.37. Использование вложенных объектов

```
<?php
  // Подключаем объявление класса
  require_once("class.contract.php");
```

```
  // Объявляем объект класса contract
  $cnt = new contract();
  $cnt->init_object(time(), // Текущая дата
                    time() + 86400*30, // Месяц спустя текущей даты
                    "описание контракта",
                    "10000.00",
                    "Иван",
                    "Иванов",
                    "Иванович",
                    "Петр",
                    "Петров",
                    "Петрович");
  // Клиент
  echo $cnt->client->surname." ".
       $cnt->client->name[0].". ".
       $cnt->client->patronymic[0].".<br>";
  // Исполнитель
  echo $cnt->employee->surname." ".
       $cnt->employee->name[0].". ".
       $cnt->employee->patronymic[0].".<br>";
?>
```

Как видно из листинга 2.37, доступ к членам вложенных объектов осуществляется при помощи цепочки операторов `->`. При получении инициалов строка рассматривается как массив символов, индексация элементов которого начинается с нуля. Таким образом, чтобы получить первый символ строки, следует обратиться к элементу массива с индексом 0.

Замечание

В дампе строки трактуются как строки, а не как массивы.

2.11. Массив объектов

Объекты выступают как обычные переменные, поэтому вполне допускается создание массива объектов. В листинге 2.38 представлен класс `point`, который моделирует точку двумерного пространства. Два его закрытых члена `$X` и `$Y` хранят координаты точки по осям абсцисс и ординат соответственно. При помощи массива объектов `point` можно построить ломаную линию; созданию такого массива и будет посвящен данный раздел.

Листинг 2.38. Реализация класса `point`

```
<?php
  class point
```

```
  {
    // Открытый интерфейс класса
    public function get_point($X, $Y)
    {
      $obj = new point();
      $obj->set_point($X, $Y);
      return $obj;
    }
    public function get_X()
    {
      return $this->X;
    }
    public function get_Y()
    {
      return $this->Y;
    }

    // Закрытые члены и методы класса
    private $X;
    private $Y;
    private function set_point($X, $Y)
    {
      $this->X = $X;
      $this->Y = $Y;
    }
  }
?>
```

Класс `point` устроен достаточно интересно. Объявив объект данного класса, невозможно инициализировать закрытые члены `$X` и `$Y`; единственный метод, позволяющий это сделать, — `set_point()` — также закрыт и может вызываться только внутри класса.

Получить инициализированный объект можно при помощи единственного открытого метода `get_point()`, который принимает два параметра для координат `X` и `Y` соответственно.

Замечание

Подобный прием оформлен в виде паттерна и называется *фабрика объектов. Паттерны* — это именованные приемы объектно-ориентированного программирования, которые широко применяются в промышленном коде.

В листинге 2.39 приводится пример, в котором при помощи метода `get_point()` класса `point` создаются два инициализированных объекта: `point1` и `point2`.

Листинг 2.39. Получение объектов класса `point`

```
<?php
  // Подключаем объявление класса
  require_once("class.point.php");

  $point1 = point::get_point(1, 1);
  $point2 = point::get_point(2, 2);
  echo "{$point1->get_X()} {$point1->get_Y()}<br>";
  echo "{$point2->get_X()} {$point2->get_Y()}<br>";
?>
```

Следует обратить внимание, что закрытым членам `$X` и `$Y` назначить значения можно лишь при создании объекта; в дальнейшем на протяжении всей жизни объекта изменить значения закрытых членов уже не получится, допускается только их чтение при помощи открытых методов `get_X()` и `get_Y()`. Такой подход исключает случайное изменение одного члена объекта (например, `$X`) без изменения второго.

Результатом работы скрипта из листинга 2.39 являются следующие строки:

```
1 1
2 2
```

Сами по себе точки не представляют никакого интереса, однако из массива таких объектов можно построить ломаную линию (листинг 2.40).

Замечание

Следует отметить, что если бы для инициализации элементов массива в листинге 2.40 не использовался метод объекта, то вместо конструкции `point::get_point()` пришлось бы употреблять конструкцию `new point()`.

Листинг 2.40. Массив объектов `point`

```
<?php
  // Подключаем объявление класса
  require_once("class.point.php");

  // Массив объектов point
  $arr = array(point::get_point(1, 1),
               point::get_point(2, 2),
               point::get_point(3, 0),
               point::get_point(4, 5));
```

```
  // Выводим дамп объекта
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.40 является дамп массива, каждый элемент которого является объектом:

```
Array
(
    [0] => point Object
        (
            [X:private] => 1
            [Y:private] => 1
        )
    [1] => point Object
        (
            [X:private] => 2
            [Y:private] => 2
        )
    [2] => point Object
        (
            [X:private] => 3
            [Y:private] => 0
        )
    [3] => point Object
        (
            [X:private] => 4
            [Y:private] => 5
        )
)
```

Полученную ломаную линию можно вывести в виде графика, например, средствами библиотеки GDLib (листинг 2.41). При обращении к объекту следует учитывать, что объект является элементом массива, поэтому сначала указывается его индекс в квадратных скобках и лишь затем при помощи оператора `->` происходит обращение к его методам. Например, для того чтобы получить координату *X* первой точки ломаной следует обратиться к методу `get_x()` первого элемента массива `$arr`: `$arr[0]->get_x()`.

Листинг 2.41. Создание массива объектов `point`

```
<?php
  // Подключаем объявление класса
  require_once("class.point.php");
```

```
  $arr = array(point::get_point(1, 1),
               point::get_point(2, 2),
               point::get_point(3, 0),
               point::get_point(4, 5));

  // Определяем минимальное и максимальное
  // значения по оси абсцисс (X) и ординат (Y)
  $x_min = $x_max = $arr[0]->get_x();
  $y_min = $y_max = $arr[0]->get_y();
  for($i = 0; $i < count($arr); $i++)
  {
    if($x_min > $arr[$i]->get_x()) $x_min = $arr[$i]->get_x();
    if($x_max < $arr[$i]->get_x()) $x_max = $arr[$i]->get_x();
    if($y_min > $arr[$i]->get_y()) $y_min = $arr[$i]->get_y();
    if($y_max < $arr[$i]->get_y()) $y_max = $arr[$i]->get_y();
  }
  // Размер выводимого изображения
  $height = 100; // высота
  $width  = 100; // ширина

  // Создаем полноцветное изображение
  $img = imagecreatetruecolor($width, $width);
  if (!$img) exit("Ошибка создания изображения");
  // Определяем черный цвет
  $black = imagecolorallocate($img, 0, 0, 0);
  // Определяем белый цвет
  $white = imagecolorallocate($img, 255, 255, 255);
  // Заливаем белым цветом фон изображения
  imagefill($img, 0, 0, $white);
  // Рисуем ломанную
  for($i = 0; $i < count($arr) - 1; $i++)
  {
    $x1 = ($arr[$i]->get_x() - $x_min)*$width/($x_max - $x_min);
    $y1 = ($y_max - $arr[$i]->get_y())*$height/($y_max - $y_min);
    $x2 = ($arr[$i + 1]->get_x() - $x_min)*$width/($x_max - $x_min);
    $y2 = ($y_max - $arr[$i + 1]->get_y())*$height/($y_max - $y_min);
    imageline($img, $x1, $y1, $x2, $y2, $black);
  }
  // Выводим изображение в окно браузера
  header("Content-type: " .image_type_to_mime_type(IMAGETYPE_PNG));
  imagepng($img);
?>
```

Для построения графика изображение будущей ломаной следует масштабировать: пересчитать координаты узлов таким образом, чтобы они вписыва-

лись в границы изображения. Для этого следует выяснить минимальное `$x_min` и максимальное `$x_max` значения координат по оси абсцисс, а также минимальное `$y_min` и максимальное `$y_max` значения координат по оси ординат. Ширину изображения `$width` следует разделить на разницу максимального и минимального значений по оси абсцисс (`$x_max – $x_min`), а высоту `$height` — на разницу максимального и минимального значений по оси ординат (`$y_max – $y_min`).

Ломаная рисуется при помощи функции `imageline()`, которая проводит линию цвета `$black` из точки (`$x1`, `$y1`) в точку (`$x2`, `$y2`). Точки вычисляются по текущим значениям массива `$arr` объектов `point` из расчета, что отсчет координат в компьютерных графических библиотеках начинается из верхнего левого угла. Результат работы скрипта из листинга 2.41 представлен на рис. 2.3.

Рис. 2.3. Изображение ломаной линии, координаты узлов которой хранятся в массиве объектов

2.12. Преобразование объекта в массив

В *разделе 2.7* демонстрировалось, что открытые элементы массива могут обходиться при помощи цикла `foreach`. В *разделах 2.9—2.11* было показано, что объекты могут содержать массивы, а массивы использовать в качестве элементов объекты. Получение дампов массивов и объектов аналогичны. Из всего вышесказанного следует, что это достаточно схожие структуры.

Между ними существует еще одна родственная связь: объекты могут преобразовываться в ассоциативные массивы, а массивы, в свою очередь, обратно в объекты. Для этого используется операция приведения типов. Напомним, что операцией приведения типа называется преобразование переменной одного типа, скажем, `float`, в переменную другого типа, например, `int`. Операция осуществляется путем указания перед выражением или переменной в круг-

лых скобках того типа, к которому необходимо привести переменную. Листинг 2.42 содержит пример преобразования числа с плавающей точкой к целому числу: для этого перед преобразуемым значением следует указать конструкцию `(int)`.

Листинг 2.42. Приведение числа с плавающей точкой к целому числу

```
<?php
  $val = 3.14;
  echo $val."<br>"; // 3.14
  $val = (int)$val;
  echo $val."<br>"; // 3
?>
```

На операции приведения типа основаны несколько приемов, например, определение четности или нечетности числа (листинг 2.43).

Листинг 2.43. Определение четности/нечетности числа

```
<?php
  $val = 5;
  if((float)($val/2) — (int)($val/2)) echo "Число $val нечетное";
  else echo "Число $val четное";
?>
```

Частное, полученное от деления нечетного числа на 2, окажется дробным при приведении результата к типу `float` и целочисленным при приведении к типу `int`. Очевидно, что при этом разница между двумя полученными значениями не равна 0, что трактуется как `TRUE`. Если число четное, то оба результата одинаковы, а их разница равна 0, что трактуется как `FALSE`.

Для того чтобы преобразовать объект в ассоциативный массив, в качестве типа в операторе преобразования следует указать ключевое слово `array`. В листинге 2.44 объект `$emp` класса `employee` преобразуется в массив `$arr`.

Листинг 2.44. Преобразование объекта в массив

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
```

```
  $emp->patronymic = "Иванович";
  $emp->set_age(23); // Присваиваем значение
                     // закрытому члену age

  // Преобразуем объект в массив
  $arr = (array)$emp;

  // Выводим структуру массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.44 является следующий дамп ассоциативного массива, в котором роль ключей выполняют имена членов объекта:

```
Array
(
    [surname] => Борисов
    [name] => Игорь
    [patronymic] => Иванович
    [employeeage] => 23
)
```

Следует обратить внимание, что в полученный массив попадают в том числе и закрытые члены. При этом к именам закрытых членов добавляется префикс в виде имени класса, так, закрытый член `age` преобразуется в элемент с ключом `employeeage`.

Если объект содержит в своем составе другие объекты, то при преобразовании его в массив вложенные объекты не подвергаются преобразованию, а соответствующие им элементы массива содержат объекты вложенного типа. Воспользуемся разработанным в *разделе 2.10* классом `contract` для описания договора между заказчиком и исполнителем. В листинге 2.45 показано его преобразование в массив.

Листинг 2.45. Преобразование вложенных объектов в массив

```
<?php
  // Подключаем объявление класса
  require_once("class.contract.php");

  // Объявляем объект класса contract
  $cnt = new contract();
```

```
  $cnt->init_object(time(), // Текущая дата
                    time() + 86400*30, // Месяц спустя текущей даты
                    "описание контракта",
                    "10000.00",
                    "Иван",
                    "Иванов",
                    "Иванович",
                    "Петр",
                    "Петров",
                    "Петрович");

  // Преобразуем объект в массив
  $arr = (array)$cnt;

  // Выводим структуру объекта
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.45 является следующий дамп ассоциативного массива:

```
Array
(
    [contractbegin_date] => 1174865360
    [contractend_date] => 1177457360
    [contractdescription] => описание контракта
    [contractprice] => 10000.00
    [client] => employee Object
        (
            [surname] => Иванов
            [name] => Иван
            [patronymic] => Иванович
            [age:private] =>
        )
    [employee] => employee Object
        (
            [surname] => Петров
            [name] => Петр
            [patronymic] => Петрович
            [age:private] =>
        )
)
```

Как можно заметить, преобразованию в массив подверглись только члены класса `contract` (т. к. все члены кроме вложенных объектов были закрытыми, к имени объекта добавлен префикс `contract`). Вложенные объекты `client` и `employee` остались не затронутыми.

Возможно и обратное преобразование элементов ассоциативного массива в объект. В листинге 2.46 объект класса `employee` преобразуется сначала в массив, а затем снова в объект. Для этого в качестве типа в операторе приведения следует указать ключевое слово `object`.

Листинг 2.46. Преобразование массива в объект

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  $emp->set_age(23); // Устанавливаем закрытый член age

  // Преобразуем объект в массив
  $arr = (array)$emp;

  // Преобразуем массив в объект
  $obj = (object)$arr;

  echo "{$obj->surname} {$obj->name} {$obj->patronymic}";

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.46 являются следующие строки:

```
Борисов Игорь Иванович
stdClass Object
(
    [surname] => Борисов
    [name] => Игорь
    [patronymic] => Иванович
    [age:private] => 23
)
```

Интересно, что член $age автоматически приобретает статус закрытого члена. При этом важно понимать, что полученный объект не является объектом типа employee, хотя и аналогичен ему по структуре.

2.13. Возвращение методом нескольких значений

В отличие от большинства других языков, в PHP методы объекта могут возвращать массивы. Например, рассмотренный ранее класс employee можно снабдить методом get_array(), возвращающим массив, элементами которого являются фамилия, имя и отчество сотрудника (листинг 2.47). Следует отметить, что массив значений формируется динамически при помощи конструкции array().

Листинг 2.47. Метод get_array() возвращает массив значений

```
<?php
  class employee
  {
    public $surname;
    public $name;
    public $patronymic;
    public function get_array()
    {
      return array($this->surname, $this->name, $this->patronymic);
    }
  }
?>
```

В листинге 2.48 демонстрируется использование метода get_array() класса employee. Результат помещается в массив $arr, дамп которого выводится в окно браузера.

Листинг 2.48. Получение массива от метода объекта

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
  $emp->surname    = "Борисов";
```

```
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";

  // Получаем массив значений
  $arr = $emp->get_array();

  // Выводим дамп массива $arr
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.48 будут следующие строки:

```
Array
(
    [0] => Борисов
    [1] => Игорь
    [2] => Иванович
)
```

Работа с массивом не всегда удобна, т. к. в качестве имени всех элементов выступает имя массива, а при получении доступа к отдельным его элементам следует использовать числовые индексы, которые не всегда наглядны. Альтернативным способом приема значений из метода, возвращающего массив, является конструкция `list()`, которая используется в сочетании с оператором = и позволяет автоматически присвоить значения элементов массива каким-либо переменным. В листинге 2.49 демонстрируется использование конструкции `list()` для приема значений из метода `get_array()` класса `employee`.

Листинг 2.49. Использование конструкции `list()`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";

  // Получаем массив значений
  list($surname, $name, $patronymic) = $emp->get_array();
```

```
  // Выводим значения
  echo "$surname $name $patronymic";
?>
```

Как видно из листинга 2.49, полученные при работе метода `get_array()` элементы массива распределяются по переменным `$surname`, `$name` и `$patronymic`. Количество элементов в конструкции `list()` и в принимаемом массиве не обязательно должны совпадать, например, в листинге 2.50 при помощи конструкции `list()` из возвращаемого массива извлекается только фамилия `$surname`, а имя `$name` и отчество `$patronymic` игнорируются.

Листинг 2.50. Количество элементов в массиве и конструкции `list()` может не совпадать

```
<?php
  ...
  // Получаем лишь фамилию, игнорируя имя и отчество
  list($surname) = $emp->get_array();
  ...
?>
```

Возможна также ситуация, когда возвращаемый массив содержит меньше значений, чем количество переменных, перечисленных в конструкции `list()`. В этом случае лишние переменные не инициализируются (это может быть опасным при включенной директиве `register_globals`, т. к. позволяет злоумышленнику заполнять их по своему усмотрению, например, через GET-параметры).

Замечание

Напомним, что GET-параметры — это элементы суперглобального массива `$_GET`, которые автоматически заполняются Web-сервером. Адресная строка для доступа к Web-сайту может иметь вид **http://www.site.ru/index.php?par1=val1&par2=val2**. Часть адреса после знака вопроса ? посвящена GET-параметрам, которые отделяются друг от друга символом амперсанда &. Слева от символа = располагается имя GET-параметра, справа — его значение. Таким образом, для приведенного выше примера в скрипте index.php будут доступны два GET-параметра: `$_GET['par1']` и `$_GET['par2']` со значениями "`val1`" и "`val2`" соответственно.

Конструкция `list()` может применяться не только в объектно-ориентированном программировании или совместно с функциями, но и с обычными массивами. В листинге 2.51 при помощи конструкций `list()` и `array()` решена классическая задача — обмен значений между двумя переменными без использования переменной-посредника.

Листинг 2.51. Обмен значений между двумя переменными с использованием конструкций `list()` и `array()`

```
<?php
  $x = 5;
  $y = 12;

  echo "x = $x<br>y = $y<br>";

  list($y, $x) = array($x, $y);

  echo "<br>x = $x<br>y = $y<br>";
?>
```

Результатом работы скрипта из листинга 2.51 являются следующие строки:

```
x = 5
y = 12

x = 12
y = 5
```

2.14. Необязательные аргументы методов

PHP является слаботипизированным языком программирования, поскольку в нем не реализован механизм перегрузки операторов и методов. Это в том числе означает, что невозможно объявить в классе два или более методов, которые имеют одинаковые названия, но отличаются друг от друга количеством (типом) аргументов или возвращаемых значений. Такое ограничение не всегда удобно при использовании объектно-ориентированного подхода. Тем не менее обойти данное ограничение можно при помощи необязательных аргументов. В языках программирования, аналогичных C, допускается инициализировать аргументы метода — такой инициализированный аргумент можно не указывать при вызове метода, при этом ему будет присвоено значение по умолчанию.

Замечание

Перегрузка методов в PHP возможна, но только в рамках использования специального метода `__call()`, который рассматривается в *разделе 3.12*, а также при построении иерархии наследующих друг другу классов. Более подробно наследование рассматривается в *главе 4*.

В листинге 2.52 приводится возможная реализация класса для описания сотрудника — `employee`, в котором имеется четыре закрытых члена, соответст-

вующих имени (`$name`), фамилии (`$surname`), отчеству (`$patronymic`) и возрасту сотрудника (`$age`). Инициализация членов класса осуществляется при помощи метода `init_object()`, принимающего четыре параметра, из которых только один — `$surname` является обязательным.

Замечание

Необязательные и обязательные параметры не могут следовать друг за другом в произвольном порядке; все необязательные параметры должны быть сосредоточены в конце списка параметров.

Листинг 2.52. Использование необязательных аргументов

```
<?php
  class employee
  {
    public function init_object($surname,
                                $name = "нет данных",
                                $patronymic = "нет данных",
                                $age = "нет данных")
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      $this->age = $age;
    }
    // Остальные методы класса
    ...

    // Закрытые члены класса
    private $surname;
    private $name;
    private $patronymic;
    private $age;
  }
?>
```

Как видно из листинга 2.52, если необязательный параметр не указывается, он принимает в качестве значения строку "нет данных" (листинг 2.53).

Листинг 2.53. Использование необязательных параметров

```
<?php
  require_once("class.employee.php");
```

```
  $emp = new employee();
  $emp->init_object("Борисов", "Игорь", "Иванович");

  echo "<pre>";
  print_r($emp);
  echo "</pre>";

  $obj = new employee();
  $obj->init_object("Егоров");

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.53 являются следующие строки:

```
employee Object
(
    [surname:private] => Борисов
    [name:private] => Игорь
    [patronymic:private] => Иванович
    [age:private] => нет данных
)
employee Object
(
    [surname:private] => Егоров
    [name:private] => нет данных
    [patronymic:private] => нет данных
    [age:private] => нет данных
)
```

Как видно из результата работы скрипта, необязательные параметры, которые не передавались методу `init_object()`, получили значение по умолчанию. Если необязательному параметру нужно передать значение, отличное от установленного по умолчанию, следует инициализировать все предшествующие ему параметры. Таким образом, при передаче возраста сотрудника `$age` необходимо передать также его имя `$name` и отчество `$patronymic`.

2.15. Присваивание одного объекта другому

До этого момента каждый объект рассматривался как самодостаточная структурная единица, допускающая вызов методов объекта и обращение к его открытым членам. Кроме этого, объекты могут взаимодействовать между собой, в частности, объекты можно сравнивать и присваивать друг другу.

Замечание

Объектно-ориентированная модель PHP не предусматривает перегрузку операторов, и программист не может переопределить их поведение для объектов.

Большинство объектов представляют собой довольно сложные конструкции, которые могут содержать массивы или другие объекты; это вызывает необходимость оперировать не самими объектами, а ссылками на области памяти, в которой они хранятся. В соответствии с этим присвоение одного объекта другому приводит к получению не двух независимых объектов, а двух имен переменных, каждая из которых содержит ссылку на одну и ту же область памяти (один и тот же объект). В листинге 2.54 этот эффект демонстрируется на примере класса `cls`, содержащего единственный открытый член `$val`.

Листинг 2.54. Присвоение одного объекта другому не приводит к созданию его копии

```
<?php
  class cls
  {
    public $val;
  }

  $fst = new cls();
  $snd = new cls();

  $fst->val = 100;
  $snd = $fst;
  $snd->val = 200;

  echo $fst->val; // 200
?>
```

В результате работы скрипта из листинга 2.54 будет выведено значение `200`. После того как объект `$fst` присвоен второму объекту `$snd`, изменение одного из объектов приводит к изменению обоих объектов одновременно.

Замечание

Данный эффект можно обойти и получить независимую копию объекта, если прибегнуть к клонированию, более подробно рассматриваемому в *главе 7*.

Присвоение объекта в качестве значения может выполняться не только явным образом при помощи оператора `=`, но и неявно, например, передачей объекта функции в качестве аргумента. Начиная с PHP 5, все объекты (как и массивы) автоматически передаются в функцию по ссылке. Это означает, что

все изменения, которые функция производит с объектом, сохраняются и после ее завершения. В листинге 2.55 приводится пример передачи объекта класса `cls` функции, где происходит изменение состояния объекта.

Листинг 2.55. Передача объекта функции в качестве аргумента

```
<?php
  class cls
  {
    public $val;
  }

  $obj = new cls();
  $obj->val = 100;
  $var = 100;
  change($var, $obj);
  echo $obj->val; // Новое значение функции
  echo $var; // 100

  function change($var, $obj)
  {
    $obj->val = "Новое значение функции";
    $var = "Новое значение функции";
  }
?>
```

Как видно из листинга 2.55, функция `change()` не может изменить значение обычной переменной `$var`, которая передается ей в качестве аргумента, тем не менее, объект `$obj` претерпевает изменения.

Замечание

Следует отметить, что хотя массивы, так же как и объекты, передаются по ссылке, они ведут себя подобно обычным переменным, т. е. не подвергаются изменению во внешнем коде, если их элементы подверглись изменению внутри функции.

2.16. Сравнение объектов друг с другом

При сравнении объектов друг с другом можно использовать один из двух операторов:

- оператор сравнения == — два объекта считаются равными друг другу, если их члены равны друг другу, а сами объекты являются экземплярами одного и того же класса;

- оператор эквивалентности `===` — два объекта считаются эквивалентными, если они ссылаются на один и тот же экземпляр одного и того же класса.

В листинге 2.56 для сравнения используются объекты класса `point`, моделирующего точку в прямоугольной декартовой системе координат.

Замечание

Сравнению подвергаются не только открытые, но и закрытые члены объектов.

Листинг 2.56. Сравнение объектов при помощи оператора ==

```
<?php
  class point
  {
    public $x;
    public $y;
  }

  $fst = new point();
  $fst->x = 100;
  $fst->y = 100;

  $snd = new point();
  $snd->x = 100;
  $snd->y = 100;

  if($fst == $snd) echo "Объекты равны<br>";
  else echo "Объекты не равны<br>";

  $arr = (array)$snd;
  $obj = (object)$arr;

  if($fst == $obj) echo "Объекты равны<br>";
  else echo "Объекты не равны<br>";

  $snd->y = 10;

  if($fst == $snd) echo "Объекты равны<br>";
  else echo "Объекты не равны<br>";
?>
```

В листинге 2.56 рассмотрены три случая:

- сравнение двух объектов одного класса, у которых совпадают значения членов;

- сравнение двух объектов разных классов, у которых совпадают тип и значения членов;
- сравнение двух объектов одного класса, у которых не совпадают значения членов.

В первом случае оператор равенства == возвращает значение `true`, в двух других — значение `false`. Скрипт из листинга 2.56 выводит следующие строки:

```
Объекты равны
Объекты не равны
Объекты не равны
```

Оператор эквивалентности === более строгий по сравнению с оператором равенства. В листинге 2.57 на примере объектов уже упоминавшегося класса `point` демонстрируется использование этого оператора.

Листинг 2.57. Сравнение объектов при помощи оператора ===

```
<?php
  class point
  {
    public $x;
    public $y;
  }

  $fst = new point();
  $fst->x = 100;
  $fst->y = 100;

  $snd = new point();
  $snd->x = 100;
  $snd->y = 100;

  if($fst === $snd) echo "Объекты равны<br>";
  else echo "Объекты не равны<br>";

  $obj = $fst;

  if($fst === $obj) echo "Объекты равны<br>";
  else echo "Объекты не равны<br>";
?>
```

Результатом работы скрипта из листинга 2.57 являются следующие строки:

```
Объекты не равны
Объекты равны
```

Для эквивалентности объектов недостаточно, чтобы они были экземплярами одного класса и имели одинаковые значения; необходимо, чтобы они ссылались на один и тот же объект.

2.17. Уничтожение объекта

Уничтожить объект, как и переменную или массив, можно при помощи конструкции `unset()`. В листинге 2.58 приводится пример уничтожения объекта класса `point`, моделирующего точку в декартовой системе координат.

Листинг 2.58. Уничтожение объекта при помощи конструкции `unset()`

```
<?php
  class point
  {
    public $x;
    public $y;
  }

  $obj = new point();
  $obj->x = 100;
  $obj->y = 100;

  echo "<pre>";
  print_r($obj); // Объект существует
  echo "</pre>";

  unset($obj);

  echo "<pre>";
  print_r($obj); // Объект не существует
  echo "</pre>";
?>
```

Скрипт выведет только один дамп объекта, соответствующий вызову функции `print_r()`, предшествующему конструкции `unset()`, которая производит удаление объекта `$obj`:

```
point Object
(
    [x] => 100
    [y] => 100
)
```

Следует отметить, что если на один и тот же объект ссылается несколько переменных, то уничтожение одной из них не приведет к уничтожению объекта (листинг 2.59).

Листинг 2.59. Удаление одной из ссылок не приводит к уничтожению объекта

```
<?php
  class point
  {
    public $x;
    public $y;
  }

  $obj = new point();
  $obj->x = 100;
  $obj->y = 100;

  $fst = $obj;

  unset($obj);

  echo "<pre>";
  print_r($fst); // Объект существует
  echo "</pre>";

  echo "<pre>";
  print_r($obj); // Объект не существует
  echo "</pre>";
?>
```

Скрипт из листинга 2.59 также выводит только один дамп, соответствующий объекту `$fst`. Объект продолжает существовать до тех пор, пока на него указывает хотя бы одна ссылка.

При помощи оператора `unset()` можно удалять не только объекты целиком, но и отдельные его члены (это особенно актуально в случае динамических членов, см. *раздел 2.5*). В листинге 2.60 демонстрируется удаление одного из членов объекта класса `point`.

Замечание

Использовать конструкцию `unset()` можно как вне объекта, так и внутри него. Правда, передачей конструкции `unset()` ссылки `$this` можно уничтожить только члены объекта, а не сам объект.

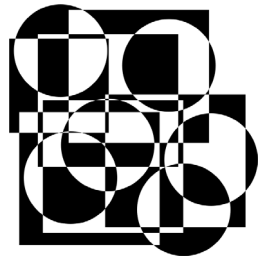
Листинг 2.60. Удаление члена класса

```
<?php
  class point
  {
    public $x;
    public $y;
  }

  $obj = new point();
  $obj->x = 100;
  $obj->y = 100;

  unset($obj->x);

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 2.60 будет следующий дамп объекта `$obj`:

```
point Object
(
    [y] => 100
)
```

ГЛАВА 3



Специальные методы классов

В предыдущей главе были рассмотрены классы и объекты, при помощи которых можно организовать своеобразные контейнеры членов и методов, работающих с членами класса. Однако объектно-ориентированный подход заключается не только в создании удобных контейнеров, но и в сокращении количества связей между классами и объектами. Существует множество приемов, позволяющих разбить программу на несколько слабосвязанных частей.

Характерной особенностью объектно-ориентированного программирования является наличие специальных методов класса, при помощи которых программист может настраивать работу объекта таким образом, чтобы его использование было удобным для внешнего программиста и отнимало как можно меньше его времени. В подавляющем большинстве случаев специальные методы вызываются автоматически в процессе жизнедеятельности объектов.

Замечание

Специальные методы `__clone()`, `__sleep()` и `__wakeup()` рассматриваются в *главе 7*.

Характерной чертой всех специальных методов является то, что они начинаются с двух символов подчеркивания.

Замечание

Полный список вспомогательных функций и их синтаксис можно найти в *приложении 2*.

Помимо специальных методов классов, в данном разделе будут рассмотрены вспомогательные функции, позволяющие проверить наличие классов, методов и членов.

3.1. Конструктор. Метод *__construct()*

Конструктор — это специальный метод класса, который автоматически выполняется в момент создания объекта до вызова всех остальных методов класса. Данный метод используется главным образом для инициализации объекта, обеспечивая согласованность его членов.

Для объявления конструктора в классе необходимо создать метод с именем `__construct()`. В листинге 3.1 приводится пример объявления класса `cls`, содержащего конструктор, который выводит сообщение **Вызов конструктора** и инициализирует закрытый член `$var`.

Замечание

В предыдущих версиях PHP конструктор объявлялся при помощи метода, имя которого совпадало с именем класса. В рамках обратной совместимости в PHP 5 допускается использование старого метода объявления конструктора, однако такой подход признан устаревшим и может быть исключен из следующих версий языка.

Листинг 3.1. Использование конструктора

```
<?php
  class cls
  {
    private $var;
    public function __construct()
    {
      echo "Вызов конструктора<br>";
      $this->var = 100;
    }
  }

  $obj = new cls();

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.1 являются следующие строки:

```
Вызов конструктора
cls Object
(
    [var:private] => 100
)
```

Важно понимать, что вызов конструктора производится автоматически во время выполнения оператора `new`. Это позволяет разработчику класса быть уверенным, что члены класса получат корректную инициализацию. Создавать специальный метод для инициализации объекта считается дурным тоном — внешний программист может забыть его вызвать.

Обычно в объектно-ориентированных языках программирования явный вызов конструктора вообще не допускается, поскольку это противоречит принципу инкапсуляции, однако в PHP конструктор можно вызвать не только в самом классе, но и из внешнего кода (листинг 3.2).

Листинг 3.2. Вызов конструктора внутри класса и из внешнего кода

```
<?php
  class cls
  {
    private $var;
    public function __construct()
    {
      echo "Вызов конструктора<br>";
      $this->var = 100;
    }
    public function anarhist()
    {
      $this->__construct();
    }
  }

  $obj = new cls();
  $obj->__construct();
  $obj->anarhist ();
?>
```

Представленный в листинге 3.2 скрипт вернет следующие строки:

```
Вызов конструктора
Вызов конструктора
Вызов конструктора
```

В первый раз конструктор вызывается неявно при создании объекта `$obj`, во второй раз — явно (метод является открытым), в третий раз вызов происходит из метода `anarhist()`. Следует избегать манипулирования конструктором напрямую. Если одни и те же действия могут выполняться как конструктором, так и каким-либо другим методом, предпочтительнее определить отдельный метод для выполнения этого набора действий. Пусть имеется класс

для описания точки `point`, при объявлении объекта которого два его члена: `$X` и `$Y`, определяющие координаты точки по оси абсцисс и ординат, получают нулевые значения. Пусть класс также содержит метод `set_null()`, позволяющий программисту вернуть координатам исходное нулевое значение. Правильным подходом будет установка нулевого значения в методе `set_null()` и вызов этого метода из конструктора, а не обнуление переменных в конструкторе и вызов его в методе `set_null()` (листинг 3.3). Назначение конструктора заключается в автоматической инициализации объекта при его создании, и его не следует использовать больше ни для каких целей — это лишь запутает и усложнит код. Конструктор может использовать методы класса в своей работе, вызывать конструктор в работе других методов крайне нежелательно.

Замечание

Явный вызов конструктора допускается только при наследовании конструктора родительского класса (см. *главу 4*).

Листинг 3.3. Конструктор может использовать методы класса; явный вызов конструктора нежелателен

```
<?php
  class point
  {
    public function __construct()
    {
      $this->set_null();
    }
    public function set_null()
    {
      $this->X = 0;
      $this->Y = 0;
    }

    private $X;
    private $Y;
  }
?>
```

3.2. Параметры конструктора

Конструктор, как и любой другой метод, может принимать параметры, которые передаются ему оператором `new` в круглых скобках, следующих после имени класса. В листинге 3.4 приводится модификация класса `point`, объект которого инициализируется при помощи конструктора.

Листинг 3.4. Передача параметров конструктору

```
<?php
  class point
  {
    public function __construct($x, $y)
    {
      $this->X = $x;
      $this->Y = $y;
    }
    public fuction get_x()
    {
      return $this->X;
    }
    public fuction get_y()
    {
      return $this->Y;
    }

    private $X;
    private $Y;
  }
?>
```

В листинге 3.5 демонстрируется использование конструктора, инициализирующего закрытые члены класса.

Замечание

Для возврата какого-либо значения в конструкторе можно использовать оператор `return`, однако он не выполняет ничего, кроме досрочной остановки работы метода. В ряде объектно-ориентированных языков программирования, например C++, попытка использования оператора `return` в конструкторе приводит к синтаксической ошибке.

Листинг 3.5. Передача параметров конструктору

```
<?php
  // Подключаем реализацию класса point
  require_once("class.point.php");

  // $obj = new point(); // Вывод предупреждения
  $obj = new point(10, 20);
  echo $obj->get_x()." ".$obj->get_y(); // 10 20
?>
```

Важно отметить, что если указать параметры при объявлении объекта, то интерпретатор PHP выведет в окно браузера предупреждение **Missing argument 1 for point::__construct() (Пропущен первый аргумент для конструктора класса point)**.

PHP не поддерживает перегрузку методов — создание нескольких разных конструкторов (или других методов) с разным количеством аргументов. Однако можно обойти это ограничение, передавая конструктору несколько аргументов, не все из которых обязательны. В листинге 3.6 класс `point` переработан таким образом, что может принимать:

- два аргумента — в этом случае значения закрытых членов `$X` и `$Y` определяются внешним программистом;
- один аргумент — при этом внешний программист определяет только значение закрытого члена `$X`, а член `$Y` получает нулевое значение;
- ни одного аргумента — оба закрытых члена `$X` и `$Y` получают нулевые значения.

Листинг 3.6. Использование параметров по умолчанию

```
<?php
  class point
  {
    public function __construct($x = 0, $y = 0)
    {
      $this->X = $x;
      $this->Y = $y;
    }
    public function get_x()
    {
      return $this->X;
    }
    public function get_y()
    {
      return $this->Y;
    }

    private $X;
    private $Y;
  }
?>
```

Если один из параметров конструктору не передается, он получает значение по умолчанию.

3.3. Закрытый конструктор

Во всех предыдущих примерах конструктор автоматически снабжался спецификатором доступа `public` и был доступен внешнему программисту. Что произойдет, если конструктор снабдить атрибутом `private`? Объект не создастся при помощи оператора `new`, т. к. попытка закончится ошибкой **Fatal error: Call to private point::__construct() from invalid context**. Однако это вовсе не означает, что объект такого класса вообще невозможно создать: можно воспользоваться конструктором при помощи открытого метода данного класса. Перепишем класс `point`, моделирующий точку в двумерной декартовой системе координат, так, чтобы он содержал закрытый конструктор (листинг 3.7).

Листинг 3.7. Закрытый конструктор

```
<?php
  class point
  {
    public function get_point($x = 0, $y = 0)
    {
      return new point($x, $y);
    }
    public function get_x()
    {
      return $this->X;
    }
    public function get_y()
    {
      return $this->Y;
    }

    private $X;
    private $Y;
    private function __construct($x = 0, $y = 0)
    {
      $this->X = $x;
      $this->Y = $y;
    }
  }
?>
```

Как видно из листинга 3.7, класс `point` содержит дополнительный метод `get_point()`, который и вызывает конструктор. В листинге 3.8 демонстрируется использование класса `point` с закрытым конструктором.

Замечание

Получение объекта при помощи его метода уже использовалось ранее в *разделе 2.11*. Приемы и алгоритмы, связанные с возвращением объектов при помощи одного из методов, называются *фабричными*.

Листинг 3.8. Получение объекта при помощи метода класса

```
<?php
  // Подключаем реализацию класса point
  require_once("class.point.php");

  $obj = point::get_point();
  echo $obj->get_x()." ".$obj->get_y(); // 0 0
?>
```

Зачем может потребоваться инициализация объекта в обход оператора `new`? Главным образом, для увеличения гибкости программы. Пусть, например, в большом вычислительном приложении создаются десятки тысяч точек в двумерной декартовой системе координат. В какой-то момент (как правило, ближе к концу сдачи проекта) принимается решение добавить новый класс `point3()` для трехмерной системы координат, конструктор которого принимает три координаты: `$X`, `$Y` и `$Z`. Просто заменить класс `point` новым нельзя: часть точек в проекте остается двумерной, а двумерные и трехмерные точки следует рассматривать как несовместимые типы, т. е. их классы должны различаться. Если для создания класса на протяжении всего проекта использовался оператор `new`, то придется просмотреть и отредактировать все точки создания объектов `point`. Если же используется фабричный метод, можно переопределить только метод `get_point()` (листинг 3.9).

Листинг 3.9. Переопределение логики фабричного метода

```
<?php
  class point
  {
    public function get_point($x = 0, $y = 0, $z = "none")
    {
      if($z == "none") return new point($x, $y);
      else return new point3($x, $y, $z);
    }
    public function get_x()
    {
      return $this->X;
    }
```

```
    public function get_y()
    {
      return $this->Y;
    }

    private $X;
    private $Y;
    private function __construct($x = 0, $y = 0)
    {
      $this->X = $x;
      $this->Y = $y;
    }
  }
?>
```

Как видно из листинга 3.9, в метод `get_point()` добавлен третий необязательный параметр для координаты Z. Это достаточно просто сделать, а на существующих вызовах наличие еще одного, необязательного параметра не отразится, т. к. для создания двумерных точек используются только первые два. Если параметр `$z` не задается пользователем, он получает значение `none`, а метод `get_point()` создает объект класса `point`. Если же параметр задан, создается объект класса `point3`, и далее класс `point` не участвует в работе объекта.

Замечание

Представленный прием сокрытия создания объекта также является примером инкапсуляции, только на более высоком уровне абстракции.

В полной мере достоинства данного приема можно оценить при использовании интерфейсов и наследования (см. *главу 4*).

3.4. Деструктор. Метод *__destruct()*

Деструктор — это специальный метод класса, который автоматически выполняется в момент уничтожения объекта. Данный метод вызывается всегда самым последним и используется главным образом для корректного освобождения занятых в конструкторе ресурсов.

Для объявления деструктора в классе необходимо создать метод с именем `__destruct()`. В листинге 3.10 приводится пример объявления класса `cls`, конструктор которого выводит сообщение **Вызов конструктора**, метод `print_msg()` — сообщение **Вызов метода**, а деструктор — **Вызов деструктора**.

Замечание

Деструктор появился только в версии PHP 5.0.0. Допускается объявление закрытого деструктора, однако при этом попытка уничтожения объекта заканчивается выводом предупреждения **Warning: Call to private point::__destruct() from context '' during shutdown ignored**.

Листинг 3.10. Использование деструктора

```
<?php
  class cls
  {
    public function __construct()
    {
      echo "Вызов конструктора<br>";
    }
    public function print_msg()
    {
      echo "Вызов метода<br>";
    }
    public function __destruct()
    {
      echo "Вызов деструктора<br>";
    }
  }
?>
```

В листинге 3.11 приводится пример создания объекта класса `cls`.

Листинг 3.11. Создание объекта, в котором реализован деструктор

```
<?php
  // Подключаем реализацию класса cls
  require_once("class.cls.php");

  $obj = new cls();
  $obj->print_msg();
  echo "Произвольный текст<br>";
?>
```

Скрипт, представленный в листинге 3.11, выводит в окно браузера следующие строки:

```
Вызов конструктора
Вызов метода
Произвольный текст
Вызов деструктора
```

Как можно видеть, деструктор выполняется в последнюю очередь и уничтожает объект при завершении работы скрипта. В отличие от других языков программирования, в PHP область существования объекта не определяется фигурными скобками, т. е. объект, созданный при выполнении кода внутри фигурных скобок, продолжает существовать и после выхода из них. Однако создание другого объекта с тем же именем приводит к вызову деструктора (листинг 3.12).

Листинг 3.12. Поведение деструктора при создании одноименного объекта

```
<?php
  // Подключаем реализацию класса cls
  require_once("class.cls.php");

  for($i = 0; $i < 3; $i++)
  {
    $obj = new cls();
  }
  echo "Произвольный текст<br>";
?>
```

Результатом работы скрипта из листинга 3.12 будут следующие строки:

```
Вызов конструктора
Вызов конструктора
Вызов деструктора
Вызов конструктора
Вызов деструктора
Произвольный текст
Вызов деструктора
```

Следует отметить, что при создании объекта сначала вызывается конструктор, создающий новый объект, затем ссылка переключается со старого объекта на новый, и только после этого вызывается деструктор. Этот порядок работы необходимо учитывать, если для создания нового объекта необходимо, чтобы старый объект освободил ресурсы (например, закрыл файл).

Явно вызывать деструктор не следует, поскольку в этом случае он будет вызван дважды. В листинге 3.13 приводится пример некорректного, явного вызова деструктора.

Замечание

Во многих объектно-ориентированных языках программирования явный вызов деструктора запрещен.

Листинг 3.13. Некорректный, явный вызов деструктора

```
<?php
  // Подключаем реализацию класса cls
  require_once("class.cls.php");

  for($i = 0; $i < 3; $i++)
  {
    $obj = new cls();
    $obj->__destruct();
  }
  echo "Произвольный текст<br>";
?>
```

Результатом работы скрипта из листинга 3.13 будут следующие строки:

```
Вызов конструктора
Вызов деструктора
Вызов конструктора
Вызов деструктора
Вызов деструктора
Вызов конструктора
Вызов деструктора
Вызов деструктора
Произвольный текст
Вызов деструктора
```

Как можно заметить, для каждого из объектов деструктор выполняется по два раза, даже если повторный вызов деструктора обработан в классе; такого использования деструктора внешний программист не ожидает, и разбор приложения может отнять у него больше времени, чем обычно. Для однозначного вызова деструктора лучше воспользоваться конструкцией `unset()`, уничтожающей объект (см. *раздел 2.17*). В листинге 3.14 приводится пример явного уничтожения объектов перед созданием нового объекта с таким же именем.

Листинг 3.14. Использование метода `unset()`

```
<?php
  // Подключаем реализацию класса cls
  require_once("class.cls.php");

  for($i = 0; $i < 3; $i++)
  {
    $obj = new cls();
```

```
    unset($obj);
  }
  echo "Произвольный текст<br>";
?>
```

Результатом работы скрипта из листинга 3.13 будут следующие строки:

```
Вызов конструктора
Вызов деструктора
Вызов конструктора
Вызов деструктора
Вызов конструктора
Вызов деструктора
Произвольный текст
```

3.5. Создание реальных объектов

Приведенные ранее примеры объектов мало применимы на практике. Рассмотрим реальные объекты, которые встречаются в промышленном коде.

Обычно конструктор и деструктор работают в паре, например, в конструкторе открывается файл или сетевое соединение, а в деструкторе — закрывается. В листинге 3.15 демонстрируется класс `file_access`, в конструкторе которого открывается файл, а многократный вызов метода `readline()` позволяет читать содержимое файла строчка за строчкой, пока не будет достигнут конец файла. Как только достигается конец файла, курсор снова устанавливается в начало файла. Таким образом происходит зацикливание файла, и при помощи метода `readline()` можно бесконечно читать строки файла. В деструкторе закрывается открытый при создании объекта файл.

Листинг 3.15. Класс, выполняющий доступ к файлу

```
<?php
  class file_access
  {
    public function __construct($filename)
    {
      // Открываем текстовый файл
      // для чтения
      $this->fd = fopen($filename, "r");
    }

    public function readline()
    {
```

```
      // Читаем 10000 символов; на самом деле
      // функция fread() прекращает чтение
      // символов, если находит перевод строки
      $str = fgets($this->fd, 10000);
      // Если достигнут конец файла, устанавливаем
      // курсор на начало
      if(feof($this->fd)) fseek($this->fd, 0);
      // Возвращаем прочитанную строку
      return $str;
    }

    public function __destruct()
    {
      // Закрываем файл
      fclose($this->fd);
    }

    // Файловый дескриптор
    private $fd;
  }

  $obj = new file_access("text.txt");
  for($i = 0; $i < 5; $i++) echo "{$obj->readline()}<br>";
?>
```

Класс, представленный в листинге 3.15, является прекрасной иллюстрацией, как *не следует* проектировать объектно-ориентированные системы. Данный код можно переписать в процедурном стиле (листинг 3.16), при этом объем кода значительно уменьшается, становится более легким для восприятия, а его логика не разбросана по нескольким методам, а сосредоточена в одном месте.

Листинг 3.16. Процедурный подход для доступа к файлу

```
<?php
   // Открываем текстовый файл для
   // чтения
   $fd = fopen("text.txt", "r");
   for($i = 0; $i < 5; $i++)
   {
     // Читаем 10000 символов; на самом деле
     // функция fread() прекращает чтение
     // символов, если находит перевод строки
     echo fgets($fd, 10000)."<br>";
```

```
      // Если достигнут конец файла, устанавливаем
      // файловый курсор на начало
      if(feof($fd)) fseek($fd, 0);
   }
   // Закрываем файл
   fclose($fd);
?>
```

Дело в том, что интерфейс доступа к файлам отточен и доведен до совершенства, и сказать здесь новое слово при помощи объектно-ориентированной технологии не удастся уже потому, что она призвана моделировать объекты реального, а не компьютерного мира. С ее помощью следует описывать структуры, которые не существуют и не могут существовать в языке программирования (люди, договоры, транспортные средства и т. п.), тогда как компьютерные объекты (переменные, файлы, области памяти, базы данных и т. п.) моделировать не нужно — они уже снабжены стандартным интерфейсом, знакомым каждому программисту. Очевидно, что если ввести для этих целей нестандартный класс, у стороннего программиста уйдет дополнительное время, чтобы понять, как он работает — налицо снижение читабельности кода.

Таким образом, использование объекта оправдано для создания отсутствующей в языке структуры, наличие которой позволило бы упростить разработку всего приложения.

Создадим классы "договор" (contract), разработка которого была начата в *разделе 2.10*, "заказчик" и "исполнитель". Вся информация о вновь регистрируемых заказчиках и исполнителях будет размещаться в отдельных строках файлов client.txt и employee.txt соответственно. Фамилию, имя или отчество будет предварять уникальный номер заказчика или исполнителя. При заключении договора в файл contract.txt помещается новая запись, содержащая текущую дату, а также уникальные номера заказчика и исполнителя. Если заказчик или исполнитель уже зарегистрированы в файлах client.txt и employee.txt, то новой записи не создается, а используется номер существующей.

Замечание

Каждая запись о заказчике или исполнителе располагается на новой строке файла, а отдельные поля в строке разделяются при помощи символа вертикальной черты |.

Таким образом создается небольшая база данных, в которой фиксируются заключаемые договоры — это позволит избавить программиста от постоянного слежения за журналированием договоров, а также от регистрации новых

заказчиков и исполнителей. Реализация базы данных сокрыта от внешнего программиста, а при увеличении нагрузки ее можно конвертировать в полноценную СУБД, сохранив интерфейс класса `contract`.

Как и в *разделе 2.10*, для реализации класса `contract` удобно разработать вспомогательные классы `client` и `employee`. В листинге 3.17 представлен класс для описания исполнителя `employee` (аналогичный по реализации класс можно использовать и для класса заказчика `client`).

Листинг 3.17. Реализация класса `employee`

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic)
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      // Проверяем, зарегистрирован ли в файле employee.txt
      // исполнитель с такими же фамилией, именем и отчеством
      $content = file_get_contents("employee.txt");
      // Если исполнителя нет в файле, добавляем
      if(strpos($content, "$surname|$name|$patronymic") === false)
      {
        // Определяем максимальный номер (предполагается,
        // что номера расположены равномерно от минимального
        // к максимальному)
        $pattern = "#([\d]+).+?$#i";
        if(preg_match($pattern, $content, $out))
        {
          $this->number = $out[1] + 1;
        }
        else $this->number = 1;
        // Производим запись в файл
        $fd = fopen("employee.txt", "a");
        $str = "{$this->number}|".
               "{$this->surname}|".
               "{$this->name}|".
               "{$this->patronymic}\r\n";
        fwrite($fd, $str);
        fclose($fd);
      }
```

```
      // Если исполнитель уже зарегистрирован в файле,
      // извлекаем его уникальный номер
      else
      {
        $pattern = "#([\d]+)\|$surname\|$name\|$patronymic#i";
        if(preg_match($pattern, $content, $out))
        {
          $this->number = $out[1];
        }
      }
    }
    public function get_surname()
    {
      return $this->surname;
    }
    public function get_name()
    {
      return $this->name;
    }
    public function get_patronymic()
    {
      return $this->patronymic;
    }
    public function get_number()
    {
      return $this->number;
    }

    private $surname;
    private $name;
    private $patronymic;
    private $number;
  }
?>
```

Класс содержит четыре закрытых члена для записи фамилии `$surname`, имени `$name`, отчества `$patronymic` и уникального номера исполнителя `$number`. Первые три переменные получают значения при помощи конструктора; уникальный номер исполнителя либо извлекается из файла employee.txt, если такой сотрудник уже зарегистрирован, либо ему присваивается новый номер, равный максимальному номеру из employee.txt, увеличенному на единицу. В последнем случае в файл employee.txt помещается новая запись. В листинге 3.18 демонстрируется использование класса `employee`.

Листинг 3.18. Использование класса `employee`

```
<?php
  // Подключаем реализацию класса
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");
  echo "<pre>";
  print_r($obj);
  echo "<pre>";
  $obj = new employee("Корнеев", "Иван", "Григорьевич");
  echo "<pre>";
  print_r($obj);
  echo "<pre>";
  $obj = new employee("Борисов", "Игорь", "Иванович");
  echo "<pre>";
  print_r($obj);
  echo "<pre>";
?>
```

Результатом работы скрипта из листинга 3.18 будут следующие строки:

```
employee Object
(
    [surname:private] => Борисов
    [name:private] => Игорь
    [patronymic:private] => Иванович
    [number:private] => 1
)
employee Object
(
    [surname:private] => Корнеев
    [name:private] => Иван
    [patronymic:private] => Григорьевич
    [number:private] => 2
)
employee Object
(
    [surname:private] => Борисов
    [name:private] => Игорь
    [patronymic:private] => Иванович
    [number:private] => 1
)
```

При этом файл employee.txt будет содержать всего две записи:

```
1|Борисов|Игорь|Иванович
2|Корнеев|Иван|Григорьевич
```

Как видно из результатов применения скрипта, создание любого уникального объекта класса `employee` приводит к его регистрации в файле employee.txt и присвоению уникального номера данному объекту. При этом создание объекта, равного уже имеющемуся, не приводит к возникновению дублирующей записи в файле employee.txt.

Объект для описания заказчика `client` строится по той же схеме, что и объект `employee`, с той разницей, что фамилия, имя и отчество заказчика помещаются в файле client.txt.

Теперь, когда вся подготовительная работа завершена, можно приступить к созданию конечного класса "договор" (`contract`) (листинг 3.19).

Листинг 3.19. Класс `contract`

```
<?php
  require_once("class.employee.php");
  require_once("class.client.php");

  class contract
  {
    public $client;
    public $employee;
    // Метод, инициирующий объект
    public function __construct($begin_date,
                          $end_date,
                          $description,
                          $price,
                          $client_name,
                          $client_surname,
                          $client_patronymic,
                          $employee_name,
                          $employee_surname,
                          $employee_patronymic)
    {
      $this->begin_date  = $begin_date;
      $this->end_date    = $end_date;
      $this->description = $description;
      $this->price       = $price;
```

```
      $this->client      = new client($client_name,
                                      $client_surname,
                                      $client_patronymic);
      $this->employee    = new employee($employee_name,
                                        $employee_surname,
                                        $employee_patronymic);
      // Помещаем данные в файл contract.txt
      $fd = fopen("contract.txt", "a");
      $str = "{$this->client->get_number()}|".
             "{$this->employee->get_number()}|".
             "{$this->begin_date}|".
             "{$this->end_date}|".
             "{$this->description}|".
             "{$this->price}\r\n";
      fwrite($fd, $str);
      fclose($fd);
    }
    // Методы, возвращающие значения закрытых
    // переменных
    public function get_begin_date()
    {
      return $this->begin_date;
    }
    public function get_end_date()
    {
      return $this->end_date;
    }
    public function get_description()
    {
      return $this->description;
    }
    public function get_price()
    {
      return $this->price;
    }
    private $begin_date;
    private $end_date;
    private $description;
    private $price;
  }
?>
```

3.6. Автозагрузка классов. Функция *__autoload()*

Обычно класс оформляется в виде отдельного файла, который вставляется в нужном месте при помощи конструкции `require_once()` (см. *раздел 2.1*). Если используется большое количество классов, то в начале скрипта выстраивается целая вереница конструкций `require_once()`, что может быть не очень удобно, особенно если путь к классам приходится часто изменять. Начиная с PHP 5, разработчику предоставляется специальная функция `__autoload()`, которая позволяет задать путь к директории с классами и автоматически подключать классы при обращении к ним в теле программы. Данная функция принимает в качестве единственного параметра имя класса.

Пусть файлы классов `employee`, `client` и `contract` (см. *раздел 3.5*) расположены в директории class, тогда их автоматическая загрузка может выглядеть так, как это представлено в листинге 3.20.

Замечание

Функция `__autoload()` не является методом класса — это независимая функция, которую PHP-разработчик может перегружать.

Листинг 3.20. Автозагрузка классов

```
<?php
  // Функция автозагрузки классов
  function __autoload($classname)
  {
    require_once("class/class.$classname.php");
  }

  $emp = new client("Борисов", "Игорь", "Иванович");
  $cnt = new employee("Корнеев", "Иван", "Григорьевич");
  $obj = new contract(time(),
                      time() + 30*24*60*60,
                      "Описание",
                      "10000",
                      "Борисов",
                      "Игорь",
                      "Иванович",
                      "Корнеев",
                      "Иван",
                      "Григорьевич");
?>
```

Важно отметить, что в начале файла class/class.contract.php уже не потребуется включать файлы class.employee.php и class.client.php: метод `__autoload()` позаботится об их загрузке.

Можно обойтись без функции `__autoload()` и подключить каждый из классов индивидуально. В листинге 3.21 приводится альтернативный способ подключения классов.

Листинг 3.21. Альтернативный способ подключения классов

```
<?php
  require_once("class/class.employee.php");
  require_once("class/class.client.php");
  require_once("class/class.contract.php");

  $emp = new client("Борисов", "Игорь", "Иванович");
  $cnt = new employee("Корнеев", "Иван", "Григорьевич");
  $obj = new contract(time(),
                      time() + 30*24*60*60,
                      "Описание",
                      "10000",
                      "Борисов",
                      "Игорь",
                      "Иванович",
                      "Корнеев",
                      "Иван",
                      "Григорьевич");
?>
```

3.7. Проверка существования класса

Проверка наличия доступа к классу из текущей точки кода осуществляется при помощи функции `class_exists()`, которая имеет следующий синтаксис:

```
bool class_exists($class [, $autoload])
```

Функция принимает в качестве параметра строку с именем класса `$class` и возвращает `true`, если класс доступен для объявления, и `false` — в противном случае. Если функции передается необязательный параметр `$autoload`, равный `true`, она пытается загрузить интерфейс при помощи функции `__autoload()` (см. *раздел 3.6*); если параметр равен `false`, такой попытки не производится.

В листинге 3.22 приводится пример использования функции `class_exitsts()`.

Листинг 3.22. Использование функции `class_exitsts()`

```
<?php
  require_once("class/class.employee.php");
  require_once("class/class.client.php");
  require_once("class/class.contract.php");

  if(class_exists("employee"))
    echo "Класс employee существует<br>";
  else
    echo "Класс employee не существует<br>";

  if(class_exists("client"))
    echo "Класс client существует<br>";
  else
    echo "Класс client не существует<br>";

  if(class_exists("contract"))
    echo "Класс contract существует<br>";
  else
    echo "Класс contract не существует<br>";

  if(class_exists("none"))
    echo "Класс none существует<br>";
  else
    echo "Класс none не существует<br>";
?>
```

В результате работы скрипта из листинга 3.22 на экран будут выведены следующие строки:

```
Класс employee существует
Класс client существует
Класс contract существует
Класс none не существует
```

Однако следует крайне осторожно использовать функцию `class_exists()` совместно с `__autoload()`, т. к. она пытается автоматически обратиться к классу, в том числе и с несуществующим именем. Если работа скрипта должна продолжаться вне зависимости от существования класса, необходимо заменить конструкцию `require_once()` на `include_once()`, (поскольку, как было указано ранее, конструкции `require()` и `require_once()` останавливают работу скрипта, если не находят требуемый файл), а перед ее вызовом поместить символ @, который подавляет вывод предупреждений (листинг 3.23).

Листинг 3.23. Совместное использование функций `__autoload()` и `class_exists()`

```
<?php
  function __autoload($classname)
  {
    @include_once("class/class.$classname.php");
  }

  if(class_exists("employee"))
    echo "Класс employee существует<br>";
  else
    echo "Класс employee не существует<br>";

  if(class_exists("client"))
    echo "Класс client существует<br>";
  else
    echo "Класс client не существует<br>";

  if(class_exists("contract"))
    echo "Класс contract существует<br>";
  else
    echo "Класс contract не существует<br>";

  if(class_exists("none"))
    echo "Класс none существует<br>";
  else
    echo "Класс none не существует<br>";
?>
```

Результат работы скрипта из листинга 3.23 полностью совпадает с результатом работы скрипта из листинга 3.22.

Иногда сложно проверить наличие каждого конкретного класса; в этом случае удобнее получить список всех доступных классов, воспользовавшись для этого функцией `get_declared_classes()`, которая не имеет параметров и возвращает массив, содержащий все доступные классы. В листинге 3.24 демонстрируется использование данной функции.

Листинг 3.24. Использование класса `get_declared_classes()`

```
<?php
  $arr = get_declared_classes();
  echo "<pre>";
```

```
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.24 будет объемный список с предопределенными классами из ядра PHP и различных расширений. Данный список может варьироваться в зависимости от того, какие расширения подключены в данный момент.

Замечание

Некоторые предопределенные классы рассматриваются в *приложении 1*.

```
Array
(
    [0] => stdClass
    [1] => Exception
    [2] => ErrorException
    [3] => COMPersistHelper
    [4] => com_exception
    [5] => com_safearray_proxy
    [6] => variant
    [7] => com
    [8] => dotnet
...
    [96] => mysqli_sql_exception
    [97] => mysqli_driver
    [98] => mysqli
    [99] => mysqli_warning
    [100] => mysqli_result
    [101] => mysqli_stmt
    [102] => PDFlibException
    [103] => PDFlib
)
```

3.8. Определение принадлежности объекта к классу

Определить, к какому классу принадлежит текущий объект, можно при помощи функции `get_class()`, которая имеет следующий синтаксис:

```
string get_class($obj)
```

Функция принимает в качестве единственного параметра объект `$obj` и возвращает строку с именем класса, к которому принадлежит объект. В листин-

ге 3.25 приводится пример скрипта, который создает массив случайных объектов классов `employee`, `client` и `contract`, после чего массив обходится в цикле и для каждого его элемента выводится имя класса.

Листинг 3.25. Использование функции `get_class()`

```
<?php
  require_once("class/class.employee.php");
  require_once("class/class.client.php");
  require_once("class/class.contract.php");

  // Формируем массив из случайных объектов
  for($i = 0; $i < 10; $i++)
  {
    switch(rand(1,3))
    {
      case 1:
        $arr[] = new client("Борисов", "Игорь", "Иванович");
        break;
      case 2:
        $arr[] = new employee("Корнеев", "Иван", "Григорьевич");
        break;
      case 3:
        $arr[] = new contract(time(),
                              time() + 30*24*60*60,
                              "Описание",
                              "10000",
                              "Борисов",
                              "Игорь",
                              "Иванович",
                              "Корнеев",
                              "Иван",
                              "Григорьевич");
        break;
    }
  }
  // Определяем принадлежность элементов
  // массива к классам
  foreach($arr as $obj)
  {
    echo get_class($obj)."<br>";
  }
?>
```

Результат работы скрипта из листинга 3.25 может выглядеть следующим образом:

```
client
contract
contract
client
client
client
employee
employee
contract
client
```

Альтернативной возможностью определения принадлежности объекта классу является использование оператора `instanceof`. Пример использования оператора приводится в листинге 3.26.

Листинг 3.26. Использование оператора `instanceof`

```
<?php
  require_once("class/class.employee.php");
  require_once("class/class.client.php");
  require_once("class/class.contract.php");

  // Формируем массив из случайных объектов
  for($i = 0; $i < 10; $i++)
  {
    switch(rand(1,3))
    {
      case 1:
        $arr[] = new client("Борисов", "Игорь", "Иванович");
        break;
      case 2:
        $arr[] = new employee("Корнеев", "Иван", "Григорьевич");
        break;
      case 3:
        $arr[] = new contract(time(),
                              time() + 30*24*60*60,
                              "Описание",
                              "10000",
                              "Борисов",
                              "Игорь",
                              "Иванович",
```

```
                                "Корнеев",
                                "Иван",
                                "Григорьевич");
            break;
        }
    }
    // Определяем принадлежность элементов
    // массива к классам
    foreach($arr as $obj)
    {
      if($obj instanceof contract) echo "Объект класса contract<br>";
      else echo "Не является объектом класса contract<br>";
    }
?>
```

В первом цикле формируется массив случайных объектов, после чего во втором цикле при помощи оператора `instanceof` определяется, является ли текущий элемент объектом класса `contract`. Результат работы скрипта из листинга 3.26 может выглядеть следующим образом:

```
Не является объектом класса contract
Не является объектом класса contract
Объект класса contract
Объект класса contract
Объект класса contract
Не является объектом класса contract
Не является объектом класса contract
Объект класса contract
Не является объектом класса contract
Не является объектом класса contract
```

3.9. Аксессоры. Методы *__set()* и *__get()*

Неписанным правилом объектно-ориентированного программирования является использование только закрытых членов, доступ к которым осуществляется через открытые методы класса. Это позволяет скрыть внутреннюю реализацию класса, ограничить диапазон значений, которые можно присваивать члену, и сделать член доступным только для чтения.

Неудобство заключается в том, что для каждого из членов приходится создавать отдельный метод для чтения и присваивания нового значения, имена которых зачастую не совпадают с именами членов.

Выходом из ситуации является использование свойств, обращение к которым выглядит точно так же, как к открытым членам класса. Для их реализации

необходимо перегрузить специальные методы __get() и __set(), которые часто называют *аксессорами*. Метод __get(), предназначенный для чтения свойства, принимает единственный параметр, который служит ключом. Метод __set() позволяет присвоить свойству новое значение и принимает два параметра, первый из которых является ключом, а второй — значением свойства.

В примере из листинга 3.27 при помощи метода __set() объекту присваиваются новые свойства, которые помещаются в массив $this->arr, а перегруженный метод __get() позволяет извлечь их из массива.

Замечание

Следует обратить внимание, что методы __set() и __get() можно объявлять как закрытыми, так и открытыми.

Листинг 3.27. Использование методов __set() и __get()

```
<?php
  class cls
  {
    private $arr = array();

    private function __get($index)
    {
      return $this->arr[$index];
    }

    private function __set($index, $value)
    {
      $this->arr[$index] = $value;
    }
 }
?>
```

Как видно из листинга 3.27, класс cls перехватывает все обращения к членам объекта и создает соответствующий элемент в закрытом массиве $arr. В листинге 3.28 демонстрируется, как обращение к члену $name приводит к созданию соответствующего элемента массива.

Листинг 3.28. Обращение к несуществующему элементу $name

```
<?php
  require_once("class.cls.php");
```

```
  $obj = new cls();
  $obj->name = "Hello world!<br>";
  echo $obj->name;
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.27 являются следующие строки:

```
Hello world!
cls Object
(
    [arr:private] => Array
        (
            [name] => Hello world!
        )
)
```

Любая попытка присвоить члену значение приводит к созданию нового элемента закрытого массива `$arr`. Интересно, что когда член в классе уже существует, то аксессоры `__set()` и `__get()` перехватывают обращение к нему, если он является закрытым (имеет спецификатор доступа `private`), и не перехватывают, если он является открытым (`public`). В листинге 3.29 в класс `cls` добавляется два члена: открытый член `$name` и закрытый `$pr_name`.

Листинг 3.29. Модифицированный класс `cls`

```
<?php
  class cls
  {
    public $name;
    private $pr_name;
    private $arr = array();

    public function __get($index)
    {
      return $this->arr[$index];
    }

    public function __set($index, $value)
    {
      $this->arr[$index] = $value;
    }
 }
?>
```

В листинге 3.30 производится обращение к открытому члену `$name` и закрытому члену `$pr_name` из внешней программы.

Листинг 3.30. Обращение к членам класса при наличии перегруженных методов `__get()` и `__set()`

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  $obj->name = "Hello world!<br>";
  $obj->pr_name = "Hello world!<br>";
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.30 станут следующие строки:

```
cls Object
(
    [name] => Hello world!

    [pr_name:private] =>
    [arr:private] => Array
        (
            [pr_name] => Hello world!
        )
)
```

Как видно из результатов выполнения программы, открытый член `$name` получил значение (при этом не был создан соответствующий элемент массива `$arr`), в то время как закрытый член `$pr_name` остался не инициализированным.

До этого момента перегрузке подвергались оба аксессора — и метод `__get()`, и метод `__set()`. Однако перегрузка обоих методов не обязательна, допускается перегружать только один из них. Например, в листинге 3.31 значения членов устанавливаются при помощи конструктора класса, после чего перегружается метод `__get()`, предоставляя доступ к членам только для чтения.

Листинг 3.31. Доступ к членам только для чтения

```
<?php
  class employee
```

```
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      $this->age = $age;
    }
    private function __get($index)
    {
      return $this->$index;
    }

    private $surname;
    private $name;
    private $patronymic;
    private $age;
  }
?>
```

В листинге 3.32 метод `__get()` предоставляет возможность чтения значений закрытых членов `$surname`, `$name`, `$patronymic` и `$age`, однако попытка передачи им значений в обход конструктора заканчивается сообщением об ошибке **Fatal error: Cannot access private property**.

Листинг 3.32. Использование класса `employee`

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");
  // $obj->surname = "Абрамов"; // Ошибка
  echo "{$obj->surname} {$obj->name} {$obj->patronymic}";
?>
```

В аксессоре `__set()` можно устанавливать различные ограничения. Например, для того чтобы пользователи класса не имели возможности установить значение несуществующего члена, перед установкой значения можно проверить при помощи конструкции `isset()`, существует член класса с таким именем или нет: это позволит предотвратить создание динамических членов класса. В листинге 3.33 приводится модификация класса `employee`, в которой при помощи метода `__set()` можно назначить новые значения только существующим членам `$surname`, `$name`, `$patronymic` и `$age`. При этом член `$age` может принимать значения лишь в диапазоне от 18 до 60.

Листинг 3.33. Присвоить значения можно только существующим членам класса

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname    = $surname;
      $this->name       = $name;
      $this->patronymic = $patronymic;
      $this->age        = $age;
    }
    private function __get($index)
    {
      return $this->$index;
    }
    private function __set($index, $value)
    {
      if(isset($this->$index))
      {
        if($index != "age")
        {
          $this->$index = $value;
        }
        else
        {
          if($value >= 18 && $value <= 60)
          {
            $this->$index = $value;
          }
        }
      }
    }

    private $surname;
    private $name;
    private $patronymic;
    private $age;
  }
?>
```

3.10. Проверка существования члена класса. Метод *__isset()*

В листинге 3.32 предыдущего раздела представлен класс `employee`, метод `__set()` которого позволяет присваивать значения только существующим членам класса. Из внешнего кода при помощи конструкции `isset()` можно убедиться в существовании члена только в том случае, если он открыт. Тем не менее при использовании аксессора `__set()` полезно знать, существует член или нет, и по отношению к закрытым переменным класса. Для решения этой задачи предусмотрен специальный метод класса `__isset()`, который принимает в качестве единственного параметра имя свойства и возвращает `true`, если свойство с таким именем существует, и `false` — в противном случае. В листинге 3.34 представлена реализация метода `__isset()` для класса `employee`.

Замечание

Метод `__isset()` доступен в PHP, начиная с версии 5.1.0.

Листинг 3.34. Перегрузка метода `__isset()`

```
<?php
  class employee
  {
    ...
    private function __isset($index)
    {
      return isset($this->$index);
    }
    ...
  }
?>
```

Теперь рассмотрим, как конструкция `isset()` будет возвращать для закрытых переменных `$name`, `$surname`, `$patronymic` или `$age` значение `true` (листинг 3.35).

Листинг 3.35. Использование конструкции `isset()`

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");
  if(isset($obj->surname)) $obj->surname = "Абрамов"; // true
```

```
  else echo "Отсутствует член класса<br>";
  if(isset($obj->surname1)) $obj->surname1 = "Абрамов"; // false
  else echo "Отсутствует член класса<br>";
  echo "{$obj->surname} {$obj->name} {$obj->patronymic}";
?>
```

Помимо конструкции `isset()` существует специальная функция `property_exists()`, предназначенная для проверки существования членов класса и имеющая следующий синтаксис:

```
bool property_exists($class, $property)
```

Функция принимает в качестве первого аргумента либо строку с именем класса, либо объект класса, а в качестве второго аргумента строку с именем члена. Если член является элементом класса, функция возвращает `true`, в противном случае — `false`.

Для демонстрации приемов работы с функцией `property_exists()` изменим класс `employee` таким образом, чтобы его члены `$surname` и `$name` стали открытыми, член `$patronymic` — закрытым, а `$age` — динамически формируемым в конструкторе (листинг 3.36).

Листинг 3.36. Модифицированный класс `employee`

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      $this->age = $age;
    }
    private function __get($index)
    {
      return $this->$index;
    }

    public $surname;
    public $name;
    private $patronymic;
  }
?>
```

В листинге 3.37 все члены класса `employee` последовательно передаются функции `property_exists()`, по результатам работы которой выводится сообщение о существовании или не существовании члена класса.

Замечание

Функция `property_exists()` введена в PHP, начиная с версии 5.1.0.

Листинг 3.37. Проверка существования членов класса `employee`

```
<?php
  require_once("class.employee.php");

  if(property_exists("employee", "surname"))
    echo "Член employee::surname существует<br>";
  else
    echo "Член employee::surname не существует<br>";

  if(property_exists("employee", "name"))
    echo "Член employee::name существует<br>";
  else
    echo "Член employee::name не существует<br>";

  if(property_exists("employee", "patronymic"))
    echo "Член employee::patronymic существует<br>";
  else
    echo "Член employee::patronymic не существует<br>";

  if(property_exists("employee", "age"))
    echo "Член employee::age существует<br>";
  else
    echo "Член employee::age не существует<br>";

  if(property_exists("employee", "none"))
    echo "Член employee::none существует<br>";
  else
    echo "Член employee::none не существует<br>";
?>
```

В результате работы скрипта из листинга 3.37 на экран будут выведены следующие строки:

```
Член employee::surname существует
Член employee::name существует
```

```
Член employee::patronymic не существует
Член employee::age не существует
Член employee::none не существует
```

Как видно из результатов работы скрипта, функции `property_exists()` доступны только открытые члены класса. Закрытые, динамические и не существующие члены класса данная функция считает не существующими.

Следует отметить, что помимо имени класса функция `property_exists()` может принимать в качестве первого параметра объект (листинг 3.38).

Листинг 3.38. Использование объекта в качестве параметра функции `property_exists()`

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");
  if(property_exists($obj, "surname"))
    echo "Член employee::surname существует<br>";
  else
    echo "Член employee::surname не существует<br>";

  if(property_exists($obj, "name"))
    echo "Член employee::name существует<br>";
  else
    echo "Член employee::name не существует<br>";

  if(property_exists($obj, "patronymic"))
    echo "Член employee::patronymic существует<br>";
  else
    echo "Член employee::patronymic не существует<br>";

  if(property_exists($obj, "age"))
    echo "Член employee::age существует<br>";
  else
    echo "Член employee::age не существует<br>";

  if(property_exists($obj, "none"))
    echo "Член employee::none существует<br>";
  else
    echo "Член employee::none не существует<br>";
?>
```

В случае использования объекта картина несколько меняется: в отличие от класса, объект содержит динамические члены, причем члены эти снабжены спецификатором `public`. Поэтому для динамического члена `$age` функция `property_exists()` возвращает `true`:

```
Член employee::surname существует
Член employee::name существует
Член employee::patronymic не существует
Член employee::age существует
Член employee::none не существует
```

Помимо метода `property_exists()` можно использовать функцию `print_r()` для получения дампа объекта (см. *раздел 2.9*). Однако зачастую требуется получить список членов класса, не прибегая к созданию объекта. В этом случае может помочь функция `get_class_vars()`, которая имеет следующий синтаксис:

```
array get_class_vars($class)
```

Функция принимает в качестве единственного параметра строку `$class` с именем класса, а возвращает ассоциативный массив, в котором в качестве ключа выступает имя члена класса (листинг 3.39).

Листинг 3.39. Использование функции `get_class_vars()`

```
<?php
  require_once("class.employee.php");

  $arr = get_class_vars("employee");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.39 будет следующий дамп массива:

```
Array
(
    [surname] =>
    [name] =>
)
```

Как видно из результатов работы скрипта, функция `get_class_vars()` помещает в результирующий массив только имена открытых членов класса, но не выводит их значения.

Если в распоряжении разработчика имеется объект класса, то получить ассоциативный массив с членами объекта и их значениями можно при помощи функции `get_object_vars()`, которая имеет следующий синтаксис:

```
array get_object_vars($obj)
```

В качестве единственного параметра функция принимает объект, а возвращает ассоциативный массив с членами объекта и их значениями (листинг 3.40).

Листинг 3.40. Использование функции `get_object_vars()`

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");
  $arr = get_object_vars($obj);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.40 будет следующий дамп массива:

```
Array
(
    [surname] => Борисов
    [name] => Игорь
    [age] => 18
)
```

Как и функция `get_class_vars()`, функция `get_object_vars()` возвращает только открытые члены класса (включая динамические).

3.11. Уничтожение члена класса. Метод *__unset()*

Для уничтожения членов класса служит специальный метод `__unset()`. Вернемся к классу `cls`, рассмотренному в листинге 3.27. Устанавливаемые при помощи метода `__set()` значения свойств помещаются в закрытый массив `$arr`. Закрытый массив не допускает уничтожения своих отдельных элементов, однако перегрузка метода `__unset()` позволяет снабдить класс такой возможностью (листинг 3.41).

Замечание

Метод `__unset()` доступен в PHP, начиная с версии 5.1.0.

Листинг 3.41. Перегрузка метода `__unset()`

```
<?php
  class cls
  {
    private $arr = array();

    private function __get($index)
    {
      return $this->arr[$index];
    }

    private function __set($index, $value)
    {
      $this->arr[$index] = $value;
    }

    private function __isset($index)
    {
      return isset($this->arr[$index]);
    }

    private function __unset($index)
    {
      unset($this->arr[$index]);
    }
 }
?>
```

Теперь, несмотря на то что массив `$arr` является закрытым членом, его элементы можно уничтожать при помощи конструкции `unset()` (листинг 3.42).

Листинг 3.42. Использование конструкции `unset()`

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  $obj->name = "Новое свойство класса";
```

```
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  unset($obj->name);

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.42 будут следующие строки:

```
cls Object
(
    [arr:private] => Array
        (
            [name] => Новое свойство класса
        )
)
cls Object
(
    [arr:private] => Array
        (
        )
)
```

Как можно видеть, использование конструкции `unset()` приводит к уничтожению динамического свойства.

3.12. Динамические методы. Метод *__call()*

Специальный метод `__call()` предназначен для создания динамических методов: если метод `__call()` перегружен в классе, обращение к несуществующему методу не приведет к ошибке, а передаст управление методу `__call()`. В качестве первого параметра метод `__call()` принимает имя вызываемого метода, а в качестве второго — массив, содержащий в качестве элементов параметры, которые были переданы при вызове метода.

Замечание

Обсуждение специального метода `__call()` затрагивается также в *разделе 2.10*.

В отличие от программирования других С-подобных языков программирования, в PHP отсутствуют функции с переменным количеством параметров.

Тем не менее, при помощи специального метода `__call()` можно эмулировать наличие таких функций в классе. В листинге 3.43 представлен класс `minmax`, который предоставляет пользователю два метода: `min()` и `max()`, принимающие произвольное количество числовых параметров и определяющие минимальное и максимальное значения соответственно.

Листинг 3.43. Использование специального метода `__call()`

```
<?php
  class minmax
  {
    private function __call($method, $arg)
    {
      if(!is_array($arg)) return false;
      $value = $arr[0];
      if($method == "min")
      {
        for($i = 0; $i < count($arg); $i++)
        {
          if($agr[$i] < $value) $value = $agr[$i];
        }
      }
      if($method == "max")
      {
        for($i = 0; $i < count($arg); $i++)
        {
          if($agr[$i] > $value) $value = $agr[$i];
        }
      }
      return $value;
    }
  }
?>
```

В примере из листинга 3.43 в зависимости от вызываемого метода — `min()` или `max()` — используется два разных алгоритма для поиска результата. Кроме того, для класса `minmax` допустим вызов метода с произвольным именем, и если оно отлично от `min` или `max`, будет возвращаться первый аргумент. Независимо от имени метода, если не передано ни одного аргумента, возвращается значение `false`.

В листинге 3.44 приводится пример использования класса `minmax` для получения максимального и минимального значений последовательности.

Листинг 3.44. Использование класса `minmax`

```
<?php
  require_once("class.minmax.php");

  $obj = new minmax();
  echo $obj->min(43, 18, 5, 61, 23, 10, 56, 36); // 5
  echo "<br>";
  echo $obj->max(43, 18, 5, 61, 23); // 61
?>
```

Важно отметить, что вызов методов при помощи оператора разрешения области видимости `::` для динамических классов заканчивается ошибкой. Для обращения к динамическим методам обязательно требуется создание объекта. В листинге 3.45 представлено ошибочное обращение, которое завершается выводом предупреждения **Fatal error: Call to undefined method minmax::min()**.

Листинг 3.45. Альтернативный вызов динамических методов класса `minmax`

```
<?php
  require_once("class.minmax.php");

  echo minmax::min(43, 18, 5, 61, 23, 10, 56, 36); // 5
  echo "<br>";
  echo minmax::max(43, 18, 5, 61, 23, 10, 56, 36); // 61
?>
```

Взаимодействие специального метода `__call()` с уже существующими в классе методами отличается от аксессоров `__get()` и `__set()`: если метод (закрытый или открытый) существует, то `__call()` не задействуется. Для демонстрации последнего правила снабдим класс `minmax` двумя методами: открытым методом `min()` и закрытым методом `max()` (листинг 3.46).

Листинг 3.46. Модифицированный вариант класса `minmax`

```
<?php
  class minmax
  {
    public function min($val, $val1, $val3)
    {
      echo "Вызов открытого метода min()";
    }
```

```
    private function max($val, $val1, $val3)
    {
      echo "Вызов открытого метода max()";
    }
    private function __call($method, $arg)
    {
      if(!is_array($arg)) return false;
      $value = $arg[0];
      if($method == "min")
      {
        for($i = 0; $i < count($arg); $i++)
        {
          if($arg[$i] < $value) $value = $arg[$i];
        }
      }
      if($method == "max")
      {
        for($i = 0; $i < count($arg); $i++)
        {
          if($arg[$i] > $value) $value = $arg[$i];
        }
      }
      return $value;
    }
  }
?>
```

При выполнении кода из листинга 3.45 при обращении к методу `min()` будет выведена фраза **Вызов открытого метода min()**, а обращение к методу `max()` заканчивается ошибкой обращения к закрытому методу: **Fatal error: Call to private method minmax::max()**.

3.13. Проверка существования метода

Как было продемонстрировано в *разделах 2.10* и *3.12*, классы могут содержать динамические методы, наличие которых неочевидно внешнему разработчику. Кроме того, в процессе эксплуатации могут создаваться разнородные массивы объектов, которые могут содержать разные методы. Для подобных ситуаций необходимы инструменты проверки существования метода в классе. В качестве такого инструмента в PHP выступает функция `method_exists()`, которая имеет следующий синтаксис:

```
bool method_exists($obj, $method_name)
```

Функция принимает в качестве первого параметра объект `$obj` или имя класса, а в качестве второго — имя метода `$method_name` и возвращает `true`, если объект или класс имеет данный метод, и `false` — в противном случае.

Для демонстрации работы функции `method_exists()` создадим класс `cls`, который будет содержать два метода: открытый метод `public_print()` и закрытый метод `private_print()` (листинг 3.47).

Листинг 3.47. Вспомогательный класс `cls`

```
<?php
  class cls
  {
    public function __construct()
    {
    }
    public function public_print()
    {
      echo "Открытый метод";
    }
    private function private_print()
    {
      echo "Закрытый метод";
    }
 }
?>
```

В листинге 3.48 при помощи функции `method_exists()` в объекте класса `cls` проверяется наличие существующих методов `public_print()`, `private_print()` и отсутствующего метода `print()`.

Замечание

Помимо функции `method_exists()` можно воспользоваться альтернативной функцией `is_callable()`, которая, в отличие от `method_exists()`, кроме проверки метода класса позволяет проверить существование функции, не входящей в состав класса.

Листинг 3.48. Использование функции `method_exists()`

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  if(method_exists($obj, "public_print"))
    echo "Метод cls::public_print() существует<br>";
```

```
  else
    echo "Метод cls::public_print() не существует<br>";

  if(method_exists($obj, "private_print"))
    echo "Метод cls::private_print() существует<br>";
  else
    echo "Метод cls::private_print() не существует<br>";

  if(method_exists($obj, "print"))
    echo "Метод cls::print() существует<br>";
  else
    echo "Метод cls::print() не существует<br>";
?>
```

Результатом работы скрипта из листинга 3.48 являются следующие строки:

```
Метод cls::public_print() существует
Метод cls::private_print() существует
Метод cls::print() не существует
```

Как видно из результата проверки, функция возвращает `true` для каждого из методов, независимо от его спецификатора доступа, `false` возвращается только в том случае, если объект не обладает ни закрытым, ни открытым методом с таким именем.

В листинге 3.48 в качестве первого аргумента функции `method_exists()` использовался объект класса `cls`, однако для проверки метода вовсе не обязательно создавать объект — достаточно передать имя класса (листинг 3.49).

Листинг 3.49. Альтернативный способ использования функции `method_exists()`

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  if(method_exists("cls", "public_print"))
    echo "Метод cls::public_print() существует<br>";
  else
    echo "Метод cls::public_print() не существует<br>";

  if(method_exists("cls", "private_print"))
    echo "Метод cls::private_print() существует<br>";
  else
    echo "Метод cls::private_print() не существует<br>";
```

```
  if(method_exists("cls", "print"))
    echo "Метод cls::print() существует<br>";
  else
    echo "Метод cls::print() не существует<br>";
?>
```

Результат работы скрипта из листинга 3.49 аналогичен полученному при работе скрипта из листинга 3.48.

При работе с методом `method_exists()` следует учитывать, что он не может определить наличие динамических методов, созданных при помощи специального метода `__call()`.

Работая со сторонним классом, программист зачастую не знает досконально всех методов данного класса. Для того чтобы получить их полный список, можно воспользоваться функцией `get_class_methods()`, которая имеет следующий синтаксис:

```
array get_class_methods($class_name)
```

Функция принимает в качестве первого аргумента имя класса `$class_name`, а возвращает массив его открытых методов. Следует подчеркнуть, что закрытые методы этой функцией не возвращаются. В листинге 3.50 приводится пример использования функции применительно к классу `cls`.

Листинг 3.50. Использование функции `get_class_methods()`

```
<?php
  require_once("class.cls.php");

  $methods = get_class_methods("cls");

  echo "<pre>";
  print_r($methods);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.50 являются следующие строки:

```
Array
(
    [0] => __construct
    [1] => public_print
)
```

Как можно видеть, закрытый метод `private_public()` не включен в результирующий массив. Динамические методы, которые эмулируются при помощи

специального метода `__call()`, также не попадают в список, получаемый при помощи функции `get_class_methods()`.

3.14. Интерполяция объекта. Метод *__toString()*

Специальный метод `__toString()` позволяет интерполировать (подставлять) объект в строку. Для подстановки значений переменных необходимо заключить строку в двойные кавычки (листинг 3.51). Следует отметить, что значение переменной не интерполируется, если вместо двойных кавычек используются одинарные.

Листинг 3.51. Интерполяция переменной

```
<?php
  $str = "12345";
  echo "str = $str<br>"; // str = 12345
  echo 'str = $str<br>'; // str = $str
?>
```

Такого же поведения можно добиться и от объекта, если реализовать в его классе метод `__toString()`, который преобразует объект в строку.

Модифицируем класс `employee` таким образом, чтобы подстановка его в строку приводила к выводу фамилии сотрудника и его инициалов (листинг 3.52).

Замечание

Следует обратить внимание, что метод `__toString()` выводит результат при помощи конструкции `return`, а не `echo`.

Листинг 3.52. Использование специального метода `__toString()`

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      $this->age = $age;
    }
```

```
    private function __toString()
    {
      return "{$this->surname} {$this->name[0]}.{$this->patronymic[0]}.";
    }
    private function __get($index)
    {
      return $this->$index;
    }

    public $surname;
    public $name;
    private $patronymic;
  }
?>
```

В листинге 3.53 приводится пример интерполяции объекта класса `employee`, при этом вместо объекта `$obj` будет подставлена фраза `Борисов Игорь Иванович`.

Листинг 3.53. Интерполяция объекта

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");

  echo "Сотрудник $obj недавно принят на работу";
?>
```

Метод `__toString()` автоматически вызывается в любом контексте, где ожидается строка, например, результат работы скрипта из листинга 3.54 полностью аналогичен тому, что был получен после работы скрипта из листинга 3.53.

Листинг 3.54. Метод `__toString()` вызывается в строковом контексте

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");

  $str = "Сотрудник ";
  $str .= $obj;
  $str .= " недавно принят на работу";
```

```
  echo $str;
?>
```

Следует отметить, что вызов объекта в строковом контексте возможен, только если его класс содержит реализацию метода `__toString()`, в противном случае попытка использовать объект в строке будет заканчиваться ошибкой **Catchable fatal error: Object of class employee could not be converted to string**.

В качестве еще одной демонстрации применения метода `__toString()` рассмотрим класс-обертку для массива `implode`, который выполняет практически те же действия, что и стандартная функция `implode()`. Конструктор класса принимает два параметра: массив `$arr` и разделитель `$delimiter`. Метод `__toString()` объединяет элементы массива `$arr` в строку, отделяя их друг от друга разделителем `$delimiter` (листинг 3.55).

Листинг 3.55. Класс `implode`

```
<?php
  class implode
  {
    public function __construct($delimiter, $arr)
    {
      $this->arr       = $arr;
      $this->delimiter = $delimiter;
    }
    private function __toString()
    {
      $str = "";
      foreach($this->arr as $value)
      {
        $str .= $value.$this->delimiter;
      }
      // Удаляем последний элемент $delimiter
      return substr($str, 0, strlen($str) - strlen($this->delimiter));
    }
 }
?>
```

В листинге 3.56 класс-обертка `implode` используется для вывода элементов массива через запятую.

Листинг 3.56. Вывод элементов массива через запятую

```
<?php
  require_once("class.implode.php");
```

```
  $obj = new implode(", ", array(11, 2, 23, 45, 18));

  echo $obj;
?>
```

Результатом работы скрипта из листинга 3.55 будет следующая строка цифр, разделенных запятой:

```
11, 2, 23, 45, 18
```

3.15. Экспорт переменных. Метод *__set_state()*

Среди многочисленных функций PHP существует функция `var_export()`, выводящая дампы переменных, массивов и объектов. В отличие от `print_r()` или `var_dump()`, она возвращает дамп в виде PHP-кода, что позволяет использовать результат ее выполнения в функции `eval()`, выполняющей PHP-код в строке. Функция `var_export()` имеет следующий синтаксис:

```
mixed var_export ($expression [, $return])
```

В качестве первого аргумента функция принимает переменную, массив или объект. По умолчанию выводится дамп объекта, а сама функция ничего не возвращает. Однако если в качестве второго необязательного параметра `$return` передается значение `true`, функция вместо вывода дампа в окно браузера возвращает его в виде строки.

В листинге 3.57 приводится пример использования функции `var_export()` для вывода дампов переменной, массива и объекта. Действие функции в этом качестве аналогично функции `print_r()`.

Листинг 3.57. Использование функции `var_export()` для вывода дампов

```
<?php
  // Переменная
  $var = 100;

  // Массив
  $arr = array(1, 2, 3, array(4, 5), array(6, 7));

  // Класс
  class cls
  {
    public function __construct($var, $val)
    {
      $this->publ_var = $var;
```

```
      $this->priv_var = $val;
    }
    public  $publ_var;
    private $priv_var;
  }
  // Объект
  $obj = new cls(12, 147);

  echo "<pre>";
  var_export($var);
  var_export($arr);
  var_export($obj);
  echo "<pre>";
?>
```

Результатом работы скрипта из листинга 3.57 будут следующие дампы переменной `$var`, массива `$arr` и объекта `$obj`:

```
100
array (
  0 => 1,
  1 => 2,
  2 => 3,
  3 => array (
    0 => 4,
    1 => 5,
  ),
  4 => array (
    0 => 6,
    1 => 7,
  ),
)
cls::__set_state(
array(
   'publ_var' => 12,
   'priv_var' => 147,
))
```

Листинг 3.58 демонстрирует использование второго, дополнительного параметра функции `var_export()`.

Листинг 3.58. Использование второго параметра функции `var_export()`

```
<?php
  $var = 100;
```

```
  $val = var_export($var, true);
  echo $val; // 100
?>
```

В листинге 3.59 приводится пример создания копии массива `$arr` при помощи функции `var_export()`.

Замечание

К сожалению, из-за ошибки реализации функции `var_export()` приходится самостоятельно заботиться об удалении последней запятой из дампов массивов и объектов. Вероятно, эта досадная ошибка будет устранена в более поздних версиях PHP.

Листинг 3.59. Создание копии массива `$arr`

```
<?php
  $arr = array(1, 2, 3, array(4, 5), array(6, 7));
  $str = var_export($arr, true);
  // Из-за ошибки реализации приходится
  // удалять последнюю запятую самостоятельно
  $str = preg_replace("|,[\s]*\)|is", ")", $str);
  // Создаем массив $copy
  eval('$copy = '.$str.';');

  echo "<pre>";
  print_r($copy);
  echo "</pre>";
?>
```

Как видно из листинга 3.59, скрипт создает копию массива `$arr` и помещает ее в массив `$copy`. Массив воссоздается путем динамического выполнения PHP-кода при помощи функции `eval()`. Результатом работы скрипта будут следующие строки:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => Array
        (
            [0] => 4
            [1] => 5
        )
```

```
    [4] => Array
        (
            [0] => 6
            [1] => 7
        )
)
```

По отношению к объекту функция `var_export()` действует несколько иным образом: поскольку невозможно предугадать заранее код, который бы воспроизводил объект, функция возвращает код вызова специального метода `__set_state()`, разработкой которого должен озаботиться автор класса. В качестве единственного параметра методу `__set_state()` передается ассоциативный массив со всеми членами объекта. В листинге 3.60 приводится пример использования метода `__set_state()`: получив значения объекта в массиве `$arr_obj`, метод `__set_state()` просто выводит все значения в окно браузера.

Замечание

Специальный метод `__set_state()` введен в PHP, начиная с версии 5.1.0.

Листинг 3.60. Использование специального метода `__set_state()`

```
<?php
  // Класс
  class cls
  {
    public function __construct($var, $val)
    {
      $this->publ_var = $var;
      $this->priv_var = $val;
    }
    public function __set_state($arr_obj)
    {
      foreach($arr_obj as $key => $value)
      {
        echo "$key => $value<br>";
      }
    }
    public  $publ_var;
    private $priv_var;
  }
  // Объект
  $obj = new cls(12, 147);
```

```
  // Возвращаем вызов метода __set_state()
  $str = var_export($obj, true);

  // Из-за ошибки реализации приходится
  // удалять последнюю запятую самостоятельно
  $str = preg_replace("|,[\s]*\)|is", ")", $str);

  // Вызываем метод __set_state()
  eval($str.';');
?>
```

Результатом работы скрипта из листинга 3.60 будет список закрытых и открытых членов класса:

```
publ_var => 12
priv_var => 147
```

Однако функция `var_export()` задумывалась для возвращения дампа массива или объекта, который совместно с функцией `eval()` позволяет воспроизвести объект. Поэтому рекомендуется использовать метод `__set_state()` именно для воссоздания объекта, а не для каких-то иных целей. В листинге 3.61 приводится альтернативная реализация класса `cls`, в которой метод `__set_state()` собирает и возвращает новый объект.

Листинг 3.61. Воссоздание объекта при помощи метода `__set_state()`

```
<?php
  // Класс
  class cls
  {
    public function __construct($var, $val)
    {
      $this->publ_var = $var;
      $this->priv_var = $val;
    }
    public function __set_state($arr_obj)
    {
      return new cls($arr_obj['publ_var'], $arr_obj['priv_var']);
    }
    public  $publ_var;
    private $priv_var;
  }
  // Объект
  $obj = new cls(12, 147);
```

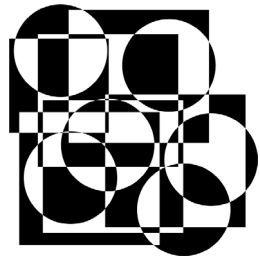
```
  // Возвращаем вызов метода __set_state()
  $str = var_export($obj, true);

  // Из-за ошибки реализации приходится
  // удалять последнюю запятую самостоятельно
  $str = preg_replace("|,[\s]*\)|is", ")", $str);

  // Создаем объект $new_obj — копию
  // объекта $obj
  eval('$new_obj = '.$str.';');

  // Выводим дамп нового объекта $new_obj
  echo "<pre>";
  print_r($new_obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 3.61 будет следующий дамп нового объекта `$new_obj`:

```
cls Object
(
    [publ_var] => 12
    [priv_var:private] => 147
)
```

Использование приведенной выше методики позволяет обойти ограничение, связанное с присваиванием объектов друг другу, после которого два объекта ссылаются на одну и ту же область памяти (см. *раздел 2.15*). Однако для получения копии объекта существует более удобная методика клонирования, которая рассматривается в *главе 7*.

ГЛАВА 4



Инкапсуляция, наследование, полиморфизм

Объектно-ориентированная программа, как было сказано в *главе 1*, должна отвечать ряду принципов: использование абстрактных типов данных (классов и объектов), инкапсуляция, наследование и полиморфизм. Создание и использование абстрактных типов данных подробно рассматривалось в трех предыдущих главах. В данной главе будут рассмотрены остальные составляющие объектно-ориентированного подхода.

4.1. Инкапсуляция

Создание класса расширяет язык программирования новым типом данных. Любой программист, разрабатывающий класс, создает собственный мини-язык программирования, которым могут пользоваться при работе другие прикладные программисты. Со временем разработчик может совершенствовать свой класс, однако при этом он должен учитывать, что, возможно, его класс используется в работе других программ. Следовательно, модификация класса не должна влиять на работу приложений, использующих его более ранние версии. Для этого вводят открытую часть (интерфейс), к которой обращаются пользователи класса, и закрытую, к которой могут получать доступ только методы самого класса (рис. 4.1).

Важность сокрытия реализации метода рассмотрим на примере класса для описания трудового договора `c` новым сотрудником `contract`, содержащего следующие члены:

- `$name` — имя сотрудника;
- `$surname` — фамилия сотрудника;
- `$patronymic` — отчество сотрудника;
- `$age` — возраст на момент приема на работу;

- `$date` — дата подписания договора;
- `$description` — описание характера работы.

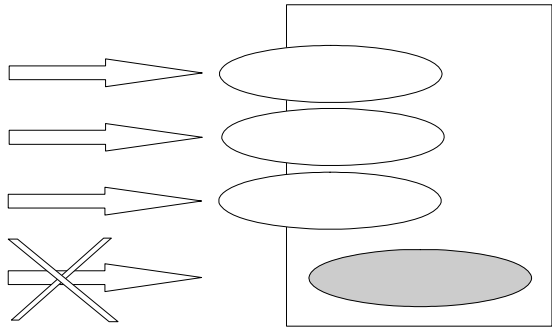


Рис. 4.1. Из внешнего кода возможен доступ только к открытым элементам интерфейса

В листинге 4.1 приводится определение класса `contract`, а также пример создания объекта данного класса.

Листинг 4.1. Создание класса `contract`

```
<?php
  // Класс "договор между организацией и сотрудником
  // о приеме последнего на работу"
  class contract
  {
     // Члены класса
     public $surname;
     public $name;
     public $patronymic;
     public $age;
     public $date;
     public $description;
     // Конструктор
     function __construct($sn, $nm, $pt, $ag, $dt, $ds)
     {
        $this->surname = $sn;
        $this->name = $nm;
        $this->patronymic = $pt;
        $this->age = $ag;
        $this->date = $dt;
        $this->description = $ds;
     }
```

```
    // Остальная реализация класса
    ...
  }
  // Создание объекта
  $obj = new contract("Сидоров",
                      "Иван",
                      "Иванович",
                      36,
                      "2003.01.15",
                      "инженер-программист");
?>
```

Полученный объект "договор" можно использовать при создании Web-приложения, управляющего документооборотом предприятия. Для вывода возраста сотрудника логично использовать поле `age` (листинг 4.2).

Листинг 4.2. Вывод возраста работника

```
<?php
  ...
  echo "Возраст сотрудника ".$obj->surname." — ".$obj->age." лет<br>";
  ...
?>
```

Таких вызовов в приложении может оказаться несколько тысяч. По прошествии нескольких лет оказывается, что вывод возраста сотрудника на момент заключения договора не устраивает пользователей системы, поскольку данные поля `$obj->age` перестали быть актуальными. Чтобы исправить эту ошибку, следует вычесть из текущего значения года значение года, содержащегося в дате заключения договора, и полученный результат прибавить к полю `$obj->age` (листинг 4.3).

Листинг 4.3. Вывод возраста работника на текущую дату

```
<?php
  ...
  $delta = date('Y') — substr($obj->date, 0, 4);
  $actual_age = $obj->age + $delta;
  echo "Возраст работника {$obj->surname} — {$actual_age}<br>";
  ...
?>
```

Таким образом, для того чтобы исправить недочет, не предусмотренный при проектировании приложения, следует заменить несколько тысяч строк Web-приложения. При этом отдельные участки кода неизбежно останутся неис-

правленными, в результате чего ситуация еще более усугубится. Кроме того, очевидно, что код по вычислению актуального возраста сотрудника должен выполняться классом, а не внешним кодом (разработчики которого не должны вникать в особенности реализации класса). Этой ситуации можно избежать, если в самом начале объявить члены класса закрытыми (`private`), а доступ к переменным осуществлять при помощи методов (листинг 4.4).

Листинг 4.4. Ограничение доступа к членам метода

```
<?php
  // Класс "договор между организацией и сотрудником
  // о приеме последнего на работу"
  class contract
  {
     // Члены класса
     private $surname;
     private $name;
     private $patronymic;
     private $age;
     private $date;
     private $description;
     // Конструктор
     __construct contract($sn, $nm, $pt, $ag, $dt, $ds)
     {
        $this->surname = $sn;
        $this->name = $nm;
        $this->patronymic = $pt;
        $this->age = $ag;
        $this->date = $dt;
        $this->description = $ds;
     }
     // Методы доступа к объектам класса
     public function getSurname()
     {
       return $this->surname;
     }
     public function getName()
     {
       return $this->name;
     }
     public function getPatronymic()
     {
       return $this->patronymic;
     }
```

```
    public function getAge()
    {
      return $this->age;
    }
    public function getDate()
    {
      return $this->date;
    }
    public function getDescription()
    {
      return $this->description;
    }
    // Остальная реализация класса
    ...
  }
?>
```

В результате становится невозможным прямое обращение к членам класса `contract`, объявленным со спецификатором доступа `private`, а их значения можно получить только через открытые методы доступа. Поэтому строка для вывода возраста сотрудника, если члены класса закрыты, должна выглядеть так, как показано в листинге 4.5.

Листинг 4.5. Использование открытых методов для доступа к закрытым членам

```
<?php
  ...
  echo "Возраст работника {$obj->getSurname()} -
        {$obj->getAge()} лет<br>";
  ...
?>
```

Таким образом, если в дальнейшем возникнет описанное выше требование, чтобы вместо возраста на момент приема сотрудника на работу возвращался его актуальный возраст, достаточно исправить метод `getAge()`. После этого в местах вызова (а их, как предполагалось, несколько тысяч) автоматически будет выводиться актуальное значение (листинг 4.6).

Листинг 4.6. Исправление метода `getAge()`

```
<?php
  ...
```

```
  public function getAge()
  {
    $delta = date('Y') — substr($this->date, 0, 4);
    $actual_age = $this->age + $delta;
    return $actual_age;
  }
  ...
?>
```

В результате одному разработчику достаточно будет исправить внутреннюю реализацию своего класса, а остальным программистам, которых могут быть десятки, не потребуется вникать в сущность ошибки и редактировать свои участки кода. Именно в этом заключается гибкость и мощность инкапсуляции — можно редактировать внутреннюю реализацию класса, не затрагивая интерфейс, которым пользуются внешние разработчики.

Если класс уже разработан, программисты пользуются полем `$obj->age` для получения возраста сотрудника и не могут изменить его интерфейс; для изменения поведения поля можно воспользоваться аксессором `__get()`, описанным в *главе 3*.

Для того чтобы изменить поведение поля `$obj->age`, закроем член `$obj->age` и перегрузим аксессор `__get()` для свойства `age` (листинг 4.7).

Листинг 4.7. Создание свойства `$obj->age`, доступного для чтения

```
<?php
  // Класс "договор между организацией и сотрудником
  // о приеме последнего на работу"
  class contract
  {
     // Члены класса
     ...
     private $age;
     ...
     private function __get($value)
     {
       if($value == 'age')
       {
         $delta = date('Y') — substr($this->date, 0, 4);
         $actual_age = $this->age + $delta;
         return $actual_age;
       }
```

```
        else return 0;
      }
      ...
  }
?>
```

Теперь обращение к свойству `$obj->age` будет перехватываться перегруженным методом `__get()`, который вернет актуальный возраст сотрудника.

Другой аспект использования инкапсуляции заключается в защите внутренних членов класса от повреждения использующим его сторонним разработчиком. Класс сам заботится о своей внутренней согласованности, тогда как внешний программист знает лишь то, как пользоваться классом. В противном случае, любой желающий сможет изменить внутреннюю реализацию класса произвольным образом, нарушив тем самым согласованность данных. Например, для всех сотрудников может быть установлен возраст, равный 0, в результате чего данные внутри объекта не будут соответствовать друг другу. Правильное использование механизма инкапсуляции исключает подобный вариант: попытка манипулирования закрытыми данными приведет к ошибке.

4.2. Наследование

Одной из главных целей объектно-ориентированного подхода является повторное использование кода. Иногда сложно определить, требуется ли такой подход к решению поставленной задачи или нет. Однако если созданный класс используется лишь однажды в одном приложении, можно утверждать, что его создание было пустой тратой времени.

Хотя при разработке новых приложений могут заимствоваться функции или участки кода из уже разработанных программ, говоря о повторном использовании кода, редко имеют в виду такой подход. Извлечение участка кода из старого проекта часто требует его серьезной переработки или подключения дополнительных файлов, которые дублируют уже существующие функциональные блоки.

Замечание

В идеале класс, созданный в одном приложении, должен свободно извлекаться из него и встраиваться в другое приложение без всякой адаптации.

Код, предназначенный для повторного использования, необходимо подготовить для этого, оформив в виде библиотеки или класса. При этом повторное использование вовсе не гарантирует, что система (особенно первая) будет занимать меньше места и разрабатываться быстрее. За любое ограничение, которое накладывается на код, приходится расплачиваться. В случае, рас-

смотренном в предыдущем разделе, платой служит более сложная организация кода и более длительные сроки его подготовки.

Объектно-ориентированная методология предоставляет два возможных способа повторного использования кода:

- включение объектов в класс (см. *раздел 3.5*);
- использование наследования.

Наследование позволяет создать новый класс на основе уже существующего, автоматически включив в новый класс все члены и методы старого. В рамках наследования "старый" класс называется *базовым*, а вновь создаваемый класс — *производным*. При объявлении производного класса необходимо указать имя базового класса с помощью ключевого слова `extends`. В листинге 4.8 создается базовый класс `base`, содержащий единственный член `$var` и метод `print_var()`. От класса `base` наследуется класс `derived`, содержащий, в свою очередь, член `$val` и метод `print_val()`.

Листинг 4.8. Наследование класса `derived` от класса `base`

```
<?php
  // Базовый класс
  class base
  {
    public $var;
    public function print_var()
    {
      echo $this->var;
    }
  }
  // Производный класс
  class derived extends base
  {
    public $val;
    public function print_val()
    {
      echo $this->val;
    }
  }
?>
```

В листинге 4.9 приводится пример, в котором объявляется объект базового класса `$bobj` и объект производного класса `$dobj`. Для каждого из объектов выводится список его членов и методов.

Листинг 4.9. Анализ объектов базового и производного классов

```
<?php
  // Подключаем базовый и производный классы
  require_once("class.base.php");

  // Объявляем объект базового класса
  $bobj = new base();

  // Члены и методы базового класса
  echo "<pre>";
  print_r($bobj);
  print_r(get_class_methods($bobj));
  echo "</pre>";

  // Объявляем объект производного класса
  $dobj = new derived();

  // Члены и методы производного класса
  echo "<pre>";
  print_r($dobj);
  print_r(get_class_methods($dobj));
  echo "</pre>";
?>
```

Выполнив скрипт из листинга 4.9, можно получить следующие дампы объектов `$bobj` и `$dobj`:

```
base Object
(
    [var] =>
)
Array
(
    [0] => print_var
)
derived Object
(
    [val] =>
    [var] =>
)
Array
(
    [0] => print_val
    [1] => print_var
)
```

Как видно из результатов работы скрипта, объект производного класса `derived` содержит не только собственный член `$val` и метод `print_val()`, но и член `$var` и метод `print_var()`, объявленные в базовом классе. Производный класс, в свою очередь, может выступать в качестве базового для других классов; в результате можно получить разветвленную иерархию классов, добиваясь расширения функциональности без повторного создания членов и методов, а используя уже имеющиеся из существующих классов.

Рассмотрим пример повторного использования кода на примере созданных ранее классов `employee` для моделирования исполнителя и `client` для моделирования заказчика (см. *раздел 3.5*), которые при обнаружении несовпадающих с уже имеющимися фамилий помещают их в файлы соответственно employee.txt и client.txt; если же вводимая фамилия в файле уже присутствует, то извлекается уникальный номер сотрудника или заказчика. В разработанном ранее варианте приходилось дублировать код одного класса в другом. Такой подход может повлечь за собой ошибки, поскольку при необходимости замены кода в нескольких местах нельзя гарантировать, что будут исправлены все его участки. Чтобы исключить дублирование кода (а следовательно, и повторное его исправление), достаточно создать для этих двух методов базовый класс `person`, который будет реализовывать общий код обоих классов (рис. 4.2).

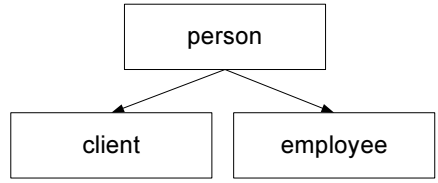


Рис. 4.2. Наследование классов `client` и `employee`

Используя подобную схему, можно не только устранить дублирование кода в двух разных, но схожих по функциональности классах, но и обеспечить базу для его повторного использования. В дальнейшем, когда потребуется ввести в систему новый класс для описания личных данных участников моделируемого процесса, исполняющих какую-либо новую роль, достаточно будет унаследовать его от класса `person`.

Например, со временем может быть введен класс "юрист" (`lawyer`): за каждым заказчиком может быть закреплен юрист, помогающий с оформлением договоров, причем каждый юрист может обслуживать нескольких заказчиков. В любом случае работа по сохранению информации в файл останется полностью сосредоточена в классе `person`, и перевод всей системы с файлового

способа хранения информации на использование СУБД сведется к переработке всего одного класса `person`, не затрагивая остальные элементы системы.

В листинге 4.10 представлена реализация класса `person` с пятью закрытыми членами: `$surname` (фамилия), `$name` (имя), `$patronymic` (отчество), `$number` (уникальный номер) и `$filename` (имя файла для журналирования). Помимо этого имеется конструктор, который и обеспечивает проверку идентичности фамилии, имени и отчества с уже зафиксированными в файле `$filename`.

Листинг 4.10. Реализация класса `person`

```
<?php
  class person
  {
    public function __construct($surname, $name, $patronymic, $filename)
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      $this->filename = $filename;
      // Проверяем, существует ли в файле $this->filename
      // лицо с такими же фамилией, именем и отчеством
      $content = file_get_contents($this->filename);
      // Если лица в файле нет — добавляем
      if(strpos($content, "$surname|$name|$patronymic") === false)
      {
        // Определяем последний номер (предполагается,
        // что номера распределяются по очереди от минимального
        // к максимальному)
        $pattern = "#([\d]+).+?$#i";
        if(preg_match($pattern, $content, $out))
        {
          $this->number = $out[1] + 1;
        }
        else $this->number = 1;
        // Производим запись в файл
        $fd = fopen($this->filename, "a");
        $str = "{$this->number}|".
               "{$this->surname}|".
               "{$this->name}|".
               "{$this->patronymic}\r\n";
        fwrite($fd, $str);
        fclose($fd);
      }
```

```
      // Если лицо уже зафиксировано в файле — извлекаем
      // его уникальный номер
      else
      {
        $pattern = "#([\d]+)\|$surname\|$name\|$patronymic#i";
        if(preg_match($pattern, $content, $out))
        {
          $this->number = $out[1];
        }
      }
    }
    public function __get($index)
    {
      return $this->$index;
    }

    private $surname;
    private $name;
    private $patronymic;
    private $number;
    private $filename;
  }
?>
```

Теперь достаточно унаследовать классы `employee` и `client` от класса `person`, чтобы они автоматически получили возможности базового класса (листинг 4.11).

Листинг 4.11. Реализация классов `employee` и `client`

```
<?php
  require_once("class.person.php");

  class client extends person
  {
    public function __construct($surname, $name, $patronymic)
    {
      person::__construct($surname, $name, $patronymic, "client.txt");
    }
  }
  class employee extends person
  {
    public function __construct($surname, $name, $patronymic)
```

```
    {
      person::__construct($surname, $name, $patronymic, "employee.txt");
    }
  }
?>
```

Как видно из листинга 4.11, производные классы не содержат ничего, кроме конструкторов, которые при помощи префикса `person::` вызывают конструктор базового класса. Порядок работы с конструкторами и деструкторами при наследовании подробнее обсуждается в следующем разделе.

В листинге 4.12 приводится пример использования объектов определенных ранее классов для получения уникальных номеров заказчика и исполнителя.

Листинг 4.12. Использование объектов класса `client` и `employee`

```
<?php
  require_once("class.client.php");
  require_once("class.employee.php");

  $obj = new client("Корнеев", "Иван", "Григорьевич");

  echo "{$obj->number}<br>";

  $obj = new employee("Борисов", "Игорь", "Иванович");

  echo "{$obj->number}<br>";
?>
```

4.3. Конструкторы, деструкторы и наследование

Одна из важнейших особенностей PHP состоит в том, что для корректной инициализации объекта производного класса из его конструктора обязателен вызов конструктора базового класса. Правда, это требование не является строгим, — допускается наследование без вызова конструктора базового класса. В листинге 4.13 представлена альтернативная реализация классов `client` и `employee` из листинга 4.11.

Листинг 4.13. Альтернативная реализация классов `client` и `employee`

```
<?php
  require_once("class.person.php");
```

```
  class client extends person
  {
    public function __construct($surname, $name, $patronymic){}
  }
  class employee extends person
  {
    public function __construct($surname, $name, $patronymic){}
  }

  $obj = new employee("Борисов", "Игорь", "Иванович");
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 4.13 будет следующий дамп объекта:

```
employee Object
(
    [surname:private] =>
    [name:private] =>
    [patronymic:private] =>
    [number:private] =>
    [filename:private] =>
)
```

Как видно из результатов работы скрипта, закрытые члены объекта остались не инициализированными, что недопустимо, т. к. нет никаких других способов их инициализации.

Ситуация с деструктором аналогична: при реализации деструктора производного класса необходимо явно вызывать деструктор базового класса (листинг 4.14).

Замечание

При вызове методов базового класса вместо его имени можно указывать специальное ключевое слово `parent`. Для обозначения текущего класса можно использовать ключевое слово `self`.

Листинг 4.14. Вызов деструктора производного класса из базового

```
<?php
  class base
```

```
  {
    private $var;
    public function __construct($var = 100)
    {
      $this->var = $var;
    }
    public function __destruct()
    {
      echo "Вызов деструктора базового класса<br>";
    }
  }
  class derived extends base
  {
    private $val;
    public function __construct($var = 100, $val = 100)
    {
      parent::__construct($var);
      $this->val = $val;
    }
    public function __destruct()
    {
      parent::__destruct();
      echo "Вызов деструктора производного класса<br>";
    }
  }

  $obj = new derived(20, 20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

В листинге 4.14 производный класс `derived` наследует от базового класса `base`. Так как класс `derived` реализует собственные конструктор и деструктор, требуется явный вызов конструктора и деструктора базового класса `base`. Результатом работы скрипта являются следующие строки:

```
derived Object
(
    [val:private] => 20
    [var:private] => 20
)
Вызов деструктора базового класса
Вызов деструктора производного класса
```

Как можно видеть, выполняется однократный вызов деструкторов базового и производного классов: первый вызывается явно, а второй — неявно.

Конструктор и деструктор базового класса вызываются автоматически без вмешательства программиста в единственном случае — когда они не реализованы в производном классе (листинг 4.15).

Листинг 4.15. Явный вызов конструктора и деструктора базового класса

```
<?php
  class base
  {
    private $var;
    public function __construct($var = 100)
    {
      $this->var = $var;
    }
    public function __destruct()
    {
      echo "Вызов деструктора базового класса<br>";
    }
  }
  class derived extends base
  {
  }

  $obj = new derived(20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Как видно из листинга 4.15, производный класс `derived` не содержит конструктора и деструктора, что приводит к неявному вызову конструктора и деструктора базового класса:

```
derived Object
(
    [var:private] => 20
)
Вызов деструктора базового класса
```

В листингах 4.14 и 4.15 в конструкторах применяются необязательные параметры, которые позволяют не указывать значение при объявлении объекта класса. Следует внимательно следить, чтобы при явном вызове конструктора

производного класса в параметрах не осталось инициализирующих значений. В листинге 4.16 демонстрируется пример ошибочного вызова конструктора базового класса, при котором инициализирующее значение не было переназначено.

Листинг 4.16. Ошибочный вызов конструктора базового класса

```
<?php
  class base
  {
    private $var;
    public function __construct($var = 0)
    {
      $this->var = $var;
    }
  }
  class derived extends base
  {
    private $val;
    public function __construct($var = 100, $val = 100)
    {
      parent::__construct($var = 0);
      $this->val = $val;
    }
  }

  $obj = new derived(20, 20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта всегда будет объект, в котором закрытая переменная `$var` имеет нулевое значение:

```
derived Object
(
    [val:private] => 20
    [var:private] => 0
)
```

Данный эффект вызван тем, что значение по умолчанию предпочтительно назначать при объявлении метода. Попытка инициализации при вызове приведет к тому, что параметр получит новое значение вне зависимости от того,

имелось ли оно у него ранее. Опасность ошибки в данном случае заключается в том, что при средних настройках тревожности PHP-интерпретатор не сообщает о попытке присвоить значение параметру метода непосредственно в точке вызова.

4.4. Спецификаторы доступа и наследование

Члены и методы, объявленные со спецификаторами доступа `public` и `private`, при наследовании ведут себя по отношению к производному классу точно так же, как и по отношению к внешней программе. Это означает, что из производного класса доступны все методы и члены, объявленные со спецификатором доступа `public`, и не доступны компоненты, объявленные как `private`. В листинге 4.17 приводится пример попытки обратиться к закрытому члену `$var` базового класса `base` из конструктора производного класса `derived`.

Листинг 4.17. Попытка обращения к закрытому члену базового класса

```
<?php
  class base
  {
    private $var;
    public function __construct($var)
    {
      $this->var = $var;
    }
    public function print_var()
    {
      echo $this->var;
    }
  }
  class derived extends base
  {
    public function __construct($var)
    {
      $this->var = $var;
    }
  }

  $obj = new derived(20);
  echo "<pre>";
  print_r($obj);
```

```
  echo "</pre>";
  echo $obj->print_var();
?>
```

В данном случае попытка обращения к закрытому члену класса не приводит к ошибке. Дело в том, что производный класс даже не видит попытки обращения к закрытому члену, он просто создает новый открытый член в объекте производного класса:

```
derived Object
(
    [var:private] =>
    [var] => 20
)
```

Именно поэтому обращение к методу `print_var()` не приводит к выводу переменной `$var` — данная переменная остается не инициализированной в рамках класса `base`.

Иногда удобно, чтобы член или метод базового класса, оставаясь закрытым для внешнего кода, был открыт для производного класса. В этом случае прибегают к специальному спецификатору доступа `protected`. Компоненты класса, снабженные спецификатором доступа `protected`, называют *защищенными*.

В листинге 4.18 на примерах базового класса `base` и производного класса `derived` демонстрируется использование защищенного члена `$var`.

Листинг 4.18. Использование спецификатора доступа `protected`

```
<?php
  class base
  {
    protected $var;
    public function __construct($var)
    {
      $this->var = $var;
    }
  }
  class derived extends base
  {
    public function __construct($var)
    {
      $this->var = $var;
    }
  }
```

```
  $obj = new derived(20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
  // echo $obj->var; // Ошибка
?>
```

Результатом работы скрипта из листинга 4.18 будут следующие строки:

```
derived Object
(
    [var:protected] => 20
)
```

Как видно из результатов работы скрипта, конструктор производного класса имеет возможность инициализировать член `$var` в обход конструктора базового класса, в то же время член `$var` остается закрытым по отношению к внешнему коду.

Использование спецификатора `protected` возможно по отношению как к членам класса, так и к методам. Например, если конструктор базового класса объявлен со спецификатором `protected`, то получить его объект невозможно; доступны будут только объекты производных классов (листинг 4.19).

Листинг 4.19. Запрет создания объектов базового класса

```
<?php
  class base
  {
    private $var;
    protected function __construct($var)
    {
      $this->var = $var;
    }
  }
  class derived extends base
  {
    public function __construct($var)
    {
      parent::__construct($var);
    }
  }

  // $obj = new base(20); // Ошибка
  $obj = new derived(20);
```

```
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Этим приемом часто пользуются, когда объект базового класса абстрактен и скрывает в себе функциональность, необходимую для производных классов, а сам по себе не имеет смысла.

4.5. Перегрузка методов

Из сказанного в *разделах 4.3* и *4.4* ясно, что конструктор и деструктор базового класса могут быть переопределены в производном классе. Это справедливо и по отношению к другим методам: в производном классе можно создать метод с таким же названием, что и в базовом классе, который заменит метод базового класса при вызове. Такая процедура называется *перегрузкой методов*.

В листинге 4.20 приводится пример перегрузки метода `print_var()` в производном классе `derived`, который наследуется от базового класса `base`.

Листинг 4.20. Перегрузка метода `print_var()`

```
<?php
  class base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() базового класса<br>";
    }
  }
  class derived extends base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() производного класса<br>";
    }
  }

  $obj = new derived();
  $obj->print_var();
?>
```

Результатом работы скрипта из листинга 4.20 будет следующая строка:

```
Вызов метода print_var() производного класса
```

Таким образом, метод `base::print_var()` не вызывается при обращении к объекту производного класса `derived`.

Однако в рамках производного класса остается возможность вызвать метод базового класса, обратившись к нему при помощи префикса `parent::`. В листинге 4.21 представлена перегрузка метода `print_var()`, при котором метод производного класса сначала вызывает метод базового класса.

Листинг 4.21. Обращение к методу базового класса

```
<?php
  class base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() базового класса<br>";
    }
  }
  class derived extends base
  {
    public function print_var()
    {
      parent::print_var();
      echo "Вызов метода print_var() производного класса<br>";
    }
  }

  $obj = new derived();
  $obj->print_var();
?>
```

Результатом работы скрипта из листинга 4.21 будут уже две строки:

```
Вызов метода print_var() базового класса
Вызов метода print_var() производного класса
```

Таким образом, при помощи перегрузки метода можно как расширить метод базового класса, так и полностью заменить его уже новым методом.

4.6. Определение имени базового класса

Для определения имени базового класса используется функция `get_parent_class()`, которая имеет следующий синтаксис:

```
string get_parent_class($obj)
```

Функция принимает в качестве единственного параметра имя класса или объект и возвращает строку с именем базового класса. Рассмотрим цепочку из трех классов `base`, `derived` и `postderived`, которые наследуют друг другу (рис. 4.3).

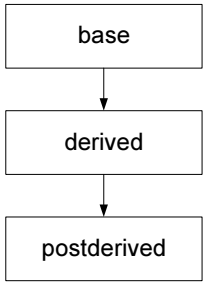


Рис. 4.3. Класс `base` является базовым для класса `derived`, который, в свою очередь, является базовым для класса `postderived`

В листинге 4.22 представлена возможная реализация классов `base`, `derived` и `postderived`. Конструктор каждого класса вызывает функцию `get_parent_class()`, передавая ей в качестве параметра ссылку на текущий объект `this`.

Листинг 4.22. Реализация классов `base`, `derived` и `postderived`

```
<?php
  class base
  {
    public function __construct()
    {
      echo get_parent_class($this);
      echo "<br>";
    }
  }
  class derived extends base
  {
    public function __construct()
    {
      echo get_parent_class($this);
```

```
      echo "<br>";
    }
  }
  class postderived extends derived
  {
    public function __construct()
    {
      echo get_parent_class($this);
      echo "<br>";
    }
  }
?>
```

Теперь при создании объектов всех трех классов в окно браузера будут выводиться имена их базовых классов (листинг 4.23).

Листинг 4.23. Создание объектов классов `base`, `derived` и `postderived`

```
<?php
  require_once("class.base.php");
  require_once("class.derived.php");
  require_once("class.postderived.php");

  $obj = new base();
  $obj = new derived();
  $obj = new postderived();
?>
```

Результатом работы скрипта из листинга 4.23 будут следующие строки:

```
base
derived
```

Как можно видеть из результатов работы скрипта, для объекта первого базового класса `base` вызов функции `get_parent_class()` возвращает пустую строку. Вместо объекта функция может принимать имя класса: таким образом путем последовательных вызовов функции `get_parent_class()` можно восстановить цепочку предков любого объекта (листинг 4.24).

Листинг 4.24. Восстановление цепочки базовых классов для объекта класса `postderived`

```
<?php
  require_once("class.base.php");
```

```
  require_once("class.derived.php");
  require_once("class.postderived.php");

  $obj = new postderived();
  echo "Основатель иерархии — ".
       get_parent_class(get_parent_class($obj))."<br>";
?>
```

Первой функции `get_parent_class()` передается объект `$obj` класса `postderived`, результатом работы которой является строка `derived`. Данная строка передается второй функции `get_parent_class()`, которая возвращает для класса `derived` имя базового класса `base`. Результатом работы скрипта из листинга 4.24 будут следующие две строки:

```
derived
Основатель иерархии — base
```

Скрипт 4.24 создавался из расчета, что вся иерархия наследования состоит из трех классов. Когда количество классов заранее не известно, лучше прибегнуть к циклам `while()` или `do ... while()`, продолжающим вызывать функцию `get_parent_class()` до тех пор, пока не будет достигнут самый первый базовый класс и функция не вернет пустую строку. Пустая строка в контексте цикла равносильна значению `false` и вызовет его остановку. Пример такого скрипта приводится в листинге 4.25.

Листинг 4.25. Восстановление цепочки базовых классов для произвольного объекта

```
<?php
  require_once("class.base.php");
  require_once("class.derived.php");
  require_once("class.postderived.php");

  $obj = new postderived();
  do
  {
    $obj = get_parent_class($obj);
    echo "$obj<br>";
  } while($obj);
?>
```

Другая схожая задача — определение, является ли класс текущего объекта производным от базового. Базовый класс может содержать определенный набор методов, и для того чтобы обращаться к ним, часто следует убедиться,

является ли текущий объект наследником базового класса. Для решения этой задачи используется функция `is_subclass_of()`, которая имеет следующий синтаксис:

```
bool is_subclass_of($obj, $class_name)
```

Функция принимает в качестве первого параметра объект `$obj`, а в качестве второго параметра имя базового класса `$class_name` и возвращает `true`, если объект является экземпляром потомка класса `$class_name`, и `false` — в противном случае. В листинге 4.26 приводится пример использования функции `is_subclass_of()`.

Листинг 4.26. Использование функции `is_subclass_of()`

```
<?php
  require_once("class.base.php");
  require_once("class.derived.php");
  require_once("class.postderived.php");

  $obj = new base();
  if(is_subclass_of($obj, "base")) echo "yes<br>"; // false
  else echo "no<br>";

  $obj = new derived();
  if(is_subclass_of($obj, "base")) echo "yes<br>"; // true
  else echo "no<br>";

  $obj = new postderived();
  if(is_subclass_of($obj, "base")) echo "yes<br>"; // true
  else echo "no<br>";
?>
```

Функция `is_subclass_of()` возвращает `true`, только если класс, которому соответствует второй аргумент, является базовым по отношению к классу объекта первого аргумента, поэтому для объекта класса `base` в листинге 4.26 возвращается значение `false`.

Для того чтобы определить, является ли объект экземпляром данного класса или экземпляром классов его наследников, можно воспользоваться функцией `is_a()`, которая имеет синтаксис, аналогичный с функцией `is_subclass_of()`:

```
bool is_a($obj, $class_name)
```

Функция принимает в качестве первого параметра объект `$obj`, а в качестве второго — имя базового класса `$class_name` и возвращает `true`, если объект является экземпляром класса `$class_name` или экземпляром его потомка, и

false — в противном случае. В листинге 4.27 приводится пример использования функции is_a().

Листинг 4.27. Использование функции is_a()

```
<?php
  require_once("class.base.php");
  require_once("class.derived.php");
  require_once("class.postderived.php");

  $obj = new base();
  if(is_a($obj, "base")) echo "yes<br>"; // true
  else echo "no<br>";

  $obj = new derived();
  if(is_a($obj, "base")) echo "yes<br>"; // true
  else echo "no<br>";

  $obj = new postderived();
  if(is_a($obj, "base")) echo "yes<br>"; // true
  else echo "no<br>";
?>
```

4.7. Полиморфизм

При использовании разветвленных наследственных иерархий все объекты производных классов автоматически снабжаются методами базового класса. Производные классы могут использовать методы базового класса без изменений, адаптировать их или заменять своей собственной реализацией. Как бы ни были реализованы эти методы, можно достоверно утверждать, что все объекты наследственной иерархии классов будут обладать некоторым количеством методов с одними и теми же названиями. Это явление называется *полиморфизмом*.

При использовании полиморфизма появляется возможность программировать работу объектов, абстрагируясь от их типа. Так, в транспортной сети, независимо от того, является ли текущий объект автомобильным, железнодорожным, воздушным или водным транспортным средством, он характеризуется способностью к движению, и для всех объектов "транспортное средство" можно ввести единый метод движения move(). Каждый объект из транспортной иерархии будет обладать этим методом. Несмотря на то что движение по асфальтовой или железной дороге, по воздуху или по реке отличается, мето-

ды будут называться одинаково, и их можно будет вызывать для каждого из объектов, чей класс является наследником класса "транспортное средство" (рис. 4.4).

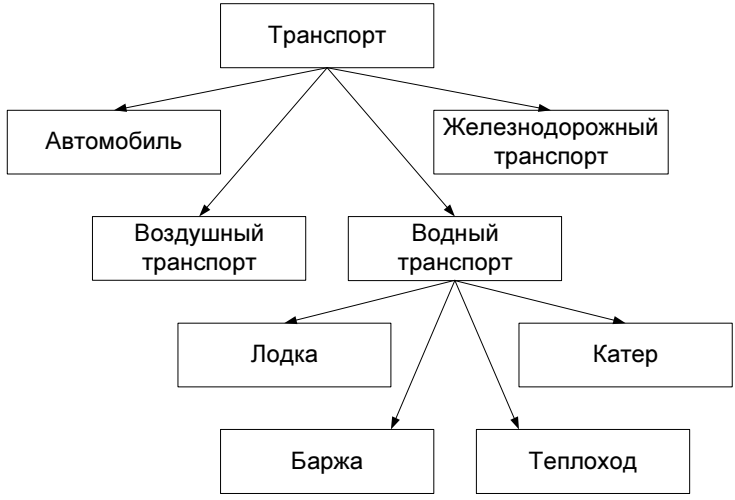


Рис. 4.4. Схема иерархии объектов "транспортные средства"

Метод `move()`, скорее всего, придется переопределить для классов "автомобиль", "воздушный транспорт", "водный транспорт" и "железнодорожный транспорт", т. к. свобода движения у этих классов транспортных средств различна и даже зависит от сезона. При дальнейшем развитии иерархии для наследников класса "водный транспорт" переопределять методы, вероятно, уже не придется, поскольку реализация метода `move()` базового класса "водный транспорт" подойдет для каждого из них.

Однако на PHP редко моделируют такие объемные системы, как транспортные сети, чаще задачи сводятся к разработке Web-инструментария. В качестве примера рассмотрим постраничную навигацию. Объемный список неудобно отображать на странице целиком, т. к. это требует значительных ресурсов. Гораздо нагляднее выводить список, например, по 10 элементов, предоставляя ссылки на оставшиеся страницы. В большинстве случаев такая задача решается без привлечения объектно-ориентированного подхода и тем более без наследуемой иерархии и полиморфизма. Однако в Web-приложении источником списка элементов, к которому следует применить постраничную навигацию, могут выступать база данных, файл со строками, директория с файлами изображений и т. п. Каждый раз создавать отдельную функцию для реализации постраничной навигации не всегда удобно, т. к. придется дублировать значительную часть кода в нескольких функциях. В данном случае

удобнее реализовать постраничную навигацию в методе базового класса `pager`, а методы, работающие с конкретными источниками, переопределить в производных классах:

- `pager_mysql` — постраничная навигация для СУБД MySQL;
- `pager_file` — постраничная навигация для текстового файла;
- `pager_dir` — постраничная навигация для файлов в директории.

Иерархия классов постраничной навигации представлена на рис. 4.5.

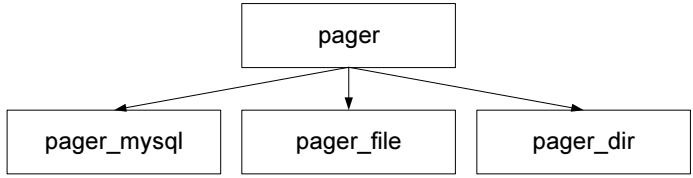


Рис. 4.5. Иерархия классов постраничной навигации

Создадим базовый класс `pager`, который при помощи перегруженного метода `__toString()` (см. *раздел 3.14*) будет выводить ссылки постраничной навигации в следующем формате:

```
[1-10]... [281-290] [291-300] [301-310] [311-320] [321-330] ... [511-517]
```

Номера отображенных на экране элементов [301—310] выделяются жирным шрифтом. Крайняя левая и крайняя правая ссылки позволяют перейти в начало и конец списка, а вокруг текущей страницы выводятся ссылки на соседние страницы.

Класс `pager()` "не знает", каким образом производные классы будут узнавать общее количество позиций в списке, сколько позиций будет отображаться на одной странице, сколько ссылок находится слева и справа от текущей страницы. Поэтому помимо метода `__ToString()` класс будет содержать четыре защищенных (`protected`) метода, которые возвращают:

- `get_total()` — общее количество позиций в списке;
- `get_pnumber()` — количество позиций на странице;
- `get_page_link()` — количество позиций слева и справа от текущей страницы;
- `get_parameters()` — строку, которую необходимо передать по ссылкам на другую страницу (например, при постраничном выводе результатов поиска по ссылкам придется передавать результаты поиска).

В листинге 4.28 представлена одна из возможных реализаций класса `pager`. Следует обратить внимание, что перечисленные выше методы не несут ника-

кой функциональности: они необходимы лишь для того, чтобы обращение к ним из метода `__toString()` не приводило к ошибке. Сами методы должны быть реализованы в производных классах.

Замечание

Конструктор класса `pager` объявлен защищенным, т. е. из внешнего кода не будет никакой возможности объявить объект класса `pager`: такая предосторожность необходима, поскольку сам по себе класс `pager` не является работоспособным, он лишь задает функциональность производных классов.

Листинг 4.28. Реализация класса `pager`

```
<?php
  class pager
  {
    protected function __construct()
    {
    }
    protected function get_total()
    {
      // Общее количество записей
    }
    protected function get_pnumber()
    {
      // Количество позиций на странице
    }
    protected function get_page_link()
    {
      // Количество ссылок слева и справа
      // от текущей страницы
    }
    protected function get_parameters()
    {
      // Дополнительные параметры, которые
      // необходимо передать по ссылке
    }

    // Ссылки на другие страницы
    public function __toString()
    {
      // Строка для возвращаемого результата
      $return_page = "";
```

```
// Через GET-параметр page передается номер
// текущей страницы      $page = $_GET['page'];
if(empty($page)) $page = 1;

// Вычисляем число страниц в системе
$number = (int)($this->get_total()/$this->get_pnumber());
if((float)($this->get_total()/$this->get_pnumber()) - $number != 0)
{
  $number++;
}
// Проверяем, есть ли ссылки слева
if($page - $this->get_page_link() > 1)
{
  $return_page .= "<a href=$_SERVER[PHP_SELF]".
        "?page=1{$this->get_parameters()}>
        [1-{$this->get_pnumber()}]
        </a>  ...  ";
  // Есть
  for($i = $page - $this->get_page_link(); $i<$page; $i++)
  {
    $return_page .= " <a href=$_SERVER[PHP_SELF]".
          "?page=$i{$this->get_parameters()}>
           [".(($i - 1)*$this->get_pnumber() + 1).
           "-".$i*$this->get_pnumber()."]
           </a> ";
  }
}
else
{
  // Нет
  for($i = 1; $i<$page; $i++)
  {
    $return_page .= " <a href=$_SERVER[PHP_SELF]".
          "?page=$i{$this->get_parameters()}>
          [".(($i - 1)*$this->get_pnumber() + 1).
          "-".$i*$this->get_pnumber()."]
           </a> ";
  }
}
// Проверяем, есть ли ссылки справа
if($page + $this->get_page_link() < $number)
{
  // Есть
  for($i = $page; $i<=$page + $this->get_page_link(); $i++)
```

```
    {
      if($page == $i)
        $return_page .= " [".
            (($i — 1) * $this->get_pnumber() + 1).
             "-".$i*$this->get_pnumber()."] ";
      else
        $return_page .= " <a href=$_SERVER[PHP_SELF]".
             "?page=$i{$this->get_parameters()}>
             [".(($i — 1)*$this->get_pnumber() + 1).
             "-".$i*$this->get_pnumber()."]
             </a> ";
    }
    $return_page .= " ...  ".
         "<a href=$_SERVER[PHP_SELF]".
         "?page=$number{$this->get_parameters()}>
         [".(($number — 1)*$this->get_pnumber() + 1).
         "-{$this->get_total()}]
         </a> ";
  }
  else
  {
    // Нет
    for($i = $page; $i<=$number; $i++)
    {
      if($number == $i)
      {
        if($page == $i)
          $return_page .= " [".
                          (($i — 1)*$this->get_pnumber() + 1).
                          "-{$this->get_total()}] ";
        else
          $return_page .= " <a href=$_SERVER[PHP_SELF]".
               "?page=$i{$this->get_parameters()}>
               [".(($i — 1)*$this->get_pnumber() + 1).
               "-{$this->get_total()}]
               </a> ";
      }
      else
      {
        if($page == $i)
          $return_page .= " [".
              (($i — 1)*$this->get_pnumber() + 1).
               "-".$i*$this->get_pnumber()."] ";
```

```
            else
              $return_page .= " <a href=$_SERVER[PHP_SELF]".
                    "?page=$i{$this->get_parameters()}>
                    [".(($i - 1)*$this->get_pnumber() + 1).
                    "-".$i*$this->get_pnumber()."]
                    </a> ";
          }
        }
      }
      return $return_page;
    }
  }
?>
```

Заранее не известно, для какой страницы будет выводиться навигация, поэтому в ссылках используется элемент суперглобального массива `$_SERVER['PHP_SELF']`, который возвращает номер текущей страницы. Нумерация страниц начинается с 1; номер текущей страницы передается через GET-параметр `page` и доступен в суперглобальном массиве `$_GET['page']`.

Замечание

Суперглобальные массивы (`$_GET`, `$_POST`, `$_SESSION`, `$_COOKIE`, `$_GLOBAL`, `$_REQUEST`) доступны в любой части скрипта, поэтому не обязательно предусматривать передачу их элементов объекту.

4.8. Файловая постраничная навигация

Реализуем постраничную навигацию при помощи производного класса `pager_file`, который будет работать с источником и перегрузит методы `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()` базового класса `pager`. Рассмотрим простейший случай, когда класс `pager_file` читает строки из текстового файла и выводит 10 строк на одной странице, предоставляя ссылки на другие страницы (листинг 4.29).

Листинг 4.29. Класс `pager_file`

```
<?php
  // Подключаем базовый класс
  require_once("class.pager.php");

  class pager_file extends pager
  {
```

```
// Имя файла
protected $filename;
// Количество позиций на странице
private $pnumber;
// Количество ссылок слева и справа
// от текущей страницы
private $page_link;
// Параметры
private $parameters;
// Конструктор
public function __construct($filename,
                            $pnumber = 10,
                            $page_link = 3,
                            $parameters = "")
{
  $this->filename   = $filename;
  $this->pnumber    = $pnumber;
  $this->page_link  = $page_link;
  $this->parameters = $parameters;
}
public function get_total()
{
  $countline = 0;
  // Открываем файл
  $fd = fopen($this->filename, "r");
  if($fd)
  {
    // Подсчитываем количество записей
    // в файле
    while(!feof($fd))
    {
      fgets($fd, 10000);
      $countline++;
    }
    // Закрываем файл
    fclose($fd);
  }
  return $countline;
}
public function get_pnumber()
{
  // Количество позиций на странице
  return $this->pnumber;
}
```

```
public function get_page_link()
{
  // Количество ссылок слева и справа
  // от текущей страницы
  return $this->page_link;
}
public function get_parameters()
{
  // Дополнительные параметры, которые
  // необходимо передать по ссылке
  return $this->parameters;
}
// Возвращает массив строк файла
// по номеру страницы $index
public function get_page()
{
  // Текущая страница
  $page = $_GET['page'];
  if(empty($page)) $page = 1;
  // Количество записей в файле
  $total = $this->get_total();
  // Вычисляем число страниц в системе
  $number = (int)($total/$this->get_pnumber());
  if((float)($total/$this->get_pnumber()) - $number != 0) $number++;
  // Проверяем, попадает ли запрашиваемый номер
  // страницы в интервал от 1 до get_total()
  if($page <= 0 || $page > $number) return 0;
  // Извлекаем позиции текущей страницы
  $arr = array();
  $fd = fopen($this->filename, "r");
  if(!$fd) return 0;
  // Номер, начиная с которого следует
  // выбирать строки файла
  $first = ($page - 1)*$this->get_pnumber();
  for($i = 0; $i < $total; $i++)
  {
    $str = fgets($fd, 10000);
    // Пока не достигнут номер $first,
    // досрочно заканчиваем итерацию
    if($i < $first) continue;
    // Если достигнут конец выборки,
    // досрочно покидаем цикл
    if($i > $first + $this->get_pnumber() - 1) break;
```

```
        // Помещаем строки файла в массив,
        // который будет возвращен методом
        $arr[] = $str;
      }
      fclose($fd);

      return $arr;
    }
  }
?>
```

Конструктор класса `page_file` принимает четыре параметра:

- `$filename` — имя текстового файла, выступающего источником строк;
- `$pnumber` — количество позиций на странице, необязательный параметр; если не указывается при создании объекта, принимает значение 10;
- `$page_link` — количество ссылок слева и справа от текущей страницы, необязательный параметр; если не указывается при создании объекта, принимает значение 3;
- `$parameters` — параметры для передачи по ссылке на другие страницы, необязательный параметр; если не указывается при создании объекта, принимает значение 3.

Данные четыре параметра передаются соответствующим закрытым членам класса. Реализация методов `get_pnumber()`, `get_page_link()` и `get_parameters()` сводится к возвращению значений соответствующих закрытых членов. Метод `get_total()` открывает файл с именем `$filename`, переданным в конструкторе, и подсчитывает количество строк в файле при каждом обращении. С точки зрения производительности было бы разумно завести закрытую переменную `$total` и присваивать ей значение только один раз в конструкторе, т. к. операция сканирования файла достаточно трудоемка. Однако это не всегда удобно, поскольку количество записей в файле может изменяться на всем протяжении существования объекта.

Подсчет строк в файле осуществляется путем чтения строк функцией `fgets()` в цикле `while()`. До тех пор, пока конец файла не достигнут, функция `feof()` возвращает значение `false`, и цикл продолжает работу. Функция `fgets()` читает из файла количество символов, указанное во втором параметре; чтение символов заканчивается, если функция встречает символ перевода строки. Обычно строки в текстовых файлах гораздо короче 10 000 символов, поэтому чтение всегда выполняется корректно.

Замечание

Для работы с текстовыми файлами можно использовать функцию `file()`, которая возвращает содержимое текстового файла в виде массива, каждый элемент которого соответствует отдельной строке файла. Однако для файлов большого объема функция `file()` не всегда подходит. Дело в том, что содержимое файла приходится полностью загружать в память скрипта, которая зачастую ограничена 8 или 16 Мбайт, а это приводит к аварийному завершению работы функции `file()`. При использовании функции `fgets()` в цикле `while()` для хранения содержимого файла скрипт в каждую секунду времени использует не больше 10 Кбайт (переменная `$str`).

Помимо указанных выше методов, объект снабжен методом `get_page()`, который возвращает массив строк файла, соответствующих текущей позиции. Функция ориентируется на GET-параметр `page`, который используется для указания номера страницы в постраничной навигации. Если этот параметр не установлен, считается, что текущей является первая страница. Перед использованием параметра проверятся, не выходит ли его значение за пределы допустимого интервала; если выходит, параметр принимает значение 0. Далее вычисляется номер строки `$first`, начиная с которой следует выбирать строки из файла и помещать их в массив `$arr`, возвращаемый методом `get_page()` в качестве результата. Пока счетчик цикла `for()` не достиг величины `$first`, итерация цикла может быть прекращена досрочно при помощи ключевого слова `continue`. Если счетчик цикла превысил величину, равную `$first` плюс количество позиций на странице, цикл прекращает работу при помощи ключевого слова `break`.

Замечание

Следует отметить, что ни класс `pager`, ни класс `pager_file` не используют ни одного оператора `echo`; результат возвращается при помощи ключевых слов `return`. Очень важно помещать в классы как можно меньше информации о дизайне и месте вывода информации на странице. Именно такой подход позволяет повторно использовать классы в других проектах. Местоположение вывода — это задача либо клиентского кода, либо специального класса вывода; абстрактный тип данных должен решать только свойственные ему задачи.

Теперь, когда готов первый производный класс постраничной навигации, можно воспользоваться им для представления файла большого объема — стандартного словаря операционной системы Linux, который можно найти в /usr/share/dict/linux.words.

Замечание

Если вы работаете в операционной системе Windows, то найти файл linux.words можно на прилагаемом к данной книге компакт-диске.

В листинге 4.30 приводится пример построения постраничной навигации по словарю linux.words.

Листинг 4.30. Постраничная навигация по файлу

```
<?php
  require_once("class.pager_file.php");

  // Объявляем объект постраничной навигации
  $obj = new pager_file("linux.words");

  // Выводим содержимое текущей страницы
  $arr = $obj->get_page();
  for($i = 0; $i < count($arr); $i++)
  {
    echo "{$arr[$i]}<br>";
  }

  // Выводим ссылки на другие страницы
  echo $obj;
?>
```

Результат работы скрипта из листинга 4.30 представлен на рис. 4.6.

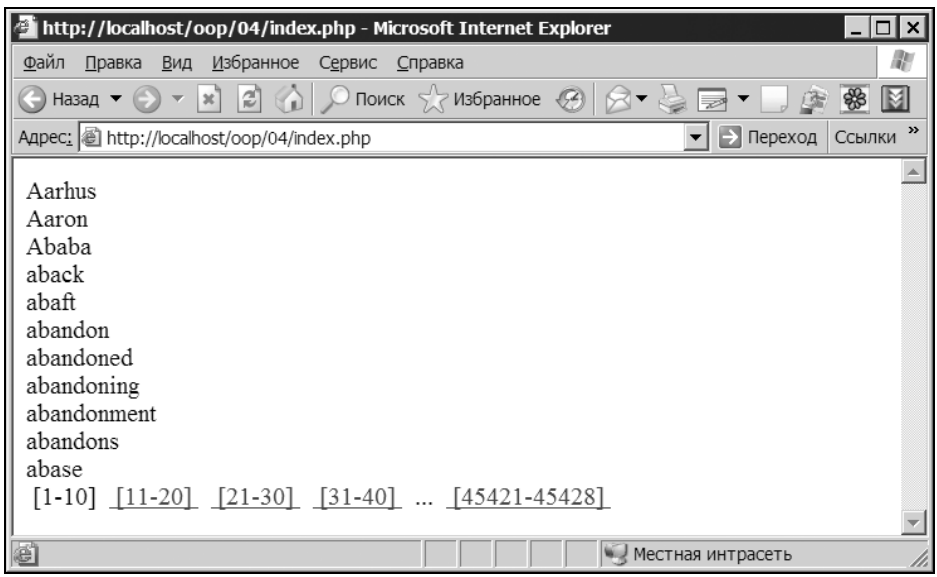


Рис. 4.6. Постраничная навигация

4.9. Постраничная навигация и поиск

Классы не следует воспринимать как нечто статическое и завершенное. Основное преимущество объектно-ориентированного подхода заключается в том, что уже существующие классы легко расширяются. Например, пользователю может понадобиться возможность вывести в окно браузера все слова из словаря, начинающиеся с введенных им символов, а результаты поиска разбить на страницы. Разработанный ранее класс `pager_file` не подходит для решения этой задачи, т. к. не учитывает дополнительное ограничение. Однако не стоит спешить его модифицировать, поскольку это может отразиться на коде, уже использующем этот класс. Более правильным будет унаследовать от класса `pager_file` новый класс `pager_file_search`, модифицировав его отдельные методы (рис. 4.7).

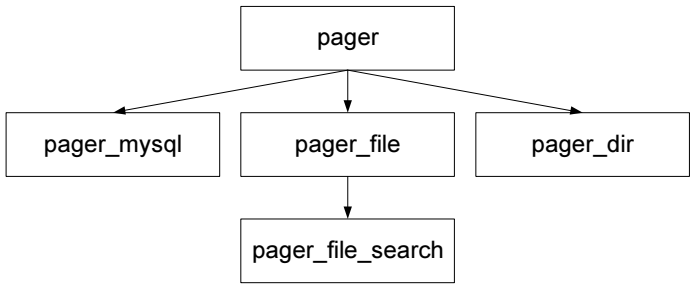


Рис. 4.7. Расширение функциональности существующей системы путем наследования

В листинге 4.31 представлена возможная реализация класса `pager_file_search`, позволяющая устанавливать фильтры на позиции из словаря.

Листинг 4.31. Класс `pager_file_search`

```
<?php
  // Подключаем базовый класс
  require_once("class.pager_file.php");

  class pager_file_search extends pager_file
  {
    // Начало слова
    private $search;
    // Конструктор
    public function __construct($search,
                                $filename,
                                $pnumber = 10,
                                $page_link = 3)
```

```
{
  parent::__construct($filename,
                      $pnumber,
                      $page_link,
                      "&search=".urlencode($search));
  $this->search   = $search;
}
public function get_total()
{
  $countline = 0;
  // Открываем файл
  $fd = fopen($this->filename, "r");
  if($fd)
  {
    // Подсчитываем количество записей
    // в файле
    while(!feof($fd))
    {
      $str = fgets($fd, 10000);
      if(preg_match("|^".preg_quote($this->search)."|i", $str))
      {
        $countline++;
      }
    }
    // Закрываем файл
    fclose($fd);
  }
  return $countline;
}
// Возвращает массив строк файла
// по номеру страницы $index
public function get_page()
{
  // Текущая страница
  $page = $_GET['page'];
  if(empty($page)) $page = 1;
  // Количество записей в файле
  $total = $this->get_total();
  // Вычисляем число страниц в системе
  $number = (int)($total/$this->get_pnumber());
  if((float)($total/$this->get_pnumber()) - $number != 0) $number++;
  // Проверяем, попадает ли запрашиваемый номер
  // страницы в интервал от 1 до get_total()
```

```
      if($page <= 0 || $page > $number) return 0;
      // Извлекаем позиции текущей страницы
      $arr = array();
      $fd = fopen($this->filename, "r");
      if(!$fd) return 0;
      // Номер, начиная с которого следует
      // выбирать строки файла
      $first = ($page - 1)*$this->get_pnumber();
      while(!feof($fd))
      {
        $str = fgets($fd, 10000);
        if(preg_match("|^".preg_quote($this->search)."|i", $str))
        {
          $countline++;
          // Пока не достигнут номер $first,
          // досрочно заканчиваем итерацию
          if($countline < $first) continue;
          // Если достигнут конец выборки,
          // досрочно покидаем цикл
          if($countline > $first + $this->get_pnumber() - 1) break;
          // Помещаем строки файла в массив,
          // который будет возвращен методом
          $arr[] = $str;
        }
      }
      // Закрываем файл
      fclose($fd);

      return $arr;
    }
  }
?>
```

Как видно из листинга 4.31, перегрузке подвергся конструктор, который первым параметром `$search` теперь принимает искомое словосочетание, вторым параметром `$filename` — имя файла-источника и лишь затем необязательные параметры, среди которых отсутствует `$parametrs`. Последний параметр формируется явно при вызове конструктора базового класса — именно с его помощью передается искомое словосочетание `$search`.

Помимо конструктора перегрузке подверглись методы `get_total()` и `get_page()`, которые при помощи регулярного выражения, выстраиваемого на основе члена `$search`, проверяют каждую строку файла на соответствие заданному условию. В листинге 4.32 приводится пример использования объек-

та класса `pager_file_search`, в котором задается вывод всех слов из словаря, начинающихся с последовательности "ab" по пять позиций на странице.

Листинг 4.32. Использование объекта класса `pager_file_search`

```
<?php
  require_once("class.pager_file_search.php");

  // Объявляем объект постраничной навигации
  $obj = new pager_file_search("ab", "linux.words", 5);

  // Выводим содержимое текущей страницы
  $arr = $obj->get_page();
  for($i = 0; $i < count($arr); $i++)
  {
    echo "{$arr[$i]}<br>";
  }

  // Выводим ссылки на другие страницы
  echo $obj;
?>
```

Теперь не представляет сложности создать небольшое Web-приложение по поиску слов в Linux-словаре (листинг 4.33).

Замечание

При поиске для передачи параметров из HTML-формы обработчику лучше пользоваться методом `GET`, чем `POST`. Во-первых, это позволяет ссылаться на результаты поиска, во-вторых, для организации постраничной навигации можно легко передавать все параметры формы через URL, не помещая их в сессию (т. к. массив для хранения данных сессий `$_SESSION` является суперглобальным и не сбрасывается автоматически, это зачастую приводит к созданию сложных систем, в которых легко допустить ошибку).

Листинг 4.33. Поиск по файлу linux.words

```
<form method=get>
<input type=text name=search
       value=<?= htmlspecialchars($_GET['search'], ENT_QUOTES); ?>>
<input type=submit value=Искать>
</form>
<?php
  require_once("class.pager_file_search.php");
```

```
  // Обработчик HTML-формы
  if(!empty($_GET))
  {
    // Объявляем объект постраничной навигации
    $obj = new pager_file_search($_GET['search'], "linux.words", 5);

    // Выводим содержимое текущей страницы
    $arr = $obj->get_page();
    for($i = 0; $i < count($arr); $i++)
    {
      echo "{$arr[$i]}<br>";
    }

    // Выводим ссылки на другие страницы
    echo $obj;
 }
?>
```

Результат работы скрипта из листинга 4.33 представлен на рис. 4.8.

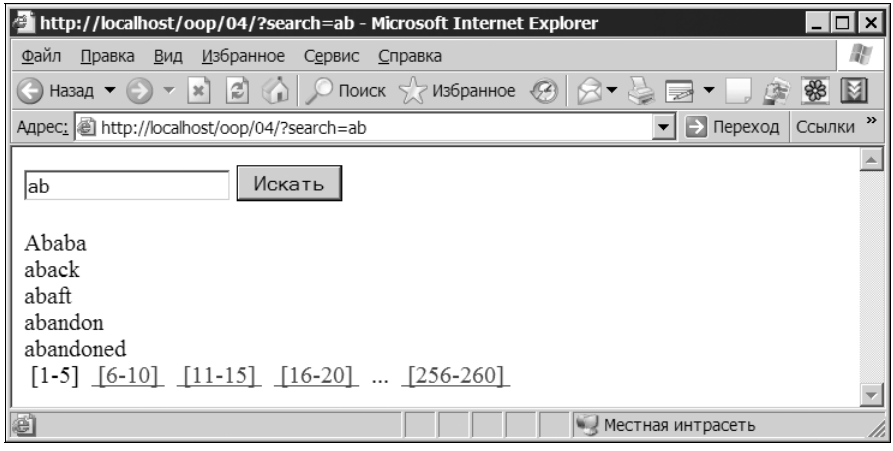


Рис. 4.8. Поиск по словарю linux.words

4.10. Постраничная навигация для директории

В качестве источника позиций, нуждающихся в постраничной навигации, может выступать директория с файлами. Сами файлы могут иметь разную природу — это может быть фотография, путь к которой следует передать ат-

рибуту `src` тега `<img>`, или текстовый файл с сообщением для гостевой книги. Поэтому класс постраничной навигации `pager_dir` должен, по аналогии с `pager_file`, возвращать массив путей к файлам, а выводить их в окно браузера. Это позволит, не сужая области применения класса, использовать его для организации постраничной навигации по любой произвольной директории.

В листинге 4.34 представлен класс `pager_dir`, который перегружает методы `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()` базового класса `pager`. Конструктор класса принимает в качестве первого параметра имя директории с файлами `$dirname`, количество позиций на текущей странице `$pnumber`, количество ссылок слева и справа от текущей страницы `$page_link` и строку с GET-параметрами `$parameters`, которую следует передавать по ссылкам.

Замечание

От класса `pager_dir` можно, в свою очередь, унаследовать класс `pager_dir_sort`, позволяющий задавать критерий сортировки файлов, или класс `pager_dir_recurse`, который осуществляет рекурсивный спуск и учитывает файлы вложенных поддиректорий.

Листинг 4.34. Класс `pager_dir`

```
<?php
  // Подключаем базовый класс
  require_once("class.pager.php");

  class pager_dir extends pager
  {
    // Имя директории
    protected $dirname;
    // Количество позиций на странице
    private $pnumber;
    // Количество ссылок слева и справа
    // от текущей страницы
    private $page_link;
    // Параметры
    private $parameters;
    // Конструктор
    public function __construct($dirname,
                                $pnumber = 10,
                                $page_link = 3,
                                $parameters = "")
```

```
{
  // Удаляем последний символ /, если он имеется
  $this->dirname    = trim($dirname, "/");
  $this->pnumber    = $pnumber;
  $this->page_link  = $page_link;
  $this->parameters = $parameters;
}
public function get_total()
{
  $countline = 0;
  // Открываем директорию
  if(($dir = opendir($this->dirname)) !== false)
  {
    while(($file = readdir($dir)) !== false)
    {
      // Если текущая позиция является файлом,
      // учитываем ее
      if(is_file($this->dirname."/".$file)) $countline++;
    }
    // Закрываем директорию
    closedir($dir);
  }
  return $countline;
}
public function get_pnumber()
{
  // Количество позиций на странице
  return $this->pnumber;
}
public function get_page_link()
{
  // Количество ссылок слева и справа
  // от текущей страницы
  return $this->page_link;
}
public function get_parameters()
{
  // Дополнительные параметры, которые
  // необходимо передать по ссылке
  return $this->parameters;
}
// Возвращает массив строк файла
// по номеру страницы $index
public function get_page()
```

```
    {
      // Текущая страница
      $page = $_GET['page'];
      if(empty($page)) $page = 1;
      // Количество записей в файле
      $total = $this->get_total();
      // Вычисляем количество страниц в системе
      $number = (int)($total/$this->get_pnumber());
      if((float)($total/$this->get_pnumber()) - $number != 0) $number++;
      // Проверяем, попадает ли запрашиваемый номер
      // страницы в интервал от 1 до get_total()
      if($page <= 0 || $page > $number) return 0;
      // Извлекаем позиции текущей страницы
      $arr = array();
      // Номер, начиная с которого следует
      // выбирать строки файла
      $first = ($page - 1)*$this->get_pnumber();
      // Открываем директорию
      if(($dir = opendir($this->dirname))  === false) return 0;
      $i = -1;
      while(($file = readdir($dir)) !== false)
      {
        // Если текущая позиция является файлом
        if(is_file($this->dirname."/".$file))
        {
          // Увеличиваем счетчик
          $i++;
          // Пока не достигнут номер $first,
          // досрочно заканчиваем итерацию
          if($i < $first) continue;
          // Если достигнут конец выборки,
          // досрочно покидаем цикл
          if($i > $first + $this->get_pnumber() - 1) break;
          // Помещаем пути к файлам в массив,
          // который будет возвращен методом
          $arr[] = $this->dirname."/".$file;
        }
      }
      // Закрываем директорию
      closedir($dir);

      return $arr;
    }
  }
?>
```

Класс pager_dir отличается от класса pager_file только реализацией метода get_total(), предназначенного для возвращения количества файлов в директории, и метода get_page(), возвращающего массив путей к файлу для текущей страницы. В основе обоих методов лежит использование функций для работы с директориями. Порядок работы с директорией точно такой же, как и с файлом: директория открывается при помощи функции opendir(), функция принимает в качестве единственного параметра имя директории и возвращает дескриптор $dir, который затем используется в качестве первого параметра для всех остальных функций, работающих с директорией. В цикле while() осуществляется последовательное чтение элементов директории при помощи функции readdir(). Функция возвращает лишь имя файла, поэтому при обращении к файлу необходимо формировать путь, добавляя перед его названием имя директории $this->dirname.

Замечание

Пользователь класса может задавать имя директории как со слэшем на конце — "photo/", так и без него — "photo". Чтобы не создавать специальной обработки подобной ситуации, в конструкторе класса pager_dir косая черта в конце строки удаляется при помощи функции trim(). По умолчанию данная функция удаляет пробелы в начале и в конце строки, однако при помощи второго параметра можно указать удаляемый символ, отличный от пробела.

Следует отметить, что результат присвоения переменной $file значения функции readdir() сравнивается со значением false. Обычно в качестве аргумента цикла while() используется выражение $file = readdir($dir), однако в данном случае это недопустимо, т. к. имя файла может начинаться с 0, что в контексте оператора while() будет рассматриваться как false и приводить к остановке цикла.

В директории могут находиться как файлы, так и поддиректории; обязательно учитываются как минимум две директории: текущая "." и родительская "..". Поэтому при подсчете количества или при формировании массива файлов для текущей страницы важно подсчитывать именно файлы, избегая директорий. Для этой цели служит функция is_file(), которая возвращает true, если переданный ей в качестве аргумента путь ведет к файлу, и false — в противном случае.

В листинге 4.35 приводится пример использования класса pager_dir для вывода фотографий из папки photo по три штуки на странице. Конструктор класса принимает в качестве первого параметра имя директории, в качестве второго, задающего количество позиций на странице, по умолчанию используется число 3.

Листинг 4.35. Использование класса `pager_dir`

```
<?php
  require_once("class.pager_dir.php");

  // Объявляем объект постраничной навигации
  $obj = new pager_dir("photo", 3);

  // Выводим содержимое текущей страницы
  $arr = $obj->get_page();
  for($i = 0; $i < count($arr); $i++)
  {
    echo "<img src={$arr[$i]}>   ";
  }
  echo "<br>";

  // Выводим ссылки на другие страницы
  echo $obj;
?>
```

Результат работы скрипта из листинга 4.35 может выглядеть так, как это представлено на рис. 4.9.

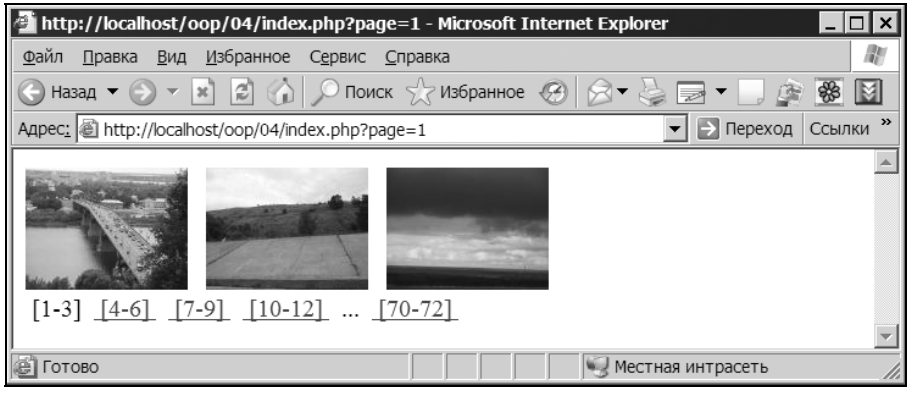


Рис. 4.9. Постраничная навигация для файлов в директории

4.11. Постраничная навигация для базы данных

Помимо файлов и директорий, постраничная навигация часто применяется для вывода позиций из базы данных. Реализуем производный класс `pager_mysql`, который расширит класс `pager` для работы с СУБД MySQL.

Замечание

Очень часто для доступа к СУБД MySQL организуют специальный класс. Это не оправданно, как и в случае с классом доступа к файлу: если приложение работает только с одним типом СУБД, незачем подменять стандартный прозрачный интерфейс доступа собственным объектно-ориентированным интерфейсом, который не будет знаком никому, кроме самого разработчика. Однако когда приложение проектируется для работы сразу с несколькими базами данных, управляемыми разными СУБД, полезно создать класс, который будет адаптировать запросы для доступа к разным типам СУБД.

Замечание

Для доступа к СУБД MySQL следует воспользоваться новой библиотекой php_mysqli, которая предоставляет объектно-ориентированный интерфейс (см. *приложение 1*). К сожалению, данная библиотека пока не получила столь же широкого распространения, как и библиотека php_mysql.

В листинге 4.36 представлена одна из возможных реализаций класса `pager_mysql`. В задачу класса не входит установка соединения с базой данных — этим будет заниматься внешний код. Конструктор класса принимает в качестве параметров имя таблицы `$tablename`, `WHERE`-условие `$where` и критерий сортировки `$order`. Конструктор класса не пытается предотвратить SQL-инъекции, т. к. предполагается, что объекту передаются уже сформированные фрагменты SQL-запроса, и внешний программист позаботился о предотвращении данного вида атаки.

Замечание

SQL-запросы зачастую формируются динамически, т. е. включают в себя одну или несколько переменных. Если значение таких переменных может быть умышленно искажено с целью изменения логики SQL-запроса, имеет место SQL-инъекция. Подробнее с данным видом атак можно ознакомиться в нашей книге "Головоломки на PHP для хакера".

Замечание

Представленный в листинге 4.36 класс `pager_mysql` работает лишь с одной таблицей, именно поэтому в нем не предусмотрены параметры для группировки при помощи ключевого слова `GROUP BY` и условий `ON` и `HAVING`. Для реализации многотабличных запросов потребуется унаследовать от `pager` новый класс `pager_mysql_multi`.

Листинг 4.36. Реализация класса `pager_mysql`

```
<?php
  // Подключаем базовый класс
  require_once("class.pager.php");
```

```
class pager_mysql extends pager
{
  // Имя таблицы
  protected $tablename;
  // WHERE-условие
  protected $where;
  // Критерий сортировки ORDER
  protected $order;
  // Количество позиций на странице
  private $pnumber;
  // Количество ссылок слева и справа
  // от текущей страницы
  private $page_link;
  // Параметры
  private $parameters;
  // Конструктор
  public function __construct($tablename,
                              $where = "",
                              $order = "",
                              $pnumber = 10,
                              $page_link = 3,
                              $parameters = "")
  {
    $this->tablename  = $tablename;
    $this->where      = $where;
    $this->order      = $order;
    $this->pnumber    = $pnumber;
    $this->page_link  = $page_link;
    $this->parameters = $parameters;
  }
  public function get_total()
  {
    // Формируем запрос на получение
    // общего количества записей в таблице
    $query = "SELECT COUNT(*) FROM {$this->tablename}
              {$this->where}
              {$this->order}";
    $tot = mysql_query($query);
    if(!$tot) exit("Ошибка подсчета количества
                    позиций<br>{mysql_error()}<br>$query");
    return mysql_result($tot, 0);
  }
```

```
public function get_pnumber()
{
  // Количество позиций на странице
  return $this->pnumber;
}
public function get_page_link()
{
  // Количество ссылок слева и справа
  // от текущей страницы
  return $this->page_link;
}
public function get_parameters()
{
  // Дополнительные параметры, которые
  // необходимо передать по ссылке
  return $this->parameters;
}
// Возвращает массив строк файла
// по номеру страницы $index
public function get_page()
{
  // Текущая страница
  $page = $_GET['page'];
  if(empty($page)) $page = 1;
  // Количество записей в файле
  $total = $this->get_total();
  // Вычисляем количество страниц в системе
  $number = (int)($total/$this->get_pnumber());
  if((float)($total/$this->get_pnumber()) - $number != 0) $number++;
  // Проверяем, попадает ли запрашиваемый номер
  // страницы в интервал от 1 до get_total()
  if($page <= 0 || $page > $number) return 0;
  // Извлекаем позиции текущей страницы
  $arr = array();
  // Номер, начиная с которого следует
  // выбирать строки файла
  $first = ($page - 1)*$this->get_pnumber();
  // Извлекаем позиции для текущей страницы
  $query = "SELECT * FROM {$this->tablename}
            {$this->where}
            {$this->order}
            LIMIT $first, {$this->get_pnumber()}";
  $tbl = mysql_query($query);
```

```
      if(!$tbl) exit("Ошибка обращения к таблице
                      позиций<br>{mysql_error()}<br>$query");
      // Если имеется хотя бы один элемент,
      // заполняем массив $arr
      if(mysql_num_rows($tbl))
      {
        while($arr[] = mysql_fetch_array($tbl));
      }
      // Удаляем последний нулевой элемент
      // массива $arr
      unset($arr[count($arr) - 1]);

      return $arr;
    }
  }
?>
```

Точно так же, как и в предыдущих классах, наиболее серьезной модификации подвергаются метод `get_total()`, возвращающий количество записей в таблице `$tablename`, и метод `get_page()`, который возвращает двумерный массив: каждый элемент массива представляет собой массив полей одной из записи.

Для подсчета количества записей таблицы в методе `get_total()` используется функция `COUNT(*)`:

```
SELECT COUNT(*) FROM tbl
```

В методе `get_page()`, возвращающем записи для текущей страницы, для получения ограниченного объема записей используется конструкция `LIMIT`: так, запрос с конструкцией `LIMIT 0, 10` возвращает первые 10 элементов таблицы, `LIMIT 10, 10` возвращает следующие 10 элементов, `LIMIT 20, 10` — следующие и т. д.

```
SELECT COUNT(*) FROM tbl LIMIT 0, 10
```

Для демонстрации возможностей класса `pager_mysql` создадим таблицу `position`, предназначенную для хранения заголовков разделов сайта (листинг 4.37).

Листинг 4.37. SQL-дамп таблицы `position`

```
CREATE TABLE position (
  id_position INT(11) NOT NULL AUTO_INCREMENT,
  name TEXT NOT NULL,
```

```
  PRIMARY KEY (id_position)
);
INSERT INTO position VALUES (1, 'C++');
INSERT INTO position VALUES (2, 'Pascal');
INSERT INTO position VALUES (3, 'Perl');
INSERT INTO position VALUES (4, 'PHP');
INSERT INTO position VALUES (5, 'C#');
INSERT INTO position VALUES (6, 'Visual Basic');
INSERT INTO position VALUES (7, 'BASH');
INSERT INTO position VALUES (8, 'Python');
INSERT INTO position VALUES (9, 'SQL');
INSERT INTO position VALUES (10, 'Fortran');
INSERT INTO position VALUES (11, 'JavaScript');
INSERT INTO position VALUES (12, 'HTML');
INSERT INTO position VALUES (13, 'UML');
INSERT INTO position VALUES (14, 'Java');
```

Таблица `position` состоит из двух полей:

- `id_position` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — название раздела.

Перед началом работы класса `pager_mysql` необходимо установить соединение с базой данных под управлением СУБД MySQL. Так как задача установки соединения с базой данных возникает достаточно часто, код, осуществляющий соединение, выделяют в отдельный файл (листинг 4.38).

Листинг 4.38. Установка соединения с базой данных под управлением СУБД MySQL (файл config.php)

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "oop";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
```

```
  if (!$dbcnx) {
   exit( "<P>В настоящий момент сервер базы данных не доступен,
             поэтому корректное отображение страницы невозможно.</P>" );
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) ) {
    exit( "<P>В настоящий момент база данных не доступна,
              поэтому корректное отображение страницы невозможно.</P>" );
  }

  // Устанавливаем кодировку соединения; следует выбрать кодировку,
  // в которой данные будут отправляться серверу MySQL
  @mysql_query("SET NAMES 'cp1251'");
?>
```

Теперь, когда весь вспомогательный код готов, выведем содержимое таблицы `position`, отсортировав ее по полю `name` (листинг 4.39).

Листинг 4.39. Постраничный вывод содержимого таблицы `position`

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Подключаем класс постраничной навигации
  require_once("class.pager_mysql.php");

  // Объявляем объект постраничной навигации
  $obj = new pager_mysql("position",
                         "",
                         "ORDER BY name");

  // Выводим содержимое текущей страницы
  $arr = $obj->get_page();
  for($i = 0; $i < count($arr); $i++)
  {
    echo "<a href=position.php?id={$arr[$i][id_postion]}>".
         "{$arr[$i][name]}</a><br>";
  }

  // Выводим ссылки на другие страницы
  echo $obj;
?>
```

Пожалуй, основным неудобством при использовании класса `pager_mysql` является раздельное формирование SQL-запроса. Большинство программистов будут вынуждены затратить значительное время для того, чтобы выяснить, что запись

```
$obj = new pager_mysql("position",
                       "",
                       "ORDER BY name");
```

эквивалентна SQL-запросу

```
SELECT * FROM position ORDER BY name
```

Такова плата за возможность повторного использования кода, которая снижает гибкость всей системы. Результат работы скрипта из листинга 4.39 продемонстрирован на рис. 4.10.

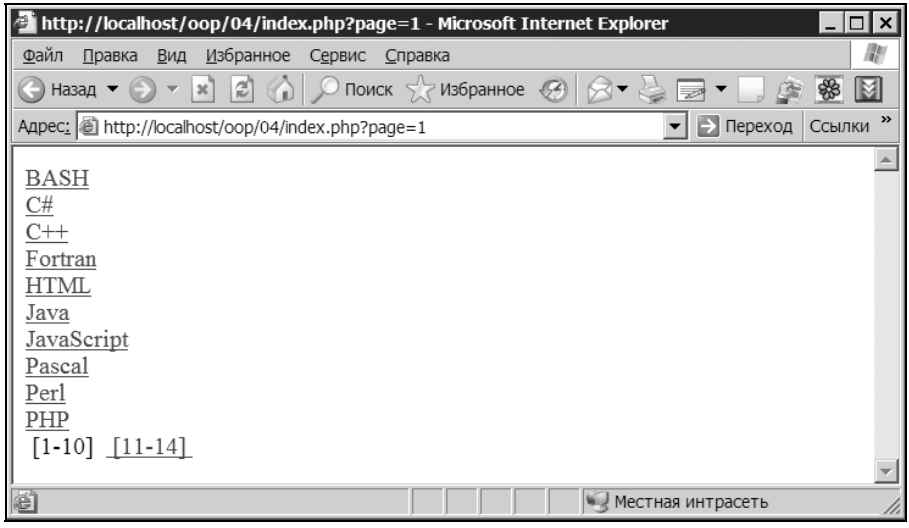


Рис. 4.10. Постраничный вывод информации из таблицы `position` базы данных под управлением MySQL

4.12. Изменение формата постраничной навигации

Пусть требуется изменить формат вывода ссылок постраничной навигации или предоставить альтернативную форму постраничной навигации, например, вместо номеров позиции выводить номера страниц:

```
<< ... < ... 3 4 5 6 7 8 9 ... > ... >>
```

Символ << является ссылкой на первую страницу, символ >> — на последнюю, а ссылки < и > перемещают пользователя с текущей страницы на одну позицию влево или вправо соответственно.

Для осуществления такого способа постраничной навигации создадим в базовом классе `pager` новый метод `print_page()` (листинг 4.40).

Замечание

В листинге 4.40 с целью экономии места базовый класс `pager` приводится не полностью.

Листинг 4.40. Альтернативный способ реализации постраничной навигации

```
<?php
  class pager
  {

    ...

    // Альтернативный способ постраничной навигации
    public function print_page()
    {
      // Строка для возвращаемого результата
      $return_page = "";

      // Через GET-параметр page передается номер
      // текущей страницы
      $page = $_GET['page'];
      if(empty($page)) $page = 1;

      // Вычисляем число страниц в системе
      $number = (int)($this->get_total()/$this->get_pnumber());
      if((float)($this->get_total()/$this->get_pnumber()) - $number != 0)
      {
        $number++;
      }

      // Ссылка на первую страницу
      $return_page .= "<a href='$_SERVER[PHP_SELF]".
                      "?page=1{$this->get_parameters()}'>".
                      "&lt;&lt;</a> ... ";
      // Выводим ссылку "Назад", если это не первая страница
      if($page != 1) $return_page .= " <a href='$_SERVER[PHP_SELF]".
                      "?page=".($page - 1)."{$this->get_parameters()}'>".
                      "&lt;</a> ... ";
```

```
// Выводим предыдущие элементы
if($page > $this->get_page_link() + 1)
{
  for($i = $page – $this->get_page_link(); $i < $page; $i++)
  {
    $return_page .= "<a href='$_SERVER[PHP_SELF]?page=$i'>$i</a> ";
  }
}
else
{
  for($i = 1; $i < $page; $i++)
  {
    $return_page .= "<a href='$_SERVER[PHP_SELF]?page=$i'>$i</a> ";
  }
}
// Выводим текущий элемент
$return_page .= "$i ";
// Выводим следующие элементы
if($page + $this->get_page_link() < $number)
{
  for($i = $page + 1; $i <= $page + $this->get_page_link(); $i++)
  {
    $return_page .= "<a href='$_SERVER[PHP_SELF]?page=$i'>$i</a> ";
  }
}
else
{
  for($i = $page + 1; $i <= $number; $i++)
  {
    $return_page .= "<a href='$_SERVER[PHP_SELF]?page=$i'>$i</a> ";
  }
}

// Выводим ссылку "вперед", если это не последняя страница
if($page != $number) $return_page .= " ... <a href='".
                     "$_SERVER[PHP_SELF]?page=".
                     ($page + 1)."{$this->get_parameters()}'>".
                     "&gt;</a>";
// Ссылка на последнюю страницу
$return_page .= " ... <a href='$_SERVER[PHP_SELF]".
                "?page=$number{$this->get_parameters()}'>".
                "&gt;&gt;</a>";
```

```
      return $return_page;
    }
  }
?>
```

Важно отметить, что новый метод автоматически начинает работать во всех производных классах `pager_file`, `pager_file_search`, `pager_dir`, `pager_mysql`, и для объектов каждого класса автоматически выбираются правильные методы `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()`.

Метод `print_page()` можно вызывать вместо содержимого метода `__toString()` или строку

```
echo $obj
```

в листингах 4.30, 4.32, 4.33, 4.35 и 4.39 заменить на строку

```
echo $obj->print_page()
```

На рис. 4.11 представлен внешний вид постраничной навигации при вызове метода `print_page()` для объекта класса `pager_dir`.

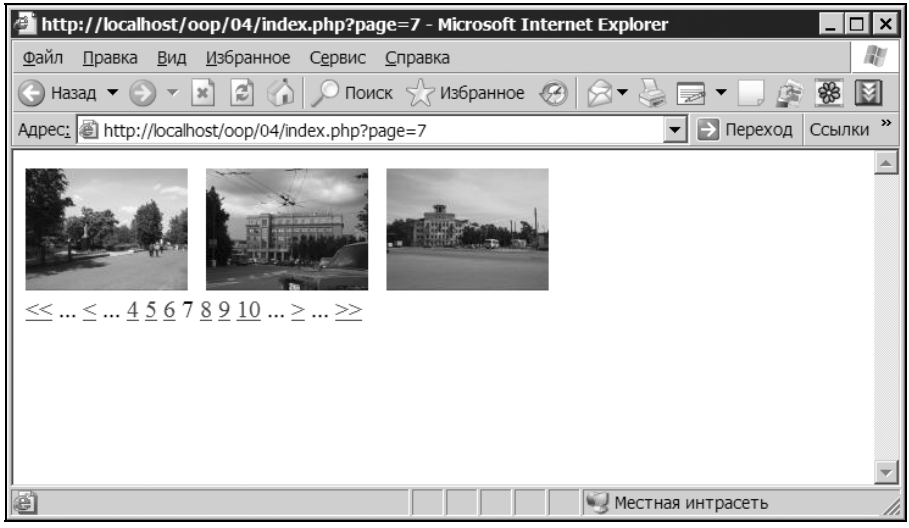


Рис. 4.11. Альтернативный вариант постраничной навигации

4.13. Абстрактные классы

В *разделах 4.7—4.12* на примере постраничной навигации было продемонстрировано построение типичной иерархии классов с применением полиморфизма. Однако объект класса `pager` не имеет смысла и не может быть исполь-

зован по назначению, и для того чтобы предотвратить объявление объекта класса `pager`, пришлось снабдить его конструктор спецификатором `protected`, запрещающим объявление объекта из внешнего кода. Однако объектно-ориентированная модель PHP предоставляет специальные инструменты для классов, появление объектов которых нежелательно: класс можно объявить абстрактным. Для этого объявление класса предваряют ключевым словом `abstract` (листинг 4.41).

Листинг 4.41. Объявление абстрактного класса

```
<?php
  abstract class pager
  {
    ...
  }
?>
```

Теперь попытка объявить экземпляр класса `pager` (листинг 4.42) будет приводить к возникновению ошибки: **Fatal error: Cannot instantiate abstract class pager (Невозможно объявить объект абстрактного класса)**.

Листинг 4.42. Попытка объявления объекта абстрактного класса

```
<?php
  require_once("class.pager.php");

  // $obj = new pager(); // Ошибка
?>
```

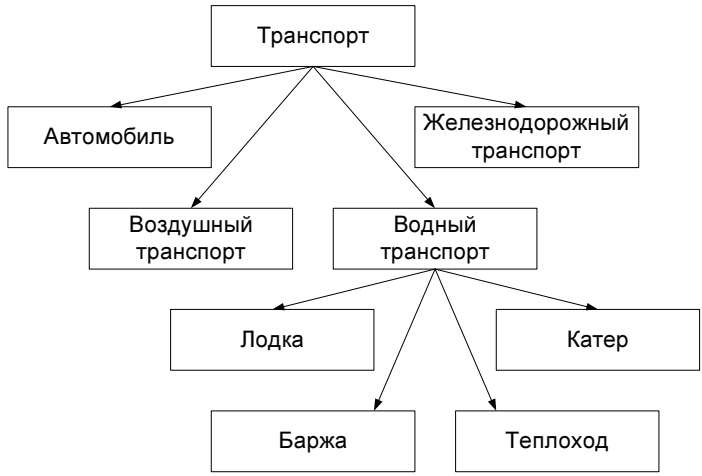


Рис. 4.12. Схема транспортной сети

Абстрактными следует объявлять классы, которые не существуют в реальности и объекты которых не понадобятся. Например, возвращаясь к обсуждаемой ранее транспортной сети, очевидно, что объекты "транспортное средство", "автомобиль", "воздушный транспорт", "водный транспорт" и "железнодорожный транспорт" не понадобятся, и их можно сделать абстрактными. В то же время конкретные классы ("лодка", "баржа", "теплоход", "катер") могут понадобиться для оценки грузо- и пассажиропотока, расчета налогообложения и т. п. (рис. 4.12).

4.14. Абстрактные методы

Если во время работы над сайтом появляется новый источник данных, к которому нужно будет применить постраничную навигацию, достаточно унаследовать от базового класса `pager` новый производный класс.

Однако продемонстрированный подход обладает некоторым изъяном: программист может забыть реализовать один из методов, который используется базовым классом. Напомним, что в базовом классе `pager` находятся только заглушки методов `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()`.

Замечание

Заглушками в программировании называют методы, которые не выполняют никакой работы.

Для решения этой задачи в объектно-ориентированной модели PHP предусмотрены абстрактные методы, объявление которых предваряется ключевым словом `abstract`. Для абстрактных методов задают их описание и список параметров без реализации. Каждый класс, который наследуется от базового класса, содержащего абстрактные методы, обязан их реализовать. Если хотя бы один метод не реализован — работа PHP-скрипта будет остановлена, а интерпретатор сообщит о необходимости реализации абстрактного метода в производном классе. При помощи абстрактных методов исключается возможность неумышленного нарушения полиморфизма для производных классов.

В листинге 4.43 методы `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()` класса `pager` объявляются абстрактными.

Листинг 4.43. Объявление абстрактных методов

```
<?php
  abstract class pager
```

```
  {
    abstract function get_total();
    abstract function get_pnumber();
    abstract function get_page_link();
    abstract function get_parameters();

    ...

  }
?>
```

Наличие в классе абстрактного метода требует, чтобы класс также был объявлен абстрактным. Невозможно объявить класс не абстрактным, если он содержит абстрактные методы.

Важно понимать, что требования реализации абстрактных методов относится лишь к обычным классам; абстрактный класс может их не реализовывать. В листинге 4.44 приводится пример класса `pager_abstract`, который является абстрактным производным классом, унаследованным от класса `pager`.

Листинг 4.44. Абстрактный класс может не реализовывать абстрактные методы

```
<?php
  // Подключаем базовый класс
  require_once("class.pager.php");

  abstract class pager_abstract extends pager
  {
    // Имя директории
    protected $dirname;
    // Количество позиций на странице
    protected $pnumber;
    // Количество ссылок слева и справа
    // от текущей страницы
    protected $page_link;
    // Параметры
    protected $parameters;
    // Конструктор
    public function __construct($dirname,
                                $pnumber = 10,
                                $page_link = 3,
                                $parameters = "")
```

```
    {
      // Удаляем последний символ /, если он имеется
      $this->dirname    = trim($dirname, "/");
      $this->pnumber    = $pnumber;
      $this->page_link  = $page_link;
      $this->parameters = $parameters;
    }
    public function get_pnumber()
    {
      // Количество позиций на странице
      return $this->pnumber;
    }
    public function get_page_link()
    {
      // Количество ссылок слева и справа
      // от текущей страницы
      return $this->page_link;
    }
    public function get_parameters()
    {
      // Дополнительные параметры, которые
      // необходимо передать по ссылке
      return $this->parameters;
    }
  }
?>
```

Если класс, унаследованный от класса `pager_abstract`, не является абстрактным, он должен реализовать все абстрактные методы всех абстрактных классов, которые предшествуют ему в иерархии классов. Например, абстрактный класс `pager_abstract` реализует три абстрактных метода класса `pager` — `get_pnumber()`, `get_page_link()` и `get_parameters()`, поэтому производным классам достаточно реализовать лишь метод `get_total()` (листинг 4.45).

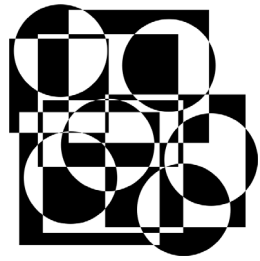
Листинг 4.45. Использование класса `pager_abstract`

```
<?php
  // Подключаем базовый класс
  require_once("class.pager_abstract.php");

  class pager_dir extends pager_abstract
  {
```

```
// Конструктор
public function __construct($dirname,
                            $pnumber = 10,
                            $page_link = 3,
                            $parameters = "")
{
  // Удаляем последний символ /, если он имеется
  $this->dirname    = trim($dirname, "/");
  $this->pnumber    = $pnumber;
  $this->page_link  = $page_link;
  $this->parameters = $parameters;
}
public function get_total()
{
  $countline = 0;
  // Открываем директорию
  if(($dir = opendir($this->dirname)) !== false)
  {
    while(($file = readdir($dir)) !== false)
    {
      // Если текущая позиция является файлом,
      // учитываем ее
      if(is_file($this->dirname."/".$file)) $countline++;
    }
    // Закрываем директорию
    closedir($dir);
  }
  return $countline;
}
// Возвращает массив строк файла
// по номеру страницы $index
public function get_page()
{
  // Текущая страница
  $page = $_GET['page'];
  if(empty($page)) $page = 1;
  // Количество записей в файле
  $total = $this->get_total();
  // Вычисляем количество страниц в системе
  $number = (int)($total/$this->get_pnumber());
  if((float)($total/$this->get_pnumber()) - $number != 0) $number++;
  // Проверяем, попадает ли запрашиваемый номер
  // страницы в интервал от 1 до get_total()
  if($page <= 0 || $page > $number) return 0;
```

```
      // Извлекаем позиции текущей страницы
      $arr = array();
      // Номер, начиная с которого следует
      // выбирать строки файла
      $first = ($page — 1)*$this->get_pnumber();
      // Открываем директорию
      if(($dir = opendir($this->dirname)) === false) return 0;
      $i = -1;
      while(($file = readdir($dir)) !== false)
      {
        // Если текущая позиция является файлом
        if(is_file($this->dirname."/".$file))
        {
          // Увеличиваем счетчик
          $i++;
          // Пока не достигнут номер $first,
          // досрочно заканчиваем итерацию
          if($i < $first) continue;
          // Если достигнут конец выборки,
          // досрочно покидаем цикл
          if($i > $first + $this->get_pnumber() — 1) break;
          // Помещаем пути к файлам в массив,
          // который будет возвращен методом
          $arr[] = $this->dirname."/".$file;
        }
      }
      // Закрываем директорию
      closedir($dir);

      return $arr;
    }
  }
?>
```

ГЛАВА 5



Интерфейсы

Наследование и полиморфизм являются центральными идеями объектно-ориентированного программирования, позволяя наиболее эффективно организовать код для иерархических систем. Обычно в реальной практике используются лишь конечные производные классы; базовые классы иерархии необходимы лишь для сокрытия реализации и формирования общего интерфейса производных классов.

В результате базовые классы зачастую становятся абстрактными, вернее, классами, которые определяют поведение производных классов, объектов которых, однако, нельзя объявить, поскольку они не смогли бы функционировать. Абстрактные классы зачастую содержат абстрактные методы, не несущие функциональности, реализовать которую должны их потомки.

Для того чтобы не использовать классы, которые ничего кроме абстрактных методов не содержат, в PHP введена специальная конструкция — интерфейс, полностью состоящая из абстрактных методов. Это позволяет внести ясность в процесс разработки: классы задают поведение объектов, а интерфейсы — поведение группы классов. Класс, реализующий интерфейс, должен перегрузить методы интерфейса. Два класса, реализующих одинаковые интерфейсы, имеют одинаковый набор определяемых ими методов. Таким образом, назначение интерфейса — это реализация полиморфизма для двух классов, не имеющих общего базового класса.

5.1. Создание интерфейса

Для создания интерфейса используется ключевое слово `interface`. В листинге 5.1 демонстрируется пример интерфейса `pager` для реализации набора классов постраничной навигации, рассматривавшихся в *главе 4*. Напомним,

что классы, реализующие постраничную навигацию, должны содержать следующие методы:

- `get_total()` — возвращает количество элементов в списке, подвергающемся разбиению на несколько страниц;
- `get_pnumber()` — возвращает количество элементов на одной странице;
- `get_page_link()` — возвращает количество ссылок слева и справа от текущей страницы;
- `get_parameters()` — возвращает дополнительные GET-параметры, передаваемые по ссылкам.

Замечание

Интерфейс может сдержать только открытые методы; при попытке добавить в интерфейс закрытый или защищенный метод возникает сообщение об ошибке.

Листинг 5.1. Интерфейс `pager`

```
<?php
  interface pager
  {
    public function get_total();
    public function get_pnumber();
    public function get_page_link();
    public function get_parameters();
  }
?>
```

Порядок создания класса, реализующего интерфейс, очень похож на наследование производного класса от базового, только вместо ключевого слова `extentds` используется ключевое слово `implements`. В листинге 5.2 объявляется два класса: `pager_sample` и `pager_example`, которые реализуют интерфейс `pager`.

Листинг 5.2. Реализация интерфейса `pager` классами `pager_sample` и `pager_example`

```
<?php
   class pager_sample implements pager
   {
      public function get_total()
      {
          // ...
      }
```

```
        public function get_pnumber()
        {
            // ...
        }
        public function get_page_link()
        {
            // ...
        }
        public function get_parameters()
        {
            // ...
        }
    }
    class pager_example implements pager
    {
        public function get_total()
        {
            // ...
        }
        public function get_pnumber()
        {
            // ...
        }
        public function get_page_link()
        {
            // ...
        }
        public function get_parameters()
        {
            // ...
        }
    }
?>
```

Оба класса `pager_sample` и `pager_example` должны реализовать методы, перечисленные в интерфейсе, если хотя бы один метод остается нереализованным, выполнение скрипта заканчивается сообщением об ошибке: **Fatal error: Access type for interface method must be omitted (Пропущен метод интерфейса)**.

5.2. Интерфейсы и наследование классов

Использование интерфейса не исключает использование наследования. В листинге 5.3 производный класс `derived` наследуется от базового класса `base`,

при этом одновременно класс `derived` реализует интерфейс `pager`, который рассматривается в *разделе 5.1*.

Листинг 5.3. Совместное использование наследования и интерфейсов

```
<?php
  require_once("interface.pager.php");

  class base
  {
    public function base_method()
    {
      echo "Метод базового класса<br>";
    }
  }
  class derived extends base implements pager
  {
    public function get_total()
    {
        // ...
    }
    public function get_pnumber()
    {
        // ...
    }
    public function get_page_link()
    {
        // ...
    }
    public function get_parameters()
    {
        // ...
    }
  }
?>
```

Объект класса `derived` позволяет вызывать не только метод `base_method()`, но и все методы интерфейса `pager`, т. к. класс `derived` обязан их реализовать.

Одна из особенностей объектно-ориентированной модели PHP заключается в том, что каждый производный класс может иметь лишь один базовый класс. Производный класс не может наследоваться сразу от двух базовых классов. Такое ограничение имеют множество объектно-ориентированных моделей. В языках программирования, допускающих множественное наследование (таких как C++), разработчики часто сознательно отказываются от возможно-

сти множественного наследования, поскольку это усложняет и запутывает код.

Тем не менее в большом приложении могут возникать ситуации, когда объекты разных иерархических схем должны реализовывать одинаковые методы. Это позволяет распространить явление полиморфизма на несколько не зависящих друг от друга иерархий наследуемых классов.

Рассмотрим две иерархии: товаров, предоставляемых компанией, и сотрудников, занятых в ней (рис. 5.1).

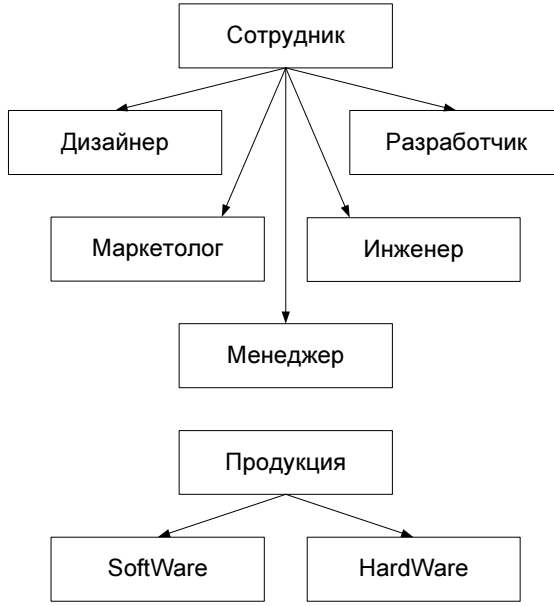


Рис. 5.1. Независимые иерархии наследуемых классов

Пусть каждый из конечных классов наследуемой иерархии должен реализовывать интерфейс `pager` для обеспечения объектов классов возможностями постраничной навигации. Было бы неразумно наследовать две эти независимые иерархии от единого абстрактного класса, который бы требовал реализации методов, необходимых для постраничной навигации. Именно в таких случаях применяются интерфейсы, которые могут требовать реализации методов в классе, не затрагивая механизм наследования.

5.3. Реализация нескольких интерфейсов

В отличие от наследования, реализация интерфейсов не ограничивается единственным интерфейсом. Класс может реализовывать любое количество

интерфейсов; для этого достаточно перечислить их через запятую после ключевого слова `implements` (листинг 5.4).

Замечание

Именно возможность реализации нескольких интерфейсов позволяет скомпенсировать отсутствие множественного наследования в объектно-ориентированной модели языка.

Листинг 5.4. Реализация нескольких интерфейсов

```
<?php
  require_once("interface.pager.php");
  require_once("interface.print_list.php");
  require_once("interface.document.php");

  class cls implements pager, print_list, document
  {
    // ...
  }
?>
```

Как видно из листинга 5.4, класс `cls` реализует сразу три интерфейса: `pager`, `print_list` и `document`.

5.4. Проверка существования интерфейса

Прежде чем обращаться к интерфейсу, часто требуется проверить его существование. Для этого можно воспользоваться функцией `get_declared_interfaces()`, которая возвращает массив всех объявленных интерфейсов (листинг 5.5).

Листинг 5.5. Использование функции `get_declared_interfaces()`

```
<?php
  interface pager
  {
    public function get_total();
    public function get_pnumber();
    public function get_page_link();
    public function get_parameters();
  }
```

```
  echo "<pre>";
  print_r(get_declared_interfaces());
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 5.5 могут быть следующие строки:

```
Array
(
    [0] => Traversable
    [1] => IteratorAggregate
    [2] => Iterator
    [3] => ArrayAccess
    [4] => Serializable
    [5] => Reflector
    [6] => RecursiveIterator
    [7] => OuterIterator
    [8] => SeekableIterator
    [9] => Countable
    [10] => SplObserver
    [11] => SplSubject
    [12] => pager
)
```

Первые 12 интерфейсов являются предопределенными; последний интерфейс `pager` декларируется в элементе массива с индексом 12.

Работа с массивом, возвращаемым функцией `get_declared_interfaces()`, не всегда удобна, т. к. зачастую требуется узнать, доступен или нет какой-либо конкретный интерфейс. Для этого используют функцию `interface_exists()`, которая имеет следующий синтаксис:

```
bool interface_exists ($interface_name [, $autoload])
```

Функция возвращает `true`, если интерфейс с именем `$interface_name` существует, и `false` — в противном случае. Если необязательный параметр `$autoload` равен `true`, функция пытается загрузить интерфейс при помощи функции `__autoload()` (см. *раздел 3.6*), если параметр равен `false`, такой попытки не производится.

В листинге 5.6 приводится пример использования функции `interface_exists()`.

Замечание

Функция `interface_exists()` введена в PHP, начиная с версии 5.0.2.

Листинг 5.6. Использование функции `interface_exists()`

```
<?php
  require_once("interface.pager.php");

  if(interface_exists("pager"))
  {
    class cls implements pager
    {
      // ...
    }
  }
?>
```

Класс `cls` из листинга 5.6 не будет объявлен, если скрипт не найдет определение интерфейса `pager`.

5.5. Наследование интерфейсов

Интерфейсы, так же как и классы, могут наследовать друг другу при помощи ключевого слова `extends`. В листинге 5.7 производный интерфейс `extended_pager` наследует методы базового интерфейса `pager`.

Листинг 5.7. Наследование интерфейсов

```
<?php
  interface pager
  {
    public function get_total();
    public function get_pnumber();
    public function get_page_link();
    public function get_parameters();
  }

  interface extended_pager extends pager
  {
    public function get_page();
  }
?>
```

Отношение интерфейсов `pager` и `extended_pager` демонстрируется на рис. 5.2. Как и для классов, не допускается множественное наследование интерфейсов: может быть лишь один базовый интерфейс, зато допускается множество производных.

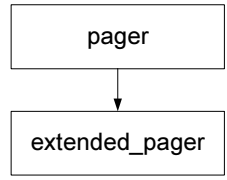


Рис. 5.2. Наследование производным интерфейсом `extended_pager` базового интерфейса `pager`

Класс, реализующий интерфейс `extended_pager`, должен перегружать не только метод `get_page()` производного класса `pager`, но и методы базового интерфейса `pager`: `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()` (листинг 5.8).

Листинг 5.8. Реализация интерфейса `extended_pager`

```
<?php
  require_once("interface.pagers.php");

  class pager_class implements extended_pager
  {
     // ...
    public function get_total()
    {
      // ...
    }
    public function get_pnumber()
    {
      // ...
    }
    public function get_page_link()
    {
      // ...
    }
    public function get_parameters()
    {
      // ...
    }
    public function get_page()
    {
      // ...
    }
  }
?>
```

Отсутствие хотя бы одного метода из интерфейсов `pager` и `extended_pager` в классе `pager_class` приведет к ошибке.

5.6. Реализует ли объект интерфейс?

Выяснить, реализует ли объект интерфейс, можно при помощи оператора `instanceof`. Данный оператор применяется для определения принадлежности объекта классу (см. *раздел 3.8*). Для демонстрации возможностей оператора создадим два интерфейса: `first` и `second`, которые будут требовать от классов реализации одноименных методов (листинг 5.9).

Листинг 5.9. Интерфейсы `first` и `second`

```
<?php
  interface first
  {
    public function first();
  }
  interface second
  {
    public function second();
  }
?>
```

В листинге 5.10 создаются три класса: `fst` — для реализации интерфейса `first`, `snd` — интерфейса `second` и `cmn`, реализующий оба интерфейса.

Листинг 5.10. Классы `fst`, `snd` и `cmn`

```
<?php
  require_once("interface.first.php");
  require_once("interface.second.php");

  class fst implements first
  {
    public function first()
    {
      echo "Метод fst:first()<br>";
    }
  }
  class snd implements second
  {
    public function second()
```

```
    {
      echo "Метод snd:second()<br>";
    }
  }
  class cmn implements first, second
  {
    public function first()
    {
      echo "Метод cmn:first()<br>";
    }
    public function second()
    {
      echo "Метод cmn:second()<br>";
    }
  }
?>
```

В листинге 5.11 создается массив случайных объектов классов `fst`, `snd` и `cmn`. Далее при помощи оператора `instanceof` определяется, какие элементы массива реализуют интерфейс `second`, а какие нет.

Листинг 5.11. Создание массива случайных объектов

```
<?php
  // Подключаем классы
  require_once("interface.first.php");

 // Формируем массив из случайных объектов
  for($i = 0; $i < 10; $i++)
  {
    switch(rand(1,3))
    {
      case 1:
        $arr[] = new fst();
        break;
      case 2:
        $arr[] = new snd();
        break;
      case 3:
        $arr[] = new cmn();
        break;
    }
  }
```

```
  // Определяем принадлежность элементов
  // массива к классам
  foreach($arr as $obj)
  {
    if($obj instanceof second) echo "Объект класса ".get_class($obj)."
                                    реализует интерфейс second<br>";
    else echo "Объект класса ".get_class($obj)." не реализует интерфейс
              second<br>";
  }
?>
```

Следует обратить внимание, что имя класса в операторе `instanceof` не заключается в кавычки. Результатом работы скрипта из листинга 5.11 будут следующие строки:

```
Объект класса snd реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса cmn реализует интерфейс second
Объект класса fst не реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса cmn реализует интерфейс second
Объект класса fst не реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса fst не реализует интерфейс second
```

Оператор `instanceof` возвращает значение `true` не только по отношению к интерфейсу, который явно реализовывался при создании класса объекта, но и для всех базовых интерфейсов. В листинге 5.12 приводится пример интерфейса `third`, который является производным интерфейсом базового интерфейса `first`. Классы, реализующие интерфейс `third`, обязательно должны реализовывать два метода: `first()` и `third()`.

Листинг 5.12. Оператор `instanceof` и производные интерфейсы

```
<?php
  require_once("interface.first.php");

  // Производный интерфейс
  interface third extends first
  {
    public function third();
  }
```

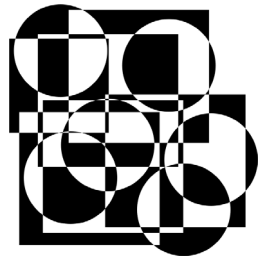
```
  // Класс thd обязательно должен реализовывать
  // методы first() и third()
  class thd implements third
  {
    public function first()
    {
      echo "Метод thd:first()<br>";
    }
    public function third()
    {
      echo "Метод thd:third()<br>";
    }
  }

  $obj = new thd();

  if($obj instanceof first) echo "Объект реализует интерфейс first<br>";
  else echo "Объект не реализует интерфейс first<br>";

  if($obj instanceof third) echo "Объект реализует интерфейс third<br>";
  else echo "Объект не реализует интерфейс third<br>";
?>
```

Результатом работы скрипта из листинга 5.12 будут следующие строки:

```
Объект реализует интерфейс first
Объект реализует интерфейс third
```

Таким образом, оператор `instanceof` определяет принадлежность объекта не только производному интерфейсу, но и базовому.

ГЛАВА 6



Статические и константные элементы класса

В процедурном стиле программирования часто используют константы — именованные постоянные значения, которые невозможно изменить (даже случайным образом). Объектно-ориентированный стиль программирования также позволяет использовать постоянные значения. Помимо констант объекты могут иметь статические элементы — члены и методы, общие для всех экземпляров класса.

6.1. Статические члены класса

Для того чтобы воспользоваться членами и методами класса, не обязательно объявлять его объект: объявление членов класса статическими с помощью ключевого слова `static` делает их доступными в любой момент без объявления объекта класса. Обращение к компоненту класса в этом случае производится при помощи оператора разрешения области видимости `::`. Одной из характерных особенностей статических членов классов является возможность их инициализации непосредственно при объявлении (листинг 6.1).

Листинг 6.1. Объявление статического члена класса

```
<?php
  class cls
  {
     public static $staticvar = 100;
  }

  echo cls::$staticvar; // 100
?>
```

К статическим членам класса нельзя обращаться через префикс `$this->` и они не отображаются в списках членов объектов при выводе дампа. В листинге 6.2 приводится модифицированный вариант класса `cls`, метод `set_staticvar()` которого осуществляет попытку изменить значение статического члена класса `$staticvar` при помощи префикса `$this->`.

Листинг 6.2. Попытка использования префикса `$this->` для доступа к статическому члену класса

```
<?php
  class cls
  {
     public static $staticvar = 100;
     public function set_staticvar($val)
     {
       $this->staticvar = $val;
     }
  }

  $obj = new cls();

  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  $obj->set_staticvar(20);

  echo cls::$staticvar; // 100

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

В результате работы скрипта из листинга 6.2 на экран будут выведены следующие строки:

```
cls Object
(
)
100
cls Object
(
    [staticvar] => 20
)
```

Таким образом, попытка обратиться к статическому члену класса через префикс `$this->` приводит к созданию нового члена класса, изменение значения которого не затрагивает значение статического члена.

Такое поведение связано с тем, что, в отличие от других членов класса, статические члены являются общими для всех объектов данного класса (рис. 6.1).

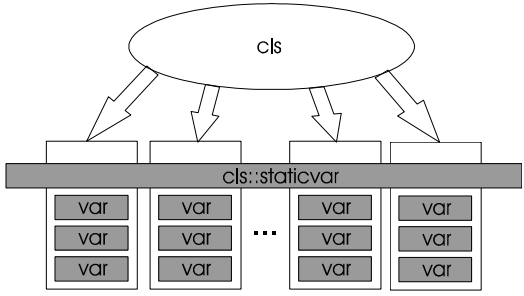


Рис. 6.1. Статический член класса является общим для всех объектов

Это означает, что изменение значения статической переменной одного объекта отражается на значении данной переменной всех остальных объектов. Эту особенность статических членов часто используют для создания счетчиков объектов или ресурсов, которые они резервируют. В листинге 6.3 представлен класс `counter`, который имеет в своем составе статический член `$count`. Изначально член `$count` имеет нулевое значение, однако в конструкторе класса оно увеличивается на единицу, а в деструкторе — уменьшается. Таким образом, в каждый момент времени можно определить количество созданных объектов.

Листинг 6.3. Счетчик объектов

```
<?php
  class counter
  {
    public static $count = 0;
    public function __construct()
    {
      counter::$count++;
    }
    public function __destruct()
    {
      counter::$count--;
    }
  }
?>
```

Использование класса `counter` демонстрируется в листинге 6.4.

Листинг 6.4. Счетчик объектов

```
<?php
  require_once("class.counter.php");

  $obj = new counter();
  echo counter::$count."<br>"; // 1

  for($i = 0; $i < 3; $i++)
  {
    $arr[] = new counter();
    echo counter::$count."<br>"; // 2, 3, 4
  }

  // Уничтожаем массив объектов
  unset($arr);

  echo counter::$count."<br>"; // 1
?>
```

6.2. Эмуляция транзакций при помощи статических членов

При совместном доступе к ресурсу нескольких клиентов неминуемо возникают конфликтные ситуации. Например, при покупке какого-либо товара в электронном магазине количество этого товара на складе уменьшается на единицу, а со счета покупателя снимаются денежные средства. Однако один и тот же товар могут одновременно приобрести несколько покупателей, при этом вероятно возникновение ситуации, когда денежные средства будут списаны со счетов всех клиентов, а подсчет оставшихся на складе товарных позиций даст отрицательный результат. Возможен также вариант, что средства окажутся списаны со счета клиента, но не зачислены на счет магазина, из-за того что файл был занят.

Для предотвращения таких конфликтов используются *транзакции* — объединение нескольких операций в единое целое. При этом для успешного завершения транзакции необходимо выполнить все операции до конца; в противном случае, если одна из операций закончится неудачей, система будет возвращена в исходное состояние.

Замечание

Транзакции хорошо знакомы разработчикам, использующим в работе базы данных. При транзакции операции зачастую проводятся в оперативной памяти — если все операции в рамках транзакции завершаются успешно, изменения записываются на жесткий диск, если где-то происходит сбой или конфликт с параллельным запросом — изменения не записываются на жесткий диск, а клиенту возвращается сообщение об ошибке.

При реализации транзакционных схем важно иметь надежный критерий, извещающий о том, что ресурс занят в настоящий момент, и здесь незаменимыми оказываются статические члены класса.

Рассмотрим реализацию транзакций на примере описанной выше схемы покупки товара в электронном магазине. Для того чтобы осуществить покупку, требуется выполнить следующие действия:

- запросить наличие выбранного товара на складе;
- запросить денежные средства клиента и оценить, хватит ли их для оплаты выбранного товара;
- уменьшить на единицу количество запрошенного товара на складе;
- увеличить количество заказанного клиентом товара;
- списать денежные средства со счета клиента;
- зачислить денежные средства на счет магазина.

Пока выполняется эта цепочка действий, другие запросы к базе данных выполняться не будут, и все остальные заказчики будут ожидать своей очереди. Это позволит предотвратить превышение имеющегося на складе товарного запаса количеством заказанных единиц товара, а также не позволит клиенту приобрести товаров на большую сумму, чем у него имеется в данный момент.

Создадим простейшую базу данных, состоящую из таблицы `warehouse`, моделирующей склад и товарные запасы на нем, таблицы `users` для счетов пользователей и таблицы `bill` для зачисления денежных средств на счет электронного магазина (листинг 6.5).

Листинг 6.5. База данных электронного магазина

```
CREATE TABLE bill (
  total DECIMAL(8,2) NOT NULL
);
INSERT INTO bill VALUES (2100.00);

CREATE TABLE users (
  id_user INT(11) NOT NULL AUTO_INCREMENT,
```

```
  fio TINYTEXT NOT NULL,
  money DECIMAL(8,2) NOT NULL,
  total INT(11) NOT NULL,
  PRIMARY KEY (id_user)
);
INSERT INTO users VALUES (1, 'Корнеев Е.Г.', 500.00, 0);
INSERT INTO users VALUES (2, 'Марычев А.А.', 0.00, 0);
INSERT INTO users VALUES (3, 'Андреев С.В.', 200.00, 1);

CREATE TABLE warehouse (
  id_position INT(11) NOT NULL AUTO_INCREMENT,
  total INT(11) NOT NULL,
  price DECIMAL(8,2) NOT NULL,
  PRIMARY KEY (id_position)
);
INSERT INTO warehouse VALUES (1, 6, 100.00);
```

Таблица `bill`, в которой хранится счет электронного магазина, содержит лишь одно поле `total`. Таблица пользователей `users` содержит четыре поля:

- `id_user` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`; данное поле необходимо, чтобы отличать посетителей друг от друга;
- `fio` — фамилия, имя и отчество покупателя;
- `money` — денежные средства на счете пользователя, которые он может использовать для оплаты товара;
- `total` — количество приобретенных единиц товара.

В настоящий момент в системе зарегистрировано три пользователя. Для упрощения модели предположим, что магазин распространяет только один вид товара. Таблица `warehouse`, моделирующая склад, состоит из трех следующих полей:

- `id_position` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`; данное поле необходимо, чтобы отличать товары друг от друга;
- `total` — количество имеющегося на складе товара;
- `price` — цена за штуку.

Базы данных обычно обладают собственным механизмом транзакций, однако его можно применить не ко всем типам таблиц. Например, в СУБД MySQL самые быстрые таблицы типа MyISAM не снабжены транзакционным механизмом; воспользоваться им можно лишь в отношении гораздо более мед-

ленных таблиц, таких как InnoDB. Поэтому иногда может потребоваться создать подобие транзакций средствами PHP.

В листинге 6.6 приводится пример класса `shop`, содержащего единственный метод `buy()`: он принимает первичный ключ посетителя магазина и количество приобретаемых товаров. Если статический член класса `$lock` имеет значение `true`, то объект устанавливает его в `false` и начинает процедуру покупки. Если член класса `$lock` при проверке имеет значение `false`, то вызывается функция `sleep()`, приостанавливающая работу объекта на одну секунду, после чего попытка получить доступ к базе данных возобновляется. Если после 30 попыток доступ к базе получить не удалось, метод возвращает значение `false`, сигнализирующее о сбое в системе оформления покупок или большом количестве обращений к сервису.

Замечание

Если имеется возможность использовать полноценные транзакции, лучше воспользоваться ими, т. к. при использовании любой эмуляции остается шанс, что два потока одновременно получат доступ к базе данных.

Листинг 6.6. Эмуляция транзакций

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  class shop
  {
    public static $lock = false;

    // Вывод диагностического сообщения
    private function error_print($str)
    {
      // Освобождаем таблицу
      shop::$lock = false;
      // Выводим диагностическое сообщение
      echo $str;
    }

    public function buy($id_user, $buy_count)
    {
      // Преобразуем параметр в целое число
      $id_user = intval($id_user);
```

```
// Проверяем, не занята ли база данных другим
// покупателем
$count = 0;
while(shop::$lock)
{
  // Приостанавливаем работу программы
  // на 1 секунду
  sleep(1);
  $count++;
  // После 30 попыток покидаем метод
  if($count > 30) return false;
}

// База данных доступна — занимаем ее
shop::$lock = true;

// Осуществляем операции транзакции

// 1. Запрашиваем количество товара на складе
$query = "SELECT * FROM warehouse
          WHERE id_position = 1";
$tot = mysql_query($query);
if(!$tot)
{
  // Выводим диагностическое сообщение
  $this->error_print("Ошибка доступа к складской таблице");
  // Покидаем метод
  return false;
}
list($id_position,$total,$price) = mysql_fetch_array($tot);
// Если товара на складе ноль или меньше,
// прекращаем осуществление сделки
if($total <= 0)
{
  $this->error_print("Закончились запасы на складе");
  return false;
}

// 2. Запросить денежные средства клиента и оценить,
// хватит ли их для оплаты товара
$query = "SELECT money FROM users WHERE id_user = $id_user";
$mon = mysql_query($query);
if(!$mon)
```

```
      {
        $this->error_print("Ошибка доступа к счету пользователя");
        return false;
      }
      if(mysql_num_rows($mon)) $money = mysql_result($mon,0);
      // Если сумма, требуемая для покупки $buy_count
      // товаров больше, чем денег на счете
      // пользователя $money, — покидаем метод
      if($buy_count*$price > $money)
      {
        $this->error_print("Не достаточно средств для покупки");
        return false;
      }

      // 3. Уменьшаем количество товаров
      // на складе на $buy_count
      $query = "UPDATE warehouse
                SET total = total — $buy_count
                WHERE id_position = 1";
      mysql_query($query);

      // 4-5. Увеличиваем количество заказанных клиентом
      // товаров на $buy_count и списываем денежные
      // средства со счета заказчика
      $query = "UPDATE users
                SET total = total + $buy_count,
                    money = money — ".($buy_count*$price)."
                WHERE id_user = $id_user";
      mysql_query($query);

      // 6. Зачисляем денежные средства на счет магазина
      $query = "UPDATE bill
                SET total = total + ".($buy_count*$price);
      mysql_query($query);

      // Освобождаем таблицу
      shop::$lock = false;
      // Возвращаем значение true, сигнализирующее об
      // успешном завершении транзакции
      return true;
    }
  }
?>
```

Как видно из листинга 6.6, для отображения диагностических сообщений вводится специальный метод `error_print()`, который сбрасывает статический флаг `$lock` в `false`. Без этой операции любой сбой в методе `buy()` будет приводить к тому, что таблица будет оставаться заблокированной без возможности снятия блокировки, т. к. объект, в обязанность которого это входило, уже завершил работу, а другие объекты ожидают освобождения базы данных.

В листинге 6.7 приводится пример покупки трех товарных позиций клиентом, первичный ключ `$is_user` которого равен единице.

Замечание

Разумно было бы переместить содержимое метода `buy()` в конструктор класса `shop`, однако в этом случае для осуществления покупки приходилось бы каждый раз создавать объект. Отдельный метод позволяет использовать один и тот же объект для осуществления нескольких покупок.

Листинг 6.7. Использование объекта класса `shop`

```
<?php
  require_once("class.shop.php");

  $obj = new shop();
  // Покупка трех товарных позиций покупателем
  // с первичным ключом id_user = 1
  $obj->buy(1, 3);
?>
```

6.3. Наследование и статические члены

Статические члены, так же как и обычные, наследуются производными классами. В листинге 6.8 демонстрируется базовый класс `base` и производный класс `derived`, каждый из которых содержит статический член `$staticvar`.

Листинг 6.8. Наследование и статические члены

```
<?php
  class base
  {
    public static $staticvar = 0;
  }
  class derived extends base
  {
    public static $staticvar = 0;
```

```
    public function __construct()
    {
      parent::$staticvar = 100;
    }
  }
  $obj = new derived();

  derived::$staticvar = 20;
  echo base::$staticvar."<br>"; // 100
  echo derived::$staticvar."<br>"; // 20
?>
```

Следует отметить, что статические члены базового и производного классов не зависят друг от друга. К статическому члену производного класса можно обращаться при помощи префикса `parent`, а к статическому члену текущего класса — применяя префикс `self`.

6.4. Статические методы класса

Статическими можно объявлять не только члены, но и методы класса. Для объявления статического метода также используется ключевое слово `static`, а для обращения к методу оператор разрешения области видимости `::`. В листинге 6.9 приводится пример класса `cls`, содержащего единственный статический метод `static_method()`.

Листинг 6.9. Объявление статического метода

```
<?php
  class cls
  {
    public static function static_method()
    {
      echo "Вызов статического члена";
    }
  }

  cls::static_method(); // Вызов статического члена
?>
```

Впрочем, для того чтобы вызывать методы при помощи оператора разрешения области видимости `::` без создания объекта, не обязательно объявлять метод статическим (см. *раздел 2.8*). Вообще говоря, вызов нестатического метода через оператор `::` должен приводить к генерации предупреждения,

однако в текущей версии PHP этого не происходит. Возможно, это будет исправлено в следующих версиях PHP.

6.5. Константы класса

Классы, наряду с членами, могут содержать константы, которые определяются при помощи ключевого слова `const`. В листинге 6.10 приводится пример класса `cls`, в состав которого включена константа `NAME`, содержащая имя класса.

Замечание

Следует обратить внимание, что имя константы не содержит символа $.

Листинг 6.10. Использование констант в классах

```
<?php
  class cls
  {
    const NAME = "cls";
    public function method()
    {
      // echo $this->NAME; // Ошибочное обращение
      echo self::NAME;
      echo "<br>";
      echo cls::NAME;
      echo "<br>";
    }
  }

  echo cls::NAME;
?>
```

Замечание

Хотя строгих правил именования констант не предусматривается, для этого обычно используются символы верхнего регистра. Лучше придерживаться данной традиции, т. к. это позволяет значительно увеличить читабельность кода — большинство программистов будут ожидать, что имена констант в программе будут записаны в верхнем регистре.

К константам, как и к статическим членам классов, нельзя обращаться при помощи оператора `->`; для обращения к ним также используется оператор разрешения области видимости `::`, который предваряется либо именем класса, либо ключевыми словами `self` и `parent`.

Существование обычных пользовательских констант, которые определяются при помощи функции `define()` (см. *раздел 2.4*), может быть проверено при помощи функции `defined()`, которая возвращает `true`, если константа существует, и `false` — в противном случае (листинг 6.11).

Замечание

При проверке классовых констант следует в обязательном порядке использовать оператор разрешения области видимости `::` и имя класса.

Листинг 6.11. Проверка существования классовых констант

```
<?php
  require_once("class.cls.php");

  if(defined("cls::NAME")) echo "Константа определена<br>"; // true
  else echo "Константа не определена<br>";

  if(defined("cls::POSITION")) echo "Константа определена<br>"; // false
  else echo "Константа не определена<br>";
?>
```

6.6. Предопределенные константы

Помимо констант, которые может вводить в класс разработчик, существуют предопределенные константы, которые определяет PHP-интерпретатор (табл. 6.1).

Таблица 6.1. *Предопределенные константы PHP*

Константа	Описание
`__LINE__`	Номер текущей строки в файле с PHP-скриптом
`__FILE__`	Полный путь к файлу с PHP-скриптом
`__FUNCTION__`	Имя функции, из которой вызывается константа
`__CLASS__`	Имя текущего класса
`__METHOD__`	Имя текущего метода класса

Замечание

Константа `__METHOD__` введена в PHP, начиная с версии 5.0.0.

В листинге 6.12 приводится пример использования предопределенных констант для идентификации текущего номера строки, файла и имени метода. Основное назначение предопределенных констант заключается в точной идентификации места возникновения ошибки.

Листинг 6.12. Использование предопределенных констант

```
<?php
  function get_point()
  {
     return "Вызов функции ".__FUNCTION__.
            "<br>файла ".__FILE__.
            "<br>в строке ".__LINE__; // 6 строка
  }

  echo get_point(); // 9 строка
?>
```

Результатом работы скрипта из листинга 6.12 будут следующие строки:

```
Вызов функции get_point
файла D:\main\oop\06\index.php
в строке 6
```

Таким образом, константа `__LINE__` подставляет номер строки вызова константы, а не функции, откуда она вызывается. Точно так же действуют и все остальные константы — включение файла из листинга 6.12 в другие файлы при помощи инструкций `include` и `require` не приводит к изменению их значений.

Как видно из листинга 6.12, предопределенные константы `__LINE__`, `__FILE__` и `__FUNCTION__` не связаны непосредственно с объектно-ориентированным подходом. Однако ничто не мешает использовать их в классах (листинг 6.13).

Листинг 6.13. Использование предопределенных констант в классе

```
<?php
  class cls
  {
    public function get_point()
    {
       return "Вызов функции ".__FUNCTION__.
              "<br>файла ".__FILE__.
              "<br>в строке ".__LINE__; // 8 строка
    }
  }
```

```
  echo cls::get_point(); // 12 строка
?>
```

Результатом работы скрипта из листинга 6.13 будут следующие строки:

```
Вызов функции get_point
файла D:\main\oop\06\class.cls.php
в строке 8
```

Однако для использования в классах предусмотрены специализированные константы: `__CLASS__` и `__METHOD__` (листинг 6.14).

Листинг 6.14. Использование констант `__CLASS__` и `__METHOD__`

```
<?php
  class cls
  {
    public function get_point()
    {
       return "Вызов метода ".__METHOD__.
              "<br>класса ".__CLASS__.
              "<br>в файле ".__FILE__.
              "<br>в строке ".__LINE__; // 9 строка
    }
  }

  echo cls::get_point(); // 13 строка
?>
```

В результате работы скрипта из листинга 6.14 на экран будут выведены следующие строки:

```
Вызов метода cls::get_point
класса cls
в файле D:\main\oop\06\class.cls.php
в строке 9
```

6.7. Самоидентификация объектов

При помощи предопределенной константы `__CLASS__` можно реализовать метод самоидентификации объектов. В листинге 6.15 представлено три класса: базовый класс `base`, производный класс `derived` и не зависящий от них класс `third`. Классы `base` и `third` содержат метод `get_name()`, который возвращает

имя класса при помощи предопределенной константы __CLASS__. Классу derived метод get_name() передается по наследству.

Листинг 6.15. Реализация методов самоидентификации

```
<?php
  class base
  {
    public function get_name()
    {
      return __CLASS__."<br>";
    }
  }
  class derived extends base {}
  class third
  {
    public function get_name()
    {
      return __CLASS__."<br>";
    }
  }

  $obj = new base();
  echo $obj->get_name(); // base

  $obj = new derived();
  echo $obj->get_name(); // base

  $obj = new third();
  echo $obj->get_name(); // derived
?>
```

Как видно из листинга 6.15, константа __CLASS__ возвращает имя класса, в котором она вызывается. Метод get_name() в классе derived вызывается в контексте класса base, поэтому константа __CLASS__ возвращает имя производного класса. Для того чтобы исправить ситуацию, потребуется переопределить метод get_name() в классе derived, полностью воспроизведя его содержимое из класса base (листинг 6.16).

Листинг 6.16. Переопределение метода get_name() в классе derived

```
<?php
  class base
```

```
  {
    public function get_name()
    {
      return __CLASS__."<br>";
    }
  }
  class derived extends base
  {
    public function get_name()
    {
      return __CLASS__."<br>";
    }
  }

  $obj = new base();
  echo $obj->get_name(); // base

  $obj = new derived();
  echo $obj->get_name(); // derived
?>
```

6.8. *Final*-методы класса

Абстрактные методы и интерфейсы предназначены для того, чтобы обеспечить обязательную перегрузку методов в производных классах; обратная задача решается при помощи ключевого слова `final`. Методы, объявленные в базовом классе с ключевым словом `final`, не могут быть перегружены в производном классе. В листинге 6.17 приводится пример базового класса `base`, снабженного `final`-методом `final_method()`, и производного от него класса `derived`.

Замечание

В ранних версиях PHP допускалось также объявление членов классов с ключевым словом `final`, в текущих версиях PHP данная возможность запрещена.

Листинг 6.17. Использование ключевого слова `final`

```
<?php
  class base
  {
    public function final_method()
```

```
    {
      echo "base::final_method()";
    }
  }
  class derived extends base
  {
    // public function final_method()
    // {
    //   echo "derived::final_method()";
    // }
  }
?>
```

Замечание

Не имеет значения, в каком порядке будут перечислены ключевые слова `final` и `public` — допускается как использование последовательности `final public`, так и `public final`.

Попытка перегрузить метод `final_method()` в производном классе `derived` заканчивается ошибкой: **Fatal error: Cannot override final method base:: final_method() (Невозможно перегрузить final-метод base::final_method())**.

Помимо обычных методов, в качестве `final`-метода можно объявлять в том числе и специальные методы. Например, в листинге 6.18 в качестве `final`-метода объявляется конструктор класса `base`, в результате чего становится невозможным наследование производного класса `derived` с переопределенным конструктором.

Листинг 6.18. Запрет на наследование

```
<?php
  class base
  {
    public final function __construct()
    {
    }
  }
//  class derived extends base
//  {
//    Переопределяется final-метод, наследование
//    невозможно
//    public function __construct()
//    {
//    }
//  }
?>
```

Однако прием, представленный в листинге 6.18, не является надежным способом запрета наследования от класса `base`. Достаточно не переопределять конструктор производного класса `derived`, чтобы наследование стало возможным (листинг 6.19).

Листинг 6.19. Обход запрета на наследование

```
<?php
  class base
  {
    public final function __construct()
    {
    }
  }
  class derived extends base
  {
     // final-метод не переопределяется, наследование
     // возможно
  }
?>
```

Для того чтобы запретить наследование полностью, следует объявить `final`-класс (см. *раздел 6.9*).

6.9. *Final*-классы

Совместно с ключевым словом `final` можно объявлять не только отдельные методы, но и целые классы. Класс, объявленный при помощи ключевого слова `final`, не может иметь наследников (листинг 6.20).

Замечание

Класс не может быть объявлен одновременно и `final`-классом, и абстрактным (`abstract`).

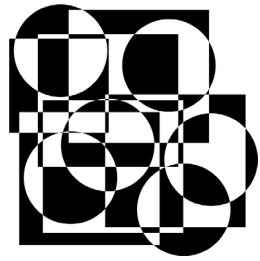
Листинг 6.20. Объявление `final`-класса

```
<?php
  final class base
  {
    public function __construct()
    {
    }
  }
```

```
//  class derived extends base
//  {
//  }
?>
```

Попытка объявить производный класс, наследующий от `final`-класса, заканчивается сообщением об ошибке: **Fatal error: Class derived may not inherit from final class (base) (Класс derived не может быть унаследован от final-класса (base))**.

ГЛАВА 7



Клонирование и серилизация объектов

Одной из особенностей PHP при выполнении в Web-среде является короткий период работы скриптов. Это приводит к тому, что объекты создаются и уничтожаются с большой частотой по сравнению с другими объектно-ориентированными системами. В таких условиях особое значение приобретает способность серилизовать объект, т. е. преобразовать образ в оперативной памяти в строку, которая может быть сохранена в файл или базу данных и из которой потом можно восстановить объект. Помимо серилизации в текущей главе также будет рассмотрен процесс клонирования объектов и управление им.

7.1. Клонирование объекта

Оператор присваивания = не приводит к созданию новой копии объекта: и старый и новый объект указывают на одну и ту же область памяти. В листинге 7.1 представлена операция присвоения одного объекта класса `cls` другому. При этом изменение члена класса нового объекта `$new_obj` отражается на старом объекте `$obj`.

Листинг 7.1. Присвоение одного объекта другому

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }
```

```
  $obj = new cls();
  $new_obj = $obj;
  $new_obj->var = 200;
  echo $obj->var; // 200
?>
```

Схематично процесс присвоения объекта `$obj` объекту `$new_obj` продемонстрирован на рис. 7.1.

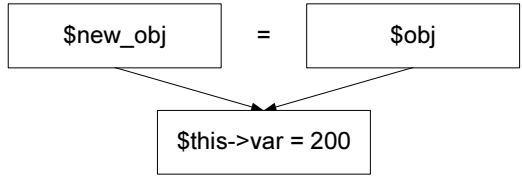


Рис. 7.1. Присвоение одного объекта другому приводит не к созданию новой копии, а к получению текущим объектом дополнительного псевдонима

Для создания копии текущего объекта используется специальная операция — *клонирование*, выполняемая с помощью ключевого слова `clone`, которое располагается непосредственно перед клонируемым объектом (листинг 7.2).

Листинг 7.2. Клонирование объекта

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }

  $obj = new cls();
  $new_obj = clone $obj;
  $new_obj->var = 200;
  echo $obj->var; // 100
?>
```

Схематически процесс клонирования объекта `$obj` и получение независимой копии `$new_obj` представлен на рис. 7.2.

Следует отметить, что оператор присваивания — это не единственный способ создания псевдонима объекта. Псевдоним объекта создается также при пере-

дache объекта внутрь функции. В отличие от обычных переменных, которые передаются по значению, массивы и объекты передаются внутрь функций по ссылке. Это означает, что изменения, произведенные над объектом внутри функции, отражаются на объекте после выполнения функции. В листинге 7.3 после вызова функции `change_obj()` член `$var` объекта `$obj` будет иметь значение `function_value`, а не 100.

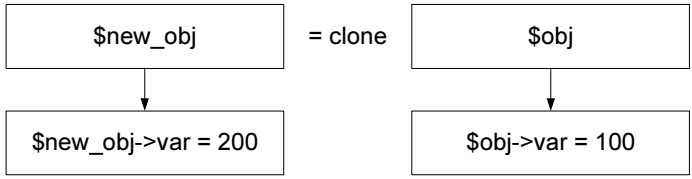


Рис. 7.2. Получение независимой копии объекта `$obj` посредством клонирования

Листинг 7.3. Передача объекта в функцию по ссылке

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }

  $obj = new cls();

  change_obj($obj);

  echo $obj->var; // function_value

  function change_obj($obj)
  {
    $obj->var = "function_value";
  }
?>
```

Если требуется передать объект по значению, как любую другую переменную, и избежать побочного эффекта, связанного с изменением объекта внутри функции, следует при вызове функции предварить объект ключевым словом `clone` (листинг 7.4).

Листинг 7.4. Передача объекта в функцию по значению

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }

  $obj = new cls();

  change_obj(clone $obj);

  echo $obj->var; // 100

  function change_obj($obj)
  {
    $obj->var = "function_value";
  }
?>
```

В листинге 7.4 внутрь функции передается не сам объект `$obj`, а лишь его копия, поэтому изменения, которые производятся над объектом внутри функции, не отражаются на объекте в основной программе.

7.2. Управление процессом клонирования. Метод *__clone()*

Клонирование объекта приводит к созданию новой копии объекта, при этом конструктор объекта не вызывается, в чем можно убедиться, поместив в конструктор вывод отладочной записи (листинг 7.5).

Листинг 7.5. Конструктор не вызывается при клонировании

```
<?php
  class cls
  {
    public $var;
    public function __construct()
```

```
    {
      $this->var = 100;
      echo "Вызов конструктора<br>";
    }
  }

  $obj = new cls();
  $new_obj = clone $obj;
?>
```

В результате работы скрипта из листинга 7.5 в окно браузера будет выведена лишь одна запись "Вызов конструктора". Тем не менее в процессе создания нового объекта может потребоваться выполнить ряд действий. Для этого PHP предоставляет специальный метод `__clone()`, который можно переопределить в классе. В листинге 7.6 приводится пример улучшенного класса `counter`, который подсчитывает количество созданных объектов и уже рассматривался в *разделе 6.1*. При создании нового объекта в конструкторе статическая переменная `$count` увеличивается на единицу, при уничтожении объекта значение переменной уменьшается на единицу. Однако, как было продемонстрировано выше, при клонировании конструктор не вызывается, и в системе появляются неучтенные объекты. Перегрузка метода `__clone()` позволяет исправить такую ситуацию.

Замечание

Метод `__clone()` не вмешивается в работу процесса клонирования, т. е. программист не должен реализовывать механизм клонирования самостоятельно, он может лишь использовать метод `__clone()` для реакции на процесс клонирования.

Листинг 7.6. Использование метода `__clone()`

```
<?php
  class counter
  {
    public static $count = 0;
    public function __construct()
    {
      counter::$count++;
    }
    public function __destruct()
    {
      counter::$count--;
    }
```

```
    public function __clone()
    {
      counter::$count++;
    }
  }
?>
```

Важной деталью, является то, что метод `__clone()` выполняется для нового объекта. Все изменения объекта, которые производятся в методе `__clone()`, будут отражаться на новом, а не на старом объекте. В листинге 7.7 в процессе клонирования единственному члену `$var` класса `cls` присваивается значение `Клон`, при этом изменения касаются лишь нового объекта.

Листинг 7.7. Метод `__clone()` вызывается для нового объекта

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
    public function __clone()
    {
      $this->var = "Клон";
    }
  }

  $obj = new cls();
  $new_obj = clone $obj;
  echo $new_obj->var; // Клон
  echo $obj->var; // 100
?>
```

7.3. Клонирование вложенного класса

Другой задачей метода `__clone()` является управление клонированием сложного объекта, который имеет в своем составе другие объекты. Дело в том, что такие вложенные объекты не клонируются — для них просто создается псевдоним. Таким образом, несмотря на то что создаются две независимые копии главного объекта, они по-прежнему указывают на один и тот же подобъект (рис. 7.3).

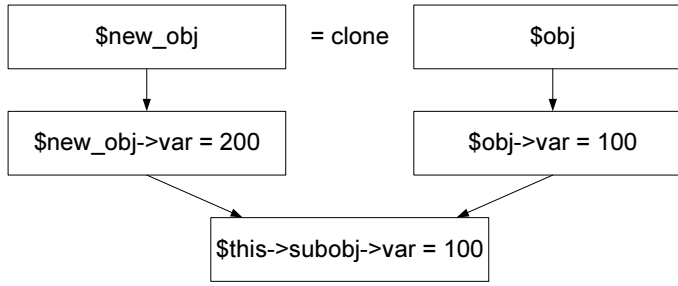


Рис. 7.3. Вложенные объекты не клонируются, для них создается псевдоним

В листинге 7.8 представлен класс `cls`, который содержит в своем составе объект `subobj` класса `subclass`.

Листинг 7.8. Вложенные классы не клонируются

```
<?php
  class subclass
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }
  class cls
  {
    public $subobj;
    public $var;
    public function __construct()
    {
      $this->subobj = new subclass();
      $this->var = 100;
    }
  }

  $obj = new cls();

  $new_obj = clone $obj;

  $new_obj->var = 200;
  $new_obj->subobj->var = 200;
```

```
  // Непосредственный член класса cls
  // клонируется
  echo $obj->var; // 100
  echo "<br>";
  // Члены вложенного класса subclass
  // не клонируются
  echo $obj->subobj->var; // 200
  echo "<br>";
?>
```

Следует отметить, что клонируются только непосредственные члены класса `cls`, внутренняя структура вложенного класса `subclass` не клонируется. Для того чтобы исправить ситуацию, объект `$subobj` необходимо клонировать явно в методе `__clone()` (листинг 7.9).

Листинг 7.9. Клонирование вложенного класса

```
<?php
  class subclass
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }
  class cls
  {
    public $subobj;
    public $var;
    public function __construct()
    {
      $this->subobj = new subclass();
      $this->var = 100;
    }
    public function __clone()
    {
      $this->subobj = clone $this->subobj;
    }
  }

  $obj = new cls();

  $new_obj = clone $obj;
```

```
  $new_obj->var = 200;
  $new_obj->subobj->var = 200;

  // Клонируются все члены, в том числе
  // и вложенных классов
  echo $obj->var; // 100
  echo "<br>";
  echo $obj->subobj->var; // 100
  echo "<br>";
?>
```

Результат работы скрипта в листинге 7.9 можно представить схемой, изображенной на рис. 7.4.

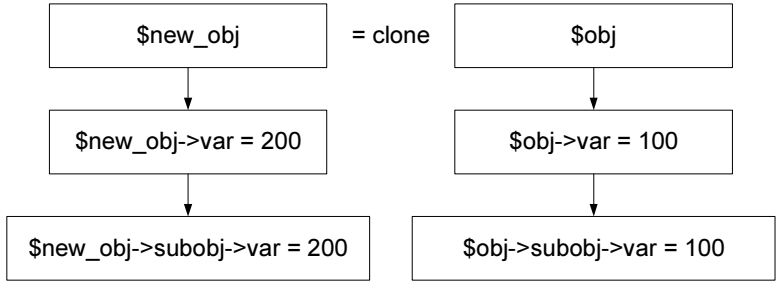


Рис. 7.4. Метод `__clone()` позволяет клонировать вложенные объекты

7.4. Серилизация объектов

Одной из особенностей PHP является то, что продолжительность жизни создаваемых с его помощью объектов, как правило, очень невелика. Это связано с тем, что все переменные и объекты уничтожаются после завершения скрипта. Протокол HTTP не является сессионным, т. е. каждое обращение к серверу воспринимается как обращение нового клиента, а история его предыдущих обращений не сохраняется. Разработчик Web-приложений должен сам реализовывать сохранение состояния приложения для каждого из клиентов, прибегая к сессиям и cookie. В таких условиях приобретает большое значение возможность передачи объекта между несколькими сеансами клиента или даже между отдельными страницами Web-приложения.

Для сохранения объекта в формате, который бы позволял в дальнейшем его восстановить, часто прибегают к *серилизации* — переводу объекта в строку при помощи функции `serialize()`. Такая строка может быть сохранена в файл или базу данных, а затем из нее можно получить сохраненный объект

при помощи обратной функции `unserialize()`. Функции имеют следующий синтаксис:

```
string serialize($obj)
object unserialize($str)
```

Для демонстрации приемов работы с функциями `serialize()` и `unserialize()` создадим вспомогательный класс `cls`, содержащий единственный открытый член `$var`, инициализация которого осуществляется в конструкторе класса (листинг 7.10).

Листинг 7.10. Класс `cls`

```
<?php
  class cls
  {
    public $var;
    public function __construct($var)
    {
      $this->var = $var;
    }
  }
?>
```

В листинге 7.11 представлен скрипт, который серилизует объект `$obj` класса `cls` в строку, а строку сохраняет в файл text.obj.

Замечание

Серилизации могут подвергаться не только объекты, но и массивы (в том числе многомерные).

Листинг 7.11. Серилизация объекта `$obj` класса `cls`

```
<?php
  // Подключаем определение класса cls
  require_once("class.cls.php");

  // Создаем объект
  $obj = new cls(100);

  // Серилизуем объект
  $text = serialize($obj);

  // Сохраняем объект в файл
  $fd = fopen("text.obj", "w");
```

```
  if(!$fd) exit("Невозможно открыть файл");
  fwrite($fd, $text);
  fclose($fd);
?>
```

Результатом работы скрипта из листинга 7.11 будет файл text.obj, содержащий следующую строку:

```
O:3:"cls":1:{s:3:"var";i:100;}
```

Данная строка предназначена для функции `unserialize()` и позволяет восстановить объект в другом файле (листинг 7.12).

Листинг 7.12. Восстановление объекта из строки

```
<?php
  // Подключаем определение класса cls
  require_once("class.cls.php");

  // Извлекаем серилизованный объект из файла
  $fd = fopen("text.obj", "r");
  if(!$fd) exit("Невозможно открыть файл");
  $text = fread($fd, filesize("text.obj"));
  fclose($fd);

  // Восстанавливаем объект
  $obj = unserialize($text);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 7.12 будет следующий дамп объекта `$obj`:

```
cls Object
(
    [var] => 100
)
```

Важно, чтобы при восстановлении объекта скрипт имел доступ к классу `cls`, иначе восстановление объекта выполнится не полностью. По сути, будет создан объект-контейнер, в котором имеются члены класса `cls`, однако отсутствуют какие бы то ни было методы:

```
__PHP_Incomplete_Class Object
(
    [__PHP_Incomplete_Class_Name] => cls
    [var] => 100
)
```

7.5. Передача объектов через сессию

Сериализованный объект можно передавать между страницами не только при помощи файла, но и при помощи сессии. В листинге 7.13 приводится пример размещения в сессии строки с сериализованным объектом.

Листинг 7.13. Помещение сериализованного объекта в сессию

```
<?php
  // Подключаем реализацию класса
  require_once("class.cls.php");

  // Инициируем сессию
  session_start();

  // Создаем объект
  $obj = new cls(100);

  // Помещаем объект в сессию
  $_SESSION['obj'] = serialize($obj);

  // Переход на вторую страницу
  echo "<a href=index1.php>ссылка</a>";
?>
```

В конце листинга 7.12 вставлена ссылка на файл index1.php, содержимое которого представлено в листинге 7.14.

Листинг 7.14. Извлечение сериализованного объекта из сессии

```
<?php
  // Подключаем реализацию класса
  require_once("class.cls.php");

  // Инициируем сессию
  session_start();
```

```
  // Восстанавливаем объект
  $obj = unserialize($_SESSION['obj']);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 7.14 является следующий дамп объекта `$obj`:

```
cls Object
(
    [var] => 100
)
```

Однако схема сохранения объекта в сессии, представленная в листингах 7.13 и 7.14, достаточно опасна. Это связано с тем, что суперглобальный массив `$_SESSION` сам подвергается серилизации при помощи функции `serialize()` перед тем, как сохраняется в файл сессии. Если помимо объекта в сессию помещаются другие переменные, то строка, полученная в результате серилизации объекта, может помешать восстановлению данных из файла сессии. Поэтому разумнее не подвергать объекты серилизации вообще — механизм сессий позаботится об этом самостоятельно (листинг 7.15).

Листинг 7.15. Передача объекта через сессию без серилизации

```
<?php
  // Подключаем реализацию класса
  require_once("class.cls.php");

  // Инициируем сессию
  session_start();

  // Создаем объект
  $obj = new cls(100);

  // Помещаем объект в сессию
  $_SESSION['obj'] = $obj;

  // Переход на вторую страницу
  echo "<a href=index1.php>ссылка</a>";
?>
```

Файл index1.php для скрипта, представленного в листинге 7.15, может выглядеть так, как в листинге 7.16.

Листинг 7.16. Извлечение объекта

```
<?php
  // Подключаем реализацию класса
  require_once("class.cls.php");

  // Инициируем сессию
  session_start();

  // Выводим дамп объекта
  echo "<pre>";
  print_r($_SESSION['obj']);
  echo "</pre>";
?>
```

7.6. Сохранение объектов в СУБД MySQL

Ни один серьезный современный Web-проект не может обходиться без СУБД MySQL. СУБД берет на себя рутинные операции с файлами, связанные с блокировкой файлов при одновременном доступе к ним нескольких потоков, сортировку и поиск данных. Эффективность СУБД практически всегда превышает эффективность приложений, непосредственно работающих с файлами, поэтому часто разумно сохранять серилизованные объекты в СУБД.

Для хранения серилизованных объектов создадим таблицу `object`. Таблица будет содержать два поля: `id_object` — первичный ключ таблицы, `object` — поле типа `BLOB`, предназначенное для хранения объекта (листинг 7.17).

Листинг 7.17. Таблица `object`

```
CREATE TABLE object (
  id_object INT(11) NOT NULL AUTO_INCREMENT,
  object BLOB NOT NULL,
  PRIMARY KEY (id_object)
);
```

В листинге 7.18 приводится пример сохранения объекта класса `cls` (листинг 7.10) в таблице `object`. Установка соединения с базой данных осуществляется в специальном файле config.php (листинг 4.38).

Листинг 7.18. Сохранение серилизованного объекта в таблицу `object`

```
<?php
  // Подключаем реализацию класса
  require_once("class.cls.php");
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Создаем объект
  $obj = new cls(100);

  // Серилизуем объект
  $object = serialize($obj);
  // Экранируем специальные символы
  $object = mysql_real_escape_string($object);

  // Сохраняем объект в таблице базы данных
  $query = "INSERT INTO object VALUES (NULL, '$object')";
  if(!mysql_query($query)) exit("Ошибка сохранения
                                 объекта в базе данных");
  else echo "Объект успешно сохранен в базе данных";
?>
```

Для извлечения объекта из таблицы базы данных можно использовать скрипт, представленный в листинге 7.19.

Листинг 7.19. Извлечение и восстановление объекта

```
<?php
  // Подключаем реализацию класса
  require_once("class.cls.php");
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Извлекаем из таблицы object
  // объект с идентификатором 1
  $query = "SELECT * FROM object WHERE id_object = 1";
  $obj = mysql_query($query);
  if(!$obj) exit("Ошибка извлечения объекта из таблицы");
  // Если запись найдена — обрабатываем ее
  if(mysql_num_rows($obj))
  {
    $table = mysql_fetch_array($obj);
```

```
    // Восстанавливаем объект
    $object = unserialize($table['object']);
    // Выводим дамп объекта
    echo "<pre>";
    print_r($object);
    echo "</pre>";
  }
?>
```

Результатом работы скрипта из листинга 7.19 будет дамп объекта класса `cls`:

```
cls Object
(
    [var] => 100
)
```

Одним из сдерживающих факторов использования объектно-ориентированного подхода в рамках Web-программирования является слабое распространение объектно-ориентированных баз данных среди хост-провайдеров. Объектно-ориентированные СУБД сильно уступают в скорости классическим реляционным базам данных, поэтому многие разработчики, а вслед за ними и хост-провайдеры стараются обходиться реляционными базами данных. В результате перед помещением в базу данных объект необходимо серилизовать, вследствие чего теряется возможность поиска по отдельным полям объекта. Альтернативой этому может служить хранение каждого объекта в отдельной таблице, предусматривающей собственное поле для всех его членов, что не всегда возможно, особенно при использовании динамических объектов.

7.7. Управление серилизацией. Методы *__sleep()* и *__wakeup()*

При сохранении и восстановлении объекта при помощи функций `serialize()` и `unserialize()` может потребоваться осуществить ряд действий, например, убрать из объекта данные, которые не должны подвергаться серилизации, или скорректировать их значения, если они теряют актуальность. Для осуществления подобных действий предназначены два специальных метода, которые могут быть перегружены в классе:

- `__sleep()` — вызывается, когда объект подвергается серилизации при помощи функции `serialize()`;
- `__wakeup()` — вызывается при восстановлении объекта при помощи функции `unserialize()`.

Методы не принимают никаких параметров.

Замечание

Название метода `__sleep()` образовано от английского глагола "спать", а метода `__wakeup()` — от глагола "пробуждаться".

Для демонстрации приемов работы со специальными методами `__sleep()` и `__wakeup()` создадим класс `user`, который будет иметь в своем составе следующие члены:

- `$name` — имя пользователя;
- `$password` — его пароль (если поле пустое, то пользователь перенаправляется на страницу авторизации);
- `$referrer` — последняя посещенная страница;
- `$time` — время авторизации пользователя.

В листинге 7.20 приводится возможная реализация класса `user`.

Листинг 7.20. Класс `user`

```
<?php
  class user
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

В листинге 7.21 демонстрируется скрипт, который подвергает серилизации объект `$obj` класса `user`.

Листинг 7.21. Серилизация объекта класса `user`

```
<?php
  // Подключаем реализацию класса
  require_once("class.user.php");

  // Создаем объект
  $obj = new user("nick", "password");

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  // Серилизуем объект
  $object = serialize($obj);

  // Выводим серилизованный объект
  echo $object;
?>
```

Результатом работы скрипта из листинга 7.21 будут следующие строки:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676349
)
O:4:"user":4:{s:4:"name";s:4:"nick";s:8:"password";s:8:"password";s:8:
"referrer";s:17:"/oop/07/index.php";s:4:"time";i:1177676349;}
```

При сериализации объекта полезно назначать паролю `$password` пустую строку, чтобы не допускать его сохранения на жестком диске в незашифрованном виде. Для решения этой задачи как нельзя лучше подходит метод `__sleep()`, обнуляющий член `$password` и возвращающий объект, который будет передан функции `serialize()`. По сути метод `__sleep()` выступает в качестве фильтра, позволяя настроить процесс серилизации на избирательное сохранение информации (рис. 7.5).

В листинге 7.21 представлен модифицированный вариант класса `user`, который обнуляет значение члена `$password` при серилизации объекта.

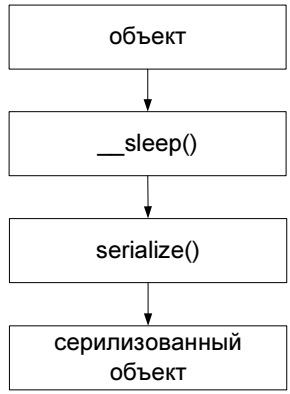


Рис. 7.5. Схема серилизации объекта

Листинг 7.22. Использование метода `__sleep()`

```
<?php
  class user
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }
    public function __sleep()
    {
      $this->password = "";
      return $this;
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

Однако теперь попытка использовать функцию `serialize()` применительно к объекту класса `user` будет приводить к обнулению члена `$password`, что может быть неудобно, если планируется дальнейшее использование объекта (листинг 7.23).

Листинг 7.23. Побочный эффект серилизации

```
<?php
  // Подключаем реализацию класса
  require_once("class.user.php");

  // Создаем объект
  $obj = new user("nick", "password");

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  // Серилизуем объект
  $object = serialize($obj);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  // Выводим серилизованный объект
  echo $object;
?>
```

Результатом работы скрипта из листинга 7.23 будут следующие строки, из которых видно, что вызов функции `serialize()` приводит к необратимой модификации объекта `$obj`:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676475
)
user Object
```

```
(
    [name] => nick
    [password] =>
    [referrer] => /oop/07/index.php
    [time] => 1177676475
)
O:4:"user":4:{s:4:"nick";N;s:0:"";N;s:17:"/oop/07/index.php";N;N;}
```

Для решения этой проблемы удобно воспользоваться клонированием, редактируя и возвращая не оригинальный объект, а его копию (листинг 7.24).

Листинг 7.24. Исключение побочного эффекта клонирования

```
<?php
  class user
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }
    public function __sleep()
    {
      $obj = clone $this;
      $obj->password = "";
      return $obj;
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

Теперь, выполнив листинг 7.23, можно убедиться в неизменности объекта, как и после вызова функции `serialize()`:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676630
)
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676630
)
O:4:"user":4:{s:4:"nick";N;s:0:"";N;s:17:"/oop/07/index.php";N;N;}
```

Восстановление объекта из серилизованного состояния при помощи функции `unserialize()` приведет к созданию объекта, в котором сохраняется время предыдущей авторизации пользователя `$time`, поэтому разумно обновить член `$time` при вызове функции `unserialize()`. Для решения этой задачи предназначен специальный метод `__wakeup()`, который вызывается сразу после восстановления объекта (рис. 7.6).

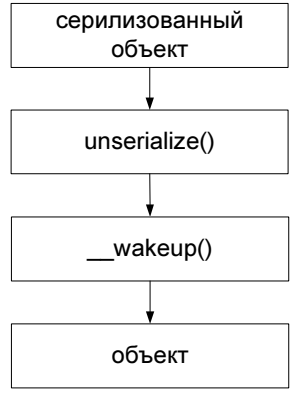


Рис. 7.6. Схема восстановления объекта

В листинге 7.25 представлен модифицированный класс `user`, который обеспечивает обновление времени авторизации пользователя при восстановлении объекта.

Листинг 7.25. Использование метода `__wakeup()`

```
<?php
  class user
```

```
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }
    public function __sleep()
    {
      $obj = clone $this;
      $obj->password = "";
      return $obj;
    }
    public function __wakeup()
    {
      $this->time = time();
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

В листинге 7.26 представлен скрипт, который демонстрирует восстановление объекта из серилизованного состояния с обновлением члена `$time`.

Листинг 7.26. Восстановление объекта

```
<?php
  // Подключаем реализацию класса
  require_once("class.user.php");

  // Серилизованный объект
  $object = 'O:4:"user":4:{s:4:"name";s:4:"nick";'.
            's:8:"password";s:8:"password";'.
            's:8:"referrer";s:17:"/oop/07/index.php";'.
            's:4:"time";i:1177676349;}';
```

```
  // Восстанавливаем объект
  $obj = unserialize($object);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 7.26 будет следующий дамп объекта класса `user`:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177677626
)
```

В листинге 7.26 был использован серилизованный вариант, который не подвергался фильтрации через метод `__sleep()`. Это вызвано тем, что в текущих версиях PHP метод `__sleep()` возвращает массив, а не объект, и попытка вернуть объект автоматически приводит его к массиву, отбрасывая имена полей. В результате после серилизации невозможно восстановить объект, класс которого содержит перегруженный метод `__sleep()` (при отсутствии этого метода восстановление происходит обычным образом). Возможно, в будущих версиях PHP эта ситуация будет исправлена, пока же для использования методов `__sleep()` и `__wakeup()` следует искать альтернативные варианты. Один из таких вариантов обсуждается в следующем разделе.

7.8. Автоматическое сохранение объекта в СУБД MySQL

Как было показано в предыдущем разделе, серилизация, которая осуществляется при помощи функции `serialize()`, не всегда проходит гладко. Кроме того, если предполагается выполнение поиска по массиву объектов, упакованный в строку объект практически бесполезен. Одним из выходов является переопределение метода `__sleep()` таким образом, чтобы каждый член объекта сохранялся в соответствующем поле специальной таблицы. Для рассматривавшегося в *разделе 7.7* класса `user` можно предусмотреть таблицу `object_user`, оператор `CREATE TABLE` для создания которой представлен в листинге 7.27.

Листинг 7.27. Структура таблицы `object_user`

```
CREATE TABLE object_user (
id_user INT NOT NULL AUTO_INCREMENT,
name TINYTEXT NOT NULL,
pass TINYTEXT NOT NULL,
referrer TINYTEXT NOT NULL,
puttime DATETIME NOT NULL,
PRIMARY KEY (id_user)
);
```

Таблица `object_user` содержит пять полей, которые имеют следующее назначение:

- `id_user` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`. Атрибут `AUTO_INCREMENT` обеспечивает автоматическое назначение полю уникального значения при передаче ему значения `NULL` в операторе `INSERT`;
- `name` — имя пользователя: в данное поле будет помещено значение члена `$this->name`;
- `pass` — пароль пользователя, куда будет заноситься значение члена `$this->password`;
- `referrer` — последняя посещенная страница, будет хранить значение члена `$this->referrer`;
- `puttime` — время авторизации, в данное поле будет помещено значение члена `$this->time`;

В листинге 7.28 представлена возможная реализация метода `__sleep()` класса `user`, объект которого при вызове функции `serialize()` вместо серилизации помещается в таблицу `object_user`.

Листинг 7.28. Сохранение объекта в СУБД MySQL

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  class user
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
```

```
    $this->password = $password;
    $this->referrer = $_SERVER['PHP_SELF'];
    $this->time     = time();
  }
  // Доступ к закрытым членам
  public function __get($value)
  {
    if(isset($this->$value)) return $this->$value;
    else return false;
  }
  // Редактирование объекта перед серилизацией
  public function __sleep()
  {
    // Экранируем специальные символы
    $name     = mysql_real_escape_string($this->name);
    $referrer = mysql_real_escape_string($this->referrer);

    // Обнуляем поля с паролем
    $password = "";

    // Преобразуем дату к формату MySQL
    $time = date("Y-m-d H:i:s", $this->time);

    // Формируем и выполняем SQL-запрос
    $query = "INSERT INTO object_user
              VALUES (NULL,
                      '$name',
                      '$password',
                      '$referrer',
                      '$time')";
    if(!mysql_query($query))
    {
      // В случае неудачи возвращает массив со значением false
      return array("false");
    }
    else
    {
      // В случае успеха возвращаем значение первичного ключа,
      // автоматически назначенного записи посредством
      // механизма AUTO_INCREMENT
      $id = mysql_insert_id();
      return array("$id"); // O:4:"user":1:{s:2:"$id";N;}
    }
  }
```

```
    // Имя пользователя
    private $name;
    // Его пароль
    private $password;
    // Последняя посещенная страница
    private $referrer;
    // Время авторизации пользователя
    private $time;
  }
?>
```

При попытке серилизовать объект класса `user` при помощи функции `serialize()` метод `__sleep()` формирует SQL-запрос `INSERT`, с помощью которого объект помещается в таблицу `object_user`. Для того чтобы не возникло конфликтов с одиночными кавычками, строковые члены класса подвергаются обработке функцией `mysql_real_escape_string()`, осуществляющей экранирование специальных символов (поле `$this->password` обнуляется, чтобы не сохранять незашифрованный пароль). Полю `id_user` передается значение `NULL`, после чего в соответствии с атрибутом `AUTO_INCREMENT` ему присваивается уникальное значение, равное максимальному значению столбца, увеличенному на единицу. Получить это значение можно при помощи специальной функции `mysql_insert_id()`.

Вместо объекта метод `__sleep()` возвращает массив, единственный элемент которого равен только что назначенному значению поля `id_user`. Однако функция `serialize()` подвергает его серилизации. Так, для `id_user = 20` результат может выглядеть следующим образом:

```
O:4:"user":1:{s:2:"20";N;}
```

Для получения конечного значения создадим в классе `user` статический метод `serialize()`, который будет отфильтровывать нужные значения при помощи регулярных выражений. Помимо этого, следует ввести метод, который бы восстанавливал объект из базы данных. Воспользоваться специальным методом `__wakeup()` уже не удастся, так как он вступает в действие в уже созданном объекте, после функции `unserialize()`. В предложенной схеме этой функции нет места, поскольку не создается строка с серилизованным объектом. В листинге 7.29 представлен конечный вариант класса `user`, который реализует альтернативную схему сохранения объекта в базу данных.

Листинг 7.29. Конечный вариант класса `user`

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
```

```
class user
{
  // Конструктор
  public function __construct($name, $password)
  {
    $this->name     = $name;
    $this->password = $password;
    $this->referrer = $_SERVER['PHP_SELF'];
    $this->time     = time();
  }
  // Доступ к закрытым членам
  public function __get($value)
  {
    if(isset($this->$value)) return $this->$value;
    else return false;
  }
  // Редактирование объекта перед серилизацией
  public function __sleep()
  {
    // Экранируем специальные символы
    $name     = mysql_real_escape_string($this->name);
    $referrer = mysql_real_escape_string($this->referrer);

    // Обнуляем поля с паролем
    $password = "";

    // Преобразуем дату к формату MySQL
    $time = date("Y-m-d H:i:s", $this->time);

    // Формируем и выполняем SQL-запрос
    $query = "INSERT INTO object_user
              VALUES (NULL,
                      '$name',
                      '$password',
                      '$referrer',
                      '$time')";
    if(!mysql_query($query))
    {
      // В случае неудачи возвращает массив со значением false
      return array(false);
    }
    else
    {
```

```
    // В случае успеха возвращаем значение первичного ключа,
    // автоматически назначенного записи посредством
    // механизма AUTO_INCREMENT
    $id = mysql_insert_id();
    return array("$id"); // O:4:"user":1:{s:2:"$id";N;}
  }
}
// Статический метод обработки результатов серилизации;
// возвращается уникальный номер объекта в базе данных
public static function serialize($ser)
{
  preg_match("|\"([\d]+)\"|", $ser, $out);
  return $out[1];
}
// Статический метод восстановления объекта из базы данных
public static function unserialize($id_user)
{
  // Приводим параметр к целому числу, предотвращая
  // SQL-инъекцию
  $id_user = intval($id_user);

  // Формируем и выполняем запрос на получение записи
  // с объектом
  $query = "SELECT * FROM object_user
            WHERE id_user = $id_user";
  $obj = mysql_query($query);
  // При возникновении ошибки возвращаем false,
  // что сигнализирует о неудачном восстановлении объекта
  if(!$obj) return false;
  // Если в таблице нет ни одной записи, также возвращаем
  // false
  if(!mysql_num_rows($obj)) return false;
  $object = mysql_fetch_array($obj);
  // Формируем и возвращаем объект
  return new user($object['name'], $object['pass']);
}

// Имя пользователя
private $name;
// Его пароль
private $password;
// Последняя посещенная страница
private $referrer;
```

```
    // Время авторизации пользователя
    private $time;
  }
?>
```

В листинге 7.30 демонстрируется сохранение и восстановление объекта из базы данных.

Листинг 7.30. Сохранение и восстановление объекта из базы данных

```
<?php
  // Подключаем реализацию класса
  require_once("class.user.php");

  // Создаем объект
  $obj = new user("nick", "password");

  // Серилизуем объект
  $id_user = user::serialize(serialize($obj));

  // Восстанавливаем объект
  $red = user::unserialize($id_user);

  // Выводим дамп восстановленного объекта
  echo "<pre>";
  print_r($red);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 7.30 будет следующий дамп объекта класса `user`:

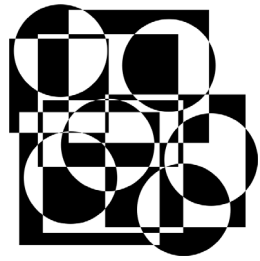
```
user Object
(
    [name:private] => nick
    [password:private] =>
    [referrer:private] => /oop/07/index.php
    [time:private] => 1177781566
)
```

Поле `$time` будет автоматически обновляться, так как метод `unserialize()` создает новый объект, в результате чего вызывается конструктор класса, где производится корректировка поля. Однако поле `$referrer` оказывается не обновленным, поскольку никаких внешних механизмов для его корректировки

не предусмотрено. Впрочем, эту проблему можно решить, если сделать метод `unserialize()` не статическим, а обычным.

Представленный подход достаточно спорный, т. к. изменяет привычную логику использования функций `serialize()` и `unserialize()`, что может запутать программиста, сопровождающего приложение. Однако он демонстрирует, каким образом можно обходить ограничения и откровенные ошибки в инструментах программирования. Подобные ошибки и несоответствия могут встречаться достаточно часто и приводить к срывам сроков и невозможности реализации требований к проекту, поэтому важно относиться к ним спокойно и уметь находить эффективные и удобные альтернативные пути.

ГЛАВА 8



Обработка ошибок и исключения

Исключения не являются непременным атрибутом объектно-ориентированного подхода, однако сопровождают каждый объектно-ориентированный язык де-факто. Дело в том, что объектно-ориентированный подход серьезно изменяет структуру кода, и использующий его программист отчасти сам становится автором нового языка программирования, разрабатывая классы в рамках предметной области. Структурные изменения кода требуют новых способов обработки ошибок (нештатных ситуаций).

Разработка класса происходит на абстрактном уровне: при этом создается не конкретная область памяти, а лишь определяется поведение объектов. Класс выступает инструментом, который, как и язык программирования, может применяться в совершенно разных областях и приложениях. Обработка нештатных ситуаций, ошибок как кода, так и ввода данных может быть различной для разных приложений: где-то достаточно вывести сообщение при помощи конструкции `echo`; где-то сообщение следует оформить в виде HTML-страницы с дизайном, согласованным с остальными страницами приложения; где-то сообщение об ошибке должно быть помещено в журнал (в файл или базу данных). Предусмотреть заранее формат ошибки невозможно, и любой формат будет сужать область применения класса и возможность его повторного использования.

Выходом из ситуации является разделение кода класса и кода обработки ошибок, что достигается при помощи специального механизма — исключений. Разработчик класса может сгенерировать исключение, а пользователь класса может его обработать по своему усмотрению.

Разработчики могут проектировать свои собственные исключения, которые являются классами, при этом генерация исключений сводится к передаче объекта исключения из точки возникновения нештатной ситуации в обработчик исключений.

Удобство применения исключительных ситуаций в объектно-ориентированном программировании вовсе не означает, что следует пренебрегать процедурными средствами обработки ошибок. В частности, в PHP имеется развитая система отслеживания и контроля за ошибками, использование которой в ряде случаев удобнее и эффективнее исключений.

Замечание

Механизм исключений введен в PHP, начиная с версии 5.0.0.

8.1. Синтаксис исключений

Для реализации механизма исключений в PHP введены следующие ключевые слова: `try` (контролировать), `throw` (генерировать) и `catch` (обрабатывать).

Замечание

Механизм исключений не обязательно должен быть привязан к объектно-ориентированной системе, допускается использование исключений применительно к структурному коду.

Ключевое слово `try` позволяет выделить в любом месте скрипта так называемый *контролируемый блок*, за которым следует один или несколько блоков обработки исключений, реализуемых с помощью ключевого слова `catch` (листинг 8.1).

Листинг 8.1. Создание контролируемого блока

```
<?php
  try
  {
    // Операторы
    ...
    // Генерация исключений
    throw Выражение_генерации_исключения;
    ...
    // Операторы
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
  }
?>
```

Обработчик (или обработчики) всегда располагаются после контролируемого оператором `try` блока кода. Среди операторов контролируемого блока могут

быть любые операторы и объявления PHP. Если в теле контролируемого блока исключение генерируется при помощи ключевого слова `throw`, то интерпретатор PHP переходит в catch-обработчик. В листинге 8.2 представлен скрипт, в котором по случайному закону либо генерируется, либо не генерируется исключение.

Листинг 8.2. Генерация исключения по случайному закону

```
<?php
  try
  {
    // Генерируем исключение по случайному закону
    if(rand(0,1))
    {
      // Генерация исключений
      throw new Exception();
    }
  }
  catch(Exception $exp)
  {
    // Фраза выводится, если было сгенерировано исключение
    exit("Произошла исключительная ситуация");
  }
  // Фраза выводится, если исключение не генерировалось
  echo "Штатная работа скрипта";
?>
```

В зависимости от того, возвращает функция `rand()` 0 или 1, выводится либо фраза **Произошла исключительная ситуация**, либо фраза **Штатная работа скрипта**. Следует обратить внимание, что если исключение не генерируется, то код в catch-блоках не выполняется.

В качестве исключения выступает объект класса `Exception`, который создается при помощи ключевого `new` непосредственно при вызове оператора `throw`. Однако объект можно подготовить заранее, как это продемонстрировано в листинге 8.3.

Листинг 8.3. Предварительное создание объекта исключения

```
<?php
  try
  {
    if(rand(0,1))
```

```
    {
      // Подготовка исключения
      $obj = new Exception();
      // Генерация исключений
      throw $obj;
    }
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    exit("Произошла исключительная ситуация");
  }
  echo "Штатная работа скрипта";
?>
```

При генерации исключения ключевое слово `throw` принимает объект класса `Exception` или производного класса.

Замечание

Создание собственных производных классов, наследующих от класса `Exception`, рассматривается в *разделе 8.8*.

8.2. Интерфейс класса *Exception*

Для того чтобы эффективно использовать класс `Exception`, следует познакомиться с его интерфейсом. В табл. 8.1 представлены члены класса `Exception`, объявленные со спецификатором доступа `protected`, доступ к которым можно получить при помощи методов класса, а также из производных классов.

Замечание

Класс `Exception` относится к так называемым предопределенным классам, т. е. классам, которые реализованы в PHP-интерпретаторе и не требуют реализации со стороны программиста. Другие предопределенные классы более подробно рассматриваются в *приложении 1*.

Таблица 8.1. *Защищенные члены класса* `Exception`

Член	Описание
`$this->message`	Текстовое сообщение, описывающее исключительную ситуацию
`$this->code`	Числовой код, присвоенный данному типу исключительных ситуаций
`$this->file`	Имя файла, в котором произошла исключительная ситуация

***Таблица 8.1** (окончание)*

Член	Описание
`$this->line`	Номер строки файла `$this->file`, в которой произошла исключительная ситуация

Так как члены класса `Exception` объявлены со спецификатором `protected`, доступ к ним осуществляется при помощи методов класса, описание которых представлено в табл. 8.2.

***Таблица 8.2.** Методы класса `Exception`*

Метод	Описание
`public function __construct ($message = null, $code = 0)`	Конструктор класса, инициализирующий члены `$message` и `$code`, оба параметра необязательны
`final function getMessage()`	Метод, возвращающий текстовое сообщение члену `$this->message`
`final function getCode()`	Метод, возвращающий числовой код `$this->code`, характеризующий исключительную ситуацию
`final function getFile()`	Метод, возвращающий имя файла `$this->file`, в котором произошла исключительная ситуация
`final function getLine()`	Метод, возвращающий номер строки `$this->line`, в которой произошла исключительная ситуация
`final function getTrace()`	Стек обработки исключительной ситуации в виде массива
`final function getTraceAsString()`	Стек обработки исключительной ситуации в виде строки
`function __toString()`	Перегрузка метода `__toString()`, возвращающего строку при использовании объекта в строковом контексте

Класс `Exception` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 8.4.

Листинг 8.4. Гипотетическая реализация класса `Exception`

```
<?php
  class Exception
```

```
{
  // Текстовое сообщение, описывающее
  // исключительную ситуацию
  protected $message = 'Unknown exception';
  // Числовой код, назначенный данному
  // типу исключительных ситуаций
  protected $code = 0;
  // Имя файла, в котором произошла
  // исключительная ситуация
  protected $file;
  // Номер строки, в которой произошла
  // исключительная ситуация
  protected $line;

  // Конструктор класса, инициализирующий
  // члены $message и $code — оба параметра
  // реализованы как необязательные
  public function __construct($message = null, $code = 0);

  // Метод, возвращающий текстовое
  // сообщение $this->message
  public final function getMessage();
  // Метод, возвращающий числовой код
  // $this->code, характеризующий
  // исключительную ситуацию
  public final function getCode();
  // Метод, возвращающий имя файла
  // $this->file, в котором произошла
  // исключительная ситуация
  public final function getFile();
  // Метод, возвращающий номер строки
  // $this->line, в которой произошла
  // исключительная ситуация
  public final function getLine();
  // Стек обработки исключительной
  // ситуации в виде массива
  public final function getTrace();
  // Стек обработки исключительной
  // ситуации в виде строки
  public final function getTraceAsString();

  // Перегрузка метода __toString(),
  // возвращающего строку при использовании
  // объекта в строковом контексте
```

```
    public function __toString();
  }
?>
```

Как видно из таблицы 8.2 и листинга 8.4, конструктор класса `Exception` позволяет инициализировать защищенные члены `$message` и `$code`, что бывает не лишним, если в контролируемом блоке генерируется несколько исключений (листинг 8.5).

Листинг 8.5. Передача сообщения и кода из точки генерации исключения

```
<?php
  try
  {
    $code = rand(0,1);
    if(!$code)
    {
      // Генерация исключений
      throw new Exception("Первая точка входа", $code);
    }
    else
    {
      // Генерация исключений
      throw new Exception("Вторая точка входа", $code);
    }
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение {$exp->getCode()} : {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
  }
?>
```

В зависимости от того, возвращает функция генерации случайного значения `rand()` число 0 или 1, скрипт из листинга 8.5 может выводить в окно браузера либо последовательность строк для первого оператора `throw`:

```
Исключение 0 : Первая точка входа
в файле D:\main\oop\08\index.php
в строке 8
```

либо для второго:

```
Исключение 1 : Вторая точка входа
в файле D:\main\oop\08\index.php
в строке 13
```

Таким образом, можно однозначно определить, в каком контексте было сгенерировано исключение, даже если контролируемый блок `try` содержит несколько вызовов оператора `throw`.

8.3. Генерация исключений в функциях

В предыдущих разделах механизм исключений демонстрировался в простейших ситуациях, когда для обработки внештатных ситуаций более уместны обычные логические операторы `if` и `switch`. Оценить полезность данного механизма можно на примере сложного скрипта, вызывающего множество функций, обработка внештатных ситуаций в которых должна производиться в одной точке.

Разработаем функцию `print_user()`, которая извлекает из таблицы `object_user` (см. листинг 7.26) список пользователей и выводит их в окно браузера (листинг 8.6).

Листинг 8.6. Функция `print_user()`

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Определяем константы для кодов исключения
  define("SQL_ERROR", 100);
  define("NONE_RECORDS", 101);

  function print_user()
  {
    $query = "SELECT * FROM object_user";
    $usr = mysql_query($query);
    // Генерируем исключение в случае
    // ошибки выполнения SQL-запроса
    if(!$usr) throw new Exception(mysql_error(), SQL_ERROR);
    // Генерируем исключение, если в
    // таблице нет ни одной записи
    if(!mysql_num_rows($usr))
```

```
    {
      throw new Exception("Отсутствуют записи в таблице", NONE_RECORDS);
    }

    // Выводим список пользователей
    while($user = mysql_fetch_array($usr))
    {
      echo $user['name']."<br>";
    }
  }
?>
```

Функция генерирует два исключения: первое — при возникновении ошибки в SQL-запросе, второе — если в таблице `object_user` нет ни одной записи. В случае возникновения ошибки в SQL-запросе через объект класса `Exception` передается текстовое описание возникшей проблемы, получаемое посредством функции `mysql_error()`, а также константа `SQL_ERROR`, значение которой определяется в том же файле. Если обнаруживается, что таблица не содержит ни одной записи, через объект класса `Exception` в обработчик передается текстовое сообщение "**Отсутствуют записи в таблице**", а также константа `NONE_RECORDS`.

Замечание

Для числовых кодов предпочтительнее вводить константы: это позволяет значительно увеличить читабельность и масштабируемость кода. Для изменения значения константы (например, чтобы исключить конфликт значений) достаточно исправить код в одной точке. С другой стороны, если используются так называемые "магические числа" — т. е. непосредственные значения чисел вместо констант, придется произвести десятки, а то и сотни исправлений, для чего, вероятно, потребуется много времени, не говоря уже о том, что это может вызвать появление ошибок в коде.

В скрипте из листинга 8.6 соединение с СУБД MySQL устанавливается при помощи вспомогательного файла config.php, обработку внештатных ситуаций в котором можно также реализовать при помощи исключений (листинг 8.7).

Листинг 8.7. Установка соединения с СУБД MySQL

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "oop";
  // Имя пользователя базы данных
  $dbuser = "root";
```

```
  // и его пароль
  $dbpasswd = "";

  // Определяем константы для кодов исключения
  define("NOT_CONNECTION", 0);
  define("NOT_DATABASE", 1);

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx)
  {
    throw new Exception("Невозможно установить
                         соединение с сервером MySQL ",
                         NOT_CONNECTION);
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) )
  {
    throw new Exception("Ошибка выбора базы данных", NOT_DATABASE);
  }

  // Устанавливаем кодировку соединения; следует выбрать ту кодировку,
  // в которой данные будут отправляться серверу MySQL
  @mysql_query("SET NAMES 'cp1251'");
?>
```

Как видно из листинга 8.7, вместо того чтобы непосредственно выводить сообщения в окно браузера в случае возникновения ошибочных ситуаций, генерируются исключения. Для кодов исключения также вводятся константы `NOT_CONNECTION` и `NOT_DATABASE`, для ошибки отсутствия соединения и ошибки выбора базы данных соответственно.

Генерация исключений позволяет "отложить" обработку ошибочных ситуаций и разделить логику приложения и логику обработки ошибок. В листинге 8.8 представлен возможный вызов функции `print_user()`.

Листинг 8.8. Вызов функции `print_user()`

```
<?php
  try
  {
    // Подключаем реализацию функции print_user()
    require_once("function.print_user.php");
```

```
    // Вызываем функцию
    print_user();
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение {$exp->getCode()} : {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
  }
?>
```

Важно отметить, что все совершенно разноплановые ошибки, которые возникают в функции `print_user()`, а также файле function.print_user.php и всех вложенных в него файлах, не только обрабатываются в одном месте, но и обрабатываются одинаково. Например, если разработчик указал неправильное имя базы, то скрипт из листинга 8.8 может вывести следующие строки:

```
Исключение 1 : Ошибка выбора базы данных
в файле D:\main\oop\08\config.php
в строке 24
```

После того как создание Web-приложения завершено, разработчик получает возможность изменить формат вывода сообщения об ошибке, исправив лишь один catch-блок (листинг 8.9).

Листинг 8.9. Изменение формата вывода сообщения об ошибке

```
<?php
  try
  {
    // Подключаем реализацию функции print_user()
    require_once("function.print_user.php");
    // Вызываем функцию
    print_user();
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    $url = "error.php?id={$exp->getCode()}";
    echo "<HTML><HEAD>
           <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$url'>
          </HEAD></HTML>";
  }
?>
```

В листинге 8.9 осуществляется переадресация на страницу error.php с передачей кода состояния исключения через GET-параметр `id`. Страницу error.php можно оформить в стиле Web-приложения, уменьшить количество технической информации и добавить извинения.

Листинг 8.10. Файл error.php

```
<?php
  echo "К сожалению, в Web-приложении произошла ошибка номер
        $_GET[id]. Пожалуйста, сообщите об этом администрации
        сайта по адресу site@site.ru.";
?>
```

8.4. Стек обработки исключительной ситуации

В предыдущем разделе демонстрировалось, как исключение, сгенерированное внутри функции или вложенного файла, может обрабатываться единственным обработчиком. При помощи функций `getFile()` и `getLine()` были получены имя файла и номер строки, в которых сгенерировано исключение. Однако файл при помощи инструкций `include`, `include_once`, `require` и `require_once` может включаться во множество других файлов проекта, поэтому локализация и воспроизведение ошибки не всегда возможны. В этой ситуации оказываются полезными функции `getTrace()` и `getTraceAsString()`, которые возвращают стек передачи исключения из файлов и функций в обработчик. Функция `getTrace()` возвращает стек в виде массива, а функция `getTraceAsString()` — в виде строки.

Перепишем вызов функции `print_user()` из листинга 8.8 так, чтобы в обработчике при помощи метода `getTraceAsString()` возвращался стек передачи исключительной ситуации (листинг 8.11).

Листинг 8.11. Использование функции `getTraceAsString()`

```
<?php
  try
  {
    // Подключаем реализацию функции print_user()
    require_once("function.print_user.php");
    // Вызываем функцию
    print_user();
  }
```

```
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение {$exp->getCode()} : {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
    echo "<pre>";
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

В случае указания неправильного имени базы данных catch-обработчик из листинга 8.11 может вывести следующий отчет:

```
Исключение 1 : Ошибка выбора базы данных
в файле D:\main\oop\08\config.php
в строке 24
#0 D:\main\oop\08\function.print_user.php(3): require_once()
#1 D:\main\oop\08\index.php(5): require_once('D:\main\oop\08\...')
#2 {main}
```

Если исключение произойдет на более высоком уровне вложенности, то количество пунктов в массиве, возвращаемом методом `getTraceAsString()`, возрастает. В случае использования функции `getTrace()` стек передачи исключения возвращается в виде многомерного массива:

```
Исключение 1 : Ошибка выбора базы данных
в файле D:\main\oop\08\config.php
в строке 24
Array
(
    [0] => Array
        (
            [file] => D:\main\oop\08\function.print_user.php
            [line] => 3
            [function] => require_once
        )
    [1] => Array
        (
            [file] => D:\main\oop\08\index.php
            [line] => 5
            [args] => Array
```

```
                (
                    [0] => D:\main\oop\08\function.print_user.php
                )
            [function] => require_once
        )
)
```

8.5. Генерация исключений в классах

Обработка внештатных ситуаций и ошибок при помощи исключений приобретает еще большее значение в случае классов. Если ошибка возникает внутри объекта, зачастую выводить сообщение в окно браузера невозможно: это может нарушить дизайн приложения, для этого придется дублировать систему ошибок приложения и т. п. Единственный правильный выход в подобной ситуации — переложить ответственность за обработку внештатных ситуаций внутри объекта на плечи внешнего разработчика.

В качестве примера рассмотрим класс `employee` ("сотрудник"), неоднократно рассматривавшийся в предыдущих главах, который реализует специальные методы `__get()` и `__set()` для обращений к закрытым членам. Напомним его упрощенную реализацию (листинг 8.12).

Листинг 8.12. Упрощенная реализация класса `employee`

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname    = $surname;
      $this->name       = $name;
      $this->patronymic = $patronymic;
      $this->age        = $age;
    }
    private function __get($index)
    {
      return $this->$index;
    }
    private function __set($index, $value)
    {
      if(isset($this->$index))
      {
          $this->$index = $value;
      }
    }
```

```
    private $surname;
    private $name;
    private $patronymic;
    private $age;
  }
?>
```

Класс содержит четыре закрытых члена: `$surname` ("фамилия"), `$name` ("имя"), `$patronymic` ("отчество") и `$age` ("возраст"). Значения закрытых членов устанавливаются при помощи конструктора класса и могут быть изменены при помощи метода `__set()`, который при этом проверяет существование запрашиваемого члена. Если член с таким именем существует — производится изменение его значения, если не существует — состояние объекта остается неизменным. Такая проверка необходима, т. к. обращение к несуществующему члену класса автоматически приводит к его созданию.

Обращение к несуществующему члену в рамках метода `__get()` не так критично, поскольку метод не изменяет состояние объекта.

Однако обращение к несуществующему члену как в методе `__set()` практически наверняка вызовет ошибку на стороне внешнего разработчика, поэтому крайне полезно сгенерировать исключение, которое сигнализировало бы внешнему разработчику об ошибке с его стороны (листинг 8.13).

Листинг 8.13. Генерация исключения при обращении к несуществующему члену

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname    = $surname;
      $this->name       = $name;
      $this->patronymic = $patronymic;
      $this->age        = $age;
    }
    private function __get($index)
    {
      if(isset($this->$index))
      {
        return $this->$index;
      }
      else new Exception("Член $index не существует");
    }
```

```
    private function __set($index, $value)
    {
      if(isset($this->$index))
      {
          $this->$index = $value;
      }
      else new Exception("Член $index не существует");
    }

    private $surname;
    private $name;
    private $patronymic;
    private $age;
  }
?>
```

Теперь достаточно поместить код, обращающийся к объектам класса `employee`, в контролируемый блок, чтобы предотвратить попытку обращения к несуществующим членам (листинг 8.14).

Листинг 8.14. Попытка обращения к несуществующему члену класса `employee`

```
<?php
  // Подключаем реализацию класса employee
  require_once("class.employee.php");

  try
  {
    // Объявляем объект класса employee
    $obj = new employee("Корнеев", "Иван", "Григорьевич");
    // Производим попытку изменения несуществующего
    // члена класса $var, что вызывает генерацию
    // исключения
    $obj->var = 100;
    // Остальные операторы
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение: {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
    echo "<pre>";
```

```
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

В листинге 8.14 производится попытка осуществить доступ к несуществующему члену `$obj`, в результате чего в методе `__set()` генерируется исключение. Перехват его в catch-блоке приводит к формированию отчета вида:

```
Исключение: Член var не существует
в файле D:\main\oop\08\class.employee.php
в строке 25
#0 D:\main\oop\08\index.php(12): employee->__set('var', 100)
#1 {main}
```

Важная особенность заключается в том, что оператор переходит в catch-блок сразу после возникновения исключительной ситуации, т. е. операторы, следующие за конструкцией `$obj->var`, выполнены не будут.

Как уже упоминалось, PHP-объекты существуют до конца работы скрипта, таким образом, объект `$obj` будет доступен в том числе и в catch-обработчике (листинг 8.15).

Замечание

В большинстве объектно-ориентированных языков программирования, реализующих механизм исключительных ситуаций, к моменту выполнения catch-обработчика объект, как правило, оказывается уже разрушенным. Объект исключения `Exception` необходим, чтобы передать в обработчик информацию о случившемся.

Листинг 8.15. Объект доступен в catch-обработчике

```
<?php
  // Подключаем реализацию класса employee
  require_once("class.employee.php");

  try
  {
    // Объявляем объект класса employee
    $obj = new employee("Корнеев", "Иван", "Григорьевич");
    // Производим попытку изменения несуществующего
    // члена класса $var, что вызывает генерацию
    // исключения
    $obj->var = 100;
  }
```

```
  catch(Exception $exp)
  {
    // Объект $obj все еще доступен
    echo "<pre>";
    print_r($obj);
    echo "</pre>";
  }
?>
```

Результатом работы скрипта из листинга 8.15 будет следующий дамп объекта:

```
employee Object
(
    [surname:private] => Корнеев
    [name:private] => Иван
    [patronymic:private] => Григорьевич
    [age:private] => 18
)
```

8.6. Генерация исключений в иерархиях классов

На первый взгляд, генерация исключений в базовых и производных классах не имеет никаких особенностей: вне зависимости от того, сгенерировано исключение в базовом или производном классе, обработчик имеет возможность его перехватить. Однако при отладке иерархии класса зачастую важно установить, где было сгенерировано исключение — в базовом или производном классе, т. к. классы могут располагаться в разных файлах.

Рассмотрим базовый класс `base`, который содержит единственный открытый метод `generate()`, задачей которого является генерация исключения (листинг 8.16).

Листинг 8.16. Базовый класс `base`

```
<?php
  class base
  {
    public function generate()
    {
      throw new Exception("Исключение базового класса");
    }
  }
?>
```

Перехват исключения, генерируемого методом `generate()`, может выглядеть так, как это представлено в листинге 8.17.

Листинг 8.17. Перехват исключения метода `generate()` базового класса

```
<?php
  require_once("class.base.php");

  try
  {
    $obj = new base();
    $obj->generate();
  }
  catch(Exception $exp)
  {
    echo "<pre>";
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

Результатом работы скрипта из листинга 8.17 будет следующий стек передачи исключения:

```
#0 D:\main\oop\08\index.php(7): base->generate()
#1 {main}
```

Пусть от базового класса `base` (который расположен в файле class.base.php) наследует производный класс `derived` (который, в свою очередь, располагается в файле class.derived.php) (листинг 8.18).

Листинг 8.18. Производный класс `derived`

```
<?php
  require_once("class.base.php");

  class derived extends base
  {
  }
?>
```

Класс `derived` не реализует никаких собственных методов, однако он наследует метод `generate()` базового класса `base` (листинг 8.19).

Листинг 8.19. Перехват исключения производного класса

```
<?php
  require_once("class.base.php");
  require_once("class.derived.php");

  try
  {
    $obj = new derived();
    $obj->generate();
  }
  catch(Exception $exp)
  {
    echo "<pre>";
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

Однако в стеке передачи исключения будет фигурировать не производный класс `derived`, а базовый класс `base`:

```
#0 D:\main\oop\08\index.php(8): base->generate()
#1 {main}
```

Это связано с тем, что метод `generate()` принадлежит базовому классу `base`. Однако если метод `generate()` перегружается в производном классе `derived`, пусть даже с вызовом метода производного класса, в стеке будет фигурировать производный класс `derived` (листинг 8.20).

Листинг 8.20. Перегрузка метода `generate()` в производном классе `derived`

```
<?php
  require_once("class.base.php");

  class derived extends base
  {
    public function generate()
    {
      parent::generate();
    }
  }
?>
```

Теперь вызов скрипта из листинга 8.19 будет приводить к следующему стеку передачи исключения:

```
#0 D:\main\oop\08\class.derived.php(8): base->generate()
#1 D:\main\oop\08\index.php(8): derived->generate()
#2 {main}
```

8.7. Использование объекта класса *Exception* в строковом контексте

Как видно из таблицы 8.2, класс `Exception` реализует метод `__toString()`, который позволяет вызывать объект в строковом контексте. В листинге 8.21 представлен переработанный скрипт из листинга 8.14, демонстрирующий обработку исключительной ситуации при попытке обратиться к несуществующему члену класса. При этом в обработчике выводится только переданный ему объект класса `Exception`.

Листинг 8.21. Использование объекта класса `Exception` в строковом контексте

```
<?php
  require_once("class.employee.php");

  try
  {
    $obj = new employee("Корнеев", "Иван", "Григорьевич");
    $obj->var = 100;
  }
  catch(Exception $exp)
  {
    echo "<pre>";
    echo $exp;
    echo "</pre>";
  }
?>
```

Результатом работы скрипта будет автоматически сгенерированный методом `__toString()` отчет, содержащий всю доступную об исключительной ситуации информацию:

```
exception 'Exception' with message 'Член var не существует' in
D:\main\oop\08\class.employee.php:25
Stack trace:
#0 D:\main\oop\08\index.php(7): employee->__set('var', 100)
#1 {main}
```

8.8. Создание собственных исключений

Класс `Exception` может выступать в качестве базового класса для пользовательских классов исключений. Создадим новый производный класс исключения `ExceptionSQL`, который бы сохранял всю информацию, полученную об исключительной ситуации, в MySQL-таблицу `exception`, содержащую следующие поля:

- `id_exception` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`;
- `message` — текстовое поле для сообщения, описывающего исключительную ситуацию;
- `code` — целочисленное поле для хранения кода, назначенного данному типу исключительных ситуаций;
- `file` — текстовое поле для имени файла, в котором произошла исключительная ситуация;
- `line` — целочисленное поле, хранящее номер строки, в которой произошла исключительная ситуация;
- `trace` — стек передачи исключительной ситуации, возвращаемый методом `getTraceAsString()`;
- `putdate` — календарный столбец для хранения даты и времени возникновения исключительной ситуации.

В листинге 8.22 представлен оператор `CREATE TABLE`, позволяющий создать таблицу `exception`.

Листинг 8.22. Таблица `exception` для хранения исключений

```
CREATE TABLE exception (
  id_exception INT(11) NOT NULL AUTO_INCREMENT,
  message TEXT NOT NULL,
  code INT(11) NOT NULL,
  file TINYTEXT NOT NULL,
  line INT(11) NOT NULL,
  trace TEXT NOT NULL,
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_exception)
);
```

Теперь можно унаследовать от класса `Exception` производный класс `ExceptionSQL`, который в конструкторе будет помещать в таблицу `exception` новую запись (листинг 8.23).

Листинг 8.23. Создание собственного типа исключений

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  class ExceptionSQL extends Exception
  {
    // Первичный ключ записи в таблице exception,
    // соответствующий текущей исключительной ситуации
    protected $id_exception;
    // Конструктор класса, инициализирующий
    // члены $message и $code — оба параметра
    // реализованы как необязательные
    public function __construct($message = null, $code = 0)
    {
      // Вызываем конструктор базового класса
      parent::__construct($message, $code);

      // Экранируем спецсимволы и приводим
      // числовые значения к целому типу
      $message = mysql_real_escape_string($this->message);
      $file    = mysql_real_escape_string($this->file);
      $trace   = mysql_real_escape_string($this->getTraceAsString());
      $code    = intval($this->code);
      $line    = intval($this->line);

      // Формируем и выполняем SQL-запрос на добавление
      // информации об исключительной ситуации в базу данных
      $query = "INSERT INTO exception
                VALUES (NULL,
                       '$message',
                        $code,
                       '$file',
                        $line,
                       '$trace',
                        NOW())";
      @mysql_query($query);

      // Извлекаем значение первичного ключа id_exception,
      // назначенного новой записи при помощи AUTO_INCREMENT
      $this->id_exception = mysql_insert_id();
    }
```

```
    // Возвращаем значение первичного ключа в таблице exception
    public function getID()
    {
      return $this->id_exception;
    }
  }
?>
```

Впрочем, поле `trace` вряд ли будет заполнено корректно, т. к. полный путь передачи исключения формируется лишь в catch-обработчике, а при создании объекта (когда вызывается конструктор) он остается незаполненным.

Помимо конструктора, класс `ExceptionSQL` содержит метод `getID()`, который позволяет получить значение закрытого члена `id_exception`, содержащего первичный ключ таблицы `exception` с информацией о текущей исключительной ситуации. В листинге 8.24 приводится пример обработки исключения `ExceptionSQL`. Следует обратить внимание на то, что в catch-обработчике для перехвата исключения указывается исключение типа `ExceptionSQL`.

Листинг 8.24. Обработка исключения `ExceptionSQL`

```
<?php
  // Подключаем реализацию класса
  require_once("class.ExceptionSQL.php");

  try
  {
    if(rand(0,1)) throw new ExceptionSQL("Случайное исключение");
  }
  catch(ExceptionSQL $exp)
  {
    // Получаем первичный ключ таблицы exception,
    // соответствующий исключению
    $id_exception = $exp->getID();
    // Извлекаем информацию об исключительной
    // ситуации
    $query = "SELECT * FROM exception
              WHERE id_exception = $id_exception";
    $ext = mysql_query($query);
    if(!$ext) exit("Ошибка извлечения исключения");
    $exception = mysql_fetch_array($ext);
    echo "<table border=1 cellpadding=5>";
    echo "<tr>
           <td>Сообщение:</td>
           <td>$exception[message]</td>
          </tr>";
```

```
    echo "<tr>
           <td>Код:</td>
           <td>$exception[code]</td>
          </tr>";
    echo "<tr>
           <td>Файл:</td>
           <td>$exception[file]</td>
          </tr>";
    echo "<tr>
           <td>Строка:</td>
           <td>$exception[line]</td>
          </tr>";
    echo "<tr>
           <td>Стек:</td>
           <td><pre>{$exception[trace]}</pre></td>
          </tr>";
    echo "<tr>
           <td>Время:</td>
           <td>$exception[putdate]</td>
          </tr>";
  }
?>
```

Результат работы скрипта из листинга 8.24 при условии, что функция `rand()` возвращает значение 1, представлен на рис. 8.1.

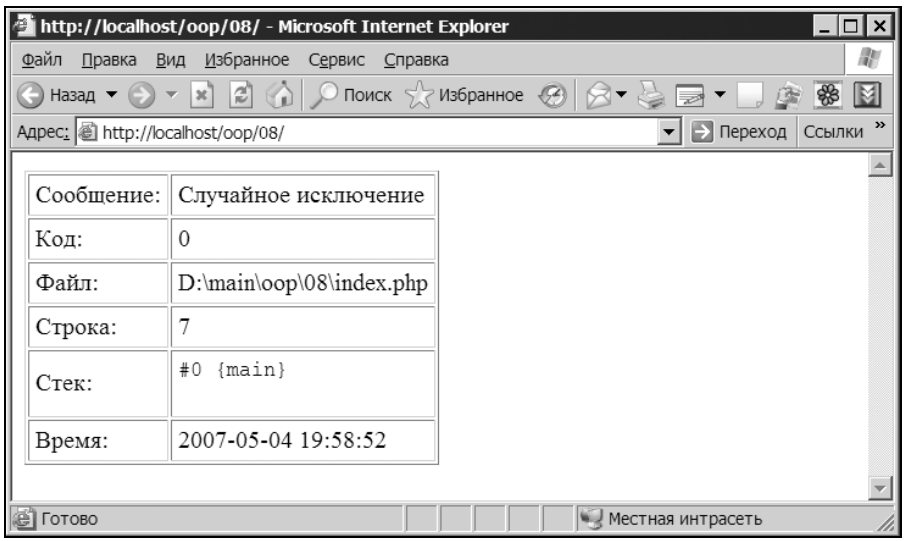


Рис. 8.1. Отчет скрипта из листинга 8.24 об исключительной ситуации

Помимо классической информации об исключительной ситуации (сообщение, код, имя файла, строка) появляется возможность получить точное время, когда произошло исключение. Можно модифицировать класс `ExceptionSQL` таким образом, чтобы сохранялась и другая ценная информация, например, IP-адрес посетителя (`$_SERVER['REMOTE_ADDR']`), адрес страницы, откуда посетитель пришел на данную страницу (`$_SERVER['HTTP_REFERER']`).

8.9. Создание новых типов исключений

Собственные классы исключений могут сами выступать в качестве базового класса для новых типов исключений. Например, от класса `ExceptionSQL` (см. *раздел 8.8*) можно унаследовать еще два класса:

- `ExceptionSQLNoneRecords` — исключительная ситуация, возникающая при отсутствии записей в таблице;
- `ExceptionSQLError` — исключительная ситуация, возникающая при возникновении ошибки обработки SQL-запроса.

Общая схема иерархии классов исключительных ситуаций представлена на рис. 8.2. Разумеется, в промышленных приложениях иерархия классов исключений может быть более сложной и разветвленной.

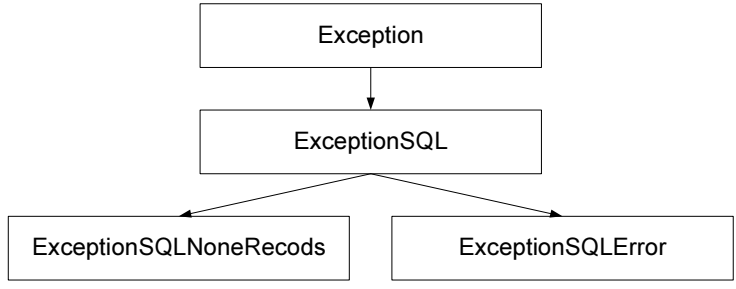


Рис. 8.2. Иерархия исключений

В листинге 8.25 представлена реализация класса `ExceptionSQLNoneRecords`. Класс перегружает конструктор, лишая его второго необязательного параметра `$code`, содержащего числовой код исключительной ситуации. Конструктору базового класса `ExceptionSQL` в качестве второго параметра передается константа 1001.

Листинг 8.25. Класс `ExceptionSQLNoneRecords`

```
<?php
  require_once("class.ExceptionSQL.php");
```

```
  class ExceptionSQLNoneRecords extends ExceptionSQL
  {
    public function __construct(
                          $message = "Запрашиваемые записи отсутствуют")
    {
      parent::__construct($message, 1001);
    }
  }
?>
```

В листинге 8.26 представлена реализация `ExceptionSQLError`, который, как и класс `ExceptionSQLNoneRecords`, перегружает конструктор, причем конструктор класса `ExceptionSQLError` не принимает ни одного параметра. Первый параметр базового класса `ExceptionSQL` формируется при помощи функции `mysql_error()`, а в качестве второго параметра ему передается константа 1002.

Листинг 8.26. Класс `ExceptionSQLError`

```
<?php
  require_once("class.ExceptionSQL.php");

  class ExceptionSQLError extends ExceptionSQL
  {
    public function __construct()
    {
      parent::__construct("Ошибка в SQL-запросе: ".mysql_error(), 1001);
    }
  }
?>
```

Для того чтобы обработать исключение, достаточно указать в catch-блоке его тип. Однако код в контролируемом блоке может генерировать большое количество разных типов исключений, для каждого из которых может потребоваться свой собственный catch-обработчик. В этом случае catch-обработчики располагаются друг под другом, как это продемонстрировано в листинге 8.27.

Листинг 8.27. Использование нескольких catch-обработчиков

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
```

```
    $query = "SELECT * FROM exception";
    $ext = mysql_query($query);
    // Если произошла ошибка — генерируем исключение
    // ExceptionSQLError()
    if(!$ext) throw new ExceptionSQLError();
    // Если в таблице exception отсутствуют
    // записи — генерируем исключение
    if(!mysql_num_rows($ext)) throw new ExceptionSQLNoneRecords();
    while($exception = mysql_fetch_array($ext))
    {
      echo $exception['message']."<br>";
    }
  }
  catch(ExceptionSQLError $exp)
  {
    // Ошибка в SQL-запросе
    echo "Произошла ошибка при выполнении SQL-запроса<br>";
    echo "{$exp->getMessage()}<br>";
  }
  catch(ExceptionSQLNoneRecords $exp)
  {
    // Отсутствуют записи
    echo "Таблица exception не содержит записей<br>";
  }
?>
```

Таким образом, для каждого из типов исключений можно предусмотреть свой собственный обработчик, количество которых не ограничено.

8.10. Перехват исключений производных классов

Исключения производных типов можно перехватывать при помощи исключений базового типа. В листинге 8.28 по случайному закону генерируется либо исключение `ExceptionSQLError`, либо исключение `ExceptionSQLNoneRecords`, однако для последнего отсутствует специализированный обработчик — его перехватывает обработчик `ExceptionSQL`.

Листинг 8.28. Перехват исключений производных классов

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");
```

```
  try
  {
    if(rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
  }
  catch(ExceptionSQLError $exp)
  {
    // Перехватываются ExceptionSQLError
    // исключения
    echo "ExceptionSQLError-исключение";
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются ExceptionSQL и
    // ExceptionSQLNoneRecords исключения
    echo "ExceptionSQL-исключение";
  }
?>
```

Если обработчик `ExceptionSQLError` отсутствует, оба типа исключений перехватываются обработчиком базового класса `ExceptionSQL` (листинг 8.29).

Листинг 8.29. Перехват исключений обработчиком базового класса

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    if(rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются ExceptionSQL, ExceptionSQLError и
    // ExceptionSQLNoneRecords исключения
    echo "ExceptionSQL-исключение ".get_class($exp);
  }
?>
```

В листинге 8.29 тип исключения в catch-обработчике определяется при помощи функции `get_class()` (см. *раздел 3.8*).

Важно помнить, что все обработчики производных классов должны быть расположены раньше обработчиков базовых классов, иначе исключения никогда не дойдут до специализированных обработчиков. В листинге 8.30 приводится скрипт, в котором никогда не срабатывает обработчик исключения `ExceptionSQLError`, т. к. исключение всегда перехватывается расположенным перед ним обработчиком исключений базового класса `ExceptionSQL`.

Листинг 8.30. Ошибочное расположение обработчиков

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    if(rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются ExceptionSQL, ExceptionSQLError и
    // ExceptionSQLNoneRecords исключения
    echo "ExceptionSQL-исключение ".get_class($exp);
  }
  catch(ExceptionSQLError $exp)
  {
    // Этот блок никогда не будет исполнен,
    // т. к. исключение перехватывается
    // предыдущим catch-обработчиком
    echo "ExceptionSQLError-исключение";
  }
?>
```

Замечание

Так как все исключения являются производными от базового класса `Exception`, то его использование позволяет перехватывать все исключения, не обработанные предыдущими обработчиками. Такая организация механизма исключительных ситуаций позволяет не вводить специальных ключевых слов вроде `__finally`, которые используются в Java.

8.11. Что происходит, когда исключения не перехватываются?

В сложных системах может сложиться ситуация, когда о некоторых возможных исключениях внешние разработчики, использующие класс, попросту не догадываются. В листинге 8.31 представлена простейшая ситуация генерации исключения без его обработки.

Листинг 8.31. Необработанное исключение

```
<?php
  throw new Exception();
?>
```

Результатом скрипта будут строки следующего вида:

```
Fatal error: Uncaught exception 'Exception' in D:\main\oop\08\index.php:2
Stack trace: #0 {main} thrown in D:\main\oop\08\index.php on line 2
```

Причем строки после **Fatal error: Uncaught** и до **thrown** формируются при помощи метода `__toString()`, в чем можно легко убедиться, перегрузив этот метод.

Листинг 8.32. Перегрузка метода `__toString()`

```
<?php
  class ExceptionSpecitial extends Exception
  {
    public function __toString()
    {
      return "Исключение ExceptionSpecitial оказалось не обработанным";
    }
  }

  throw new ExceptionSpecitial();
?>
```

Результатом работы скрипта из листинга 8.32 будут строки вида:

```
Fatal error: Uncaught Исключение ExceptionSpecitial оказалось
не обработанным thrown in D:\main\oop\08\index.php on line 10
```

Таким образом, изменить реакцию на необработанное исключение при помощи перегрузки метода `__toString()` не удастся, т. к. он предназначен исключительно для штатного использования. Тем более, разработчик класса

вряд ли сможет предположить, какая реакция приложения на необработанное исключение будет удобна внешнему разработчику.

В связи с этим в PHP, начиная с версии 5.0, введена функция `set_exception_handler()`, которая позволяет переопределить обработчик для необработанных исключений. В качестве единственного параметра она принимает строку с именем функции обратного вызова, которое может быть произвольным, не совпадающим с именами уже существующих функций. Функция обратного вызова должна быть подготовлена внешним разработчиком и принимать единственный параметр — объект исключения. В листинге 8.33 приводится пример переопределения обработчика необработанных исключений.

Листинг 8.33. Использование функции `set_exception_handler()`

```
<?php
  function exception_handler($exp)
  {
    echo "Не обработано сообщение ".get_class($exp);
  }

  set_exception_handler("exception_handler");

  throw new Exception();

  echo "Это строка никогда не будет выведена";
?>
```

Результатом работы скрипта из листинга 8.33 будет предупреждение: **Не обработано сообщение Exception**. Важно отметить, что при генерации исключения интерпретатор PHP покидает исполняемый модуль, и оставшаяся часть кода не выполняется.

8.12. Вложенные контролируемые блоки

Контролируемые блоки могут быть вложены друг в друга. Например, листинг 8.28 может быть переписан следующим образом (листинг 8.34).

Листинг 8.34. Вложенные контролируемые блоки

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");
```

```
  try
  {
    try
    {
      if(rand(0,1)) throw new ExceptionSQLError();
      else throw new ExceptionSQLNoneRecords();
    }
    catch(ExceptionSQLError $exp)
    {
      // Перехватываются ExceptionSQLError
      // исключения
      echo "ExceptionSQLError-исключение";
    }
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются ExceptionSQL и
    // ExceptionSQLNoneRecords исключения
    echo "ExceptionSQL-исключение";
  }
?>
```

Исключения, которые не перехватываются внутренними контролирующими блоками, передаются во внешние контролирующие блоки. Если исключение не было обработано ни одним из обработчиков, то оно считается не обработанным и передается функции обратного вызова, назначаемой функцией `set_exception_handler()` (см. *раздел 8.11*).

Разумеется, вложенные контролируемые блоки редко используются в контексте, как это сделано в листинге 8.34, где более уместным было бы определить один контролируемый блок с несколькими catch-обработчиками. Вложенные контролируемые блоки используются в ситуациях, когда класс или функция часть исключений обрабатывают сами, а часть передают внешнему коду.

Пусть имеется файл generator.php, который по случайному закону генерирует одно из трех исключений: `ExceptionSQL`, `ExceptionSQLError` и `ExceptionSQLNoneRecords` (листинг 8.35).

Листинг 8.35. Файл generator.php

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");
```

```
  switch(rand(0,2))
  {
    case 0:
      throw new ExceptionSQL();
    case 1:
      throw new ExceptionSQLError();
    case 2:
      throw new ExceptionSQLNoneRecords();
  }
?>
```

Как видно из листинга 8.35, в case-блоках оператора `switch` не используются традиционные операторы `break`, позволяющие досрочно выйти из `switch`. Генерация исключения приводит к тому, что PHP-интерпретатор прекращает выполнение оставшейся части кода и покидает файл generator.php.

В листинге 8.36 представлена функция `catch_exception()`, которая обрабатывает два вида исключений: `ExceptionSQLError` и `ExceptionSQLNoneRecords`.

Замечание

Вместо отдельной функции `catch_exception()` может выступать метод класса.

Листинг 8.36. Функция обработки исключений

```
<?php
  function catch_exception()
  {
    try
    {
      require_once("generate.php");
    }
    catch(ExceptionSQLError $exp)
    {
      echo "ExceptionSQLError-исключение";
    }
    catch(ExceptionSQLNoneRecords $exp)
    {
      echo "ExceptionSQLNoneRecords-исключение";
    }
  }
?>
```

Следует отметить, что, если файл generate.php генерирует исключение ExceptionSQL(), функция catch_exception() не обрабатывает его самостоятельно, а передает во внешний код, который обладает такой возможностью (листинг 8.37).

Листинг 8.37. Перехват исключения ExceptionSQL

```
<?php
  require_once("function.catch_exception.php");

  try
  {
    catch_exception();
  }
  catch(ExceptionSQL $exp)
  {
    echo "ExceptionSQL-исключение";
  }
?>
```

Таким образом, можно организовывать каскады обработки исключений, вкладывая контролируемые блоки друг в друга. Разумеется, обработчики исключений производных классов должны располагаться на более глубоком уровне, чем обработчики исключений базовых классов.

8.13. Повторная генерация исключений

Исключение, перехваченное одним catch-обработчиком, может быть регенерировано для передачи его следующему по каскаду обработчику. Это позволяет нескольким обработчикам получить доступ к одному и тому же исключению. Для повторной генерации исключения достаточно вызвать оператор throw в catch-блоке, передав ему значение объекта исключения.

В листинге 8.38 обработчик исключений базового класса ExceptionSQL предшествует обработчикам исключений производных типов ExceptionSQLError и ExceptionSQLNoneRecords.

Листинг 8.38. Повторная генерация исключений

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");
```

```
  try
  {
    try
    {
      if(rand(0,1)) throw new ExceptionSQLError();
      else throw new ExceptionSQLNoneRecords();
    }
    catch(ExceptionSQL $exp)
    {
      echo "ExceptionSQL-исключение ".get_class($exp)."<br>";
      // Передача исключения далее по каскаду
      throw $exp;
    }
  }
  catch(ExceptionSQLError $exp)
  {
    echo "ExceptionSQLError-исключение";
  }
  catch(ExceptionSQLNoneRecords $exp)
  {
    echo "ExceptionNoneRecords-исключение";
  }
?>
```

В зависимости от сгенерированного исключения, скрипт из листинга 8.38 возвратит либо эти строки:

```
ExceptionSQL-исключение ExceptionSQLError
ExceptionSQLError-исключение
```

либо такие:

```
ExceptionSQL-исключение ExceptionSQLNoneRecords
ExceptionNoneRecords-исключение
```

Таким образом, какое бы из исключений производного класса не было бы сгенерировано, catch-обработчик базового класса в любом случае будет выполнен.

Важно отметить, что передать заново сгенерированное исключение можно только внешнему контролирующему блоку. Повторная генерация исключения в рамках одного контролирующего блока с несколькими catch-обработчиками не вызовет перехода к следующим catch-обработчикам. В листинге 8.39 повторно сгенерированное исключение не будет обработано ни одним из обработчиков.

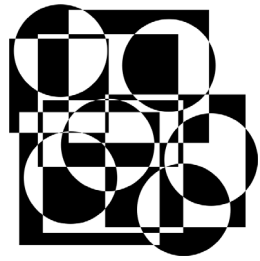
Листинг 8.39. Повторно сгенерированное исключение останется не обработанным

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    if(rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
  }
  catch(ExceptionSQL $exp)
  {
    echo "ExceptionSQL-исключение ".get_class($exp)."<br>";
    // Передача исключения далее по каскаду
    throw $exp;
  }
  catch(ExceptionSQLError $exp)
  {
    echo "ExceptionSQLError-исключение";
  }
  catch(ExceptionSQLNoneRecords $exp)
  {
    echo "ExceptionNoneRecords-исключение";
  }
?>
```

Результат работы скрипта из листинга 8.39 может выглядеть следующим образом:

```
ExceptionSQL-исключение ExceptionSQLError
Fatal error: Uncaught exception 'ExceptionSQLError' with message 'Ошибка
в SQL-запросе: ' in D:\main\oop\08\index.php:7 Stack trace: #0 {main}
thrown in D:\main\oop\08\index.php on line 7
```

ГЛАВА 9



Отражения

Механизм отражений предоставляет разработчику возможность исследования как пользовательских, так предопределенных классов, выясняя статус и состав отдельных членов, методов, классов и даже расширений PHP, а также объявления объектов классов и выполнения над ними манипуляций. Кроме этого, отражения позволяют автоматически генерировать документацию иерархии классов по схеме javadoc, применяемой в технологии Java.

9.1. Иерархия классов отражения

Механизм отражений включает иерархию классов, каждый из которых (кроме класса `Reflection`) реализует интерфейс `Reflector`. Интерфейс `Reflector` не требует реализации никаких методов и предназначен лишь для объединения классов отражения в одну группу таким образом, чтобы проверка объекта при помощи оператора `instanceof` на принадлежность интерфейсу `Reflector` возвращала значение `true`.

Замечание

Класс механизма отражения относится к так называемым *предопределенным классам*, т. е. классам, которые реализованы в PHP-интерпретаторе и не требуют реализации со стороны программиста. Другие предопределенные классы более подробно рассматриваются в *приложении 1*.

В табл. 9.1 представлен список классов механизма отражения и их краткое описание.

Некоторые классы данной иерархии базовые, другие — производные; так, класс `ReflectionException` является производным класса `Exception`, класс `ReflectionMethod` наследует от `ReflectionFunction`, а `ReflectionObject` — от `ReflectionClass` (рис. 9.1).

Таблица 9.1. *Список классов механизма отражения*

Класс	Описание
`Reflection`	Содержит вспомогательные статические методы для обслуживания объектов классов механизма отражения
`ReflectionException`	Класс исключений для обработки исключительных ситуаций (см. *главу 8*), возникающих при работе с отражениями
`ReflectionFunction`	Предназначен для представления (отражения) функции
`ReflectionParameter`	Обеспечивает представление (отражение) параметра функции
`ReflectionMethod`	Выполняет представление (отражение) метода класса
`ReflectionClass`	Предназначен для представления (отражения) класса
`ReflectionObject`	Служит для представления (отражения) объекта
`ReflectionProperty`	Разработан для представления (отражения) члена класса
`ReflectionExtension`	Предназначен для представления (отражения) расширения PHP

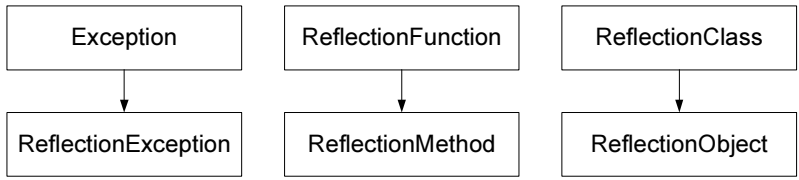


Рис. 9.1. Отношение наследования среди классов механизма отражения

9.2. Отражение функции. Класс *ReflectionFunction*

Объект класса `ReflectionFunction` содержит информацию о функции, имя которой передается ее конструктору в качестве параметра. При помощи методов объекта можно получить самую разнообразную информацию об этой функции, начиная с имени и заканчивая количеством и типом параметров, которые функция может принимать.

В листинге 9.1 представлена демонстрационная функция `hello()`, принимающая единственный параметр `$name`, который подставляется в выводимую ею приветственную строку.

Листинг 9.1. Демонстрационная функция `hello()`

```
<?php
  function hello($name)
  {
     echo "Hello, $name!";
  }
?>
```

Для того чтобы объявить отражение функции `hello()`, достаточно создать объект класса `ReflectionFunction`, инициированный строкой `"hello"` (листинг 9.2).

Листинг 9.2. Создание отражения функции `hello()`

```
<?php
  require_once("function.hello.php");

  // Получаем отражение функции
  $obj = new ReflectionFunction("hello");
  // Выводим название функции
  echo "{$obj->getName()}<br>";
  // Выполняем функцию, передавая в качестве аргумента
  // строку "world"
  $obj->invoke("world");
?>
```

Скрипт из листинга 9.2 создает объект отражения `$obj` для функции `hello()`. При помощи полученного объекта производится вывод названия функции, а также ее выполнение. Результатом работы скрипта из листинга 9.2 будут следующие строки:

```
hello
Hello, world!
```

В табл. 9.2 представлен полный список методов класса `ReflectionFunction`, позволяющий получать самую разнообразную информацию о функции.

Таблица 9.2. *Список методов класса* `ReflectionFunction`

Метод	Описание
`object __construct($name)`	Конструктор класса, принимающий в качестве единственного параметра `$name` имя функции

Таблица 9.2 *(продолжение)*

Метод	Описание
`string __toString()`	Выводит строковое представление объекта при его интерполяции
`static string export($name, $return)`	Аналогичен функции `__toString()`: возвращает информацию о функции `$name` в виде строки или, если необязательный параметр `$retrun` равен `false`, выводит ее в окно браузера. В отличие от метода `__toString()`, метод `export()` является статическим и может вызываться без объявления объекта класса, например, `ReflectionFunction::export('hello')`
`string getName()`	Возвращает название метода
`bool isInternal()`	Возвращает `true` для предопределенных функций (входящих в ядро PHP) и `false` — для пользовательских
`bool isUserDefined()`	Возвращает `true` для пользовательских функций и `false` — для предопределенных (входящих в ядро PHP)
`string getFileName()`	Возвращает полный путь к файлу, в котором определена функция
`int getStartLine()`	Возвращает номер строки в файле, соответствующий началу функции
`int getEndLine()`	Возвращает номер строки в файле, соответствующий концу функции
`string getDocComment()`	Возвращает содержимое многострочного комментария, расположенного непосредственно перед функцией
`array getStaticVariables()`	Возвращает ассоциативный массив всех статических переменных функции и их текущие значения: имена переменных служат ключами массива, значения переменных — значениями элементов
`mixed invoke($arg1, $arg2 ...)`	Выполняет функцию, передавая ей в качестве аргументов параметры `$arg1`, `$arg2` и т. д.
`mixed invokeArgs($arr)`	Выполняет функцию, однако, в отличие от метода `invoke()`, параметры передаются в виде массива `$arr`
`bool returnsReference()`	Возвращает `true`, если функция возвращает результат по ссылке, и `false` — в противном случае

***Таблица 9.2** (окончание)*

Метод	Описание
`ReflectionParameter[] getParameters()`	Возвращает массив объектов класса `ReflectionParameter` (см. *раздел 9.3*), представляющих собой отражение параметров функции
`int getNumberOfParameters()`	Возвращает количество параметров функции
`int getNumberOfRequiredParameters()`	Возвращает количество обязательных параметров функции

Класс `ReflectionFunction` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 9.3.

Листинг 9.3. Гипотетическая реализация класса `ReflectionFunction`

```
<?php
  class ReflectionFunction implements Reflector
  {
    final private function __clone();
    public function __construct($name);
    public function __toString();
    public function static export();
    public function getName();
    public function isInternal();
    public function isUserDefined();
    public function getFileName();
    public function getStartLine();
    public function getEndLine();
    public function getDocComment();
    public function getStaticVariables();
    public function invoke($args, ...);
    public function invokeArgs($arr);
    public function returnsReference();
    public function getParameters();
    public function getNumberOfParameters();
    public function getNumberOfRequiredParameters();
  }
?>
```

Одним из самых интересных методов является метод `getDocComment()`, который позволяет восстанавливать многострочный комментарий перед функ-

цией, предоставляя разработчикам аналог javadoc из технологии Java для построения автоматической документации к системе. Модифицируем файл function.hello.php с функцией `hello()` так, как это представлено в листинге 9.4.

Замечание

Важным условием распознавания комментария методом `getDocComment()` является начало комментария с последовательности `/**`.

Листинг 9.4. Многострочный комментарий перед функцией `hello()`

```
<?php
  /** Метод hello() принимает единственный параметр
    $name и выводит в окно браузера приветствие
    "Hello, $name!"
  */

  function hello($name)
  {
     echo "Hello, $name!";
  }
?>
```

Теперь, обращаясь к методу `getDocComment()` объекта отражения функции `hello()`, можно получить содержимое комментария, предваряющего функцию в файле function.hello.php (листинг 9.5).

Листинг 9.5. Использование метода `getDocComment()` класса `ReflectionFunction`

```
<?php
  require_once("function.hello.php");

  // Получаем отражение функции
  $obj = new ReflectionFunction("hello");
  // Выводим комментарий к функции
  echo "<pre>";
  echo $obj->getDocComment();
  echo "</pre>";
  // Выводим название файла и номера строк,
  // с которых функция начинается и заканчивается
  echo "Файл: {$obj->getFileName()}<br>";
  echo "Строки с {$obj->getStartLine()}
        по {$obj->getEndLine()}<br>";
?>
```

Результат выполнения скрипта из листинга 9.5 представлен на рис. 9.2.

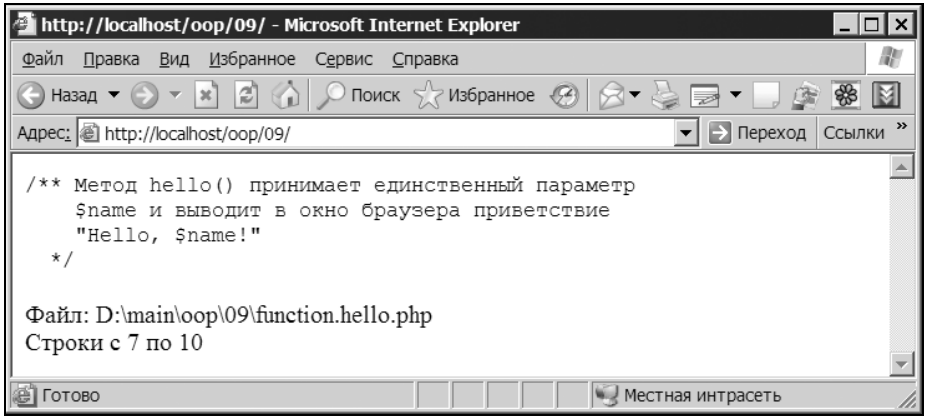


Рис. 9.2. Результат выполнения скрипта из листинга 9.5

9.3. Отражение параметра функции. Класс *ReflectionParameter*

Объект класса `ReflectionParameter` содержит информацию о параметре функции, имя которого передается конструктору в качестве параметра. В результате методы объекта могут вернуть о параметре разнообразную информацию. Полный список методов класса представлен в табл. 9.3.

Таблица 9.3. *Методы класса* `ReflectionParameter`

Метод	Описание
`object __construct($function, $parameter)`	Конструктор класса, создающий отражение для параметра `$paramter` функции `$function`. Можно задавать как название параметра `$parameter`, так и его порядковый номер (нумерация начинается с 0)
`string __toString()`	Выводит строковое представление объекта при его интерполяции
`static string export($function, $parameter, $return)`	Аналогичен функции `__toString()`: возвращает информацию о параметре `$parameter` функции `$function` в виде строки или, если необязательный параметр `$retrun` равен `false`, выводит ее в окно браузера. В отличие от метода `__toString()`, метод `export()` является статическим и может вызываться без объявления объекта класса, например, `ReflectionParameter::export('hello', 0)`
`string getName()`	Возвращает название параметра

Таблица 9.3 *(окончание)*

Метод	Описание
`ReflectionFunction getDeclaringFunction()`	Возвращает объект класса `ReflectionFunction` для функции, которой принадлежит текущий параметр
`ReflectionClass getDeclaringClass()`	Возвращает объект класса `ReflectionClass` (см. *раздел 9.4*), которому принадлежит метод, содержащий текущий параметр
`ReflectionClass getClass()`	Аналог метода `getDeclaringClass()`
`bool isArray()`	Возвращает `true`, если параметр представляет собой массив, и `false` — в противном случае
`bool allowsNull()`	Возвращает `true`, если параметр может принимать значение NULL или является необязательным, и `false` — в противном случае
`bool isPassedByReference()`	Возвращает значение `true`, если параметр передается по ссылке, и `false` — в противном случае
`bool isOptional()`	Возвращает `true`, если параметр является необязательным, и `false` — в противном случае
`bool isDefaultValueAvailable()`	Возвращает `true`, если у параметра есть значение по умолчанию, и `false` — в противном случае
`mixed getDefaultValue()`	Возвращает значение по умолчанию, если оно предусмотрено для параметра

Класс `ReflectionParameter` — предопределенный и, следовательно, не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 9.6.

Листинг 9.6. Гипотетическая реализация класса `ReflectionParameter`

```
<?php
class ReflectionParameter implements Reflector
{
    final private function __clone();
    public function __construct($function, $parameter);
    public function __toString();
    public static function export($function, $parameter, $return);
    public function getName();
    public function getDeclaringFunction();
    public function getDeclaringClass();
    public function getClass();
```

```
    public function isArray();
    public function allowsNull();
    public function isPassedByReference();
    public function isOptional();
    public function isDefaultValueAvailable();
    public function getDefaultValue();
}
?>
```

В листинге 9.7 приводится пример создания отражения для единственного параметра функции `hello()` (листинг 9.4).

Листинг 9.7. Создание отражение для параметра функции `hello()`

```
<?php
  require_once("function.hello.php");

  $obj = new ReflectionParameter("hello", 0);
  // Выводим название параметра
  echo $obj->getName();
?>
```

Однако способ создания отражения параметра, представленный в листинге 9.7, практически не используется. В подавляющем большинстве случаев методом `getParameters()` объекта класса `ReflectionFunction` возвращается массив объектов класса `ReflectionParameter`. Такой массив, как правило, обходится при помощи цикла, пример которого приводится в листинге 9.8.

Листинг 9.8. Получение и обработка массива отражений параметров функции

```
<?php
  require_once("function.hello.php");

  // Получаем отражение функции
  $obj = new ReflectionFunction("hello");
  // Получаем массив объектов отражений
  // параметров функции
  $arr = $obj->getParameters();
  // Обрабатываем отражения
  // параметров в цикле
  foreach($arr as $par)
  {
    // Выводим название очередного
    // параметра
```

```
    echo $par->getName()."<br>";
  }
?>
```

9.4. Отражение класса. Класс *ReflectionClass*

Объект класса `ReflectionClass` содержит информацию о классе, имя которого задается в качестве параметра конструктора. При помощи методов данного объекта можно вернуть о классе разнообразную информацию, а ряд его специальных методов позволяет получить доступ к массиву отражений методов и членов класса. Полный список методов класса `ReflectionClass` представлен в табл. 9.4.

Таблица 9.4. *Методы класса* `ReflectionClass`

Метод	Описание
`object __construct($name)`	Конструктор класса, позволяющий создавать отражение для класса с именем `$name`
`string __toString()`	Вывод строкового представления объекта при его интерполяции
`static string export($class, $return)`	Аналогичен функции `__toString()`: возвращает информацию о классе `$class` в виде строки или , если необязательный параметр `$retrun` равен `false`, выводит ее в окно браузера. В отличие от метода `__toString()`, метод `export()` является статическим и может вызываться без объявления объекта класса, например, `ReflectionClass::export('class_name')`
`string getName()`	Возвращает имя класса
`bool isInternal()`	Возвращает `true` для предопределенных классов (входящих в ядро PHP) и `false` — для пользовательских
`bool isUserDefined()`	Возвращает `true` для пользовательских классов и `false` — для предопределенных (входящих в ядро PHP)
`bool isInstantiable()`	Возвращает `true`, если класс не является абстрактным (возможно создание объекта данного класса), и `false` — в противном случае
`bool isAbstract()`	Возвращает `true`, если класс является абстрактным (невозможно создание объекта данного класса), и `false` — в противном случае

Таблица 9.4 *(продолжение)*

Метод	Описание
`bool isInterface()`	Возвращает `true`, если класс является интерфейсом, и `false` — в противном случае
`bool isFinal()`	Возвращает `true`, если класс не допускает наследования (создание производных классов), и `false` — в противном случае
`bool isIterateable()`	Возвращает `true`, если класс может быть использован в качестве массива в цикле `foreach`, и `false` — в противном случае возвращается
`bool isInstance($object)`	Возвращает `true`, если объект `$object` является экземпляром класса отражения, и `false` — в противном случае. Метод аналогичен оператору `instanceof`
`bool isSubclassOf($class)`	Возвращает `true`, если класс с именем `$class` является базовым для текущего класса, и `false` — в противном случае
`bool hasConstant($name)`	Возвращает `true`, если класс содержит константу с именем `$name`, и `false` — в противном случае
`bool hasMethod($name)`	Возвращает `true`, если класс содержит метод с именем `$name`, и `false` — в противном случае
`bool hasProperty($name)`	Возвращает `true`, если класс содержит член с именем `$name`, и `false` — в противном случае
`bool implementsInterface($name)`	Возвращает `true`, если класс реализует интерфейс с именем `$name`, и `false` — в противном случае
`string getFileName()`	Возвращает полный путь к файлу, в котором определен класс
`int getStartLine()`	Возвращает номер строки в файле, с которой начинается описание класса
`int getEndLine()`	Возвращает номер строки в файле, которая завершает описание класса
`string getDocComment()`	Возвращает содержимое многострочного комментария, расположенного непосредственно перед описанием класса
`ReflectionMethod getConstructor()`	Возвращает отражение (объект класса `ReflectionMethod`) для конструктора класса
`ReflectionMethod getMethod($name)`	Возвращает отражение (объект класса `ReflectionMethod`) для метода с именем `$name`

Таблица 9.4 *(продолжение)*

Метод	Описание
`ReflectionMethod[] getMethods()`	Возвращает массив отражений для методов класса
`ReflectionProperty getProperty($name)`	Возвращает отражение (объект класса `ReflectionProperty`) для члена класса с именем `$name`
`ReflectionProperty[] getProperties()`	Возвращает массив отражений для членов класса
`array getConstants()`	Возвращает ассоциативный массив констант класса: в качестве ключей выступают названия констант, а в качестве значений элементов — значения констант
`mixed getConstant($name)`	Возвращает значение константы с именем `$name`
`ReflectionClass[] getInterfaces()`	Возвращает массив отражений для интерфейсов, которые реализует класс
`int getModifiers()`	Возвращает битовую маску, соответствующую атрибуту класса, например, 00100000 (32) — абстрактный (`abstract`) класс, 01000000 (64) — константный (`final`) класс, 10010000 (144) — интерфейс (`interface`)
`stdclass newInstance($arg1, $arg2, ...)`	Создает объект класса, используя в качестве параметров конструктора аргументы `$arg1`, `$arg2` и т. д.
`stdclass newInstanceArgs($arr)`	Создает объект класса, используя в качестве параметров конструктора элементы массива `$arr`
`ReflectionClass getParentClass()`	Возвращает отражение (объект класса `ReflectionClass`) базового класса
`array getStaticProperties()`	Возвращает ассоциативный массив статических членов класса: в качестве ключей выступают названия членов, а в качестве значений элементов — значения статических членов
`mixed getStaticPropertyValue($name [, $default])`	Возвращает значение статической переменной с именем `$name`
`void setStaticPropertyValue($name, $value)`	Позволяет передать статической переменной `$name` новое значение `$value`
`array getDefaultProperties()`	Возвращает значения членов класса по умолчанию
`ReflectionExtension getExtension()`	Возвращает отражение расширения PHP (объект класса `ReflectionExtension`), которому принадлежит текущий класс

Таблица 9.4 *(окончание)*

Метод	Описание
`string getExtensionName()`	Возвращает имя расширения PHP, которому принадлежит текущий класс

Класс `ReflectionClass` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 9.9.

Листинг 9.9. Гипотетическая реализация класса `ReflectionClass`

```
<?php
  class ReflectionClass implements Reflector
  {
    final private function __clone();
    public function __construct($name);
    public function __toString();
    public static function export($class, $return);
    public function getName();
    public function isInternal();
    public function isUserDefined();
    public function isInstantiable();
    public function isAbstract();
    public function isInterface();
    public function isFinal();
    public function isInstance($object);
    public function isSubclassOf($class);
    public function hasConstant($name);
    public function hasMethod($name);
    public function hasProperty($name);
    public function getFileName();
    public function getStartLine();
    public function getEndLine();
    public function getDocComment();
    public function getConstructor();
    public function getMethod($name);
    public function getMethods();
    public function getProperty($name);
    public function getProperties();
    public function getConstants();
    public function getConstant($name);
```

```
    public function getInterfaces();
    public function getModifiers();
    public function newInstance($args1, ...);
    public function newInstanceArgs($arr);
    public function getParentClass();
    public function getStaticProperties();
    public function getStaticPropertyValue($name [, $default]);
    public function setStaticPropertyValue($name, $value);
    public function getDefaultProperties();
    public function isIterateable();
    public function implementsInterface($name);
    public function getExtension();
    public function getExtensionName();
  }
?>
```

Для демонстрации приемов работы с отражением класса создадим демонстрационный класс `example` (листинг 9.10).

Листинг 9.10. Демонстрационный класс `example`

```
<?php
  /** Тестовый класс для демонстрации
      возможностей отражения класса. */
  class example
  {
    const NAME = "cls";

    private $first;
    public $second;
    private $third;

    static $count = 0;

    public function __construct($first,
                                $second,
                                $third)
    {
      $this->first  = $first;
      $this->second = $second;
      $this->third  = $third;
    }
```

```
    public function getFirst()
    {
      return $this->first;
    }

    public function helloWorld()
    {
      return "Hello world";
    }
  }
?>
```

В листинге 9.11 приводится пример получения отражения класса `example`. Отражение конструктора получается при помощи метода `getConstructor()`, а метод `getParameters()` — предоставляет массив отображений параметров. В результате работы цикла скрипт выводит список параметров конструктора.

Листинг 9.11. Получение списка параметров конструктора

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Получаем массив отражений параметров конструктора
  $arr = $obj->getConstructor()->getParameters();

  foreach($arr as $par)
  {
    echo $par->getName()."<br>";
  }
?>
```

Листинг 9.12 предоставляет сведения о файле, в котором определен класс, а также номера начальной и конечной строк описания класса.

Замечание

При помощи соответствующего представления `ReflectionMethod` можно получить также многострочные комментарии, предваряющие методы класса.

Листинг 9.12. Получение сведений о локализации класса

```
<?php
  require_once("class.example.php");
```

```
  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Выводим комментарий к классу
  echo "<pre>";
  echo $obj->getDocComment();
  echo "</pre>";
  // Выводим название файла и номера строк,
  // с которых начинается и заканчивается описание класса
  echo "Файл: {$obj->getFileName()}<br>";
  echo "Строки с {$obj->getStartLine()}
        по {$obj->getEndLine()}<br>";
?>
```

При помощи метода `newInstance()` допускается неявное создание объекта класса. В листинге 9.13 демонстрируется создание объекта класса `example` и вызов его метода `getFirst()`.

Листинг 9.13. Неявное создание объекта класса `example`

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Неявное создание объекта класса example
  $o = $obj->newInstance(1, 2, 3);

  // Вызов метода объекта класса example
  echo $o->getFirst();
?>
```

При помощи набора методов `getStaticProperties()`, `setStaticPropertyValue()` и `getStaticPropertyValue()` можно управлять статическими членами класса (листинг 9.14).

Листинг 9.14. Управление статическим членом `count` класса `example`

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");
```

```
  // Выводим список статических членов класса
  echo "<pre>";
  print_r($obj->getStaticProperties());
  echo "</pre>";

  // Присваиваем новое значение статическому
  // члену класса count
  $obj->setStaticPropertyValue("count", 100);

  // Выводим значение статического члена класса
  // count
  echo $obj->getStaticPropertyValue("count");

?>
```

9.5. Отражение объекта. Класс *ReflectionObject*

Класс `ReflectionClass`, рассмотренный в предыдущем разделе, позволяет создать отражение произвольного класса, однако в иерархии классов механизма отражения предусмотрен дополнительный класс `ReflectionObject`, позволяющий создать отражение объекта. Класс `ReflectionObject` является производным класса `ReflectionClass` и переопределяет только конструктор и статический метод `export()`. Если бы класс `ReflectionObject` не был предопределенным, то его описание могло бы выглядеть так, как это представлено в листинге 9.15.

Листинг 9.15. Гипотетическая реализация класса `ReflectionObject`

```
<?php
  class ReflectionObject extends ReflectionClass
  {
    public function __construct($object);
    public static function export($object, $return);
  }
?>
```

Остальные методы класс `ReflectionObject` наследует от класса `ReflectionClass` (см. табл. 9.4).

9.6. Отражение метода класса. Класс *ReflectionMethod*

Класс `ReflectionMethod` позволяет создать отражение метода класса. Сам класс `ReflectionMethod` большинство методов наследует от базового класса `ReflectionFunction` (см. *раздел 9.2*). Если бы класс `ReflectionMethod` не был предопределенным, то его описание могло бы выглядеть так, как это представлено в листинге 9.16.

Листинг 9.16. Гипотетическая реализация класса `ReflectionMethod`

```
<?php
  class ReflectionMethod extends ReflectionFunction
  {
    public function __construct($class, $name);
    public function __toString();
    public static function export($class, $name, $return);
    public function invoke($object, $args);
    public function invokeArgs($object, $arr);
    public function isFinal();
    public function isAbstract();
    public function isPublic();
    public function isPrivate();
    public function isProtected();
    public function isStatic();
    public function isConstructor();
    public function isDestructor();
    public function getModifiers();
    public function getDeclaringClass();
  }
?>
```

В табл. 9.5 представлено описание методов класса `ReflectionMethod`, которые класс реализует помимо наследуемых от класса `ReflectionFunction`.

***Таблица 9.5.** Методы класса `ReflectionMethod`*

Метод	Описание
`__construct($class, $name)`	Конструктор класса, создающий отражение метода `$name` или объекта `$class`
`string __toString()`	Вывод строкового представления объекта при его интерполяции

Таблица 9.5 *(окончание)*

Метод	Описание
`static string export($class, $name, $return)`	Аналогичен функции `__toString()`: возвращает информацию о методе `$name` класса `$class` в виде строки или, если необязательный параметр `$retrun` равен `false`, выводит ее в окно браузера. В отличие от метода `__toString()`, метод `export()` является статическим и может вызываться без объявления объекта класса, например, `ReflectionMethod::export('example', 'getFirst')`
`mixed invoke($object, $arg1, $arg2, ...)`	Выполняет метод объекта `$object` с параметрами `$arg1`, `$arg2` и т. д.
`mixed invokeArgs($object, $arr)`	Выполняет метод объекта `$object` с параметрами, которые задаются в виде массива `$arr`
`bool isFinal()`	Возвращает `true`, если метод снабжен атрибутом `final`, и `false` — в противном случае
`bool isAbstract()`	Возвращает `true`, если метод является абстрактным (снабжен атрибутом `abstract`), и `false` — в противном случае
`bool isPublic()`	Возвращает `true`, если метод является открытым (снабжен атрибутом `public`), и `false` — в противном случае
`bool isPrivate()`	Возвращает `true`, если метод является закрытым (снабжен атрибутом `private`), и `false` — в противном случае
`bool isProtected()`	Возвращает `true`, если метод является защищенным (снабжен атрибутом `protected`), и `false` — в противном случае
`bool isStatic()`	Возвращает `true`, если метод является статическим (снабжен атрибутом `static`), и `false` — в противном случае
`bool isConstructor()`	Возвращает `true`, если метод является конструктором, и `false` — в противном случае
`bool isDestructor()`	Возвращает `true`, если метод является деструктором, и `false` — в противном случае
`int getModifiers()`	Возвращает битовую маску, соответствующую атрибуту метода (`public`, `private`, `protected`, `final`, `static`)
`ReflectionClass getDeclaringClass()`	Возвращает отражение класса (объект класса `ReflectionClass`), которому принадлежит метод

В листинге 9.17 приводится пример создания отражения метода `helloWorld()` класса `example` (листинг 9.10) и его неявное выполнение.

Листинг 9.17. Создание отражения метода `helloWorld()` класса `example`

```
<?php
  require_once("class.example.php");

  // Создаем объект класса example
  $exp = new example(1, 2, 3);
  // Получаем отражение метода
  $obj = new ReflectionMethod($exp, "helloWorld");
  // Осуществляем неявное выполнение метода
  echo $obj->invoke($exp);
?>
```

Несмотря на то, что класс `ReflectionMethod` допускает создание отражения при помощи конструктора, чаще создают отражение метода при помощи отражения класса, используя метод `getMethod()` или `getMethods()` (табл. 9.4). В листинге 9.18 демонстрируется вызов метода `helloWorld()` класса `example`.

Листинг 9.18. Получение отражения метода через отражение класса

```
<?php
  require_once("class.example.php");

  // Создаем объект класса example
  $exp = new example(1, 2, 3);

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Осуществляем неявный вызов метода
  echo $obj->getMethod("helloWorld")->invoke($exp);
?>
```

9.7. Отражение члена класса. Класс *ReflectionProperty*

Класс `ReflectionProperty` позволяет создать отражение члена класса. Методы класса `ReflectionProperty` представлены в табл. 9.6.

Таблица 9.6. *Методы класса* `ReflectionProperty`

Метод	Описание
`__construct($class, $name)`	Конструктор, создающий отражение члена класса `$name` или объекта `$class`
`string __toString()`	Вывод строкового представления объекта при его интерполяции
`static string export($class, $name, $return)`	Аналогичен функции `__toString()`: возвращает информацию о члене `$name` класса `$class` в виде строки или, если необязательный параметр `$retrun` равен `false`, выводит ее в окно браузера. В отличие от метода `__toString()`, метод `export()` является статическим и допускает вызов без объявления объекта класса, например, `ReflectionProperty::export('example', 'first')`
`string getName()`	Возвращает имя члена
`bool isPublic()`	Возвращает `true`, если член является открытым (снабжен атрибутом `public`), и `false` — в противном случае
`bool isPrivate()`	Возвращает `true`, если член является закрытым (снабжен атрибутом `private`), и `false` — в противном случае
`bool isProtected()`	Возвращает `true`, если член является защищенным (снабжен атрибутом `protected`), и `false` — в противном случае
`bool isStatic()`	Возвращает `true`, если член является статическим (снабжен атрибутом `static`), и `false` — в противном случае
`bool isDefault()`	Возвращает `true`, если член имеет значение по умолчанию, и `false` — в противном случае
`int getModifiers()`	Возвращает битовую маску, соответствующую атрибутам метода (`public`, `private`, `protected`, `final`, `static`)
`mixed getValue ($object)`	Возвращает текущее значение члена объекта `$object`
`void setValue ($object, $value)`	Передает новое значение `$value` члену объекта `$object`
`ReflectionClass getDeclaringClass()`	Возвращает отражение класса, которому принадлежит член
`string getDocComment()`	Возвращает содержимое многострочного комментария, расположенного непосредственно перед классом

Класс `ReflectionProperty` является предопределенным и не требует объявления. Однако при реализации внешним разработчиком он мог бы выглядеть так, как это представлено в листинге 9.19.

Листинг 9.19. Гипотетическая реализация класса `ReflectionProperty`

```
<?php
  class ReflectionProperty implements Reflector
  {
    public function __construct($class, $name);
    public function __toString();
    public static function export($class, $name, $return);
    public function getName();
    public function isPublic();
    public function isPrivate();
    public function isProtected();
    public function isStatic();
    public function isDefault();
    public function getModifiers();
    public function getValue($object);
    public function setValue($object, $value);
    public function getDeclaringClass();
    public function getDocComment();
  }
?>
```

В листинге 9.20 открытому члену `$second` передается новое значение.

Листинг 9.20. Неявная установка нового значения члена при помощи отражения

```
<?php
  require_once("class.example.php");

  // Создаем объект класса example
  $exp = new example(1, 2, 3);

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Получаем отражение члена $second
  $snd = $obj->getProperty("second");
  // Устанавливаем значение
  $snd->setValue($exp, 100);
```

```
  // Выводим новое значение
  echo $exp->second; // 100
?>
```

Важно отметить, что при помощи метода `setValue()` невозможно установить значение закрытой переменной — в этом случае генерируется исключение `ReflectionException` (рис. 9.3).

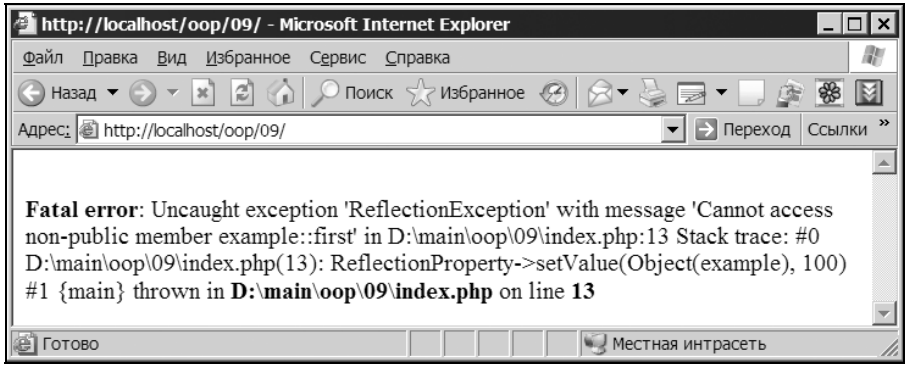


Рис. 9.3. При попытке доступа к закрытому члену генерируется исключение `ReflectionException`

9.8. Исключения механизма отражения

В предыдущем разделе была приведена ситуация, в которой генерируется исключение `ReflectionException`. Исключение данного типа предназначено для обработки всех исключительных ситуаций механизма отражения. Класс `ReflectionException` наследуется от базового класса `Exception`, но не содержит никаких дополнительных методов. В листинге 9.21 представлен пример обработки исключительной ситуации при использовании механизма отражения.

Листинг 9.21. Обработка исключительной ситуации `ReflectionException`

```
<?php
  require_once("class.example.php");

  try
  {
    // Создаем объект класса example
    $exp = new example(1, 2, 3);
```

```
    // Получаем отражение класса
    $obj = new ReflectionClass("example");

    // Получаем отражение члена $first
    $fst = $obj->getProperty("first");
    // Пытаемся передать новое значение закрытому члену
    $fst->setValue($exp, 100);
    // Выводим новое значение
    echo $exp->getFirst();
  }
  catch(ReflectionException $exc)
  {
    echo "Исключение: {$exc->getMessage()}<br>";
    echo "в файле {$exc->getFile()}<br>";
    echo "в строке {$exc->getLine()}<br>";
  }
?>
```

В листинге 9.21 производится попытка передать новое значение закрытому члену `$first`, что приводит к генерации исключительной ситуации (рис. 9.4).

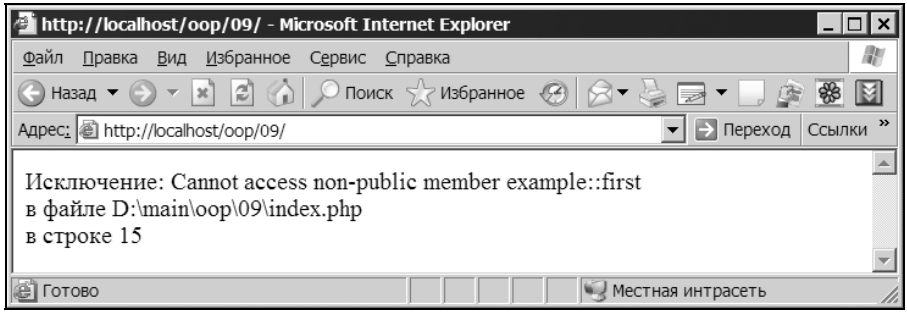


Рис. 9.4. Обработка исключительной ситуации

9.9. Отражение расширения. Класс *ReflectionExtension*

Отражение `ReflectionExtension` предназначено для получения информации о расширениях PHP.

Замечание

Язык PHP имеет модульную структуру, включающую ядро и ряд расширений, которые могут подключаться в конфигурационном файле php.ini при помощи директивы `extension`.

В табл. 9.7 представлен список методов класса `ReflectionExtension`.

Таблица 9.7. *Методы класса* `ReflectionExtension`

Метод	Описание
`__construct($name)`	Конструктор класса, создающий отражение для расширения с именем `$name`
`string __toString()`	Вывод строкового представления объекта при его интерполяции
`static string export($name, $return)`	Аналогичен функции `__toString()`: возвращает информацию о расширении `$name` в виде строки или, если необязательный параметр `$retrun` равен `false`, выводит ее в окно браузера. В отличие от метода `__toString()`, метод `export()` является статическим и может вызываться без объявления объекта класса, например, `ReflectionExtension::export('standard')`
`string getName()`	Возвращает имя расширения
`string getVersion()`	Возвращает версию расширения
`ReflectionFunction[] getFunctions()`	Возвращает массив отражений для функций, входящих в состав расширения
`array getConstants()`	Возвращает ассоциативный массив с предопределенными константами, входящими в состав расширения
`array getINIEntries()`	Возвращает ассоциативный массив со значениями всех директив конфигурационного файла php.ini, которые влияют на работу расширения
`ReflectionClass[] getClasses()`	Возвращает массив отражений для классов, входящих в состав расширения
`array getClassNames()`	Возвращает массив с названиями классов, входящих в состав расширения

Класс `ReflectionExtension` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 9.22.

Листинг 9.22. Гипотетическая реализация класса `ReflectionExtension`

```
<?php
  class ReflectionExtension implements Reflector
  {
    public function __construct($name);
    public function __toString();
```

```
    public static function export($name, $return);
    public function getName();
    public function getVersion();
    public function getFunctions();
    public function getConstants();
    public function getINIEntries();
    public function getClasses();
    public function getClassNames();
  }
?>
```

Отражение `ReflectionExtension` может применяться не только для расширений, но и для ядра PHP: для этого конструктору класса следует передать имя `standard`. В листинге 9.23 демонстрируется, как при помощи отражения расширения можно выяснить версию PHP.

Листинг 9.23. Получение версии PHP

```
<?php
  // Получаем отражение ядра PHP
  $obj = new ReflectionExtension("standard");
  // Выводим в окно браузера версию PHP
  echo $obj->getVersion();
?>
```

В листинге 9.24 приводится пример вывода директив конфигурационного файла php.ini и списка функций для расширения php_mysql, предназначенного для взаимодействия с СУБД MySQL.

Замечание

Получить список загруженных в настоящий момент расширений PHP можно при помощи функции `get_loaded_extensions()`.

Листинг 9.24. Получение отражения расширения php_mysql

```
<?php
  // Получаем отражение расширения
  // для работы с MySQL
  $obj = new ReflectionExtension("mysql");
  // Выводим директивы конфигурационного
  // файла php.ini
  echo "<pre>";
  print_r($obj->getINIEntries());
  echo "</pre>";
```

```
  // Выводим список функций расширения
  $functions = $obj->getFunctions();
  foreach($functions as $funct)
  {
    echo $funct->getName()."<br>";
  }
?>
```

Результатом работы скрипта из листинга 9.24 будут следующие строки:

```
Array
(
    [mysql.allow_persistent] => 1
    [mysql.max_persistent] => -1
    [mysql.max_links] => -1
    [mysql.default_host] =>
    [mysql.default_user] =>
    [mysql.default_password] =>
    [mysql.default_port] =>
    [mysql.default_socket] =>
    [mysql.connect_timeout] => 60
    [mysql.trace_mode] =>
)
mysql_connect
mysql_pconnect
mysql_close
...
mysql_tablename
mysql_table_name
```

Важно отметить, что наряду с функциями, доступными внешнему программисту, в списке приводятся внутренние функции, предназначенные для использования внутри библиотеки php_mysql.

9.10. Вспомогательный класс *Reflection*

Вспомогательный класс `Reflection` не реализует интерфейс `Reflector` и, следовательно, не является отражением. В состав класса входят лишь две статические функции, описание которых представлено в табл. 9.8.

В листинге 9.25 демонстрируется использование метода `getModifierNames` для расшифровки битовой маски, получаемой для метода `getFirst()` класса `example` (листинг 9.10).

Таблица 9.8. *Методы класса* `Reflection`

Метод	Описание
`public static array getModifierNames ($modifiers)`	Метод принимает битовую маску классов отражений, получаемую методом `getModifiers()`, и возвращает массив атрибутов в текстовом, а не числовом представлении
`public static export ($ref, $return)`	Метод принимает объект отражения `$refl` и вызывает его метод `__toString()`. Если необязательный параметр `$return` равен `true`, метод возвращает строку, в противном случае результат выводится непосредственно в окно браузера

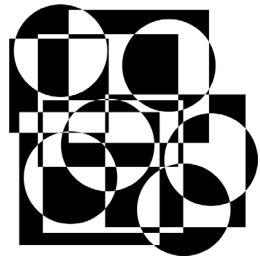
Листинг 9.25. Использование метода `getModifierNames()`

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Получаем битовую маску
  $mask = $obj->getMethod("getFirst")->getModifiers();

  // Расшифровываем содержимое битовой маски
  echo "<pre>";
  print_r(Reflection::getModifierNames($mask));
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 9.25 будет следующий дамп массива свойств:

```
Array
(
    [0] => public
)
```

ГЛАВА 10



Набор классов. Framework

Создавая объектно-ориентированные приложения, следует тщательно следить, чтобы классы и объекты могли повторно использоваться, увеличивали производительность и гибкость системы. Классы и объекты не должны создаваться только ради того, чтобы приложение оправдало звание объектно-ориентированного.

Например, разрабатывать собственный объектно-ориентированный интерфейс для доступа к СУБД MySQL в приложении, не использующем никаких других СУБД, не имеет смысла — код усложнится, увеличатся трудозатраты на его создание и сопровождение, придется отказаться от эффективных уникальных особенностей MySQL.

Замечание

Объектно-ориентированный подход для доступа к СУБД MySQL реализован в новом PHP расширением php_mysqli, которое описывается в *приложении 1*.

Создавать уникальные классы, например, содержащие методы для реализации гостевой книги или блока "Новости", также не имеет смысла, т. к. объем кода, необходимый для создания таких классов, превосходит объем кода не объектно-ориентированного приложения, а сам класс никогда повторно использоваться не будет. Трудно предположить, что для внесения изменения в класс гостевой книги сторонний разработчик будет наследовать новый класс, а не отредактирует существующий. Тем более, редактирование как содержимого подобного класса, так и его интерфейса ничем не ограничивается — у такого класса нет наследников, его не используют десятки других приложений.

Очень часто возникает вопрос: когда же необходимо применять объектно-ориентированный подход в PHP? Ведь протоколы Интернета, СУБД и все сетевое окружение ориентированы на структурный подход.

Объектно-ориентированный подход оправдывает себя только в том случае, когда один и тот же набор классов используют десятки или сотни приложений, т. е. налицо повторное использование кода. Если класс используют десятки приложений, изменить его интерфейс уже невозможно, однако можно воспользоваться его функциональностью, унаследовав производный класс.

При этом классы не должны зависеть от бизнес-логики приложения или от его дизайна, иначе использовать его в других приложениях станет невозможно. Последнее означает, что, совершенствуя и модифицируя набор классов, можно в приложении двухлетней давности заменить его новой версией (снабженной дополнительной функциональностью и с исправленными ошибками), при этом ни дизайн, ни бизнес-логика приложения не пострадают.

Разделение движка, бизнес-логики и дизайна приложения часто описывают в рамках архитектуры MVC (Model, View, Controller) — Модель, Вид, Контроллер. Архитектура MVC популярна на Западе и удобна в рамках сложных оконных приложений, однако к сетевым приложениям эта модель обычно не применяется. Да и сами компоненты Web-приложения часто называют немного по-другому: "движок", "бизнес-логика" и "дизайн". На рис. 10.1 представлена схема соответствия архитектуры MVC и компонентов Web-приложения.

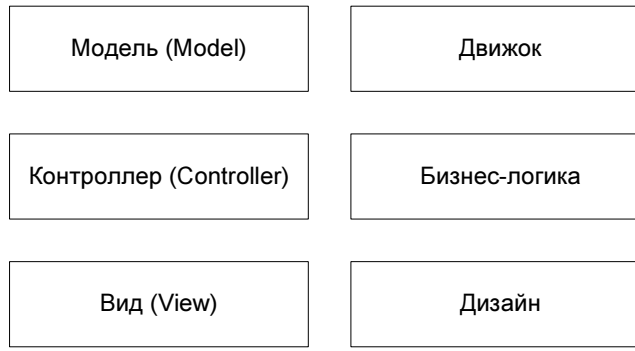


Рис. 10.1. Сравнение архитектуры MVC и компонентов Web-приложения

Движок (Модель) реализует базовые возможности приложения, на основе которых строится бизнес-логика приложения (Контроллер). Причем для того, чтобы дизайнеры и разработчики могли работать раздельно, часто прибегают к шаблонам или стилевым таблицам (Вид), позволяющим одномоментно менять дизайн всего приложения, состоящего из тысяч страниц.

Единая архитектура MVC в Web-приложениях чаще всего разделяется на компоненты в соответствии с одной из следующих моделей:

- разделение дизайна и программного кода, которое часто реализуется в виде шаблонов (template);

- разделение бизнес-логики и реализации рутинных операций, обычно реализуемое в виде фреймворка (framework) — набора классов, который реализует остов приложения и предоставляет разработчику готовые программные блоки, соединяемые между собой в зависимости от бизнес-логики конкретного приложения.

Замечание

Шаблоны достаточно популярны среди Web-разработчиков, однако их рассмотрение выходит за рамки данной книги. Среди достоинств шаблонов можно отметить то, что они позволяют значительно сократить размер файлов, состоящих из смешанного кода HTML и PHP. К недостаткам шаблонов можно отнести отсутствие стандартизации — существует большое количество разных систем, не совместимых друг с другом ни по синтаксису, ни по идеологии. Более того, стандартизации шаблонов не предвидится, т. к. их роль должны выполнять каскадные таблицы стилей (CSS), использование которых совместно с XML в идеале должно приводить к разделению оформления документа (дизайна) и его структуры.

Остановимся подробнее на создании движка или Framework-системы, т. к. эта часть Web-приложения идеально подходит для применения объектно-ориентированного подхода. Однажды разработанный набор классов может применяться для создания большого количества приложений, а его старые версии могут заменяться без ущерба для логики приложений новыми версиями.

Замечание

При создании систем, автоматизирующих рутинные операции, важно определить грань, которая позволит значительно сократить время разработки приложений и в то же время не уменьшит гибкость системы. Чем быстрее система позволяет разрабатывать приложения, тем труднее отклониться от шаблона, диктуемого системой. Наиболее гибким подходом является использование PHP, HTML и JavaScript без каких-либо готовых систем и наборов классов, однако это и самый трудоемкий и длительный путь, что в условиях высокой динамики Web-среды не всегда приемлемо.

Большая часть времени Web-разработчиков тратится на создание всевозможных HTML-форм и их обработчиков. Можно добиться значительного сокращения времени разработки, если создать готовые компоненты HTML-форм, в которых автоматически будет осуществляться проверка правильности ввода и обработка результатов при последующей вставке в базу данных. Объекты элементов управления HTML-формы сами могут становиться членами класса HTML-формы. Таким образом можно добиться высокой степени повторного использования кода и в то же время предоставить разработчикам широкие возможности для внесения изменений в набор классов за счет наследования и включения уже существующих объектов.

Замечание

Набор классов, представленный в данной главе, к моменту написания книги был опробован сотрудниками студии SoftTime для построения десятка сайтов. Для того чтобы не путать данный набор классов (Framework) с другими разработками в этой области, назовем его SoftTime Framework. Обновленные версии дистрибутива можно будет найти в разделе Downloads на сайте **http://www.softtime.ru**. Версию SoftTime Framework, рассматриваемую в данной книге, можно найти на поставляемом вместе с ней компакт-диске.

В данной главе будет разработан набор классов для быстрого построения HTML-форм, а также продемонстрирован способ расширения данного набора за счет разработки новых элементов управления с использованием уже существующих.

10.1. Требования к набору классов

Конечной целью SoftTime Framework будет набор классов, позволяющий создавать HTML-формы высокой степени сложности. Такой набор предоставляет разработчику ряд преимуществ:

- автоматически осуществляется проверка правильности ввода информации;
- автоматически выполняется экранирование специальных символов перед помещением информации в базу данных, что предотвращает любую возможность SQL-инъекций;
- HTML-форму (вместе с введенной пользователем информацией) можно серилизовать и в любой момент восстановить (это может быть удобным, когда требуется продолжить работу с формой после регистрации пользователя).

HTML-формы могут быть совершенно разными по форме и содержанию. В своем составе HTML-формы могут содержать текстовые области, выпадающие списки, переключатели, флажки (рис. 10.2). Обязательные для заполнения элементы управления отмечаются символом *.

Как видно из рис. 10.2, типичная HTML-форма содержит расположенные друг под другом элементы управления. Справа располагается название элемента, слева сам элемент управления.

Замечание

В данной главе не будут рассматриваться элементы управления HTML-формы, использующие для своего функционирования элементы JavaScript.

Встречаются и более сложные формы, например, между элементами управления могут располагаться текстовые параграфы и заголовки, а сам элемент управления может снабжаться пояснительной надписью или ссылкой на статью с пояснениями (рис. 10.3).

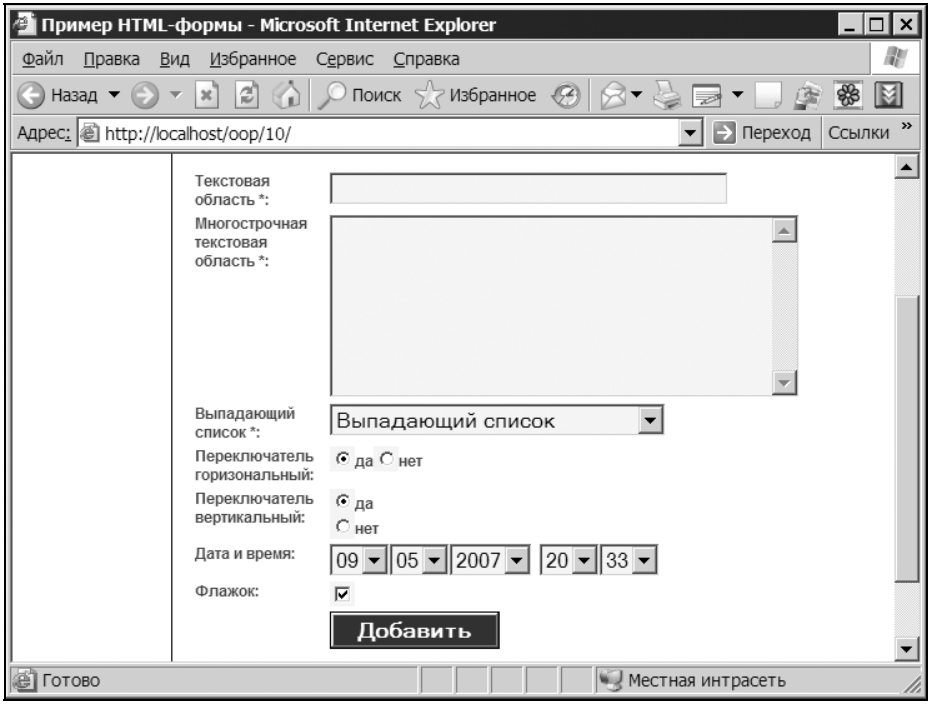


Рис. 10.2. Типичная HTML-форма

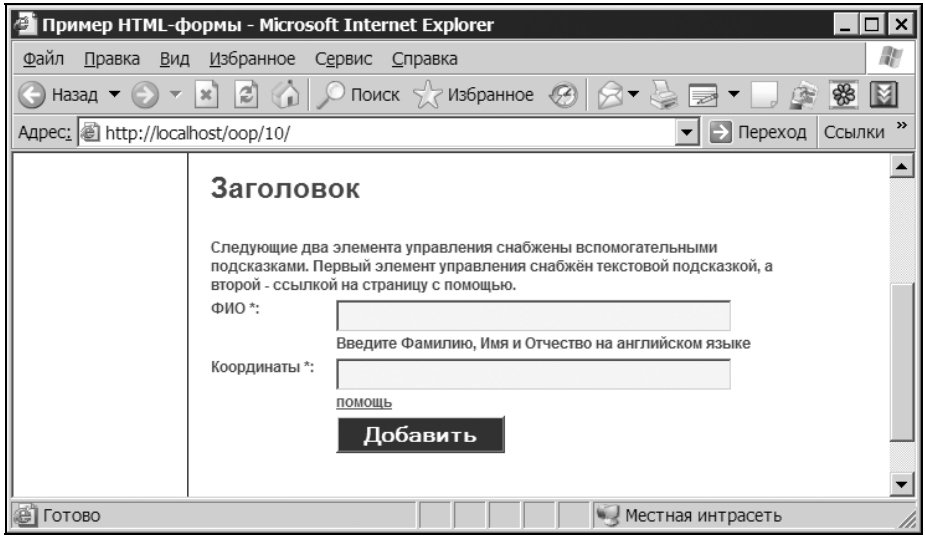


Рис. 10.3. Подсказки для элементов управления HTML-формы

Помимо видимых элементов управления, HTML-форма может содержать скрытые элементы для передачи информации, которые могут быть как обязательными, так и не обязательными, одни должны содержать лишь числовые значения, для других допускаются текстовые.

Каждому элементу управления будет соответствовать отдельный класс. Массив объектов разных классов будет составлять содержимое HTML-формы. Данный массив будет передаваться конструктору HTML-формы, которая будет содержать метод вывода HTML-формы в окно браузера в виде таблицы.

Замечание

Класс HTML-формы будет самым узким элементом системы, т. к. именно он будет определять структуру и внешний вид HTML-формы. Разумеется, при помощи каскадных таблиц стилей можно будет поменять внешний вид HTML-формы, однако если потребуются структурные изменения, например, расположить элементы управления в два столбца, придется унаследовать от него производный класс и перегрузить в нем метод, ответственный за вывод HTML-формы в окно браузера.

10.2. HTML-форма и ее обработчик

Так как основной целью создаваемого набора классов будет HTML-форма, рассмотрим подробнее процесс ее создания и обработки. HTML-формы создаются при помощи парных тегов `<form>` и `</form>`, между которыми располагаются теги элементов управления. В листинге 10.1 представлен HTML-код формы, содержащей два элемента управления с однострочной текстовой областью `text` и кнопку подтверждения `submit`.

Листинг 10.1. HTML-форма

```
<form method=POST>
  <input type=text name=first><br>
  <input type=text name=second><br>
  <input type=submit value="Отправить">
</form>
```

Результатом интерпретации HTML-кода из листинга 10.1 будет простейшая HTML-форма, представленная на рис. 10.4.

Элементы управления, как правило, подписываются слева, что может приводить к искажениям, нарушающим дизайн HTML-формы (рис. 10.5).

Для того чтобы предотвратить искажение, представленное на рис. 10.5, элементы управления помещают в ячейки HTML-таблицы (листинг 10.2).

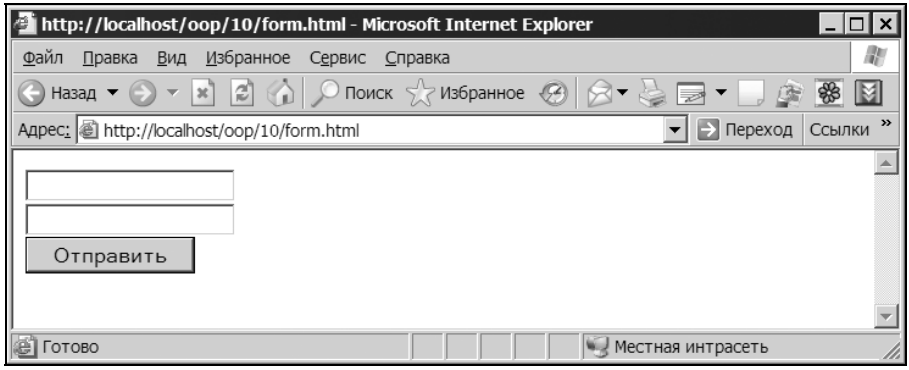


Рис. 10.4. HTML-форма в окне браузера

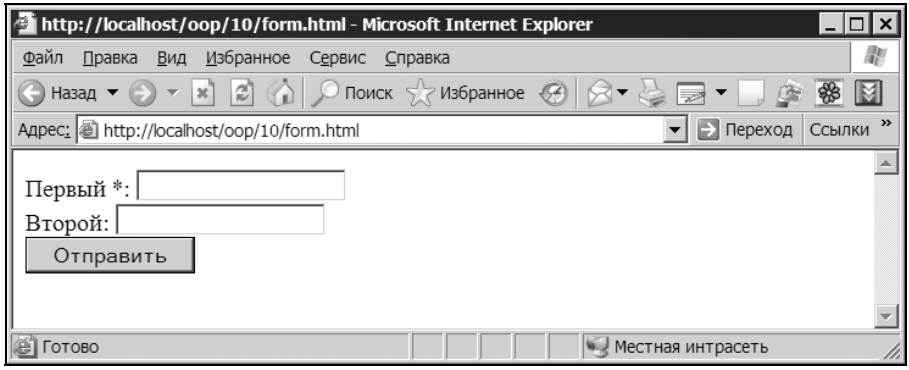


Рис. 10.5. Искажение дизайна HTML-формы

Листинг 10.2. Использование таблицы для структурирования элементов формы

```
<form method=POST>
  <table>
    <tr>
      <td>Первый *:</td>
      <td><input type=text name=first></td>
    </tr>
    <tr>
      <td>Второй:</td>
      <td><input type=text name=second></td>
    </tr>
    <tr>
      <td> </td>
      <td><input type=submit value="Отправить"></td>
    </tr>
```

```
  </table>
</form>
```

Результат интерпретации HTML-кода из листинга 10.2 представлен на рис. 10.6.

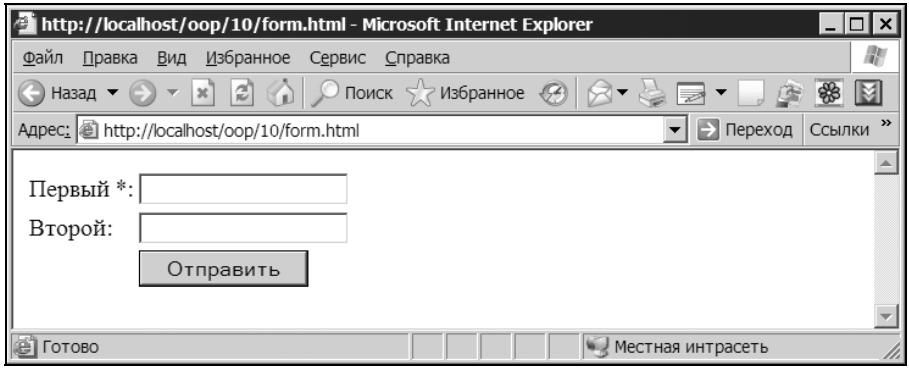


Рис. 10.6. Выравнивание элементов управления HTML-формы

В листингах 10.1 и 10.2 тег `<form>` содержит атрибут `method`, который устанавливает в качестве метода передачи метод `POST`. Помимо метода `POST` применяется также метод `GET`. Главное их отличие заключается в том, что в случае метода `GET` данные из HTML-формы передаются через адресную строку (то есть в HTTP-заголовках). В случае использования метода `POST` данные из HTML-формы передаются в теле HTTP-документа. Метод `POST` используется для передачи большого объема информации и файлов, в то время как метод `GET` применяется для передачи небольшого объема данных, на которые можно указать при помощи ссылки (например, на результаты поиска). В табл. 10.1 представлены основные атрибуты, позволяющие управлять поведением HTML-формы.

Таблица 10.1. *Атрибуты тега* `<form>`

Атрибут	Описание
`action`	Указывает адрес обработчика, которому передаются данные из HTML-формы. Если тег `<form>` не содержит атрибута `action`, то данные отправляются в файл, в котором описывается HTML-форма
`enctype`	Определяет формат отправляемых данных при использовании метода передачи данных `POST`. По умолчанию используется формат `application/x-www-form-urlendoded`. Если HTML-форма содержит элемент управления `file`, предназначенный для передачи файлов на сервер, то следует указать формат `multipart/form-data`

Таблица 10.1 *(окончание)*

Атрибут	Описание
`method`	Определяет метод передачи данных (`POST` или `GET`) из HTML-формы обработчику. По умолчанию, если не указывается атрибут `method`, применяется метод `GET`
`name`	Определяет имя HTML-формы, которое может использоваться для доступа к элементам управления в скриптах, выполняющихся на стороне клиента (например, скриптах JavaScript)
`target`	Указывает окно для вывода результата, полученного от обработчика HTML-формы. Атрибут может принимать следующие значения: • `_blank` — результат открывается в новом окне; • `_self` — результат открывается в текущем окне; данный режим используется по умолчанию, если атрибут `target` не указан явно; • `_parent` — результат открывается в родительском фрейме; при отсутствии фреймов режим аналогичен `_self`; • `_top` — отменяет все фреймы, если они имеются, и загружает страницу в полном окне браузера; при отсутствии фреймов работает как `_self`

Как видно из табл. 10.1, адрес обработчика указывается в атрибуте `action`. Различают два подхода к созданию обработчика HTML-формы:

- ❒ обработчик расположен в отдельном файле, после выполнения манипуляций над полученными данными они направляются на главную страницу;
- ❒ обработчик располагается в том же самом файле, где находится HTML-форма.

Рассмотрим каждый из случаев более подробно. В листинге 10.3 приводится пример HTML-формы, расположенной в файле index.php и содержащей единственное текстовое поле `first` и кнопку.

Листинг 10.3. HTML-форма (файл index.php)

```
<form action=handler.php method=POST>
<input type=text name=first>
<input type=submit value="Отправить">
</form>
```

Данные отправляются обработчику, расположенному в файле handler.php, который осуществляет запись введенной в поле `first` строки в файл text.txt (листинг 10.4).

Листинг 10.4. Обработчик HTML-формы (файл handler.php)

```
<?php
  // Если поле first не заполнено, выводим сообщение
  // об ошибке
  if(empty($_POST['first'])) exit("Текстовое поле не заполнено");

  // Открываем файл text.txt на запись
  $fd = fopen("text.txt", "a");
  if(!$fd) exit("Невозможно открыть файл");
  // Записываем введенную пользователем строку
  // в текстовый файл
  fwrite($fd, "$_POST[first]\r\n");
  // Закрываем файл
  fclose($fd);

  // Осуществляем редирект на главную страницу
  @header("Location: index.php");
?>
```

Обработчик handler.php проверяет при помощи функции `empty()`, не является ли значение поля `first` пустым. Если поле пустое, работа скрипта останавливается с выдачей сообщения об ошибке. Если поле заполнено корректно, файл text.txt открывается для записи и в него помещается новая строка с содержимым элемента суперглобального массива `$_POST['first']`.

Замечание

Значения элементов управления, переданных из HTML-формы, извлекаются из суперглобального массива `$_POST` или `$_GET` в зависимости от выбранного метода передачи данных. Для файлов, загружаемых на сервер, предназначен отдельный массив `$_FILES`.

При помощи функции `header()` отправляется HTTP-заголовок `Location`, который требует, чтобы браузер пользователя загрузил страницу index.php.

Замечание

Даже если результат должен отображаться на странице handler.php, стоит сразу после обработки перегрузить страницу при помощи HTTP-заголовка `Location`: это сбросит POST-данные, и перезагрузка страницы не будет приводить к повторному выполнению обработчика.

Размещение HTML-формы и обработчика в разных файлах позволяет структурировать код, однако это не всегда удобно. Например, если пользователь забыл ввести данные, то узнает он об этом только на странице обработчика. В результате пользователь вынужден возвращаться обратно и заполнять HTML-форму заново, что может оказаться очень утомительным, если HTML-форма содержит множество обязательных полей.

Выходом из данной ситуации является расположение обработчика непосредственно в файле HTML-формы (листинг 10.5).

Листинг 10.5. Расположение HTML-формы и обработчика в одном файле

```
<?php
  // Обработчик HTML-формы
  $error = array();
  if(!empty($_POST))
  {
    // Если поле first не заполнено, выводим сообщение
    // об ошибке
    if(empty($_POST['first'])) $error[] = "Текстовое поле не заполнено";

    // Если нет ошибок, начинаем обработку данных
    if(empty($error))
    {
      // Открываем файл text.txt на дозапись
      $fd = fopen("text.txt", "a");
      if(!$fd) exit("Невозможно открыть файл");
      // Записываем введенную пользователем строку
      // в текстовый файл
      fwrite($fd, "$_POST[first]\r\n");
      // Закрываем файл
      fclose($fd);

      // Перегружаем текущую страницу
      @header("Location: $_SERVER[PHP_SELF]");
      // Останавливаем работу скрипта, чтобы после
      // перенаправления не грузилась HTML-форма
      exit();
    }
  }

  // Выводим сообщения об ошибках, если они имеются
  if(!empty($error))
  {
    foreach($error as $err)
```

```
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // HTML-форма
?>
<form method=POST>
<input type=text name=first
       value="<?= htmlspecialchars($_POST['first'], ENT_QUOTES); ?>">
<input type=submit value="Отправить">
</form>
```

Такой подход позволяет не только вывести сообщения об ошибках непосредственно перед HTML-формой, но и сохранить все введенные ранее данные. При проверке данных сообщения об ошибках сохраняются в массив `$error`. Если он оказывается пуст, начинается обработка; если массив содержит сообщения об ошибках, то происходит повторная загрузка HTML-формы с выводом списка обнаруженных ошибок в цикле `foreach`.

10.3. Обработка исключительных ситуаций

По мере работы с набором классов неминуемо будут возникать исключительные ситуации. Поэтому, прежде чем перейти к разработке набора классов, следует создать иерархию исключений, обрабатывающих следующие ошибки:

- `ExceptionMember` — ошибки класса, связанные с обращением к несуществующим членам классов. Как правило, это исключение генерируется в специальных методах `__get()` и `__set()`;
- `ExceptionObject` — ошибки класса, связанные с использованием переменных, не являющихся объектами. Класс HTML-формы будет выступать в качестве контейнера объектов элементов управления. Данный тип исключения будет генерироваться, если вместо такого объекта ошибочно будет передана переменная или объект недопустимого типа;
- `ExceptionMySQL` — ошибки класса, связанные с обращением к СУБД MySQL. Некоторые сложные элементы управления, например выпадающий список городов Российской Федерации, потребуют использования СУБД MySQL. Данное исключение предназначено для обработки ошибок обращения к базе данных.

На рис. 10.7 представлена схема наследования классов `ExceptionMember`, `ExceptionObject` и `ExceptionMySQL`.

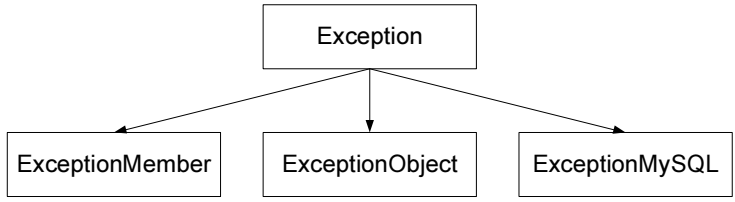


Рис. 10.7. Классы исключений, используемые в SoftTime Framework

В листинге 10.6 представлена реализация класса `ExceptionMember`, который дополняет класс `Exception` защищенным членом `$key`, предназначенным для хранения имени несуществующего члена.

Листинг 10.6. Исключение `ExceptionMember`

```
<?php
  class ExceptionMember extends Exception
  {
    // Имя не существующего члена
    protected $key;

    public function __construct($key, $message)
    {
      $this->key = $key;

      // Вызываем конструктор базового класса
      parent::__construct($message);
    }

    public function getKey()
    {
      return $this->key;
    }
  }
?>
```

В отличие от базового класса `Exception`, параметр `$message`, предназначенный для сопроводительного текста, является обязательным.

В листинге 10.7 представлена реализация класса `ExceptionObject`, который предназначен для исключительных ситуаций, связанных с обращением к несуществующему объекту.

Листинг 10.7. Исключение `ExceptionObject`

```
<?php
  class ExceptionObject extends Exception
```

```
  {
    // Имя объекта
    protected $key;

    public function __construct($key, $message)
    {
      $this->key = $key;

      // Вызываем конструктор базового класса
      parent::__construct($message);
    }

    public function getKey()
    {
      return $this->key;
    }
  }
?>
```

В листинге 10.8 представлена реализация класса `ExceptionMySQL`. В отличие от двух предыдущих классов, конструктор содержит три параметра:

- `$mysql_error` — сообщение об ошибке, возвращаемое функций `mysql_error()`;
- `$sql_query` — SQL-запрос, без которого иногда сложно понять причину возникающей ошибки;
- `$message` — пользовательский комментарий.

Листинг 10.8. Исключение ExceptionMySQL

```
<?php
  class ExceptionMySQL extends Exception
  {
    // Сообщение об ошибке
    protected $mysql_error;
    // SQL-запрос
    protected $sql_query;

    public function __construct($mysql_error, $sql_query, $message)
    {
      $this->mysql_error = $mysql_error;
      $this->sql_query = $sql_query;
```

```
      // Вызываем конструктор базового класса
      parent::__construct($message);
    }

    public function getMySQLError()
    {
      return $this->mysql_error;
    }
    public function getSQLQuery()
    {
      return $this->sql_query;
    }
  }
?>
```

10.4. Базовый класс *field*

Все классы элементов управления будут унаследованы от единого базового класса `field` (рис. 10.8). Так как данный класс является прототипом и его экземпляров (объектов) не потребуется, объявим его абстрактным.

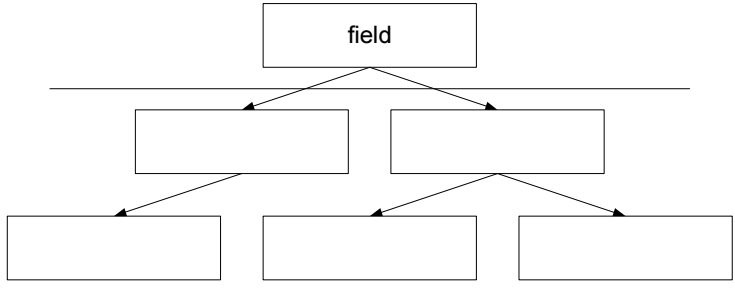


Рис. 10.8. Класс `field` является базовым классом для всей иерархии элементов управления

Класс `field` будет содержать ряд защищенных членов, инициализируемых через конструктор (табл. 10.2).

Таблица 10.2. *Члены класса* `field`

Член класса `field`	**Описание**
`protected $name`	Имя элемента управления (атрибут `name`)
`protected $type`	Тип элемента управления (атрибут `type`)
`protected $caption`	Название, которое располагается слева от элемента управления

Таблица 10.2 *(окончание)*

Член класса `field`	Описание
`protected $value`	Значение элемента управления (атрибут `value`)
`protected $is_required`	Принимает значение `true`, если элемент управления обязателен к заполнению, и `false` — в противном случае. По умолчанию принимает значение `false`
`protected $parameters`	Строка дополнительных параметров; данное поле является зарезервированным для дальнейших расширений. По умолчанию принимает значение пустой строки
`protected $help`	Текст подсказки; по умолчанию принимает значение пустой строки
`protected $help_url`	Ссылка на подсказку; по умолчанию член принимает значение пустой строки
`public $css_class`	Класс CSS
`public $css_style`	Стиль CSS

Следует обратить внимание на то, что члены `$css_class` и `$css_style` объявлены открытыми. Это может быть удобным для быстрой смены оформления HTML-форм (коллекций элементов управления).

Помимо конструктора, класс `field` будет содержать два абстрактных метода: `check()` и `get_html()`, предназначенные для перегрузки в производных классах. Метод `check()` будет проверять корректность введенной информации и возвращать при успешной проверке пустую строку, в противном случае — сообщение об ошибке. Метод `get_html()` будет возвращать массив с названием элемента управления, подсказок и тега элемента управления. Используя этот массив, класс, отображающий HTML-форму, будет формировать элемент управления.

В листинге 10.9 представлена возможная реализация класса `field`.

Листинг 10.9. Реализация класса `field`

```
<?php
  abstract class field
  {
    ///////////////
    // Члены класса
    ///////////////
    // Имя элемента управления
    protected $name;
```

```
// Тип элемента управления
protected $type;
// Название слева от элемента управления
protected $caption;
// Значение элемента управления
protected $value;
// Обязателен ли элемент к заполнению
protected $is_required;
// Строка дополнительных параметров
protected $parameters;
// Подсказка
protected $help;
// Ссылка на подсказку
protected $help_url;

// Класс CSS
public $css_class;
// Стиль CSS
public $css_style;


////////////////
// Методы класса
////////////////
// Конструктор класса
function __construct($name,
                $type,
                $caption,
                $is_required = false,
                $value = "",
                $parameters = "",
                $help = "",
                $help_url = "")
{
  $this->name        = $this->encodestring($name);
  $this->type        = $type;
  $this->caption     = $caption;
  $this->value       = $value;
  $this->is_required = $is_required;
  $this->parameters  = $parameters;
  $this->help        = $help;
  $this->help_url    = $help_url;
}
```

```
    // Метод для проверки корректности заполнения поля
    abstract function check();
    // Абстрактный метод, возвращающий название поля
    // и самого тега элемента управления (каждый наследник
    // должен переопределить этот метод)
    abstract function get_html();

    // Доступ к закрытым и защищенным элементам класса
    // (предназначен только для чтения)
    public function __get($key)
    {
      if(isset($this->$key)) return $this->$key;
      else
      {
        throw new ExceptionMember($key,
              "Член ".__CLASS__."::$key не существует");
      }
    }

    // функция перевода текста с русского языка в транслит
    protected function encodestring($st)
    {
      // Сначала заменяем "односимвольные" фонемы.
      $st=strtr($st,"абвгдеезийклмнопрстуфхъыэ_",
      "abvgdeeziyklmnoprstufh'iei");
      $st=strtr($st,"АБВГДЕЕЗИЙКЛМНОПРСТУФХЪЫЭ_",
      "ABVGDEEZIYKLMNOPRSTUFH'IEI");
      // Затем – "многосимвольные".
      $st=strtr($st,
                      array(
                          "ж"=>"zh", "ц"=>"ts", "ч"=>"ch", "ш"=>"sh",
                          "щ"=>"shch","ь"=>"", "ю"=>"yu", "я"=>"ya",
                          "Ж"=>"ZH", "Ц"=>"TS", "Ч"=>"CH", "Ш"=>"SH",
                          "Щ"=>"SHCH","Ь"=>"", "Ю"=>"YU", "Я"=>"YA",
                          "ї"=>"i", "Ї"=>"Yi", "є"=>"ie", "Є"=>"Ye"
                          )
              );
      // Возвращаем результат.
      return $st;
    }
  }
?>
```

Помимо абстрактных методов и конструктора в класс введен метод `encodestring()`, который преобразует русский текст в транслит. Последнее необходимо для того, чтобы в качестве названия элемента управления случайно не было использовано русское название.

Для доступа к защищенным членам класса перегружается специальный метод `__get()`, который при попытке обратиться к несуществующему члену класса генерирует исключение типа `ExceptionMember`.

10.5. Текстовое поле. Класс *field_text*

Текстовое поле предназначено для ввода пользователем строки текста, как правило, не очень большой. Для создания текстового поля необходимо поместить в HTML-форме между тегами `<form>` и `</form>` тег следующего вида:

```
<input type=text>
```

Следует отметить, для атрибута `type` тип элемента управления `text` является значением по умолчанию. Поэтому, если атрибут `type` отсутствует, а также если ему присвоено неизвестное или ошибочное значение, браузер интерпретирует элемент управления как текстовое поле.

Помимо атрибута `type` тег `<input>` может содержать дополнительные атрибуты, представленные в табл. 10.3.

Таблица 10.3. *Атрибуты текстового поля*

Атрибут	Описание
`maxlength`	Определяет максимально допустимую длину текстовой строки. При отсутствии атрибута количество символов не ограничено
`name`	Имя элемента управления, предназначенное для идентификации в обработчике. Имя должно быть уникальным в пределах формы, т. е. отличаться от имен других элементов управления, т. к. используется в качестве ключа для доступа к значению поля в обработчике
`size`	Ширина элемента управления, определяющая физический размер элемента управления на странице сайта
`value`	Начальный текст, содержащийся в поле

Для моделирования текстового поля потребуется унаследовать от базового класса `field` новый класс `field_text` (рис. 10.9).

Как видно из табл. 10.3, по сравнению с обобщенным элементом управления, который моделирует класс `field` (см. *раздел 10.4*), однострочное текстовое поле имеет два уникальных атрибута: `maxlength` и `size`. Последнее означает,

что при создании производного класса `field_text` необходимо будет ввести два новых элемента управления для хранения значений этих атрибутов (листинг 10.10).

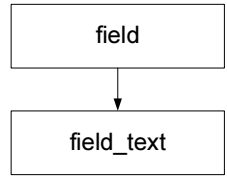


Рис. 10.9. Наследование производного класса `field_text` от базового класса `field`

Листинг 10.10. Класс текстового поля `field_text`

```
<?php
  ////////////////////////////////////////////////////////////
  // Текстовое поле text
  ////////////////////////////////////////////////////////////

  class field_text extends field
  {
    // Размер текстового поля
    public $size;
    // Максимальный размер вводимых данных
    public $maxlength;
    // Конструктор класса
    function __construct($name,
                  $caption,
                  $is_required = false,
                  $value = "",
                  $maxlength = 255,
                  $size = 41,
                  $parameters = "",
                  $help = "",
                  $help_url = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct($name,
                  "text",
                  $caption,
                  $is_required,
                  $value,
```

```
                $parameters,
                $help,
                $help_url);
    // Инициализируем члены класса
    $this->size = $size;
    $this->maxlength = $maxlength;
  }

  // Метод для возврата имени поля
  // и самого тега элемента управления
  function get_html()
  {
    // Если элементы оформления не пусты, учитываем их
    if(!empty($this->css_style))
    {
      $style = "style=\"".$this->css_style."\"";
    }
    else $style = "";
    if(!empty($this->css_class))
    {
      $class = "class=\"".$this->css_class."\"";
    }
    else $class = "";

    // Если определены размеры, учитываем их
    if(!empty($this->size)) $size = "size=".$this->size;
    else $size = "";
    if(!empty($this->maxlength))
    {
      $maxlength = "maxlength=".$this->maxlength;
    }
    else $maxlength = "";

    // Формируем тег
    $tag = "<input $style $class
                type=\"".$this->type."\"
                name=\"".$this->name."\"
                value=\"".
                htmlspecialchars($this->value, ENT_QUOTES)."\"
                $size $maxlength>\n";

    // Если поле обязательно, отмечаем этот факт
    if($this->is_required) $this->caption .= " *";
```

```
      // Формируем подсказку, если она имеется
      $help = "";
      if(!empty($this->help))
      {
        $help .= "<span style='color:blue'>".
                        nl2br($this->help)."</span>";
      }
      if(!empty($help)) $help .= "<br>";
      if(!empty($this->help_url))
      {
        $help .= "<span style='color:blue'><a href=".
                  $this->help_url.">помощь</a></span>";
      }

      return array($this->caption, $tag, $help);
    }

    // Метод, проверяющий корректность переданных данных
    function check()
    {
      // Обезопасить текст перед внесением в базу данных
      if (!get_magic_quotes_gpc())
      {
        $this->value = mysql_escape_string($this->value);
      }

      // Если поле обязательно для заполнения
      if($this->is_required)
      {
        // Проверяем, не пусто ли оно
        if(empty($this->value))
        {
          return "Поле \"".$this->caption."\" не заполнено";
        }
      }

      return "";
    }
  }
?>
```

Конструктор класса `field_text` совпадает по своей структуре с конструктором базового класса `field`, однако принимает два дополнительных необяза-

тельных параметра `$maxlength` и `$size` для максимального количества символов в строке и размера элемента управления соответственно. Параметр `$maxlength` принимает по умолчанию значение 255, а `$size` — 41. Только последние два члена подвергаются инициализации в конструкторе класса `field_text`; все остальные члены инициализируются при помощи вызова конструктора базового класса `field`. Тип элемента управления `$type` указывается жестко при помощи константы `"text"`.

Метод `check()` экранирует специальные символы во введенной пользователем строке `$value` при помощи функции `mysql_escape_string()`. Последнее связано с тем, что все значения из HTML-форм на протяжении оставшейся части книги будут помещаться в СУБД MySQL. Обработка текстовых строк при помощи функции `mysql_escape_string()` позволит избежать синтаксических ошибок при вставке в SQL-запрос, а также пресечет SQL-инъекции.

Замечание

Вообще метод `check()` не совсем корректен с точки зрения методологии программирования, т. к. решает, по сути, две задачи: экранирование текстовых значений перед помещением в СУБД MySQL и проверку корректности ввода данных. Как в структурном, так и в объектно-ориентированном программировании стараются проектировать функции и методы таким образом, чтобы они решали только одну задачу. В данном случае правило нарушено сознательно, чтобы инкапсулировать в классе как можно больше рутинных задач.

Метод `check()` осуществляет также корректность ввода данных. Если поле обязательно для заполнения, выполняется проверка, содержит ли член `$value` какие-либо символы. Если поле пустое, возвращается текстовое сообщение, если заполненное — пустая строка. Генерировать исключения для обработки данной ситуации не целесообразно, т. к. отсутствие значения в поле представляет собой рядовой случай, а не исключительный. Кроме того, HTML-форма может содержать несколько элементов управления, обязательных для заполнения, и пользователю необходимо сообщить обо всех незаполненных полях. В случае использования исключения можно будет обработать только один элемент управления.

Метод `get_html()` формирует массив, элементы которого используются для генерации HTML-кода элемента управления. Первый элемент массива содержит подпись, второй — тег элемента управления, а третий — текст подсказки или ссылку на статью с пояснениями. Возвращать строку, представляющую только элемент управления, нежелательно, т. к. при формировании HTML-формы может потребоваться расположить фрагменты нестандартным образом.

Следует отметить, что значение члена `$value` при формировании атрибута `value`, во-первых, должно быть заключено в кавычки, иначе от фразы, содер-

жащей символы пробела, останется только первое слово, а во-вторых, оно должно быть обработано при помощи функции `htmlspecialchars()`. Последнее необходимо для предотвращения XSS-атак и искажения HTML-формы при передаче угловых скобок и HTML-фрагментов.

Для того чтобы продемонстрировать работу класса `field_text`, необходимо создать класс HTML-формы `form`, который будет выступать в качестве контейнера элементов управления. Построению такого класса посвящен следующий раздел.

10.6. Класс *form*

HTML-форма может содержать множество элементов управления разного типа, поэтому целесообразно включить в состав класса, моделирующего форму, массив объектов `$fields`. Член `$fields` будет объявлен открытым для удобного доступа к элементам управления.

Все HTML-формы, которые представлены в данной книге, будут содержать одну кнопку типа `submit`, отправляющую данные обработчику формы. Для названия кнопки в классе будет предусмотрен защищенный член `$button_name`.

Сама HTML-форма будет строиться при помощи таблицы, как это описывается в *разделе 10.2*. Для оформления ячеек таблицы будут использованы два члена: `$css_td_class` и `$css_td_style`, определяющие класс (`class`) и стиль (`style`) каскадных таблиц CSS. Кроме того, два члена `$css_fld_class` и `$css_fld_style` будут определять класс и стиль каскадных таблиц CSS для элементов управления формы (эти поля являются открытыми в классах, производных от `field`).

Помимо конструктора, предназначенного для инициализации членов метода, класс `form` будет содержать следующие методы:

- `print_form()` — выводит HTML-форму в окно браузера;
- `__toString()` — перегруженный специальный метод, аналогичный по действию методу `print_form()`;
- `check()` — вызывает методы проверки всех элементов управления и формирует массив с сообщениями об ошибках.

В листинге 10.11 представлена возможная реализация класса `form`. Пока класс учитывает особенности только одного элемента управления `field_text`, однако по мере развития иерархии потребуется определенная модификация класса.

Листинг 10.11. Класс HTML-формы `form`

```
<?php
  ////////////////////////////////////////////////////////
  // Класс HTML-формы
  ////////////////////////////////////////////////////////

  class form
  {
    // Массив элементов управления
    public $fields;
    // Название кнопки HTML-формы
    protected $button_name;

    // Класс CSS ячейки таблицы
    protected $css_td_class;
    // Стиль CSS ячейки таблицы
    protected $css_td_style;
    // Класс CSS элемента управления
    protected $css_fld_class;
    // Стиль CSS элемента управления
    protected $css_fld_style;

    // Конструктор класса
    public function __construct($flds,
                          $button_name,
                          $css_td_class = "",
                          $css_td_style = "",
                          $css_fld_class = "",
                          $css_fld_style = "")
    {
      $this->fields       = $flds;
      $this->button_name  = $button_name;

      $this->css_td_class = $css_td_class;
      $this->css_td_style = $css_td_style;
      $this->css_fld_class = $css_fld_class;
      $this->css_fld_style = $css_fld_style;

      // Проверяем, являются ли элементы массива $flds
      // производными класса field
      foreach($flds as $key => $obj)
      {
        if(!is_subclass_of($obj, "field"))
```

```
    {
      throw new ExceptionObject($key,
            "\"$key\" не является элементом управления");
    }
  }
}

// Вывод HTML-формы в окно браузера
public function print_form()
{
  $enctype = "";
  if(!empty($this->fields))
  {
    foreach($this->fields as $obj)
    {
      // Назначаем всем элементам управления единый стиль
      if(!empty($this->css_fld_class))
      {
        $obj->css_class = $this->css_fld_class;
      }
      if(!empty($this->css_fld_class))
      {
        $obj->css_style = $this->css_fld_style;
      }
      // Проверяем, нет ли среди элементов управления
      // поля file; если есть — включаем строку
      // enctype='multipart/form-data'
      if($obj->type == "file")
      {
        $enctype = "enctype='multipart/form-data'";
      }
    }
  }

  // Если элементы оформления не пусты — учитываем их
  if(!empty($this->css_td_style))
  {
    $style = "style=\"".$this->css_td_style."\"";
  }
  else $style = "";
  if(!empty($this->css_td_class))
  {
    $class = "class=\"".$this->css_td_class."\"";
  }
  else $class = "";
```

```
  // Выводим HTML-форму
  echo "<form name=form $enctype method=post>";
  echo "<table>";
  if(!empty($this->fields))
  {
    foreach($this->fields as $obj)
    {
      // Получаем название поля и его HTML-представление
      list($caption, $tag, $help, $alternative) = $obj->get_html();
      if(is_array($tag)) $tag = implode("<br>",$tag);
      echo "<tr>
              <td width=100
                  $style $class valign=top>$caption:</td>
              <td $style $class valign=top>$tag </td>
            </tr>\n";
      if(!empty($help))
      {
         echo "<tr>
                 <td> </td>
                 <td $style $class valign=top>$help</td>
               </tr>";
      }
    }
  }
  // Выводим кнопку подтверждения
  echo "<tr>
          <td $style $class></td>
          <td $style $class>
            <input class=button
                   type=submit
     value=\"".htmlspecialchars($this->button_name, ENT_QUOTES)."\">
          </td>
        </tr>\n";
  echo "</table>";
  echo "</form>";
}

// Перегрузка специального метода __toString()
public function __toString()
{
  $this->print_form();
}

// Метод, проверяющий корректность ввода данных в форму
public function check()
```

```
    {
      // Последовательно вызываем метод check для каждого
      // объекта field, принадлежащего классу
      $arr = array();
      if(!empty($this->fields))
      {
        foreach($this->fields as $obj)
        {
          $str = $obj->check();
          if(!empty($str)) $arr[] = $str;
        }
      }
      return $arr;
    }
  }
?>
```

Конструктор класса `form` принимает в качестве первого параметра массив объектов, классы которых должны быть производными классами `field`; в противном случае генерируется исключение `ExceptionObject`. Такой подход позволяет быть совершенно уверенным, что любой элемент массива элементов управления содержит реализацию абстрактных методов `check()` и `get_html()`, необходимых для нормального функционирования класса `form`.

Метод `check()` вызывает одноименные методы элементов управления, формируя массив сообщений о некорректном заполнении полей. Метод `print_form()` формирует таблицу с HTML-формой, получая HTML-фрагменты элемента управления при помощи метода `get_html()`. Элементы возвращенного массива распределяются по переменным при помощи конструкции `list()`.

Замечание

Количество элементов в конструкции `list()` и массиве могут не совпадать; в листинге 10.11 предусмотрена дополнительная переменная `$alternative`, которая может потребоваться в дальнейшем для более сложных элементов управления, чей метод `get_html()` возвращает массив из четырех фрагментов.

В настоящий момент все готово, чтобы приступить к созданию простейших HTML-форм с использованием разработанных ранее классов. Этому будет посвящен следующий раздел.

10.7. Пример HTML-формы

Очень важно на самых ранних этапах проектирования систем любой сложности иметь возможность запустить и проверить работоспособность создаваемого кода. Чем раньше появляется возможность отлаживать систему — тем устойчивее и надежнее она окажется в результате. На самом деле классы `field`, `field_text` и `form` создаются одновременно, чтобы исправлять взаимные ошибки и опечатки, которые неминуемо возникают в ходе набора кода.

Для отладки классов и проверки их работоспособности в момент разработки обычно создается небольшое приложение. В качестве такого приложения выберем регистрацию пользователей, от которых потребуется ввод имени и пароля. Полученные данные будут помещаться в таблицу `users`, содержащую следующие четыре поля:

- `id_users` — первичный ключ таблицы, значение которого автоматически генерируется при помощи механизма `AUTO_INCREMENT`;
- `name` — имя пользователя;
- `pass` — пароль пользователя, подвергающийся необратимому кодированию по алгоритму MD5;
- `putdate` — время регистрации пользователя.

Оператор `CREATE TABLE`, создающий таблицу `users`, представлен в листинге 10.12.

Листинг 10.12. Таблица `users`

```
CREATE TABLE users (
  id_user INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pass TINYTEXT NOT NULL,
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_user)
)
```

Для доступа к СУБД MySQL создадим конфигурационный файл config.php, который в случае возникновения проблем с соединением или выбором текущей базы данных будет генерировать исключение `ExceptionMySQL` (листинг 10.13).

Листинг 10.13. Создание соединения с СУБД MySQL

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
```

```
  // Имя базы данных на хостинге или локальной машине
  $dbname = "oop";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx)
  {
    throw new ExceptionMySQL(mysql_error(),
                             "connection",
                             "Невозможно установить
                              соединение с MySQL-сервером");
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) )
  {
    throw new ExceptionMySQL(mysql_error(),
                             "connection",
                             "Ошибка выбора базы данных");
  }

  // Устанавливаем кодировку,
  // в которой данные будут отправляться MySQL-серверу
  @mysql_query("SET NAMES 'cp1251'");
?>
```

Пока доступно лишь текстовое поле `text`, поэтому при создании HTML-формы создадим два обязательных для заполнения элемента управления:

- `$name` — имя пользователя;
- `$pass` — пароль пользователя.

В дальнейшем, когда будет разработан класс для поля типа `password`, можно будет использовать специализированный элемент управления для ввода пароля.

В листинге 10.14 представлена возможная реализация HTML-формы для регистрации нового пользователя.

Замечание

Для оформления страницы в приложении используются файлы utils/top.php и utils/bottom.php. Найти их можно на компакт-диске, поставляемом вместе с книгой. Более подробно эти файлы описываются в *главе 11*.

Листинг 10.14. Регистрация нового пользователя

```
<?php
  require_once("class/class.field.php");
  // Подключаем класс текстового поля
  require_once("class/class.field.text.php");
  // Подключаем класс формы
  require_once("class/class.forms.php");
  // Исключения
  require_once("class/exception.object.php");
  require_once("class/exception.member.php");
  require_once("class/exception.mysql.php");

  // Параметры формы
  $button_name = "Добавить";
  $class_name = "field";

  ////////////////////////////////////////
  // 1. Формирование HTML-формы
  ////////////////////////////////////////
  $name = new field_text("name",
                         "Имя",
                          true,
                          $_POST['name']);
  $pass = new field_text("pass",
                         "Пароль",
                          true,
                          $_POST['pass']);
  try
  {
    $form = new form(array("name" => $name,
                           "pass" => $pass),
                     $button_name,
                     $class_name);
  }
  catch(ExceptionObject $exc)
  {
    // Перехватываем исключение, если производится
    // попытка передать классу form некорректный
    // элемент управления

    // Включаем заголовок страницы
    require_once("utils/top.php");
```

```
    echo "<p class=help>Произошла исключительная
          ситуация (ExceptionObject) — попытка
          использования в качестве элемента управления
          объекта, класс которого не является
          производным от базового класса field.
          {$exc->getMessage()}.</p>";
    echo "<p class=help>Ошибка в файле {$exc->getFile()}
          в строке {$exc->getLine()}.</p>";

    // Включаем завершение страницы
    require_once("utils/bottom.php");
    exit();
  }

  ////////////////////////////////////////
  // 2. Обработчик HTML-формы
  ////////////////////////////////////////
  if(!empty($_POST))
  {
    try
    {
      // Устанавливаем соединение с базой данных
      require_once("config.php");
      // Проверяем корректность заполнения HTML-формы
      // и обрабатываем текстовые поля
      $error = $form->check();
      if(empty($error))
      {
        // Записываем полученные результаты в таблицу
        $query = "INSERT INTO users
                  VALUES(NULL,
                         '{$form->fields[name]->value}',
                         MD5('{$form->fields[pass]->value}'),
                         NOW())";
        if(!mysql_query($query))
        {
          throw new ExceptionMySQL(mysql_error(),
                                   $query,
                                   "Ошибка регистрации пользователя");
        }

        // Перегружаем страницу для сброса POST-данных
        header("Location: $_SERVER[PHP_SELF]");
```

```
        exit();
      }
    }
    catch(ExceptionMember $exc)
    {
      // Перехватываем исключение, если выполняется
      // обращение к несуществующему элементу управления

      // Включаем заголовок страницы
      require_once("utils/top.php");

      echo "<p class=help>Произошла исключительная
            ситуация (ExceptionMember) — попытка
            обращения к несуществующему члену класса.
            {$exc->getMessage()}.</p>";
      echo "<p class=help>Ошибка в файле {$exc->getFile()}
            в строке {$exc->getLine()}.</p>";

      // Включаем завершение страницы
      require_once("utils/bottom.php");
      exit();
    }
    catch(ExceptionMySQL $exc)
    {
      // Обрабатываем исключения, возникающие при
      // обращении к СУБД MySQL

      // Включаем заголовок страницы
      require_once("utils/top.php");

      echo "<p class=help>Произошла исключительная
            ситуация (ExceptionMySQL) при обращении
            к СУБД MySQL.</p>";
      echo "<p class=help>{$exc->getMySQLError()}<br>
           ".nl2br($exc->getSQLQuery())."</p>";
      echo "<p class=help>Ошибка в файле {$exc->getFile()}
            в строке {$exc->getLine()}.</p>";

      // Включаем завершение страницы
      require_once("utils/bottom.php");
      exit();
    }
  }
```

```
  ////////////////////////////////////////
  // 3. Видимая часть страницы
  ////////////////////////////////////////
  // Включаем заголовок страницы
  require_once("utils/top.php");

  // Выводим сообщения об ошибках, если они имеются
  if(!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("utils/bottom.php");
?>
```

Как видно из листинга 10.14, приложение для регистрации пользователей условно можно разбить на три части:

1. Формирование HTML-формы.
2. Обработчик HTML-формы.
3. Видимая часть страницы.

В первой части, посвященной формированию HTML-формы, создаются два текстовых поля `$name` и `$pass` для ввода имени пользователя и его пароля соответственно. Оба элемента передаются объекту `$form` в качестве элементов массива.

Для того чтобы сообщить классу `field_text`, что элементы обязательны для заполнения, третьему параметру присваивается значение `true`. Четвертому параметру присваивается значение элемента управления из суперглобального массива `$_POST`. Это позволяет сохранить ранее введенные пользователем данные, если обработчик нашел ошибки ввода. Дело в том, что за время работы приложения объект формы создается два раза: первый раз — при отображении HTML-формы, второй — при обработке результатов ввода. Если метод `$form->check()` из второй части приложения возвращает хотя бы одно сообщение об ошибке, обработчик опускается и вместо него выводится HTML-форма с ранее введенными значениями и сообщениями об ошибках. В листинге 10.15 демонстрируется ситуация, в которой пользователь не заполнил поле для ввода пароля.

Замечание

Массив `$error` можно дополнить своими собственными сообщениями об ошибках, например, результатами проверки совпадения имени нового пользователя с зарегистрированными ранее. Если в таблице `users` обнаруживается запись с совпадающим именем, можно добавить в массив `$error` соответствующее сообщение. При этом обработчик не сработает, а клиенту среди прочих будет выведено сообщение о том, что пользователь с таким именем уже зарегистрирован.

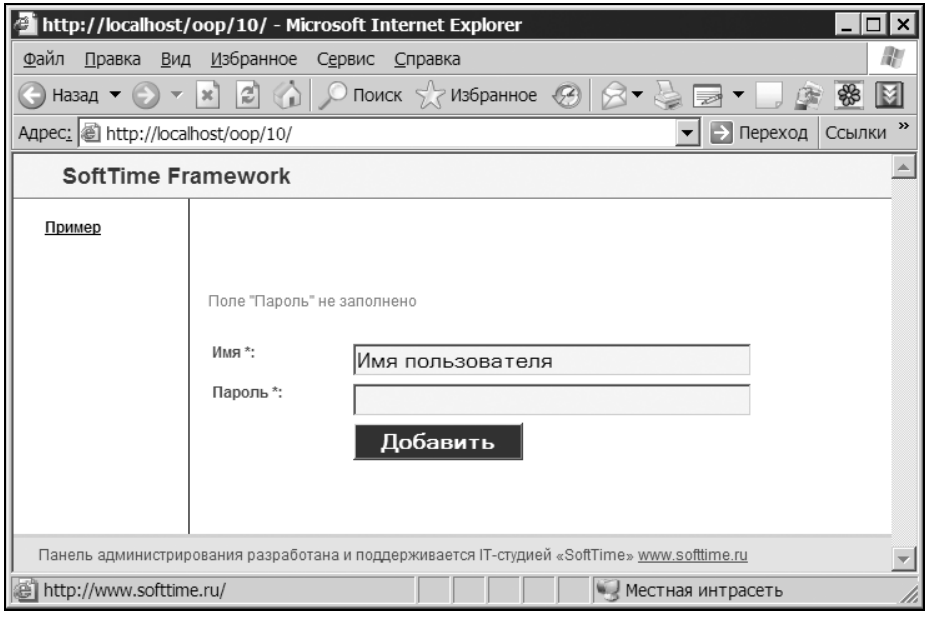


Рис. 10.10. Если одно из обязательных полей не заполнено, ранее введенные данные сохраняются

Участок кода, ответственный за создание объекта класса `form`, помещается в контролируемый блок `try ... catch`, который позволяет обработать исключительную ситуацию, связанную с попыткой передать конструктору класса `form` массива элементов управления, содержащего некорректные объекты элементов управления. Для того чтобы продемонстрировать возникновение исключительной ситуации, добавим в массив элементов управления несуществующий объект `third` (листинг 10.15).

Замечание

В листинге 10.15 приведен только тот фрагмент программы, который подвергся изменению и отличается от представленного в листинге 10.14.

Листинг 10.15. Попытка передачи конструктору класса `form` объекта несуществующего элемента управления `third`

```
<?php
  ...
  try
  {
    $form = new form(array("name" => $name,
                           "pass" => $pass,
                           "third" => $third),
                     $button_name,
                     $class_name);
  }
  catch(ExceptionObject $exc)
  {
    ...
?>
```

Результатом работы программы из листинга 10.15 будет выполнение фрагмента кода из catch-блока (рис. 10.11).

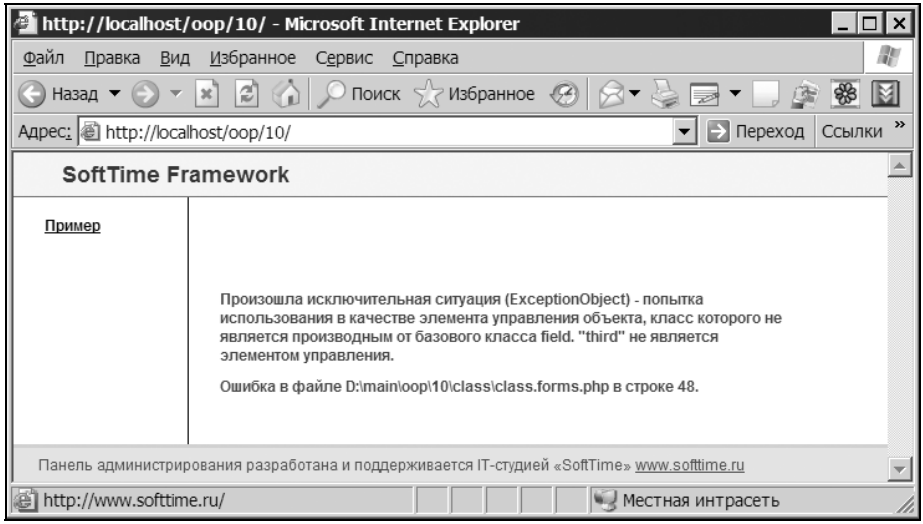


Рис. 10.11. Обработка исключения `ExceptionObject`

Второй блок (обработчик HTML-формы) в листинге 10.14 вступает в действие, только если скрипту передаются данные методом POST. В контролируемом блоке устанавливается соединение с СУБД MySQL, данные HTML-формы автоматически обрабатываются и с их помощью формируется запрос INSERT на вставку новой записи. Если выполнение запроса заканчивается

неудачей, генерируется исключение `ExceptionMySQL`, которое перехватывается catch-обработчиком (рис. 10.12).

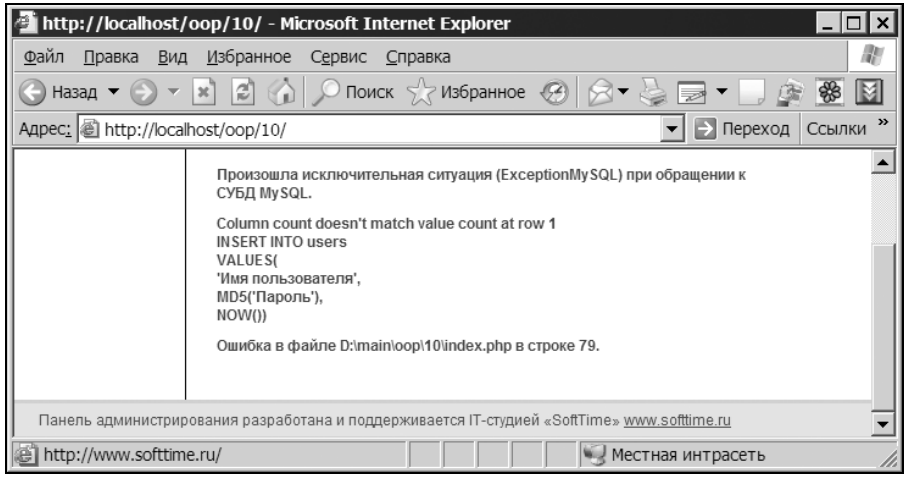


Рис. 10.12. Обработка исключения `ExceptionMySQL`

Последняя третья часть ответственна за вывод HTML-формы и сообщений об ошибках. Ошибки выводятся в цикле `foreach` (если массив `$error` содержит хотя бы одно сообщение), а HTML-форма выводится при помощи метода `print_form()` класса `form`.

10.8. Поле для пароля. Класс *field_password*

Для ввода пароля обычно используют не текстовое поле, как это было продемонстрировано в *разделе 10.9*, а специальное поле типа `password`, которое совпадает по атрибутам и внешнему виду с текстовым полем `text`, однако вводимый текст остается скрытым за символами звездочек или точек.

Так как атрибуты элементов управления `text` и `password` аналогичны, то класс поля для пароля `field_password` можно унаследовать от разработанного ранее в *разделе 10.5* класса `field_text` (листинг 10.16).

Листинг 10.16. Класс поля для пароля `field_password`

```
<?php
  ////////////////////////////////////////////////////////
  // Текстовое поле для пароля password
  ////////////////////////////////////////////////////////

  class field_password extends field_text
```

```
  {
    // Конструктор класса
    function __construct($name,
                  $caption,
                  $is_required = false,
                  $value = "",
                  $maxlength = 255,
                  $size = 41,
                  $parameters = "",
                  $help = "",
                  $help_url = "")
    {
      // Вызываем конструктор базового класса field_text
      // для инициализации его данных
      parent::__construct($name,
                  $caption,
                  $is_required,
                  $value,
                  $maxlength,
                  $size,
                  $parameters,
                  $help,
                  $help_url);
       // Класс field_text присваивает члену type
       // значение text, для пароля этом члену
       // следует присвоить значение password
       $this->type = "password";
    }
  }
?>
```

Как видно из листинга 10.16, класс `field_password` не перегружает методы `check()` и `get_html()`, т. к. они реализованы в базовом классе `field_text`. Можно было бы не перегружать и конструктор класса, т. к. количество и характер параметров совпадают, однако в классе `field_text` значение члена `$type` указано явно (`text`), а поле, содержащее пароль, должно иметь тип `password`. Перегрузка конструктора позволяет изменить значение защищенного члена `$type` на новое.

Иерархия классов элементов управления HTML-формы теперь выглядит так, как это представлено на рис. 10.13.

В листинге 10.17 приводится фрагмент системы регистрации пользователя, рассмотренной ранее в листинге 10.14, в которой поле для пароля формируется при помощи нового класса `field_password`.

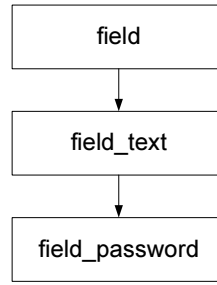


Рис. 10.13. Иерархия классов элементов управления

Листинг 10.17. Использование класса `field_password`

```
<?php
  ...
  ////////////////////////////////////////
  // 1. Формирование HTML-формы
  ////////////////////////////////////////
  $name = new field_text("name",
                         "Имя",
                          true,
                          $_POST['name']);
  $pass = new field_password("pass",
                         "Пароль",
                          true,
                          $_POST['pass']);
  ...
?>
```

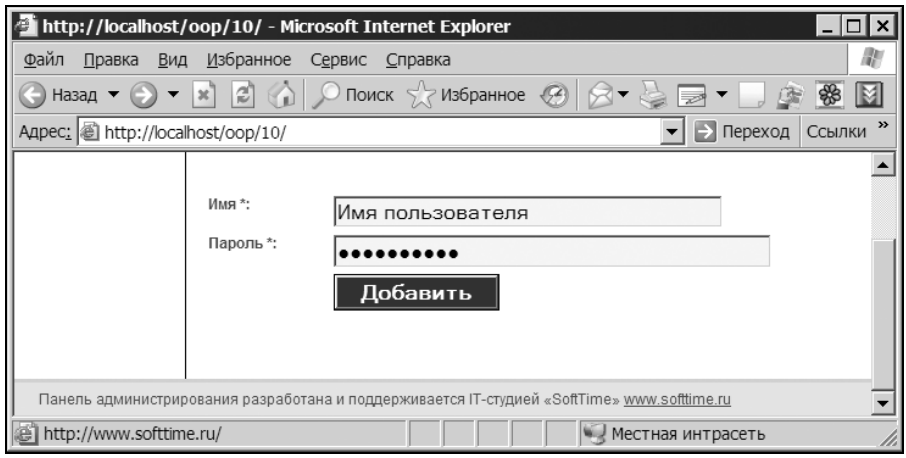


Рис. 10.14. Использование для ввода пароля специализированного поля

Результат модификации приложения для регистрации пользователей продемонстрирован на рис. 10.14.

10.9. Поле для ввода английского текста. Класс *field_text_english*

На текст, вводимый в текстовое поле, зачастую может накладываться большое количество ограничений, связанных с особенностями текущего Web-приложения. Одно из распространенных ограничений — требование, чтобы вводимый текст содержал только символы латинского алфавита.

Создадим новый класс `field_text_english`, который унаследуем от класса текстового поля `field_text` (рис. 10.15).

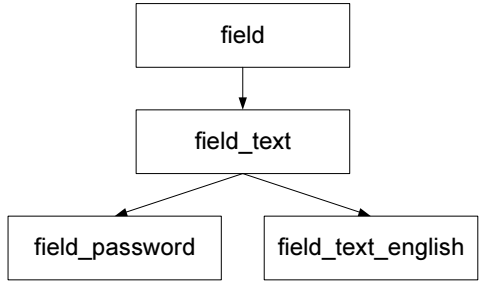


Рис. 10.15. Иерархия классов элементов управления

В листинге 10.18 приводится реализация класса `field_text_english`. Так как конструктор и метод `get_html()` класса `field_text_english` полностью эквивалентны методам класса `field_text`, перегрузке подвергается только метод `check()`. При помощи регулярных выражений он осуществляет проверку, не содержит ли введенный текст символов, отличных от латинских.

Листинг 10.18. Класс поля с английским текстом `field_text_english`

```
<?php
  ////////////////////////////////////////////////////////////
  // Текстовое поле для английского текста
  ////////////////////////////////////////////////////////////

  class field_text_english extends field_text
  {
    // Метод, проверяющий корректность переданных данных
    function check()
```

```
    {
      // Обезопасить текст перед внесением в базу данных
      if (!get_magic_quotes_gpc())
      {
        $this->value = mysql_escape_string($this->value);
      }
      if($this->is_required) $pattern = "|^[a-z]+$|i";
      else $pattern = "|^[a-z]*$|i";

      // Проверяем символы в поле value
      // на принадлежность латинскому алфавиту
      if(!preg_match($pattern, $this->value))
      {
        return "Поле \"{$this->caption}\"
                должно содержать только символы латинского алфавита";
      }

      return "";
    }
  }
?>
```

10.10. Поле для ввода целых чисел. Класс *field_text_int*

Очень часто встречается задача ввода целых чисел, принадлежащих определенному диапазону. Создать такое поле можно, расширяя класс `field_text` путем ввода в него двух новых членов:

- `$min_value` — минимальное значение, которое может принимать вводимое пользователем число;
- `$max_value` — максимальное значение, которое может принимать вводимое пользователем число.

Будем считать, что если оба значения совпадают, то ограничения по диапазону отсутствуют, и класс `field_text_int` будет лишь приводить текст к целому значению.

На рис. 10.16 приводится схема иерархии классов и место класса `field_text_int` в этой иерархии.

В листинге 10.19 приводится возможная реализация класса `field_text_int`.

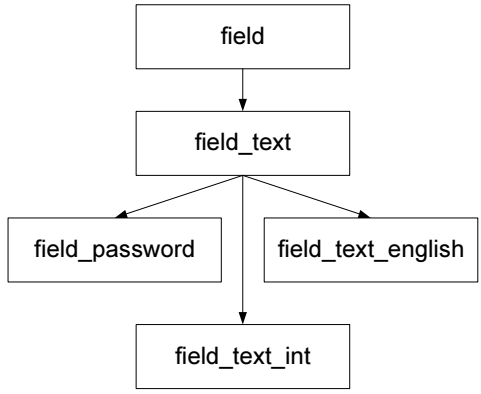


Рис. 10.16. Место класса `field_text_int` в иерархии классов элементов управления

Листинг 10.19. Класс для ввода целых чисел `field_text_int`

```
<?php
  ////////////////////////////////////////////////////////////
  // Текстовое поле с целочисленными значениями
  ////////////////////////////////////////////////////////////

  class field_text_int extends field_text
  {
    // Минимальное значение поля
    protected $min_value;
    // Максимальное значение поля
    protected $max_value;
    // Конструктор класса
    function __construct($name,
                  $caption,
                  $is_required = false,
                  $value = "",
                  $min_value = 0,
                  $max_value = 0,
                  $maxlength = 255,
                  $size = 41,
                  $parameters = "",
                  $help = "",
                  $help_url = "")
    {
      // Вызываем конструктор базового класса field_text
      // для инициализации его данных
```

```
    parent::__construct($name,
                $caption,
                $is_required,
                $value,
                $maxlength,
                $size,
                $parameters,
                $help,
                $help_url);
    $this->min_value = intval($min_value);
    $this->max_value = intval($max_value);

    // Минимальное значение должно быть больше максимального
    if($this->min_value > $this->max_value)
    {
      throw Exception("Минимальное значение должно
                       быть больше максимального
                       значения. Поле \"".$this->caption."\".");
    }
  }

  // Метод, проверяющий корректность переданных данных
  function check()
  {
    $pattern = "|^[-\d]*$|i";
    if($this->is_required)
    {
      // Проверяем поле value на максимальное и минимальное значение
      if($this->min_value != $this->max_value)
      {
        if($this->value < $this->min_value ||
           $this->value > $this->max_value)
        {
          return "Поле \"".$this->caption."\"
                  должно быть больше ".$this->min_value."
                  и меньше ".$this->max_value."";
        }
      }
      $pattern = "|^[-\d]+$|i";
    }
    // Проверяем, является ли введенное значение
    // целым числом
```

```
      if(!preg_match($pattern, $this->value))
      {
        return "Поле \"".$this->caption."\"
                должно содержать только цифры";
      }

      return "";
    }
  }
?>
```

При создании объекта проверяется, чтобы минимальное значение `$min_value` было меньше или равно максимальному значению `$max_value`, в противном случае генерируется исключение.

Так как класс содержит два дополнительных члена, конструктор подвергается перегрузке для их корректной инициализации. Помимо этого, перегружается метод `check()`, который контролирует, чтобы введенное число содержало только цифры, а значение не выходило за рамки интервала (`$min_value`, `$max_value`). В методе `check()`для проверки, является ли введенное значение целым числом, используются два регулярных выражения. Первое регулярное выражение `"|^[-\d]+$|i"` используется, когда поле обязательно (`$is_required == true`), и требует, чтобы значение содержало хотя бы одно целое значение или знак минус `"-"`. Второе регулярное выражение `"|^[-\d]*$|i"` используется, когда поле необязательно (`$is_required == true`), и требует, чтобы значение было либо пустым, либо содержало цифры и знак минус `"-"`.

10.11. Поле для ввода электронной почты. Класс *field_text_email*

Другой распространенной задачей является ввод адреса электронной почты, который имеет строгий фиксированный формат. Каждый раз формировать регулярное выражение для проверки вводимого значения утомительно, поэтому разумно дополнить иерархию классов дополнительным классом `field_text_email`, который будет осуществлять автоматическую проверку синтаксиса адреса электронной почты (рис. 10.17).

При наследовании класса `field_text_email` от класса `field_text` достаточно перегрузить метод `check()`, в котором при помощи регулярного выражения проверить формат адреса электронной почты (листинг 10.20).

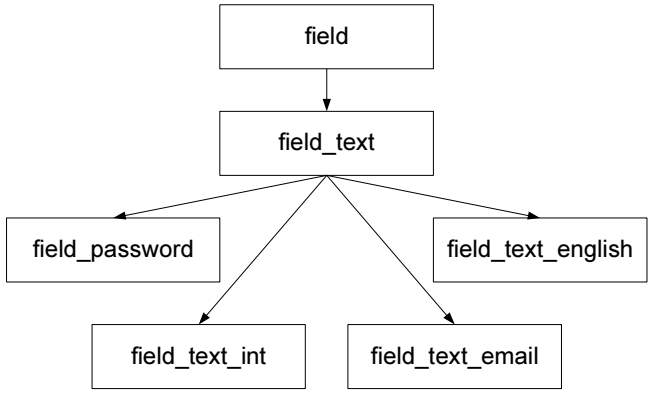


Рис. 10.17. Место класса `field_text_email` в иерархии классов элементов управления

Листинг 10.20. Класс для ввода электронной почты `field_text_email`

```
<?php
  ////////////////////////////////////////////////////////////
  // Текстовое поле для e-mail
  ////////////////////////////////////////////////////////////

  class field_text_email extends field_text
  {
    // Метод, проверяющий корректность переданных данных
    function check()
    {
      if($this->is_required || !empty($this->value))
      {
        $pattern = "#^[-0-9a-z_\.]+@[0-9a-z_^\.]+\.[a-z]{2,6}$#i";
        if (!preg_match($pattern, $this->value))
        {
          return "Введите e-mail в виде <i>something@server.com</i>";
        }
      }

      return "";
    }
  }
?>
```

Будем исходить из того, что адрес должен иметь вид *something@server.com*. У адреса имеются две составляющие — имя пользователя и имя домена, ко-

торые разделены символом "@". В имени пользователя могут присутствовать символы латинского алфавита в нижнем и верхнем регистрах, цифры, знаки подчеркивания, минуса и точки. Для проверки разделителя между именем пользователя и именем домена в выражение требуется добавить @. Далее следует доменное имя, которое тоже может состоять из символов латинского алфавита в нижнем и верхнем регистрах, цифр, знаков препинания, минуса и точки. Таким образом, регулярное выражение, проверяющее имя пользователя и наличие разделителя и наличия доменного имени, имеет следующий вид:

```
"[-0-9a-z_]+@[0-9a-z_^\.]+"
```

Для проверки доменного имени первого уровня (.ru, .com) необходимо добавить следующее выражение:

```
"\.[a-z]{2,6}"
```

Замечание

Важно отметить, что недавно были добавлены несколько доменных имен первого уровня, в частности, info и travel, поэтому трех символов, как раньше, стало недостаточно. Для проверки правильности ввода e-mail под домен первого уровня необходимо зарезервировать от двух до шести символов.

Символ "." в регулярных выражениях используется для обозначения любого символа, поэтому для поиска соответствия точки в регулярном выражении этот символ экранируется обратным слэшем: "\.".

Объединяя эти строки, можно получить следующее регулярное выражение в формате Perl для проверки адресов электронной почты:

```
"|[-0-9a-z_]+@[0-9a-z_^\.]+\.[a-z]{2,6}|i"
```

Замечание

Модификатор "i" в регулярном выражении сообщает интерпретатору, что поиск соответствия должен выполняться без учета регистра.

Описанное регулярное выражение интерпретирует e-mail, однако если строка должна содержать адрес электронной почты и ничего более, регулярное выражение необходимо снабдить привязкой к началу ^ и концу текста $:

```
"|^[-0-9a-z_]+@[0-9a-z_^\.]+\.[a-z]{2,6}$|i"
```

10.12. Текстовая область. Класс *field_textarea*

Текстовая область предназначена для ввода нескольких строк текста и создается при помощи парного тега <textarea>. В отличие от текстового поля

<input> в текстовой области допускается создание переводов строк (абзацев). Тег <textarea> имеет следующий синтаксис:

```
<textarea>...</textarea>
```

Как можно заметить, текстовая область определяется не с помощью атрибута, а при помощи собственного тега. Допустимые атрибуты тега <textarea> приводятся в табл. 10.4.

***Таблица 10.4.** Атрибуты текстовой области*

Атрибут	Описание
cols	Ширина текстовой области, равная количеству символов моноширинного шрифта
disabled	Блокирует возможность редактирования и выделения текста в текстовой области, при этом сам элемент управления окрашивается в серый цвет
name	Имя элемента управления, предназначенное для идентификации в обработчике. Имя должно быть уникальным в пределах формы, т. е. отличаться от имен других элементов управления, поскольку используется как ключ для доступа к значению поля в обработчике
readonly	Блокирует возможность редактирования текстовой области, однако, в отличие от атрибута disabled, цвет элемента управления остается неизменным
rows	Высота текстовой области, равная количеству отображаемых строк без прокрутки содержимого
wrap	Требует от браузера, чтобы перенос текста на следующую строку осуществлялся только при нажатии клавиши <Enter>, в противном случае должна появляться горизонтальная полоса прокрутки

Для моделирования текстовой области создадим класс field_textarea, который унаследуем от базового класса field (рис. 10.18).

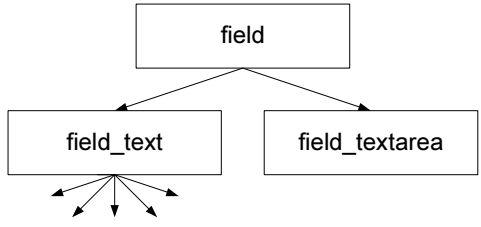


Рис. 10.18. Место класса field_textarea в иерархии классов элементов управления

Как видно табл. 10.4, помимо членов, входящих в обобщенный класс field, для класса field_textarea необходимо предусмотреть пять дополнительных

членов для задания ширины и высоты текстовой области, а также логические члены для атрибутов `disabled`, `readonly` и `wrap`. В листинге 10.21 приводится возможная реализация класса `field_textarea`, наследуемого от базового класса `field`.

Листинг 10.21. Класс текстовой области

```
<?php
  ////////////////////////////////////////////////////////
  // Текстовая область textarea
  ////////////////////////////////////////////////////////

  class field_textarea extends field
  {
    // Размер текстового поля
    protected $cols;
    // Максимальное количество вводимых данных
    protected $rows;
    // Блокировка поля
    protected $disabled;
    // Только для чтения
    protected $readonly;
    // Отсутствие автоперевода строк
    protected $wrap;

    // Конструктор класса
    function __construct($name,
                  $caption,
                  $id_required = false,
                  $value = "",
                  $cols = 35,
                  $rows = 7,
                  $disabled = false,
                  $readonly = false,
                  $wrap = false,
                  $parameters = "",
                  $help = "",
                  $help_url = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct($name,
                  "textarea",
                  $caption,
```

```
                $id_required,
                $value,
                $parameters,
                $help,
                $help_url);
    // Инициируем члены класса field_text
    $this->cols     = $cols;
    $this->rows     = $rows;
    $this->disabled = $disabled;
    $this->readonly = $readonly;
    $this->wrap     = $wrap;
  }

  // Метод для возврата имени поля
  // и самого тега элемента управления
  function get_html()
  {
    // Если элементы оформления не пусты, учитываем их
    if(!empty($this->css_style))
    {
      $style = "style=\"".$this->css_style."\"";
    }
    else $style = "";
    if(!empty($this->css_class))
    {
      $class = "class=\"".$this->css_class."\"";
    }
    else $class = "";

    // Если определены размеры, учитываем их
    if(!empty($this->cols))
    {
      $cols = "cols=".$this->cols;
    }
    else $cols = "";
    if(!empty($this->rows))
    {
      $rows = "rows=".$this->rows;
    }
    else $rows = "";

    // Атрибуты текстовой области
    if($this->disabled) $disabled = "disabled";
    else $disabled = "";
```

```
    if($this->readonly) $readonly = "readonly";
    else $readonly = "";
    if($this->wrap) $wrap = "wrap";
    else $wrap = "";

    if(is_array($this->value))
    {
      $this->value = implode("\r\n",$this->value);
    }
    if(!get_magic_quotes_gpc())
    {
      $output = str_replace('\r\n',"\r\n",$this->value);
    }
    else $output = $this->value;
    $tag = "<textarea $style $class
            name=\"".$this->name."\"
            $cols $rows $disabled $readonly $wrap>".
            htmlspecialchars($output, ENT_QUOTES).
           "</textarea>\n";

    // Если поле обязательно для заполнения,
    // отмечаем этот факт
    if($this->is_required) $this->caption .= " *";

    // Формируем подсказку, если она имеется
    $help = "";
    if(!empty($this->help))
    {
      $help .= "<span style='color:blue'>".
                nl2br($this->help)."</span>";
    }
    if(!empty($help)) $help .= "<br>";
    if(!empty($this->help_url))
    {
      $help .= "<span style='color:blue'>
                  <a href=".$this->help_url.">помощь</a>
                </span>";
    }

    return array($this->caption, $tag, $help);
  }

  // Метод, проверяющий корректность переданных данных
  function check()
```

```
    {
      // Обеспечение безопасности текста
      // перед внесением в базу данных
      if (!get_magic_quotes_gpc())
      {
        $this->value = mysql_escape_string($this->value);
      }
      if($this->is_required)
      {
        if(empty($this->value))
        {
          return "Поле \"".$this->caption."\" не заполнено";
        }
      }

      return "";
    }
  }
?>
```

Конструктор класса `field_textarea` инициирует члены класса, а также вызывает конструктор базового класса `field` для инициализации его членов классов. Метод `get_html()` формирует фрагменты HTML-кода, которые затем используются классом `form` для создания строки таблицы с элементом управления `<textarea>`. Метод `check()` ничем не отличается от ранее рассмотренного метода для класса текстового поля `field_text`.

В *разделе 10.7* было рассмотрено простейшее Web-приложение для регистрации пользователей. Расширим возможности этого приложения: пользователь получит возможность ввести при регистрации дополнительную информацию о себе в текстовую область, кроме этого, от него потребуется указать адрес электронной почты, отличный от всех зарегистрированных ранее.

Модифицируем таблицу `users` так, как это продемонстрировано в листинге 10.22.

Листинг 10.22. Модифицированная таблица `users`

```
CREATE TABLE users (
  id_user INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pass TINYTEXT NOT NULL,
  email TINYTEXT NOT NULL,
  description TEXT NOT NULL,
```

```
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_user)
)
```

Дополнительное поле `email` предназначено для хранения адреса электронной почты, а поле `description` — дополнительной информации о пользователе.

Чтобы сократить объем приложения, укажем все файлы с классами элементов управления, исключений и HTML-форм в специальном файле config/class.config.php. Кроме этого, для обработчиков исключений введем специальные файлы exception_member.php, exception_mysql.php и exception_object.php, которые включим в catch-блок при помощи инструкции `require`.

Замечание

Файлы config/class.config.php, exception_member.php, exception_mysql.php и exception_object.php можно найти на компакт-диске, поставляемом вместе с книгой.

В листинге 10.23 приводится возможная реализация расширенной регистрации пользователей.

Листинг 10.23. Расширенная регистрация пользователей

```
<?php
  // Подключаем все необходимые классы
  require_once("config/class.config.php");

  // Параметры формы
  $button_name = "Добавить";
  $class_name = "field";

  ////////////////////////////////////////
  // 1. Формирование HTML-формы
  ////////////////////////////////////////
  $name = new field_text("name",
                         "Имя",
                          true,
                          $_POST['name']);
  $pass = new field_password("pass",
                         "Пароль",
                          true,
                          $_POST['pass']);
  $pass_again = new field_password("pass_again",
                         "Повтор пароля",
```

```
                          true,
                          $_POST['pass_again']);
$email = new field_text_email("email",
                         "E-mail",
                          true,
                          $_POST['email']);
$about = new field_textarea("about",
                         "О себе",
                          false, // Поле не обязательное
                          $_POST['about']);
try
{
  $form = new form(array("name"       => $name,
                         "pass"       => $pass,
                         "pass_again" => $pass_again,
                         "email"      => $email,
                         "about"      => $about),
                   $button_name,
                   $class_name);
}
catch(ExceptionObject $exc) { require("exception_object.php"); }

////////////////////////////////////////
// 2. Обработчик HTML-формы
////////////////////////////////////////
if(!empty($_POST))
{
  try
  {
    // Устанавливаем соединение с базой данных
    require_once("config.php");
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();

    // Проверяем идентичность паролей
    if($form->fields['pass']->value !=
       $form->fields['pass_again']->value)
    {
      $error[] = "Неверный пароль";
    }
    // Проверяем, не регистрировался ли ранее пользователь
    // с идентичным электронным адресом
```

```
    $query = "SELECT COUNT(*) FROM users
              WHERE email = '{$form->fields[email]->value}'";
    $mal = mysql_query($query);
    if(!$mal)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                               "Ошибка регистрации пользователя");
    }
    if(mysql_result($mal, 0))
    {
      $error[] = "Пользователь с электронным адресом
                  {$form->fields[email]->value} уже
                  зарегистрирован";
    }

    if(empty($error))
    {
      // Записываем полученные результаты в таблицу
      $query = "INSERT INTO users
                VALUES(NULL,
                       '{$form->fields[name]->value}',
                       MD5('{$form->fields[pass]->value}'),
                       '{$form->fields[email]->value}',
                       '{$form->fields[description]->value}',
                       NOW())";
      if(!mysql_query($query))
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                 "Ошибка регистрации пользователя");
      }

      // Перегружаем страницу для сброса POST-данных
      header("Location: $_SERVER[PHP_SELF]");

      exit();
    }
  }
  catch(ExceptionMember $exc) { require("exception_member.php"); }
  catch(ExceptionMySQL $exc) { require("exception_mysql.php"); }
}
```

```
  ////////////////////////////////////////
  // 3. Видимая часть страницы
  ////////////////////////////////////////
  // Включаем заголовок страницы
  require_once("utils/top.php");

  // Выводим сообщения об ошибках, если они имеются
  if(!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("utils/bottom.php");
?>
```

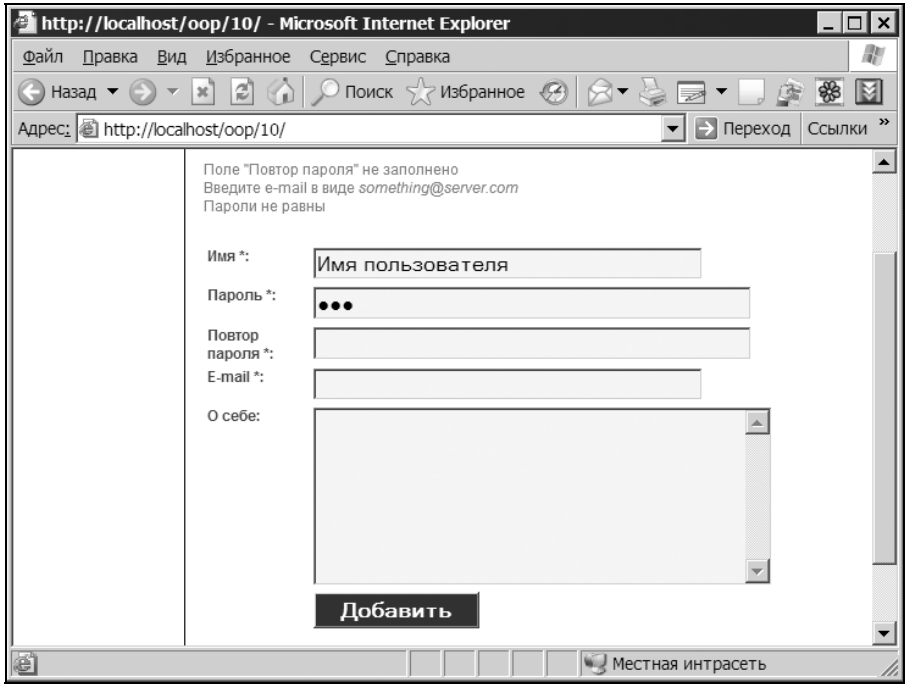


Рис. 10.19. Расширенная регистрация пользователей

Результат обращения к скрипту из листинга 10.23 продемонстрирован на рис. 10.19.

Для формирования HTML-формы используется объект текстового поля `field_text`, два объекта класса `field_password`, поле для ввода пароля `field_text_email` и текстовая область `field_textarea`.

Отличительной особенностью данного приложения являются дополнительные условия, размещенные между вызовом метода `check()` объекта `$form` и началом обработки. Это позволяет дополнить список сообщений об ошибках, возвращаемых методом `check()`, отрицательными результатами проверок, реализовать которые в классе `form` не представляется возможным.

Замечание

Как и класс текстового поля `field_text`, класс текстовой области `field_textarea` можно использовать в качестве базового класса для создания новых элементов управления с различными ограничениями, накладываемыми на формат вводимой информации.

10.13. Скрытое поле. Класс *field_hidden*

Скрытое поле служит для передачи незаметно от пользователя служебной информации. Скрытые поля не отображаются на странице.

Замечание

Дело в том, что протокол HTTP не сессионный, и проблема передачи информации от страницы к странице стоит в нем достаточно остро. Помимо скрытых полей передавать информацию можно при помощи механизма сессий (session), однако такой подход не всегда удобен, т. к. сессии носят глобальный характер и могут искажаться другой частью приложения.

Скрытое поле создается при помощи input-тега, атрибут `type` которого принимает значение `hidden`:

```
<input type="hidden">
```

Помимо атрибута `type`, скрытое поле поддерживает атрибут `name` для уникального имени элемента управления и атрибут `value` для его значения.

Для представления скрытого поля унаследуем от базового класса `field` производный класс `field_hidden` (рис. 10.20).

В листинге 10.24 представлена реализация класса `field_hidden`. Так как у скрытых полей отсутствуют представления, его конструктор имеет только три параметра:

- `$name` — имя скрытого поля;

- `$is_required` — логическое значение, определяющее обязательность элемента управления;
- `$value` — значение скрытого поля.

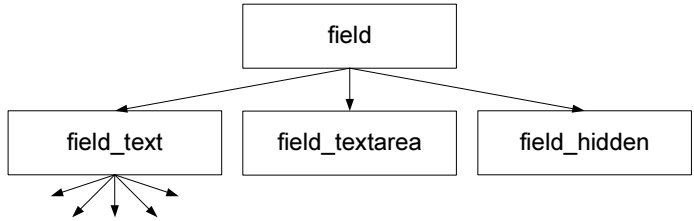


Рис. 10.20. Место класса `field_hidden` в иерархии классов элементов управления

Листинг 10.24. Класс скрытого поля `field_hidden`

```
<?php
  ////////////////////////////////////////////////////////////
  // Скрытое поле hidden
  ////////////////////////////////////////////////////////////

  class field_hidden extends field
  {
    // Конструктор класса
    function __construct($name,
                         $id_required = false,
                         $value = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct($name,
                  "hidden",
                  "-",
                  $id_required,
                  $value,
                  $parameters,
                  "",
                  "");
    }

    // Метод для возврата имени названия поля
    // и самого тега элемента управления
    function get_html()
```

```
    {
      $tag = "<input type=\"".$this->type."\"
                     name=\"".$this->name."\"
                     value=\"".
                     htmlspecialchars($this->value, ENT_QUOTES)."\">\n";
      return array("", $tag);
    }
    // Метод, проверяющий корректность переданных данных
    function check()
    {
      // Обеспечение безопасности текста
      // перед внесением в базу данных
      if (!get_magic_quotes_gpc())
      {
        $this->value = mysql_escape_string($this->value);
      }
      if($this->is_required)
      {
        if(empty($this->value)) return "Скрытое поле не заполнено";
      }

      return "";
    }
  }
?>
```

Метод `check()` осуществляет стандартную проверку наличия в поле непустого значения и полностью совпадает с методом `check()` рассмотренного ранее класса `field_text`. Метод `get_html()`, формирующий HTML-фрагменты, передает классу `form` массив из двух элементов, причем первый из них, используемый для описания, пуст. Во избежание формировании лишней пустой строки при включении в HTML-форму скрытого поля необходимо так модифицировать метод `print_form()` класса `form` (листинг 10.11), чтобы поле типа `hidden` обрабатывалось специальным образом. Для этого введем в цикл обработки элементов управления оператор `switch()`, который, в зависимости от значения члена `$type`, будет выбирать тот или иной способ представления элементов управления (листинг 10.25).

Замечание

Полный вариант класса `form`, обрабатывающий все рассматриваемые в книге элементы управления, можно найти на компакт-диске, поставляемом вместе с книгой.

Листинг 10.25. Условный вывод элементов управления в методе `print_form()` класса `form`

```
<?php
  ...
  switch($obj->type)
  {
    case "hidden":
      // Скрытое поле
      echo $tag;
      break;
    default:
      // Элементы управления по умолчанию
      echo "<tr>
              <td width=100
                  $style $class valign=top>$caption:</td>
              <td $style $class valign=top>$tag </td>
            </tr>\n";
      if(!empty($help))
      {
        echo "<tr>
                <td> </td>
                <td $style $class valign=top>$help</td>
              </tr>";
      }
      break;
  }
  ...
?>
```

Оператор `switch()` проверяет содержимое члена `$type` класса элемента управления, и, если класс относится к скрытому полю (`hidden`), обрабатывает его специальным образом, не создавая строку таблицы. Все остальные элементы по умолчанию обрабатываются как текстовые поля или области.

Замечание

Использование оператора `switch` в объектно-ориентированных программах считается признаком плохого тона, и следовало бы вместо `switch` использовать набор классов-адаптеров. Однако в книге мы будем использовать в классе `form` оператор `switch`, чтобы снизить количество используемых классов и повысить читабельность кода.

10.14. Скрытое поле для целых значений. Класс *field_hidden_int*

Большое значение, особенно при работе с СУБД MySQL, приобретают скрытые поля, передающие целочисленные значения. Дело в том, что записям в таблицах баз данных присваиваются первичные ключи, при помощи которых можно связать таблицы друг с другом или отличить одну запись от другой. Значения первичных ключей часто становятся объектом передачи в скрытых полях. Унаследуем от `field_hidden` производный класс `field_hidden`, который будет передавать только целые значения (рис. 10.21).

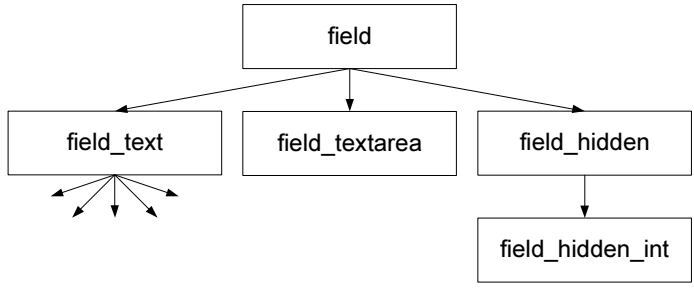


Рис. 10.21. Место класса `field_hidden_int` в иерархии классов элементов управления

В листинге 10.26 представлена возможная реализация класса `field_hidden_int`.

Листинг 10.26. Класс скрытого поля для передачи целочисленных значений `field_hidden_int`

```
<?php
  ////////////////////////////////////////////////////////////
  // Скрытое поле с целочисленными значениями hidden
  ////////////////////////////////////////////////////////////

  class field_hidden_int extends field_hidden
  {
    // Метод, проверяющий корректность переданных данных
    function check()
    {
      if($this->is_required)
      {
        // Поле обязательно к заполнению
        if(!preg_match("|^[\d]+$|",$this->value))
```

```
        {
          return "Скрытое поле должно быть целым числом";
        }
      }
      // Поле не обязательно к заполнению
      if(!preg_match("|^[\d]*$|",$this->value))
      {
        return "Скрытое поле должно быть целым числом";
      }

      return "";
    }
  }
?>
```

Для демонстрации работы скрытых полей обратимся к рассмотренной в *разделе 10.12* системе регистрации пользователей. Пусть на странице выводится список всех пользователей в виде гиперссылок, переход по которым позволяет редактировать все личные данные пользователя (пароль, e-mail, дополнительная информация), исключая имя, служащее для авторизации.

В листинге 10.27 приводится пример скрипта, выводящего список пользователей.

Листинг 10.27. Список пользователей (list.php)

```
<?php
  // Включаем заголовок страницы
  require_once("utils/top.php");

  // Подключаем все необходимые классы
  require_once("config.php");

  try
  {
    $query = "SELECT * FROM users";
    $usr = mysql_query($query);
    if(!$usr)
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка обращения к списку пользователей");

    // Если имеется хотя бы одна запись,
    // выводим список пользователей
```

```
    if(mysql_num_rows($usr))
    {
      while($user = mysql_fetch_array($usr))
      {
        echo "<a href=edituser.php?id_user=$user[id_user]>".
                htmlspecialchars($user['name'], ENT_QUOTES).
             "</a><br>";
      }
    }
  }
  catch(ExceptionMySQL $exc) { require("exception_mysql.php"); }

  // Включаем завершение страницы
  require_once("utils/bottom.php");
?>
```

Каждый из элементов списка указывает на файл edituser.php, которому через GET-параметр передается первичный ключ `id_user`, соответствующий текущему пользователю.

В файле edituser.php (листинг 10.28) формируется HTML-форма, которая позволяет отредактировать данные пользователя. Так как данные передаются методом POST, то первичный ключ следует передать через скрытое поле `id_user`. В противном случае обработчик не сможет определить, какую из записей таблицы `users` следует подвергнуть редактированию.

Листинг 10.28. Форма для редактирования данных пользователя (edituser.php)

```
<?php
  try
  {
    // Подключаем все необходимые классы
    require_once("config/class.config.php");

    // Параметры формы
    $button_name = "Добавить";
    $class_name = "field";

    // Если это первое обращение к HTML-форме,
    // извлекаем информацию из таблицы users
    if(empty($_POST))
    {
      $_GET['id_user'] = intval($_GET['id_user']);
```

```
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  $query = "SELECT * FROM users
            WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
  if(!$usr)
  {
    throw new ExceptionMySQL(mysql_error(),
                             $query,
                             "Ошибка извлечения
                              данных пользователя");
  }
  $_REQUEST = mysql_fetch_array($usr);
  $_REQUEST['pass_again'] = $_REQUEST['pass'];
}

////////////////////////////////////////
// 1. Формирование HTML-формы
////////////////////////////////////////
$pass = new field_password("pass",
                       "Пароль",
                        true,
                        $_REQUEST['pass']);
$pass_again = new field_password("pass_again",
                       "Повтор пароля",
                        true,
                        $_REQUEST['pass_again']);
$email = new field_text_email("email",
                       "E-mail",
                        true,
                        $_REQUEST['email']);
$description = new field_textarea("description",
                       "О себе",
                        false, // Поле не обязательное
                        $_REQUEST['description']);
$id_user = new field_hidden_int("id_user",
                                true,
                                $_REQUEST['id_user']);
try
{
  $form = new form(array("pass"        => $pass,
                         "pass_again"  => $pass_again,
                         "email"       => $email,
```

```
                        "description" => $description,
                        "id_user"     => $id_user),
                    $button_name,
                    $class_name);
}
catch(ExceptionObject $exc) { require("exception_object.php"); }

////////////////////////////////////////
// 2. Обработчик HTML-формы
////////////////////////////////////////
if(!empty($_POST))
{
  try
  {
    // Устанавливаем соединение с базой данных
    require_once("config.php");
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();

    // Проверяем, равны ли пароли
    if($form->fields['pass']->value !=
       $form->fields['pass_again']->value)
    {
      $error[] = "Пароли не равны";
    }
    // Проверяем, не регистрировался ли ранее пользователь
    // с запрашиваемым электронным адресом
    $query = "SELECT COUNT(*) FROM users
              WHERE email = '{$form->fields[email]->value}'";
    $mal = mysql_query($query);
    if(!$mal)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                               "Ошибка обновления
                                пользовательских данных");
    }
    if(mysql_result($mal, 0))
    {
      $error[] = "Пользователь с электронным адресом
                  {$form->fields[email]->value} уже
                  зарегистрирован";
    }
```

```
      if(empty($error))
      {
        // Обновляем запись пользователя
        $query = "UPDATE users
                  SET pass = MD5('{$form->fields[pass]->value}'),
                      email = '{$form->fields[email]->value}',
                   description = '{$form->fields[description]->value}'
                  WHERE id_user = {$form->fields[id_user]->value}";
        if(!mysql_query($query))
        {
          throw new ExceptionMySQL(mysql_error(),
                                   $query,
                                   "Ошибка обновления
                                    пользовательских данных");
        }

        // Перегружаем страницу для сброса POST-данных
        header("Location: list.php");

        exit();
      }
    }
    catch(ExceptionMember $exc) { require("exception_member.php"); }
  }
}
catch(ExceptionMySQL $exc) { require("exception_mysql.php"); }

////////////////////////////////////////
// 3. Видимая часть страницы
////////////////////////////////////////
// Включаем заголовок страницы
require_once("utils/top.php");

// Выводим сообщения об ошибках, если они имеются
if(!empty($error))
{
  foreach($error as $err)
  {
    echo "<span style=\"color:red\">$err</span><br>";
  }
}
// Выводим HTML-форму
$form->print_form();
```

```
  // Включаем завершение страницы
  require_once("utils/bottom.php");
?>
```

В HTML-форме, представленной в листинге 10.28, при формировании объектов элементов управления вместо суперглобального массива `$_POST` используется суперглобальный массив `$_REQUEST`. Это связано с тем, что поля формы должны быть заполнены во время первого обращения к форме, тогда использование суперглобального массива `$_POST` приведет к срабатыванию обработчика. Кроме того, поле `id_user` сначала передается методом GET, а затем — методом POST. В данной ситуации использование суперглобального массива `$_RESUEST` позволяет избежать дополнительных преобразований. Результат работы скрипта из листинга 10.28 представлен на рис. 10.22.

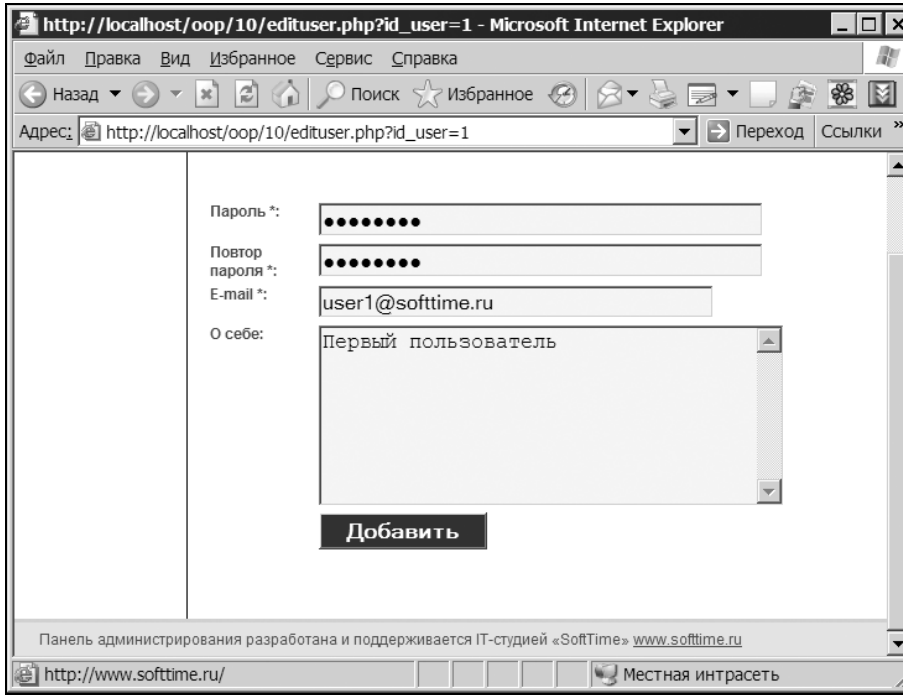


Рис. 10.22. Редактирование пользователя

Скрытое обязательное поле `id_user` защищено от SQL-инъекций, т. к. значение поля автоматически приводится к целому значению при помощи функции `intval()`.

10.15. Флажок. Класс *field_checkbox*

Флажок — это элемент управления, позволяющий представлять логическое значение в HTML-форме, находясь в установленном или снятом состоянии. Синтаксис элемента управления выглядит следующим образом:

```
<input type=checkbox>
```

Помимо атрибута `type`, флажок поддерживает атрибут `name` для уникального имени элемента управления, атрибут `value` для его значения и атрибут `checked`, наличие которого означает, что флажок установлен.

Для представления флажка унаследуем от базового класса `field` производный класс `field_checkbox` (рис. 10.23).

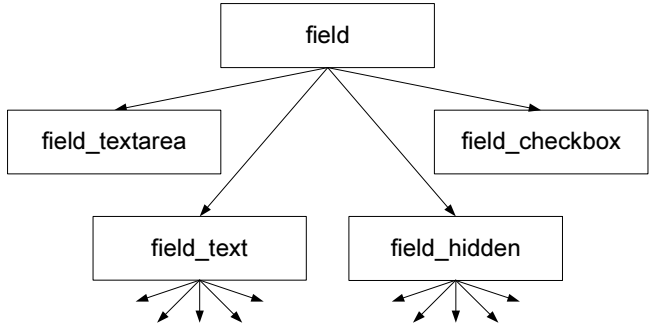


Рис. 10.23. Место класса `field_checkbox` в иерархии классов элементов управления

В листинге 10.29 представлена возможная реализация класса `field_checkbox`. В отличие от предыдущих элементов управления, конструктор класса `field_checkbox` не содержит параметра `$is_required`, определяющего обязательность элемента управления, т. к. это не имеет смысла применительно к флажку.

Листинг 10.29. Класс флажка `field_checkbox`

```
<?php
  ////////////////////////////////////////////////////////////
  // Флажок checkbox
  ////////////////////////////////////////////////////////////

  class field_checkbox extends field
  {
    // Конструктор класса
    function __construct($name,
                         $caption,
```

```
                $value = false,
                $parameters = "",
                $help = "",
                $help_url = "")
  {
    // Вызываем конструктор базового класса field
    // для инициализации его данных
    parent::__construct($name,
                "checkbox",
                $caption,
                false,
                $value,
                $parameters,
                $help,
                $help_url);
    // Инициализируем члены класса
    if($value == "on") $this->value = true;
    else if($value === true) $this->value = true;
    else $this->value = false;
  }

  // Метод для возврата имени названия поля
  // и самого тега элемента управления
  function get_html()
  {
    // Если элементы оформления не пусты, учитываем их
    if(!empty($this->css_style))
    {
      $style = "style=\"".$this->css_style."\"";
    }
    else $style = "";
    if(!empty($this->css_class))
    {
       $class = "class=\"".$this->css_class."\"";
    }
    else $class = "";

    // Проверяем, установлен ли флажок
    if($this->value) $checked = "checked";
    else $checked = "";

    // Формируем тег
    $tag = "<input $style $class
                  type=\"".$this->type."\"
```

```
                    name=\"".$this->name."\"
                    $checked>\n";

      // Формируем подсказку, если она имеется
      $help = "";
      if(!empty($this->help))
      {
        $help .= "<span style='color:blue'>".
                    nl2br($this->help)
                ."</span>";
      }
      if(!empty($help)) $help .= "<br>";
      if(!empty($this->help_url))
      {
        $help .= "<span style='color:blue'>
                    <a href=".$this->help_url.">помощь</a>
                  </span>";
      }

      return array($this->caption, $tag, $help);
    }

    // Метод, проверяющий корректность переданных данных
    function check()
    {
      return "";
    }
  }
?>
```

По умолчанию если атрибут `value` флажка не заполнен, то значение параметра, соответствующего флажку, принимает значение `on`, если флажок установлен, или отсутствует, если флажок снят. Во внутреннем представлении класса значение флажка выступает либо как `true`, либо как `false` (это немного ограничивает класс решаемых задач, зато увеличивает устойчивость элемента управления). Если конструктору класса передается значение `on`, то оно автоматически преобразуется к `true`.

Метод `check()` всегда возвращает пустую строку, т. к. ошибиться с вводом в случае флажка невозможно: он всегда либо установлен, либо снят.

10.16. Список. Класс *field_select*

Список позволяет выбрать одно или несколько значений из определенного набора и имеет следующий синтаксис:

```
<select>
  <option>Первый пункт</option>
  <option>Второй пункт</option>
  <option>Третий пункт</option>
</select>
```

Между тегами `<select>` и `</select>` располагаются пункты списка, которые оформляются в виде option-тегов. В приведенном выше примере список имеет три пункта.

Помимо традиционного атрибута `name`, тег `<select>` может иметь атрибуты `multiple` и `size`. Параметр `multiple` позволяет выбрать несколько пунктов списка, отмечая их правой кнопкой мыши при одновременном удержании клавиши <Ctrl>. Атрибут `size` определяет высоту списка в пунктах.

Тег `<select>` не имеет атрибута `value` — этот атрибут располагается в теге `<option>`. Помимо этого, тег `<option>` может иметь атрибут `selected` для обозначения выделения текущего пункта.

На рис. 10.24 приведены два варианта списков: первый вариант — множественный список, который использует атрибут `multiple` и имеет высоту, равную 3 (по количеству элементов), второй список является выпадающим и позволяет выбрать только одно значение.

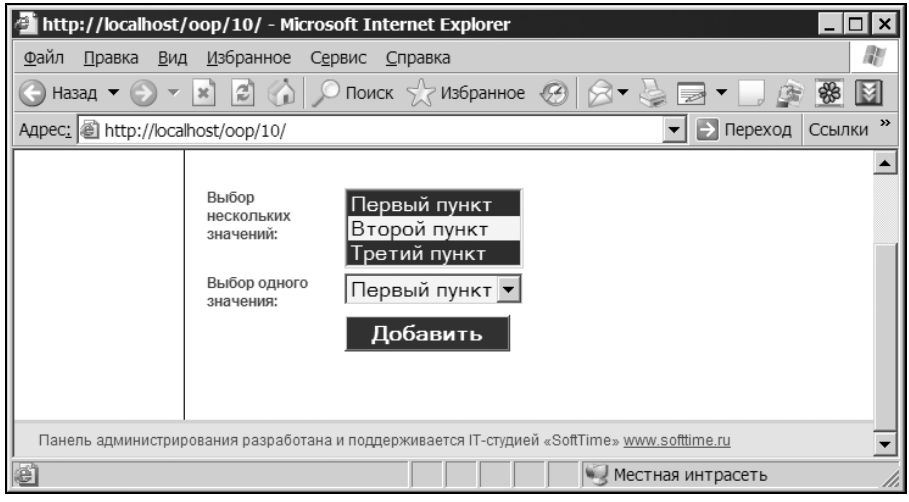


Рис. 10.24. Множественный и выпадающий списки

Для создания элемента управления списка унаследуем от базового класса `field` новый производный класс `field_select`. На рис. 10.25 представлена расширенная схема иерархии классов элементов управления.

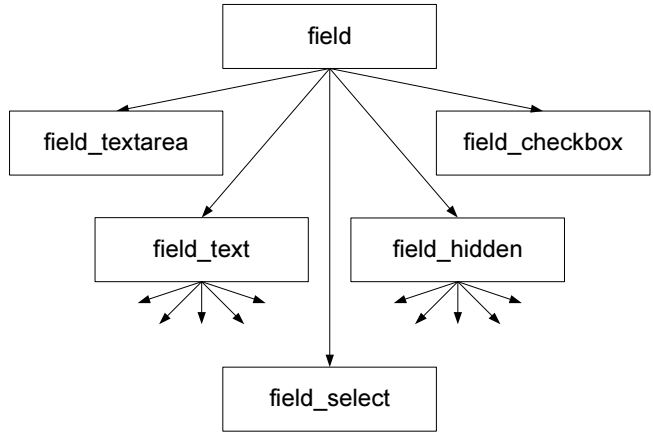


Рис. 10.25. Место класса `field_select` в иерархии классов элементов управления

В листинге 10.30 представлена возможная реализация класса `field_select`. Производный класс `field_select` необходимо расширить введением новых членов:

- `$options` — массив со значениями пунктов списка; ключи массива помещаются в атрибут `value`;
- `$multi` — логическая переменная, принимающая значение `true`, если список допускает выбор нескольких пунктов, и `false`, если список выпадающий;
- `$select_size` — высота для множественных списков.

Замечание

Конструктор класса списка `field_select` также не снабжается параметром `$is_required`, т. к. обязательность выбора элемента не имеет смысла в рамках элемента управления и реализуется на логическом уровне приложения.

Листинг 10.30. Реализация класса списка `field_select`

```
<?php
  ////////////////////////////////////////////////////////////
  // Список select
  ////////////////////////////////////////////////////////////

  class field_select extends field
```

```
{
  // Размер текстового поля
  protected $options;
  // Является ли список множественным
  protected $multi;
  // Высота списка
  protected $select_size;
  // Конструктор класса
  function __construct($name,
                $caption,
                $options = array(),
                $value,
                $multi = false,
                $select_size = 4,
                $parameters = "")
  {
    // Вызываем конструктор базового класса field для
    // инициализации его данных
    parent::__construct($name,
                "select",
                $caption,
                false,
                $value,
                $parameters);
    // Инициализируем члены класса
    $this->options     = $options;
    $this->multi       = $multi;
    $this->select_size = $select_size;
  }

  // Метод для возврата имени поля
  // и самого тега элемента управления
  function get_html()
  {
    // Если элементы оформления не пусты, учитываем их
    if(!empty($this->css_style))
    {
      $style = "style=\"".$this->css_style."\"";
    }
    else $style = "";
    if(!empty($this->css_class))
    {
      $class = "class=\"".$this->css_class."\"";
    }
    else $class = "";
```

```
if($this->multi && $this->select_size)
{
  $multi = "multiple size=".$this->select_size;
  $this->name = $this->name."[]";
}
else $multi = "";
// Формируем тег
$tag = "<select $style $class name=\"".$this->name."\" $multi>\n";
if(!empty($this->options))
{
  foreach($this->options as $key => $value)
  {
    if(is_array($this->value))
    {
      if(in_array($key,$this->value)) $selected = "selected";
      else $selected = "";
    }
    else if($key == trim($this->value)) $selected = "selected";
    else $selected = "";
    $tag .= "<option value='".
              htmlspecialchars($key, ENT_QUOTES)
             ."' $selected>".
              htmlspecialchars($value, ENT_QUOTES)
             ."</option>\n";
  }
}
$tag .= "</select>\n";

// Формируем подсказку, если она имеется
$help = "";
if(!empty($this->help))
{
  $help .= "<span style='color:blue'>".
              nl2br($this->help)
          ."</span>";
}
if(!empty($help)) $help .= "<br>";
if(!empty($this->help_url))
{
  $help .= "<span style='color:blue'>
              <a href=".$this->help_url.">помощь</a>
            </span>";
}
```

```
      return array($this->caption, $tag, $help);
    }

    // Метод, проверяющий корректность переданных данных
    function check()
    {
      // Получаем список возможных значений
      if(!in_array($this->value,array_keys($this->options)))
      {
        if(empty($this->value))
        {
          return "Поле \"".$this->caption."\"
                  содержит недопустимое значение";
        }
      }
      if (!get_magic_quotes_gpc())
      {
        for($i = 0; $i < count($this->value); $i++)
        {
          $this->value[$i] = mysql_escape_string($this->value[$i]);
        }
      }

      return "";
    }
    // Выбранный элемент
    function selected()
    {
      return $this->value[0];
    }
  }
?>
```

Член `$option` хранит ассоциативный массив с пунктами списка, член `$value` является либо массивом (если допускается выбор нескольких значений), либо первичным ключом (если объект представляет выпадающий список). HTML-форма, представленная на рис. 10.24, может быть сформирована при помощи кода из листинга 10.31.

Листинг 10.31. Использование списков

```
<?php
  $fst = new field_select("fst",
```

```
                          "Выбор множества<br> значения",
                           array("Первый пункт",
                                 "Второй пункт",
                                 "Третий пункт"),
                           array(0, 2),
                           true,
                           3);
  $snd = new field_select("snd",
                          "Выбор одного<br> значения",
                           array("Первый пункт",
                                 "Второй пункт",
                                 "Третий пункт"),
                           0);
  $form = new form(array("fst"       => $fst,
                         "snd"       => $snd),
                   $button_name,
                   $class_name);
?>
```

Метод `check()` класса `field_select()`, помимо всего прочего, проверяет, являются ли выбранные значения ключами массива `$options`. Кроме того, введен дополнительный метод `select()`, позволяющий получить выделенный элемент управления, если формируется выпадающий список.

10.17. Переключатели. Класс *field_radio*

Переключатель, или, иначе, радиокнопка — элемент управления, позволяющий выбрать из набора утверждений только одно. Он имеет следующий синтаксис:

```
<input type=radio>
```

Для формирования набора утверждений используется несколько переключателей, которым присваивается одно и то же имя через атрибут `name`. Помимо традиционных атрибутов `type` и `name`, переключатели могут быть снабжены атрибутом `value` для передачи значения и атрибутом `checked` для того, чтобы отметить один из переключателей по умолчанию.

Набор переключателей будет задаваться в SoftTime Framework при помощи одного элемента управления. При этом утверждения будут передаваться в виде массива точно так же, как в классе `field_select` (см. *раздел 10.15*).

На рис. 10.26 приведены два набора переключателей с горизонтальным и вертикальным расположением переключателей.

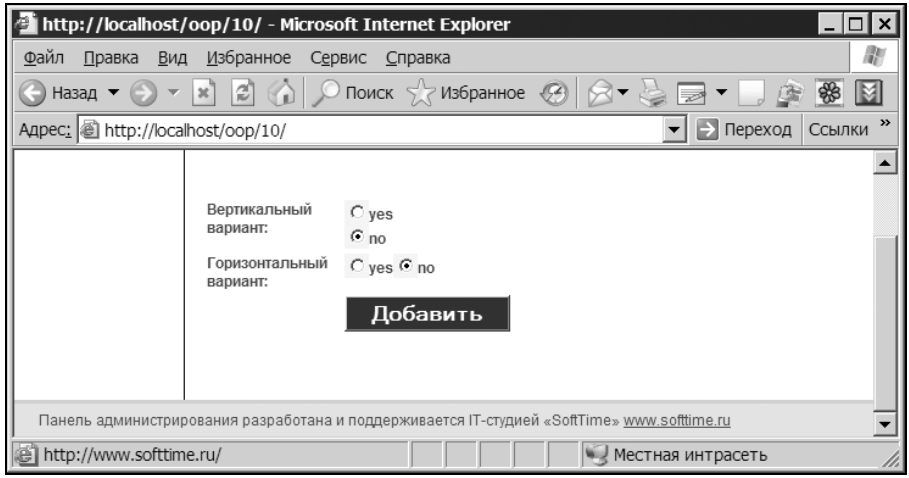


Рис. 10.26. Горизонтальный и вертикальный варианты набора переключателей

Для моделирования переключателей унаследуем от базового класса `field` производный класс `field_radio` (рис. 10.27).

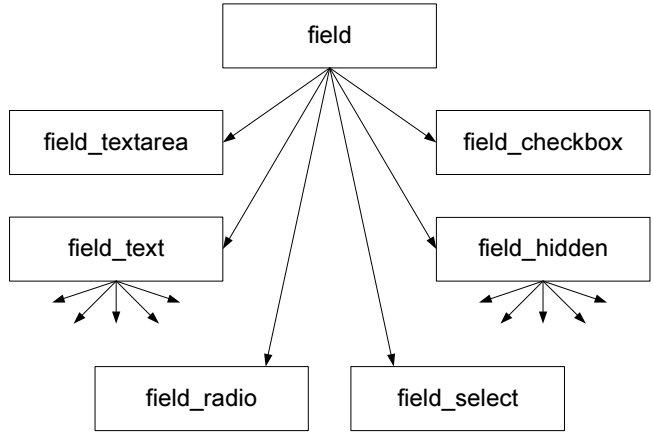


Рис. 10.27. Место класса `field_radio` в иерархии классов элементов управления

В листинге 10.32 представлена возможная реализация класса `field_radio`, сильно напоминающая реализацию класса списка `field_select`.

Замечание

Конструктор класса списка `field_select` также не снабжается параметром `$is_required`, т. к. обязательность выбора элемента не имеет смысла в рамках элемента управления и реализуется на логическом уровне приложения.

Листинг 10.32. Класс набора переключателей `field_radio`

```
<?php
  ////////////////////////////////////////////////////////////
  // Переключатель radio
  ////////////////////////////////////////////////////////////

  class field_radio extends field
  {
    // Варианты ответов
    protected $radio;
    // Конструктор класса
    function __construct($name,
                  $caption,
                  $radio = array(),
                  $value,
                  $parameters = "", // horizontal
                  $help = "",
                  $help_url = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct($name,
                  "radio",
                  $caption,
                  false,
                  $value,
                  $parameters,
                  $help,
                  $help_url);
      // Инициализируем члены класса
      if($this->radio != "radio_rate") $this->radio = $radio;
    }

    // Метод для возврата имени поля
    // и самого тега элемента управления
    function get_html()
    {
      // Если элементы оформления не пусты, учитываем их
      if(!empty($this->css_style))
      {
        $style = "style=\"".$this->css_style."\"";
      }
```

```
else $style = "";
if(!empty($this->css_class))
{
  $class = "class=\"".$this->css_class."\"";
}
else $class = "";

$this->type = "radio";
// Формируем тег
$tag = "";
if(!empty($this->radio))
{
  foreach($this->radio as $key => $value)
  {
    if($key == $this->value) $checked = "checked";
    else $checked ="";
    if(strpos($this->parameters, "horizontal") !== false)
    {
      $tag .= "<input $style $class
                type=".$this->type."
                name=".$this->name."[]
                $checked value='$key'>$value";
    }
    else
    {
      $tag[] = "<input $style $class
                 type=".$this->type."
                 name=".$this->name."[]
                 $checked value='$key'>$value\n";
    }
  }
}

// Формируем подсказку, если она имеется
$help = "";
if(!empty($this->help))
{
  $help .= "<span style='color:blue'>".
              nl2br($this->help)
          ."</span>";
}
if(!empty($help)) $help .= "<br>";
if(!empty($this->help_url))
```

```
      {
        $help .= "<span style='color:blue'>
                    <a href=".$this->help_url.">помощь</a>
                  </span>";
      }

      return array($this->caption, $tag, $help);
    }

    // Метод, проверяющий корректность переданных данных
    function check()
    {
      // Получаем список возможных значений
      if (!get_magic_quotes_gpc())
      {
        $this->value = mysql_escape_string($this->value);
      }
      if(!@in_array($this->value,array_keys($this->radio)))
      {
        if(empty($this->value))
        {
          return "Поле \"".$this->caption."\" содержит
                  недопустимое значение";
        }
      }

      return "";
    }
  }
?>
```

В данном классе задействуется зарезервированный член `$parameters`. Если его значение содержит в своем составе подстроку "`horizontal`" — переключатели располагаются горизонтально, в противном случае — вертикально. Названия переключателей и их значения задаются при помощи элементов и ключей массива `$radio`. HTML-форма, представленная на рис. 10.26, может быть сформирована при помощи кода из листинга 10.33.

Листинг 10.33. Использование переключателей

```
<?php
  $fst = new field_radio("fst",
                         "Вертикальный вариант",
```

```
                          array("yes",
                                "no"),
                          1);
  $snd = new field_radio("snd",
                         "Горизонтальный вариант",
                          array("yes",
                                "no"),
                          1,
                          "horizontal");
  $form = new form(array("fst"       => $fst,
                         "snd"       => $snd),
                   $button_name,
                   $class_name);
?>
```

10.18. Поле для загрузки файла на сервер. Класс *field_file*

Для загрузки пользовательских файлов на сервер используется специальный элемент управления, позволяющий указать путь к загружаемому файлу (при помощи кнопки **Обзор**). Элемент управления имеет следующий синтаксис:

```
<input type=file>
```

Помимо атрибута `type`, элемент управления допускает указания атрибутов `name` и `size`.

Особенность загрузки файлов на сервер заключается в том, что после загрузки файл помещается во временную директорию, откуда PHP-разработчик должен его скопировать в директорию назначения при помощи функции `copy()` или `move_uploaded_file()`.

Замечание

Атрибут HTML-формы `enctype` должен принимать значение `multipart/form-date`, в противном случае файл не будет загружен на сервер.

Характеристики загруженного файла доступны через двумерный суперглобальный массив `$_FILES`. При этом переменная со значениями этого массива может иметь следующий вид:

- `$_FILES['filename']['name']` — содержит исходное имя файла на клиентской машине;
- `$_FILES['filename']['size']` — содержит размер загруженного файла в байтах;

- `$_FILES['filename']['type']` — содержит MIME-тип файла;
- `$_FILES['filename']['tmp_file']` — содержит имя временного файла, в который сохраняется загруженный файл.

Здесь `filename` — имя элемента управления (атрибут `name`).

Унаследуем от базового класса `field` новый производный класс `field_file`, расширенный двумя новыми членами:

- `$dir` — директория назначения, куда будет помещен файл;
- `$prefix` — префикс, которым при необходимости будет снабжаться файл. Такой префикс можно формировать уникальным — это позволит не перезаписывать файлы с одинаковыми названиями.

На рис. 10.28 демонстрируется текущая иерархия классов элементов управления.

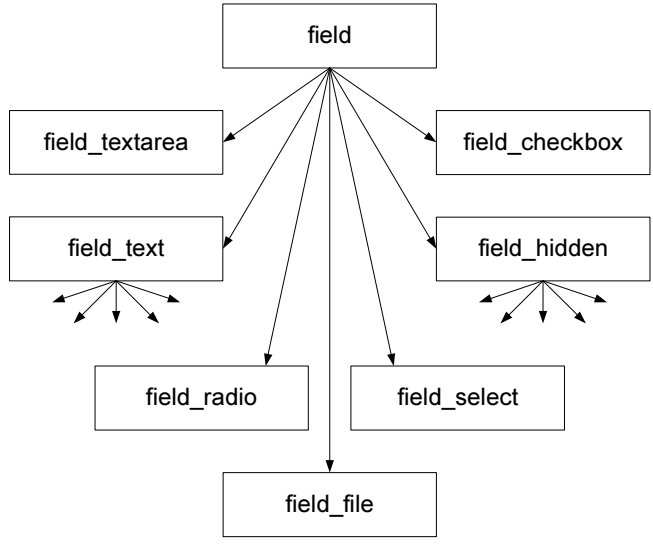


Рис. 10.28. Место класса `field_file` в иерархии классов элементов управления

Название файла может включать символы русского алфавита. В то же время не все UNIX-системы поддерживают русскоязычные наименования файлов и директорий, поэтому название будет преобразовываться в транслит методом `encodestring()` базового класса `field`. Получить конечное имя файла можно при помощи дополнительного метода `get_filename()`.

В листинге 10.34 приводится возможная реализация класса `field_file`.

Листинг 10.34. Класс загрузки файла на сервер `field_file`

```
<?php
  ////////////////////////////////////////////////////////////
  // Загрузка файла на сервер file
  ////////////////////////////////////////////////////////////

  class field_file extends field
  {
    // Директория назначения
    protected $dir;
    // Префикс
    protected $prefix;
    // Конструктор класса
    function __construct($name,
                  $caption,
                  $is_required = false,
                  $value, // $_FILES
                  $dir,
                  $prefix = "",
                  $help = "",
                  $help_url = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct($name,
                  "file",
                  $caption,
                  $is_required,
                  $value,
                  "",
                  $help,
                  $help_url);
      $this->dir = $dir;
      $this->prefix = $prefix;

      if(!empty($this->value))
      {
        // Если файл является скриптом PHP
        // или Perl или представляет собой html-страницу,
        // преобразуем его в формат .txt
        $extentions = array("#\.php#is",
                            "#\.phtml#is",
```

```
                        "#\.php3#is",
                        "#\.html#is",
                        "#\.htm#is",
                        "#\.hta#is",
                        "#\.pl#is",
                        "#\.xml#is",
                        "#\.inc#is",
                        "#\.shtml#is",
                        "#\.xht#is",
                        "#\.xhtml#is");
    // Заменяем символы русского алфавита на транслит
    $this->value[$this->name]['name'] =
          $this->encodestring($this->value[$this->name]['name']);
    // Извлекаем из имени файла расширение
    $path_parts = pathinfo($this->value[$this->name]['name']);
    $ext = ".".$path_parts['extension'];
    $path = basename($this->value[$this->name]['name'],$ext);
    $add = $ext;
    foreach($extentions AS $exten)
    {
      if(preg_match($exten, $ext)) $add = ".txt";
    }
    $path .= $add;
    $path = str_replace("//","/",$dir."/".$prefix.$path);
    // Перемещаем файл из временной директории сервера
    // в директорию /files Web-приложения
    if (copy($this->value[$this->name]['tmp_name'], $path))
    {
      // Уничтожаем файл во временной директории
      @unlink($this->value[$this->name]['tmp_name']);
      // Изменяем права доступа к файлу
      @chmod($path, 0644);
    }
  }
}

// Метод для возврата имени поля
// и самого тега элемента управления
function get_html()
{
  // Если элементы оформления не пусты, учитываем их
  if(!empty($this->css_style))
```

```
    {
      $style = "style=\"".$this->css_style."\"";
    }
    else $style = "";
    if(!empty($this->css_class))
    {
      $class = "class=\"".$this->css_class."\"";
    }
    else $class = "";

    // Формируем тег
    $tag = "<input $style $class
                   type=\"".$this->type."\"
                   name=\"".$this->name."\">\n";

    // Если поле обязательно, отмечаем этот факт
    if($this->is_required) $this->caption .= " *";

    // Формируем подсказку, если она имеется
    $help = "";
    if(!empty($this->help))
    {
      $help .= "<span style='color:blue'>".
                  nl2br($this->help)
              ."</span>";
    }
    if(!empty($help)) $help .= "<br>";
    if(!empty($this->help_url))
    {
      $help .= "<span style='color:blue'>
                  <a href=".$this->help_url.">помощь</a>
                </span>";
    }

    return array($this->caption, $tag, $help);
  }

  // Метод, проверяющий корректность переданных данных
  function check()
  {
    if($this->is_required)
    {
      if(empty($this->value[$this->name]))
```

```
        {
          return "Поле \"".$this->caption."\" не заполнено";
        }
      }

      return "";
    }

    // Возврат перекодированного имени файла
    function get_filename()
    {
      if(!empty($this->value))
      {
        if(!empty($this->value[$this->name]['name']))
        {
          return mysql_escape_string($this->encodestring(
            $this->prefix.$this->value[$this->name]['name']));
        }
        else return "";
      }
      else return "";
    }
  }
?>
```

Конструктору класса `field_file` в качестве элемента `value` передается суперглобальный массив `$_FILES`. Если на сервер не загружено ни одного файла, массив остается пустым, а конструктор не выполняет никаких действий. Однако если массив не пуст, производится загрузка файла в директорию назначения `$dir`. Путь к загруженному файлу можно получить при помощи специального метода `get_filename()`.

Замечание

Пример с использованием класса `field_file` будет рассмотрен в *главе 11*.

10.19. Заголовок. Класс *field_title*

HTML-формы могут иметь достаточно сложную структуру и включать несколько логических разделов. Было бы удобно иметь возможность вставлять в HTML-форму свои заголовки, которые отделяли бы одни элементы управления от других. Введение статических элементов управления позволило бы придать дополнительную гибкость приложению, которой приходится жерт-

вовать ради увеличения скорости разработки при автоматическом построении HTML-форм.

Унаследуем от базового класса `field` производный класс `field_title`, который будет поддерживать шесть видов заголовков, соответствующих HTML-тегам `<H1>`, `<H2>`, `<H3>`, `<H4>`, `<H5>` и `<H6>`. Возможная реализация класса `field_title` представлена в листинге 10.35.

Конструктор класса `field_title` содержит три необязательных параметра:

- `$value` — текст заголовка;
- `$h_type` — размер заголовка;
- `$parameters` — параметр, зарезервированный для будущих расширений.

Листинг 10.35. Класс заголовка `field_title`

```
<?php
  ////////////////////////////////////////////////////////////
  // Заголовок (текст)
  ////////////////////////////////////////////////////////////

  class field_title extends field
  {
    // Размер заголовка 1, 2, 3, 4, 5, 6 для
    // h1, h2, h3, h4, h5, h6 соответственно
    protected $h_type;
    // Конструктор класса
    function __construct($value = "",
                         $h_type = 3,
                         $parameters = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct("",
                  "title",
                  "",
                  false,
                  $value,
                  $parameters,
                  "",
                  "");
      if($h_type > 0 && $h_type < 7) $this->h_type = $h_type;
      // По умолчанию присваиваем значение 3
      else $this->h_type = 3;
    }
```

```
    // Метод для возврата имени поля
    // и самого тега элемента управления
    function get_html()
    {
      // Формируем тег
      $tag = htmlspecialchars($this->value, ENT_QUOTES);
      $pattern = "#\[b\](.+)\[\/b\]#isU";
      $tag = preg_replace($pattern,'<b>\\1</b>',$tag);
      $pattern = "#\[i\](.+)\[\/i\]#isU";
      $tag = preg_replace($pattern,'<i>\\1</i>',$tag);
      $pattern = "#\[url\][\s]*((?=http:)[\S]*)[\s]*\[\/url\]#si";
      $tag = preg_replace($pattern,
                          '<a href="\\1" target=_blank>\\1</a>',$tag);
      $pattern =
     "#\[url[\s]*=[\s]*((?=http:)[\S]+)[\s]*\][\s]*([^\[]*)\[/url\]#isU";
      $tag = preg_replace($pattern,
                           '<a href="\\1" target=_blank>\\2</a>',
                           $tag);
      if (get_magic_quotes_gpc()) $tag = stripcslashes($tag);
      $tag = "<h".$this->h_type.">".$this->value."</h".$this->h_type.">";

      return array($this->caption, $tag);
    }

    // Метод, проверяющий корректность переданных данных
    function check()
    {
      return "";
    }
  }
?>
```

Для указания типа заголовка в класс `filed_title` вводится член `$h_type`, который может принимать значения от 1 до 6 (для заголовков от H1 до H6 соответственно). Метод `check()`, который необходимо перегружать в силу того, что он является абстрактным в классе `field`, всегда возвращает пустую строку (отсутствие ошибок), т. к. элемент управления заголовок является статическим и служит для отображения информации.

Метод `get_html()` преобразует bbCode в обычные HTML-теги. Поддерживаются следующие bbCode-теги:

- [b]...[/b] — текст, заключенный в эти теги, выделяется жирным шрифтом, их использование эквивалентно ...;

- [i]...[/i] — текст, заключенный в эти теги, выделяется курсивом (наклонным шрифтом), их использование эквивалентно ...;
- [url]http://www.site.ru[/url] — ссылка, преобразующаяся к виду http://www.site.ru;
- [url=http://www.site.ru]текст[/url] — ссылка, преобразующаяся к виду текст.

Использование тегов bbCode вместо обычных HTML-тегов необходимо для того, чтобы предотвратить искажение последних функцией `htmlspecialchars()`, использование которой необходимо для предотвращения XSS-инъекций.

Рассматриваемые до этого элементы управления занимали две ячейки таблицы. В случае заголовка удобнее, если он будет располагаться в двух ячейках, объединенных при помощи атрибута `colspan` тега `<td>`. Для этого модифицируем метод `print_form()` класса `form` (см. листинг 10.11) так, чтобы элементы, чей член `$type` принимает значение `title`, обрабатывались специальным образом (листинг 10.36).

Листинг 10.36. Специальная обработка заголовка в методе `print_form()` класса `form`

```
<?php
  ...
  switch($obj->type)
  {
    case "hidden":
      // Скрытое поле
      echo $tag;
      break;
    case "title":
      // Заголовок
      echo "<tr>
              <td $style $class colspan=2 valign=top>$tag</td>
              </tr>\n";
      break;
    default:
      // Элементы управления по умолчанию
      echo "<tr>
              <td width=100
                  $style $class valign=top>$caption:</td>
              <td $style $class valign=top>$tag </td>
            </tr>\n";
      if(!empty($help))
```

```
      {
        echo "<tr>
                <td> </td>
                <td $style $class valign=top>$help</td>
              </tr>";
      }
      break;
  }
  ...
?>
```

В листинге 10.37 приводится пример использования заголовков: с их помощью обязательные параметры отделяются от необязательных.

Листинг 10.37. Использование заголовков

```
<?php
  $titl_req = new field_title("1. Обязательные параметры");

  $fst = new field_text("fst",
                        "Первое поле",
                        true,
                        "");
  $snd = new field_text("snd",
                        "Второе поле",
                        true,
                        "");

  $titl_non_req = new field_title("2. Необязательные параметры");

  $thd = new field_text("thd",
                        "Третье поле",
                        false,
                        "");
  $fth = new field_text("fth",
                        "Четвертое поле",
                        false,
                        "");

  $form = new form(array("titl_req"     => $titl_req,
                         "fst"          => $fst,
                         "snd"          => $snd,
                         "titl_non_req" => $titl_non_req,
```

```
                    "thd"          => $thd,
                    "fth"          => $fth),
                $button_name,
                $class_name);
?>
```

Результат работы скрипта из листинга 10.37 представлен на рис. 10.29.

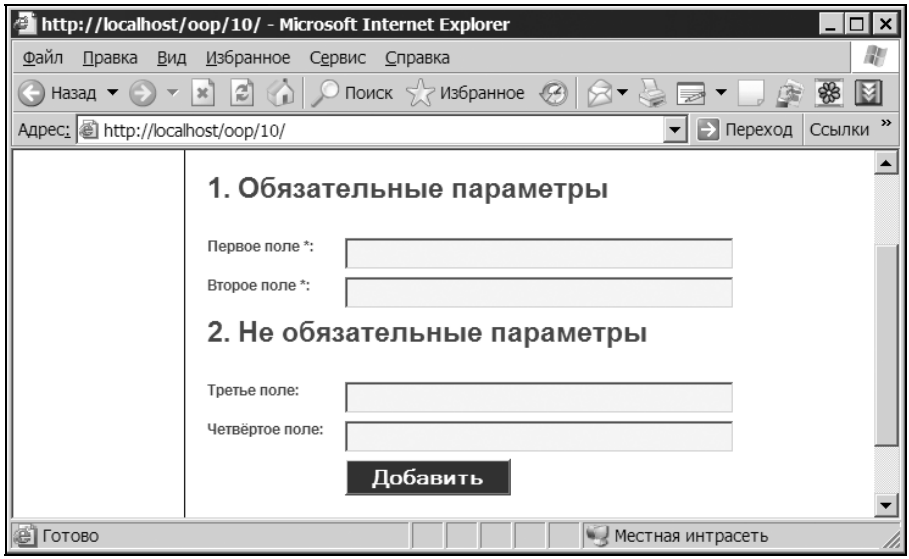


Рис. 10.29. Использование заголовков

10.20. Параграф. Класс *field_paragraph*

Естественным дополнением статических элементов будет класс параграфа, который допускал бы размещение текстового комментария между элементами управления. Унаследуем от базового класса `field` производный класс `field_paragraph` (листинг 10.38). Класс `field_paragraph` практически полностью аналогичен классу `field_title`, за исключением отсутствия члена `$h_type`, т. к. размер текста всех надписей одинаков. Кроме того, в методе `get_html()` при возврате фрагментов для HTML-формы текст параграфа подвергается преобразованию при помощи функции `nl2br()`, которая заменяет переводы строк на тег `<br />`. Последнее необходимо, чтобы структура перевода строк параграфа сохранялась при выводе в окно браузера.

Листинг 10.38. Класс параграфа `field_paragraph`

```
<?php
  ////////////////////////////////////////////////////////////
  // Параграф (текст)
  ////////////////////////////////////////////////////////////

  class field_paragraph extends field
  {
    // Конструктор класса
    function __construct($value = "",
                         $parameters = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct("",
                  "paragraph",
                  "",
                  false,
                  $value,
                  $parameters,
                  "",
                  "");
    }

    // Метод для возврата имени поля
    // и самого тега элемента управления
    function get_html()
    {
      // Формируем тег
      $tag = htmlspecialchars($this->value, ENT_QUOTES);
      $pattern = "#\[b\](.+)\[\/b\]#isU";
      $tag = preg_replace($pattern,'<b>\\1</b>',$tag);
      $pattern = "#\[i\](.+)\[\/i\]#isU";
      $tag = preg_replace($pattern,'<i>\\1</i>',$tag);
      $pattern = "#\[url\][\s]*((?=http:)[\S]*)[\s]*\[\/url\]#si";
      $tag = preg_replace($pattern,
                          '<a href="\\1" target=_blank>\\1</a>',$tag);
      $pattern = "#\[url[\s]*=[\s]*((?=http:)[\S]+)[\s]*\][\s]*([^\[]*)\
                                                          [/url\]#isU";
      $tag = preg_replace($pattern,
                           '<a href="\\1" target=_blank>\\2</a>',
                           $tag);
      if (get_magic_quotes_gpc()) $tag = stripcslashes($tag);
```

```
      return array($this->caption, nl2br($tag));
    }

    // Метод, проверяющий корректность переданных данных
    function check()
    {
      return "";
    }
  }
?>
```

Для того чтобы обработка параграфа в методе `print_form()` класса `form` протекала правильно, необходимо модифицировать switch-оператор, включив в него case-оператор для параграфа (листинг 10.39).

Листинг 10.39. Специальная обработка параграфа в методе `print_form()` класса `form`

```
<?php
  ...
  switch($obj->type)
  {
    case "hidden":
      // Скрытое поле
      echo $tag;
      break;
    case "paragraph":
    case "title":
      // Заголовок
      echo "<tr>
              <td $style $class colspan=2 valign=top>$tag</td>
              </tr>\n";
      break;
    default:
      // Элементы управления по умолчанию
      echo "<tr>
              <td width=100
                  $style $class valign=top>$caption:</td>
              <td $style $class valign=top>$tag </td>
            </tr>\n";
      if(!empty($help))
      {
        echo "<tr>
                <td> </td>
```

```
                <td $style $class valign=top>$help</td>
              </tr>";
      }
      break;
  }
  ...
?>
```

10.21. Выбор даты и времени. Класс *field_datetime*

До этого момента новые элементы управления в основном дублировали существующие в HTML элементы управления и может казаться не совсем понятным, как, дублируя одну систему кодирования другой, можно добиться увеличения скорости разработки Web-приложений.

Преимущество набора классов заключается в его расширяемости — можно создать достаточно крупные блоки со сложными условиями проверки корректности ввода. В качестве одного из таких крупных блоков, которые часто используются при создании Web-приложений, может выступать набор выпадающих списков для выбора даты и времени (рис. 10.30).

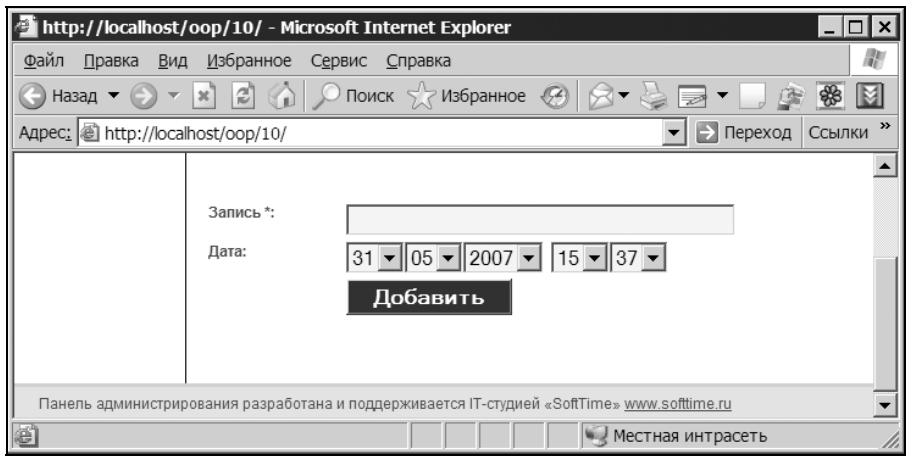


Рис. 10.30. Выбор даты и времени

Обработка и проверка корректности ввода таких списков, особенно если дата ограничена сверху и снизу, может отнимать много сил Web-разработчика. Поэтому разумно унаследовать от базового класса `field` новый класс `field_datetime` и поручить ему все рутинные операции проверки корректности (листинг 10.40).

Класс `field_datetime` снабжен тремя закрытыми членами:

- `$time` — время в формате UNIXSTAMP (количество секунд, прошедших с первого января 1970 года);
- `$begin_year` — минимальный допустимый год;
- `$end_year` — максимальный допустимый год.

Листинг 10.40. Класс выбора даты и времени `field_datetime`

```
<?php
  ////////////////////////////////////////////////////////////
  // Элемент управления для выбора даты и времени
  ////////////////////////////////////////////////////////////

  class field_datetime extends field
  {
    // Время в time
    protected $time;
    // Начальный год
    protected $begin_year;
    // Конечный год
    protected $end_year;
    // Конструктор класса
    function __construct($name,
                  $caption,
                  $time,
                  $begin_year = 2000,
                  $end_year = 2020,
                  $parameters = "",
                  $help = "",
                  $help_url = "")
    {
      // Вызываем конструктор базового класса field
      // для инициализации его данных
      parent::__construct($name,
                  "datetime",
                  $caption,
                  false,
                  $value,
                  $parameters,
                  $help,
                  $help_url);
```

```
    if(empty($time)) $this->time = time();
    else if(is_array($time))
    {
      $this->time     = mktime($time['hour'],
                               $time['minute'],
                               0,
                               $time['month'],
                               $time['day'],
                               $time['year']);
    }
    else $this->time  = $time;
    $this->begin_year = $begin_year;
    $this->end_year   = $end_year;
  }

  // Дата в формате MySQL
  function get_mysql_format()
  {
    return date("Y-m-d H:i:s", $this->time);
  }

  // Метод для возврата имени поля
  // и самого тега элемента управления
  function get_html()
  {
    // Если элементы оформления не пусты, учитываем их
    if(!empty($this->css_style))
    {
      $style = "style=\"".$this->css_style."\"";
    }
    else $style = "";
    if(!empty($this->css_class))
    {
      $class = "class=\"".$this->css_class."\"";
    }
    else $class = "";

    // Формируем тег
    $date_month  = @date("m", $this->time);
    $date_day    = @date("d", $this->time);
    $date_year   = @date("Y", $this->time);
    $date_hour   = @date("H", $this->time);
    $date_minute = @date("i", $this->time);
```

```
// Выпадающий список для выбора числа
$tag = "<select title='Число'
         $style $class type=text
         name='".$this->name."[day]'>\n";
for($i = 1; $i <= 31; $i++)
{
  if($date_day == $i) $temp = "selected";
  else $temp = "";
  $tag .= "<option value=$i $temp>".sprintf("%02d", $i);
}
$tag .= "</select>";
// Выпадающий список для выбора месяца
$tag .= "<select title='Месяц'
          $style $class type=text
          name='".$this->name."[month]'>";
for($i = 1; $i <= 12; $i++)
{
  if($date_month == $i) $temp = "selected";
  else $temp = "";
  $tag .= "<option value=$i $temp>".sprintf("%02d", $i);
}
$tag .= "</select>";
// Выпадающий список для выбора года
$tag .= "<select title='Год'
          $style $class type=text
          name='".$this->name."[year]'>";
for($i = 2004; $i <= 2017; $i++)
{
  if($date_year == $i) $temp = "selected";
  else $temp = "";
  $tag .= "<option value=$i $temp>$i";
}
$tag .= "</select>";
// Выпадающий список для указания часа
$tag .= "  <select
         title='Часы' $style $class
         type=text name='".$this->name."[hour]'>";
for($i = 0; $i <= 23; $i++)
{
  if($date_hour == $i) $temp = "selected";
  else $temp = "";
  $tag .= "<option value=$i $temp>".sprintf("%02d",$i);
}
```

```
    $tag .= "</select>";
    // Выпадающий список для указания минут
    $tag .= "<select title='Минуты'
              $style $class
              type=text
              name='".$this->name."[minute]'>";
    for($i = 0; $i <= 59; $i++)
    {
      if($date_minute == $i) $temp = "selected";
      else $temp = "";
      $tag .= "<option value=$i $temp>".sprintf("%02d",$i);
    }
    $tag .= "</select>";

    // Если поле обязательно, отмечаем этот факт
    if($this->is_required) $this->caption .= " *";

    // Формируем подсказку, если она имеется
    $help = "";
    if(!empty($this->help))
    {
      $help .= "<span style='color:blue'>".
                 nl2br($this->help)
              ."</span>";
    }
    if(!empty($help)) $help .= "<br>";
    if(!empty($this->help_url))
    {
      $help .= "<span style='color:blue'>
                  <a href=".$this->help_url.">помощь</a>
                </span>";
    }

    return array($this->caption, $tag, $help);
  }

  // Метод, проверяющий корректность переданных данных
  function check()
  {
    if(date('Y', $this->time) > $this->end_year ||
       date('Y', $this->time) < $this->begin_year)
    {
```

```
            return "Поле \"".$this->caption."\" содержит
                    недопустимое значение (его значение
                    должно лежать в диапазоне ".
                    $this->begin_year."-".$this->end_year.")";
        }

        return "";
    }
  }
?>
```

Конструктор класса `field_datetime` принимает в качестве параметра дату и время в UNIXSTAMP-формате `$time`, а также максимальный `$end_year` и минимальный `$begin_year` допустимый год. Метод `get_html()` формирует набор

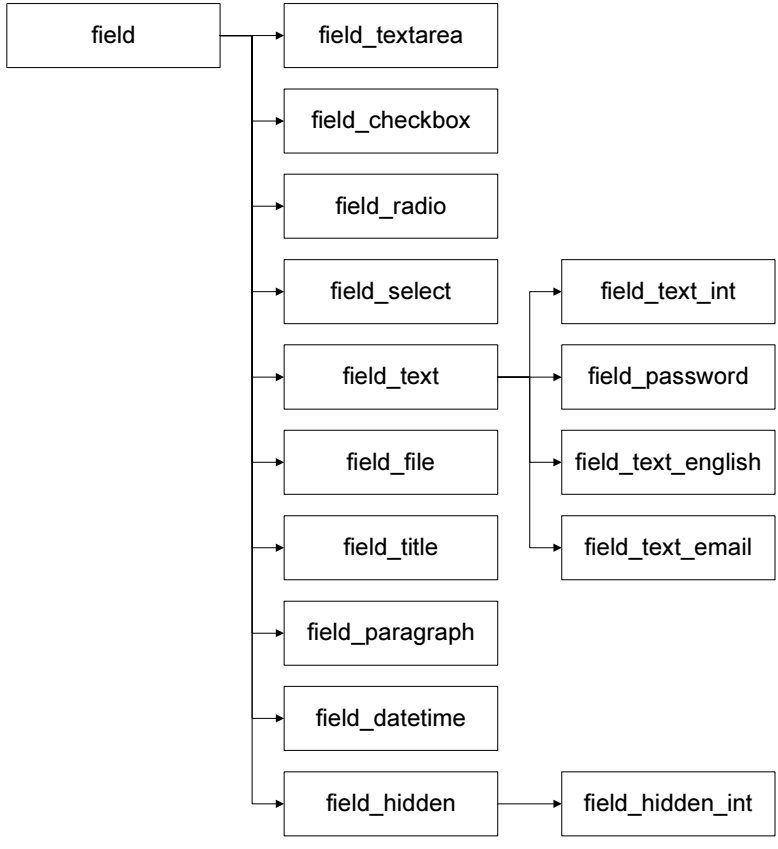


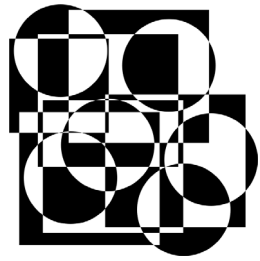
Рис. 10.31. Иерархия классов SoftTime Framework

выпадающих списков для года, месяца, даты, минут и секунд. Метод `check()` осуществляет проверку, не выходят ли текущие дата и время за интервал, задаваемый параметрами `$begin_year` и `$end_year`.

10.22. Обзор элементов управления

В завершение главы приведем общую схему элементов управления SoftTime Framework (рис. 10.31). Главная особенность данной схемы заключается в том, что она не является завершенной — в любой момент ее можно расширить новыми элементами управления, включающими как один HTML-элемент управления со специфической обработкой (и даже JavaScript-вставками), так и комбинированные элементы управления, содержащие несколько HTML-элементов (см. *раздел 10.21*).

ГЛАВА 11



Создание системы управления сайтом (CMS)

В последнее время получили большое распространение системы управления содержимым сайта (Content Management System — CMS). CMS — это система, которая позволяет владельцу сайта управлять текстовой и графической информацией сайта без кодирования с использованием технологий PHP, MySQL, HTML, JavaScript, Flash и т. д.

Появление самых разнообразных CMS вызвано двумя причинами:

- программисты отходят от практики разработки сайта как произведения искусства; жесткие сроки заставляют их переходить к конвейерному производству (по крайней мере, в области системы администрирования и баз данных);
- заказчики больше не воспринимают Web-сайты как игрушку, демонстрирующую состоятельность компании. В сайт вкладывают средства, и зачастую он служит для расширения бизнеса и извлечения прибыли. Все это приводит к тому, что владельцы сайта желают иметь полный контроль над каждым элементом сайта через систему администрирования.

Стандарта в области построения CMS нет: это может быть как простейшая форма для изменения e-mail и адреса в блоке обратной связи, так и система, включающая десятки блоков и сотни настроек, позволяющих изменять содержимое меню, тексты статей и даже структуру сайта. Система управления может быть вынесена как в отдельный блок, так и быть интегрированной в блоки управления на сайте и включаться при входе на сайт администратора. CMS может быть ориентирована на администратора и редактора (удобство использования) или на разработчика (удобство расширения и введения новых блоков). Скорее всего, создание универсальной CMS невозможно — сайты (да и сами CMS) предназначены для решения разнообразных задач — учет всех возможностей приведет к созданию слишком сложного интерфейса,

изучение которого станет непреодолимым препятствием для большинства пользователей и разработчиков.

В этой главе мы сосредоточимся на создании системы управления содержимым сайта (CMS), предназначенной для Web-разработчика. В основе системы будет лежать созданный в предыдущей главе SoftTime FrameWork. Главной задачей системы будет ускорение разработки сайтов и их систем управления, поэтому она будет иметь развитую базу данных и систему администрирования, тогда как система представления будет лишь демонстрационной (практически не развитой). Последнее связано с тем, что система представления часто требует значительной переработки под дизайн сайта — создание готовой системы неизбежно диктует шаблоны, что крайне раздражает заказчиков и дизайнеров.

Замечание

Система администрирования — это область сайта, доступная только администраторам и редакторам. Система представления — область сайта, доступная посетителям. Обе системы работают с одной и той же базой данных. Системы могут быть интегрированы между собой, а могут быть разделены и находиться в разных папках. Мы будем использовать последний подход, позволяющий минимизировать зависимость от дизайна сайта.

Замечание

В данной главе будет рассмотрен ограниченный набор блоков, т. к. для рассмотрения полноценной CMS потребовалась бы отдельная книга. Полноценная CMS, включающая большое количество блоков, рассматривается в нашей книге "PHP 5. Практика создания Web-сайтов".

11.1. Структура системы управления сайтом (CMS)

CMS структурно разделяется на несколько блоков:

- база данных — набор таблиц, в которых хранится вся конструктивная информация сайта;
- система администрирования — блок, позволяющий осуществлять редактирование базы данных;
- система представления — набор программных блоков, с которыми имеет дело посетитель: информация извлекается и помещается в базу данных;
- конфигурационные (настроечные) файлы — директория config, в которой будут располагаться конфигурационные файлы: config.php для установки соединения с СУБД MySQL, class.config.php для хранения путей к классам

блока представления и class.config.dmn.php — путей к классам для блока администрирования;

- FrameWork — набор общих классов, которые используются при построении системы администрирования и представления (см. *главу 10*).

Схема отношения отдельных блоков системы управления содержимым сайта представлена на рис. 11.1.

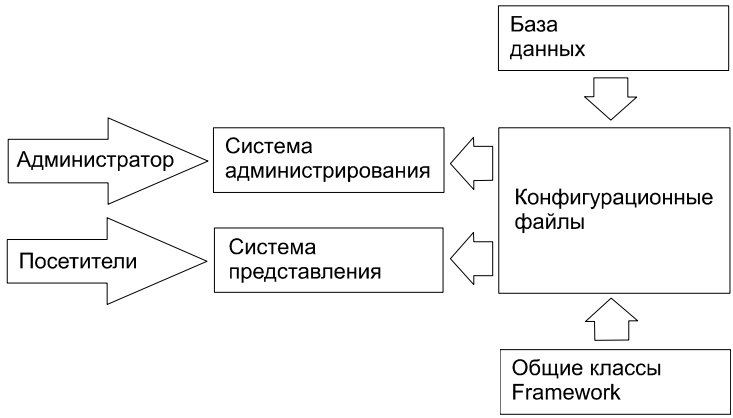


Рис. 11.1. Схема отношения блоков системы управления содержимым сайта (CMS)

Каждый из блоков CMS располагается в отдельной директории, при этом файлы блока представления находятся в корне сайта, а все остальные блоки — в подкаталогах:

```
/ — блок представления
  /class — Общие классы FramWork
  /config — конфигурационные файлы
  /dmn — система администрирования
```

Таблицы базы данных будут начинаться с префикса `system_`, при этом в программном коде они будут обозначаться переменной с префиксом `$tbl_`. Список всех таблиц будет размещен в файле config/config.php, который будет отвечать за соединение с СУБД MySQL. В листинге 11.1 представлен фрагмент файла config.php, который содержит переменные с названиями таблиц.

Замечание

Новейшие нововведения в СУБД MySQL, такие как хранимые процедуры и функции, триггеры, планировщик заданий, представления, не будут использоваться в целях обеспечения максимальной совместимости со старыми версиями СУБД MySQL.

Листинг 11.1. Названия таблиц базы данных

```
<?php
  ...
  // Новости
  $tbl_news             = 'system_news';
  // Каталог
  $tbl_catalog          = 'system_catalog';
  // Позиция каталога
  $tbl_position         = 'system_position';
  ...
?>
```

Если вводится новая таблица, например, для имен загружаемых на сервер файлов, то в файл config/config.php добавляется новая переменная `$tbl_files` с названием новой таблицы "`system_files`". Такая система именований позволяет избежать конфликтов, вызываемых хранением в одной базе данных таблиц с одинаковыми именами. Если в базу данных потребуется добавить таблицу с именем, совпадающим с именем уже существующей таблицы, допускается переименование таблицы с конфликтным именем. Изменения, произведенные в файле config/config.php, отразятся на всей системе.

Система администрирования является наиболее развитым блоком CMS и имеет собственную структуру. Для каждого блока выделяется отдельный каталог:

```
/
  /dmn — система администрирования
    /system_accounts — управление аккаунтами
    /article — управление статьями
    /news — управление новостями
    /utils — общие файлы и авторизация
    /index.php — индексный файл
```

Индексный файл осуществляет переадресацию на один из блоков системы администрирования. В листинге 11.2 приводится пример файла index.php, выполняющего переадресацию на блок управления новостями.

Листинг 11.2. Содержимое индексного файла index.php

```
<?php
  // Переадресация на блок управления новостями
  header("Location: news/index.php");
?>
```

Основная сложность при построении универсальных систем управления сайтами заключается в том, что разным сайтам требуются совершенно разные блоки. Можно создать десятки и сотни блоков, однако это будет запутывать пользователя и разработчика — зачем в системе администрирования управление электронным магазином и блоком голосования, если ни того, ни другого на сайте нет?

Ориентируясь на быструю разработку Web-сайта, необходимо предусмотреть простой вариант удаления ненужных блоков из дистрибутива CMS, а также быстрый способ создания и подключения уникальных для данного сайта компонентов. Так как основным критерием выступает простота операций, регистрация блоков в файле или таблице базы данных сразу исключается — неподготовленному разработчику потребуется время на поиск такого файла или таблицы и выяснение, как корректно удалить или вставить блок, чтобы это не затронуло остальные части системы. Идеальной была бы ситуация, когда создание или удаление поддиректории в директории /dmn приводило бы к ее автоматической интеграции или к исключению блока из системы администрирования. При этом для создания нового блока сторонний Web-разработчик мог бы скопировать содержимое одного из существующих блоков и преобразовать его по аналогии.

Описанный выше подход требует, чтобы каждый из блоков, помещенный в отдельный каталог, сам себя описывал системе при помощи специального файла. При отсутствии данного файла блок назывался бы именем поддиректории.

Самоидентификация будет проводиться при помощи специального файла .htdir. В случае Web-сервера Apache имена файлов, которые начинаются с префикса .ht, недоступны для просмотра из Web-браузера. За это отвечает контейнер `<Files>` в конфигурационном файле httpd.conf (листинг 11.3).

Листинг 11.3. Файлы, начинающиеся с префикса .ht, недоступны для просмотра из Web-браузера

```
...
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy All
</Files>
...
```

Вообще говоря, данный контейнер предназначен для защиты содержимого конфигурационных файлов .htaccess и .htpasswd от любопытных глаз. Однако

можно воспользоваться такой защитой и с целью автоматически защитить собственные конфигурационные файлы.

Файл .htdir будет содержать две строки: первая с названием блока, вторая с его кратким описанием. Данная информация будет использоваться в меню системы администрирования. В листинге 11.4 представлено содержимое файла .htdir для блока управления новостями.

Листинг 11.4. Содержимое файла .htdir

```
Блок новости
Размещение, редактирование и удаление новостей
```

Таким образом, каждый блок предоставляет информацию о себе; если такая информация отсутствует, система пытается назвать его самостоятельно. Удаление поддиректории приводит к его автоматическому исключению из системы администрирования.

Конфигурационные файлы содержат служебную настроечную информацию. Помимо файла config.php, осуществляющего соединение с СУБД MySQL, блок конфигурационных файлов содержит файл class.config.php, который включает все ранее разработанные классы SoftTime FramWork (листинг 11.5).

Замечание

По мере разработки системы (добавление новых элементов управления или исключений) в данный файл необходимо включать новые файлы.

Листинг 11.5. Содержимое файла class.config.php

```
<?php
  require_once("class/class.field.php");
  require_once("class/class.field.text.php");
  require_once("class/class.field.text.english.php");
  require_once("class/class.field.text.int.php");
  require_once("class/class.field.text.email.php");
  require_once("class/class.field.password.php");
  require_once("class/class.field.textarea.php");
  require_once("class/class.field.hidden.php");
  require_once("class/class.field.hidden.int.php");
  require_once("class/class.field.radio.php");
  require_once("class/class.field.select.php");
  require_once("class/class.field.select.city.php");
  require_once("class/class.field.checkbox.php");
  require_once("class/class.field.file.php");
```

```
  require_once("class/class.field.date.php");
  require_once("class/class.field.datetime.php");
  require_once("class/class.field.paragraph.php");
  require_once("class/class.field.title.php");

  require_once("class/class.forms.php");

  require_once("class/exception.member.php");
  require_once("class/exception.mysql.php");
  require_once("class/exception.object.php");
?>
```

Теперь достаточно включить в файл блока представления лишь один файл class.config.php, чтобы скрипту был доступен весь набор классов для построения HTML-форм. Система администрирования и блок представления находятся на разных уровнях вложения, поэтому для системы представления следует создать отдельный файл class.config.dmn.php, включающий файлы для построения HTML-форм (листинг 11.6).

Замечание

В операционной системе UNIX текущий каталог обозначается при помощи символа точки ".", родительский каталог обозначается при помощи двух точек "..". Подобная же система была заимствована операционными системами DOS и Windows. В UNIX скрытые файлы и директории не имеют специального атрибута — все элементы файловой системы, начинающиеся с точки, считаются скрытыми. Именно поэтому конфигурационные файлы .htaccess, .htpasswd и наш файл .htdir также начинаются с точки.

Листинг 11.6. Содержимое файла class.config.dmn.php

```
<?php
  require_once("../../class/class.field.php");
  require_once("../../class/class.field.text.php");
  require_once("../../class/class.field.text.english.php");
  require_once("../../class/class.field.text.int.php");
  require_once("../../class/class.field.text.email.php");
  require_once("../../class/class.field.password.php");
  require_once("../../class/class.field.textarea.php");
  require_once("../../class/class.field.hidden.php");
  require_once("../../class/class.field.hidden.int.php");
  require_once("../../class/class.field.radio.php");
  require_once("../../class/class.field.select.php");
  require_once("../../class/class.field.select.city.php");
  require_once("../../class/class.field.checkbox.php");
```

```
  require_once("../../class/class.field.file.php");
  require_once("../../class/class.field.date.php");
  require_once("../../class/class.field.datetime.php");
  require_once("../../class/class.field.paragraph.php");
  require_once("../../class/class.field.title.php");

  require_once("../../class/class.forms.php");

  require_once("../../class/exception.member.php");
  require_once("../../class/exception.mysql.php");
  require_once("../../class/exception.object.php");
?>
```

11.2. Общие файлы системы администрирования

Остановимся подробнее на системе администрирования CMS, вернее, на директории общих файлов /dmn/utils. В данной директории сосредоточены служебные файлы, такие как шапка и завершение страницы, стилевые таблицы, системы авторизации, поиска модулей и формирования меню и другие вспомогательные файлы. Ниже приводится список основных файлов:

- bottom.php — завершение страницы;
- cms.css — каскадные таблицы стилей;
- menu.php — блок формирования меню (поиск блоков и чтение их .htdir-файлов);
- security_mod.php — блок безопасности, осуществляющий авторизацию администраторов и редакторов (см. *раздел 11.3*);
- top.php — шапка страницы;

В листинге 11.7 представлена типичная структура файла системы администрирования.

Листинг 11.7. Типичная структура файла системы администрирования

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем классы формы
  require_once("../../config/class.config.dmn.php");
```

```
  // Данные переменные определяют название страницы и подсказку.
  $title    = 'Название страницы';
  $pageinfo = '<p class=help>Здесь можно добавить описание страницы</p>';

  // Включаем заголовок страницы
  require_once("../utils/top.php");

  // Содержимое страницы

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
```

На рис. 11.2 представлен результат работы скрипта из листинга 11.7.

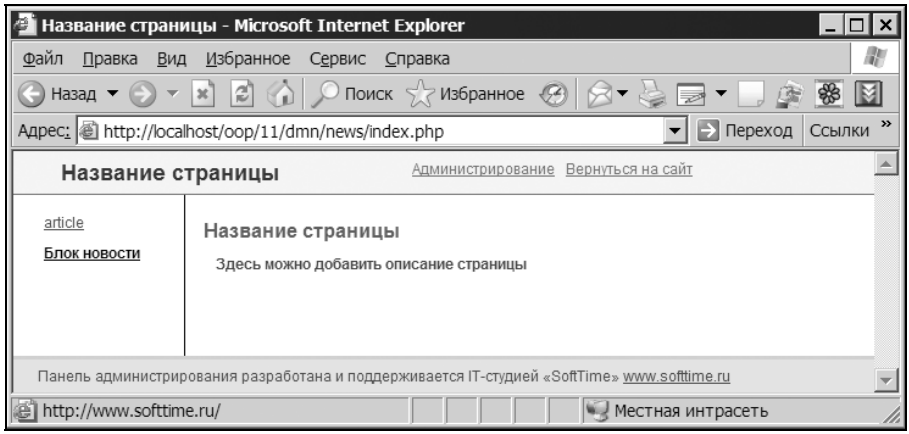


Рис. 11.2. Простейшая страница системы администрирования

Как видно из рис. 11.2, слева располагается меню, а справа — основное содержимое страницы, которое начинается с названия страницы и ее описания (оба этих элемента могут быть опущены, если не определены переменные `$title` и `$pageinfo`).

В листинге 11.8 приводится содержимое файла top.php, который отвечает за формирование заголовка страницы.

Листинг 11.8. Шапка страницы, top.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
?>
```

```
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html;
      charset=windows-1251">
<title><?php echo $title; ?></title>
<link rel="StyleSheet" type="text/css" href="../utils/cms.css">
</head>
<body leftmargin="0"
      marginheight="0"
      marginwidth="0"
      rightmargin="0"
      bottommargin="0"
      topmargin="0" >
<table width="100%"
       border="0"
       cellspacing="0"
       cellpadding="0"
       height="100%">
  <tr valign="top">
    <td colspan="3">
      <table class=topmenu border=0>
        <tr>
          <td width=5%> </td>
          <td>
            <h1 class=title><?php echo $title; ?></h1>
          </td>
          <td>

            <a href="../index.php"
               title="Вернуться на страницу администрирования сайта">
                 Администрирование</a>  

            <a href="../../index.php"
               title="Вернуться на головную страницу сайта" >
                 Вернуться на сайт</a>

          </td>
        </tr>
      </table>
    </td>
  </tr>
  <tr valign=top>
    <td class=menu>
```

```
      <?php
        // Формируем меню системы администрирования
        include "menu.php";
      ?>
    </td>
  <td class=main height=100%>
    <h1 class=namepage>
      <?php echo htmlspecialchars($title, ENT_QUOTES) ?>
    </h1>
    <?php echo $pageinfo ?><br><br>
```

Как видно из листинга 11.8, в файле top.php при помощи HTML-таблиц формируется шапка страницы, в которой выводится название страницы `$title` и ее описание `$pageinfo`.

В листинге 11.9 приводится содержимое файла bottom.php, который формирует завершение страницы.

Листинг 11.9. Завершение страницы, bottom.php

```
<br><br></td><td width=10%> </td></tr>
<tr class=authors>
  <td colspan="3">
     Панель администрирования разработана
     и поддерживается IT-студией "SoftTime"
     <a href="http://www.softtime.ru">www.softtime.ru</a></td></tr>
</table>
</body>
</html>
```

Шапка страницы top.php (листинг 11.8) включает в себя файл menu.php, который несет ответственность за поиск блоков в системе администрирования и формирование меню (листинг 11.10).

Листинг 11.10. Формирование меню, menu.php

```
<?php
  // Анализируем содержимое директории системы
  // администрирования для формирования меню

  // Открываем каталог /dmn
  $dir = opendir("..");
  // В цикле проходим по всем файлам
  // и поддиректориям
```

```
while (($file = readdir($dir)) !== false)
{
  // Обрабатываем только директории,
  // игнорируя файлы
  if(is_dir("../$file"))
  {
    // Исключаем текущую ".", родительскую ".."
    // директории, а также директорию utils
    if($file != "." && $file != ".." && $file != "utils")
    {
      // Ищем в директории файл с описанием
      // блока .htdir
      if(file_exists("../$file/.htdir"))
      {
        // Если файл .htdir существует,
        // читаем название блока и его
        // описание
        list($block_name,
             $block_description) = file("../$file/.htdir");
      }
      else
      {
        // Если файл .htdir не существует,
        // в качестве его названия
        // выступает имя поддиректории
        $block_name        = "$file";
        $block_description = "";
      }

      // Отмечаем текущий пункт другим стилем
      if(strpos($_SERVER['PHP_SELF'], $file) !== false)
      {
        $style = 'class=active';
      }
      else $style = '';

      // Формируем пункт меню
      echo "<div $style>
              <a class=menu
                 href='../$file'
                 title='$block_description'>
                 $block_name
              </a>
```

```
                </div>";
            }
        }
    }
    // Закрываем директорию
    closedir($dir);
?>
```

Скрипт из листинга 11.10 открывает директорию /dmn и последовательно читает файлы и поддиректории. При этом все файлы, а также поддиректории ".", ".." и "utils" игнорируются, все остальные поддиректории рассматриваются как блоки системы администрирования. Если файл файла .htdir присутствует в поддиректории, скрипт извлекает из него данные для отображения названия и описания во всплывающей подсказке. В противном случае вместо названия блока используется название поддиректории.

Текущий блок системы администрирования выделяется отличным от остальных блоков стилем. Методика определения текущей директории заключается в поиске ее названия по содержимому элемента суперглобального массива `$_SERVER['PHP_SELF']`, который возвращает адрес текущей страницы. Такая система накладывает определенные ограничения на названия директорий: они должны быть уникальными и каждое из них не должно быть частью названия других директорий, в противном случае текущим стилем будет выделяться два и более блоков.

На рис. 11.2 демонстрируется ситуация, когда система администрирования содержит два блока: `news`, в котором присутствует файл с описанием .htdir, и `article`, где такой файл отсутствует.

11.3. Ограничение доступа к системе администрирования

Система администрирования является центральным пультом управления сайтом, где можно добавлять, редактировать и удалять информацию. Разумеется, доступ к ней должен быть ограничен и защищен паролем. Для защиты приложения каждый его файл включает скрипт security_mod.php, который осуществляет аутентификацию и авторизацию администратора. Реализовать авторизацию удобно при помощи базовой HTTP-аутентификации, внешний вид которой демонстрируется на рис. 11.3.

Для реализации данного вида авторизации необходимо послать браузеру клиента HTTP-заголовки

```
WWW-Authenticate: Basic realm="Admin Page"
HTTP/1.0 401 Unauthorized
```

которые отобразят форму для ввода имени пользователя и пароля, представленную на рис. 11.3. Имя пользователя будет помещено браузером в переменную суперглобального массива `$_SERVER['PHP_AUTH_USER']`, а пароль — в `$_SERVER['PHP_AUTH_PW']`.

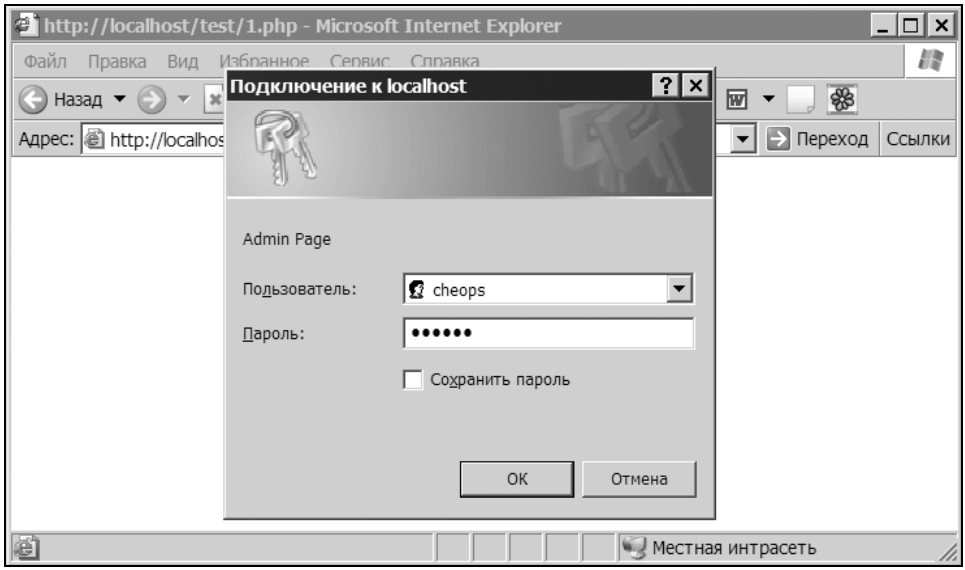


Рис. 11.3. Базовая HTTP-аутентификация

Замечание

Элементы суперглобального массива `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']` доступны, только если PHP установлен в качестве модуля, а не CGI-приложения. Определить тип подключения можно из отчета функции `phpinfo()`: если поле `Server API` принимает значение "Apache", PHP установлен в качестве модуля, если значение равно "CGI", — в качестве внешнего CGI-приложения. В последнем случае разумно удалить из файла security_mod.php код и защитить систему администрирования при помощи конфигурационных файлов .htaccess и .htpasswd (см. **http://www.softtime.ru/article/index.php?id_page=9**).

Имена пользователей и их пароли будем хранить в таблице `system_accounts` (листинг 11.11), которая в скриптах будет носить имя `$tbl_accounts`.

Замечание

Для того чтобы в систему можно было войти сразу, в листинге 11.11 при помощи оператора `INSERT` добавляется аккаунт с именем `root` и паролем "root".

Листинг 11.11. Таблица system_accounts ($tbl_accounts)

```
CREATE TABLE system_accounts (
  id_account INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pass TINYTEXT NOT NULL,
  PRIMARY KEY (id_account)
);
INSERT INTO system_accounts
VALUES (1, 'root', '63a9f0ea7bb98050796b649e85481845');
```

Таблица system_accounts обладает тремя полями, которые имеют следующие значения:

- id_account — первичный ключ таблицы, обладающий атрибутом AUTO_INCREMENT;
- name — имя пользователя;
- pass — его пароль, зашифрованный по алгоритму MD5.

Файл security_mod.php, использующийся для аутентификации и авторизации в системе управления содержимым сайта (CMS), может выглядеть так, как это представлено в листинге 11.12.

Листинг 11.12. Файл security_mod.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Если пользователь не авторизовался — авторизуем
  if(!isset($_SERVER['PHP_AUTH_USER']))
  {
    Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
    Header("HTTP/1.0 401 Unauthorized");
    exit();
  }
  else
  {
    // Проверяем переменные $_SERVER['PHP_AUTH_USER'] и
    // $_SERVER['PHP_AUTH_PW'], чтобы предотвратить
    // SQL-инъекцию
    if (!get_magic_quotes_gpc())
    {
      $_SERVER['PHP_AUTH_USER'] =
                          mysql_escape_string($_SERVER['PHP_AUTH_USER']);
```

```
      $_SERVER['PHP_AUTH_PW'] =
                          mysql_escape_string($_SERVER['PHP_AUTH_PW']);
    }

    $query = "SELECT pass FROM $tbl_accounts
              WHERE name='".$_SERVER['PHP_AUTH_USER']."'";
    $lst = @mysql_query($query);
    // При ошибке в SQL-запросе выводим окно
    if(!$lst)
    {
      Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если такого пользователя нет — выводим окно авторизации
    if(mysql_num_rows($lst) == 0)
    {
      Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если все проверки пройдены, сравниваем хэши паролей
    $pass = @mysql_fetch_array($lst);
    if(md5($_SERVER['PHP_AUTH_PW']) != $pass['pass'])
    {
      Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
  }
?>
```

Как видно из листинга, при неудачной авторизации клиенту отсылается повторное приглашение для ввода пароля:

```
WWW-Authenticate: Basic realm="Admin Page"
HTTP/1.0 401 Unauthorized
```

Далее работа скрипта останавливается при помощи функции `exit()`. Можно реализовать ограничение числа попыток ввода пароля: для этого достаточно вместо приведенных выше HTTP-заголовков послать заголовок **Страница не найдена**:

```
HTTP/1.0 404 Not Found
```

После того как модуль защиты security_mod.php создан, необходимо включить его при помощи директивы `require_once()` в начале защищаемых страниц (листинг 11.13).

Листинг 11.13. Защита страницы

```
<?php
  // Модуль безопасности
  require_once("security_mod.php");
  echo "Данные страницы, к которой необходимо получить доступ";
?>
```

Добавлять, редактировать и удалять новых пользователей системы администрирования при помощи SQL-запросов довольно неудобно, особенно если администрированием занимается пользователь незнакомый или не имеющий доступа к СУБД MySQL. Поэтому первым блоком системы администрирования разработаем блок управления пользователями, который расположим в поддиректории system_accounts. Внешний вид системы управления пользователями представлен на рис. 11.4.

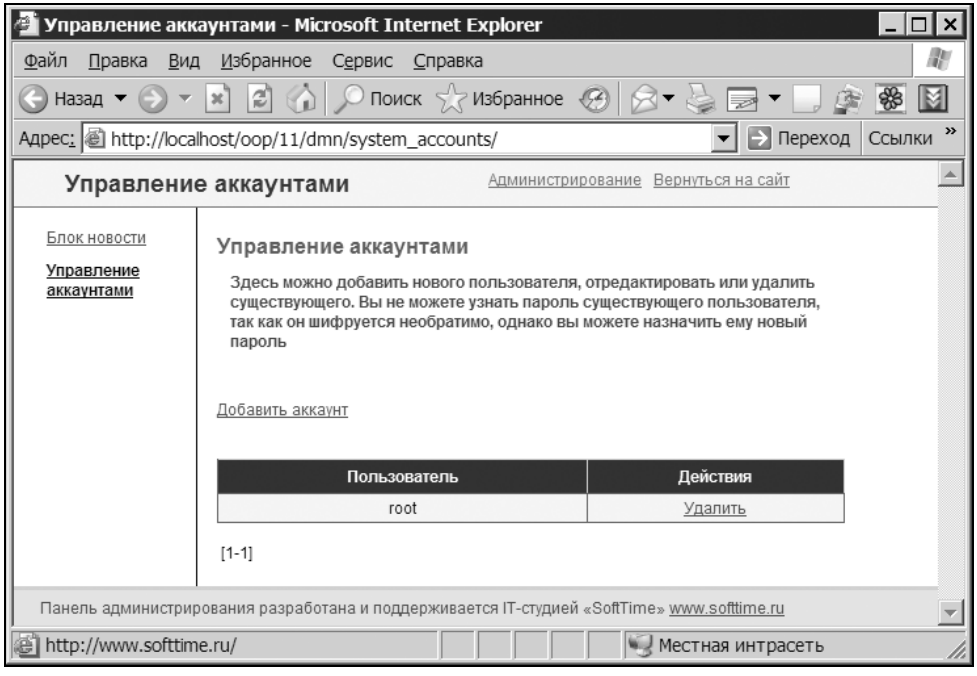


Рис. 11.4. Внешний вид системы управления аккаунтами

Замечание

При построении системы управления аккаунтами, а также в последующих блоках будет использоваться модифицированный вариант классов постраничной навигации, которые рассматривались в *главе 4*. Основное отличие новой иерархии классов заключается в том, что обработка ошибок обращения к СУБД MySQL осуществляется при помощи исключения `ExceptionMySQL` (см. *раздел 10.3*). Для этого в файлы config/class.config.php и config/class.config.dmn.php добавляются соответствующие инструкции включения. Полный вариант системы можно найти на компакт-диске, поставляемом вместе с книгой.

Система управления представляет собой таблицу с именами зарегистрированных пользователей, напротив каждого из которого имеется ссылка для его удаления. Последнего пользователя удалить нельзя (иначе нельзя будет войти в систему). При помощи управляющей ссылки "Добавить аккаунт" можно создать нового пользователя.

Блок "Управление аккаунтами", помимо файла с описанием, содержит три следующих файла:

- index.php — главная страница блока управления аккаунтами, представленная на рис. 11.4;
- addaccount.php — HTML-форма для добавления нового пользователя;
- delaccount.php — удаление пользователя.

В листинге 11.14 представлена реализация главной страницы index.php.

Листинг 11.14. Главная страница блока управления аккаунтами, index.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Данные переменные определяют название страницы и подсказку.
  $title = 'Управление аккаунтами';
  $pageinfo = '<p class=help>Здесь можно добавить нового
               пользователя, удалить или отредактировать
               данные существующего. Вы не можете узнать
               пароль существующего пользователя, т. к.
               его шифрование необратимо, однако вы можете
               назначить ему новый пароль</p>';
```

```
// Включаем заголовок страницы
require_once("../utils/top.php");

try
{
  // Количество ссылок в постраничной навигации
  $page_link = 3;
  // Количество позиций на странице
  $pnumber = 10;
  // Объявляем объект постраничной навигации
  $obj = new pager_mysql($tbl_accounts,
                         "",
                         "ORDER BY name",
                         $pnumber,
                         $page_link);

  // Добавить аккаунт
  echo "<a href=addaccount.php?page=$_GET[page]
           title='Добавить новый аккаунт'>
           Добавить аккаунт</a><br><br>";

  // Получаем содержимое текущей страницы
  $accounts = $obj->get_page();
  // Если имеется хотя бы одна запись — выводим ее
  if(!empty($accounts))
  {
    ?>
    <table width="100%"
           class="table"
           border="0"
           cellpadding="0"
           cellspacing="0">
      <tr class="header" align="center">
        <td>Пользователь</td>
        <td>Действия</td>
      </tr>
    <?php
    for($i = 0; $i < count($accounts); $i++)
    {
      // Выводим строку таблицы
      echo "<tr>
              <td align=center>{$accounts[$i][name]}</td>
              <td align=center>
```

```
                <a href=#
                   onClick=\"delete_account('".
                  "delaccount.php?page=$_GET[page]&".
                  "id_account={$accounts[$i][id_account]}');\"
                   title='Удалить пользователя'>Удалить</a></td>
              </tr>";
      }
      echo "</table><br>";
    }

    // Выводим ссылки на другие страницы
    echo $obj;
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
<script language="JavaScript"  type='text/javascript'>
<!--
  function delete_account(url)
  {
    if(confirm("Вы действительно хотите удалить аккаунт?"))
    {
      location.href=url;
    }
    return false;
  }
//-->
</script>
```

Как видно из листинга 11.14, вывод аккаунтов из таблицы `system_accounts` (`$tbl_accounts`) осуществляется при помощи объекта класса `pager_mysql` (см. *главу 4*), который обеспечивает постраничное представление результата SELECT-запроса. Возможные ошибки MySQL-сервера обрабатываются при помощи исключения типа `ExceptionMySQL` (рис. 11.5).

Перед таблицей с результатами запроса выводится ссылка на скрипт добавления нового пользователя addaccount.php. В самой таблице результатов ссылки на скрипт удаления пользователей delaccount.php оформляются в виде

функции JavaScript, которая перед удалением запрашивает подтверждение (рис. 11.6).

Рис. 11.5. Обработка исключительной ситуации `ExceptionMySQL`

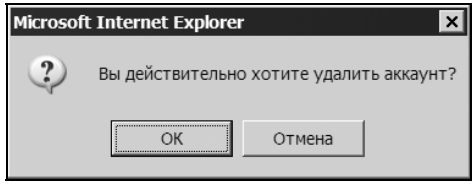


Рис. 11.6. Запрос на подтверждение удаления аккаунта

Формирование HTML-формы для добавления нового пользователя осуществляется по схеме, представленной в *разделе 10.7*. В листинге 11.15 представлена возможная реализация файла addaccount.php, ответственного за добавление нового пользователя.

Листинг 11.15. Добавление нового пользователя, addaccount.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");
  // Подключаем генератор паролей
  require_once("../utils/utils.password.php");
```

```
// Генерируем новый пароль
$pass_example = generate_password(10);

// Параметры формы
$button_name = "Добавить";
$class_name = "field";

$name = new field_text_english("name",
                               "Имя пользователя",
                               true,
                               $_POST['name']);
$pass = new field_password("pass",
                           "Пароль",
                           true,
                           $_POST['pass'],
                           255,
                           41,
                           "",
                           "Например, $pass_example");
$passag = new field_password("passag",
                           "Повтор пароля",
                           true,
                           $_POST['passag'],
                           255,
                           41,
                           "",
                           " Например, $pass_example");
$page = new field_hidden_int("page",
                             false,
                             $_REQUEST['page']);

try
{
  $form = new form(array("name"   => $name,
                         "pass"   => $pass,
                         "passag" => $passag,
                         "page"   => $page),
                   $button_name,
                   $class_name);
}
catch(ExceptionObject $exc)
{
  require("../utils/exception_object.php");
}
```

```
if(!empty($_POST))
{
  try
  {
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();

    if($form->fields['pass']->value != $form->fields['passag']->value)
    {
      $error[] = "Пароли не равны";
    }
    // Проверяем, не регистрировался ли ранее пользователь
    // с таким электронным адресом
    $query = "SELECT COUNT(*) FROM $tbl_accounts
            WHERE name = '{$form->fields[name]->value}'";
    $acc = mysql_query($query);
    if(!$acc)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                               "Ошибка добавления нового
                                пользователя");
    }
    if(mysql_result($acc, 0))
    {
      $error[] = "Пользователь с именем
                  {$form->fields[name]->value} уже
                  зарегистрирован";
    }

    // Если ошибок нет, добавляем нового пользователя
    if(empty($error))
    {
      $query = "INSERT INTO $tbl_accounts
                VALUES (NULL,
                        '{$form->fields[name]->value}',
                         MD5('{$form->fields[pass]->value}'))";
      if(!mysql_query($query))
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка добавления нового
                                 пользователя");
      }
```

```
        // Перегружаем страницу для сброса POST-данных
        header("Location: index.php?page=".$form->fields['page']->value);

        exit();
      }
    }
    catch(ExceptionMySQL $exc)
    {
      require("../utils/exception_mysql.php");
    }
  }

  // Включаем заголовок страницы
  $title = "Добавление аккаунта";
  $pageinfo = '<p class=help>Имя пользователя и пароль могут содержать
               только символы латинского алфавита</p>';
  require_once("../utils/top.php");

  // Если при работе произошли ошибки — выводим сообщения
  if(!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
```

Скрипт addaccount.php из листинга 11.15 формирует HTML-форму, состоящую из нескольких элементов управления: текстового поля `name` для ввода имени пользователя, два поля типа `password` для ввода пароля `pass` и его повтора `passage`, а также скрытое поле `page` для передачи номера текущей страницы при постраничной навигации, чтобы вернуться на ту же страницу, откуда был осуществлен переход на страницу добавления нового аккаунта.

Если при вводе данных не возникает никаких ошибок (введено незарегистрированное ранее имя пользователя, пароли, причем пароли совпадают друг с другом), пользователь добавляется в СУБД MySQL при помощи INSERT-запроса. При этом пароль необратимо шифруется по алгоритму MD5, что по-

зволяет значительно снизить риск подбора пароля, даже в случае воровства злоумышленником базы данных.

На рис. 11.7 представлен внешний вид HTML-формы для добавления нового аккаунта.

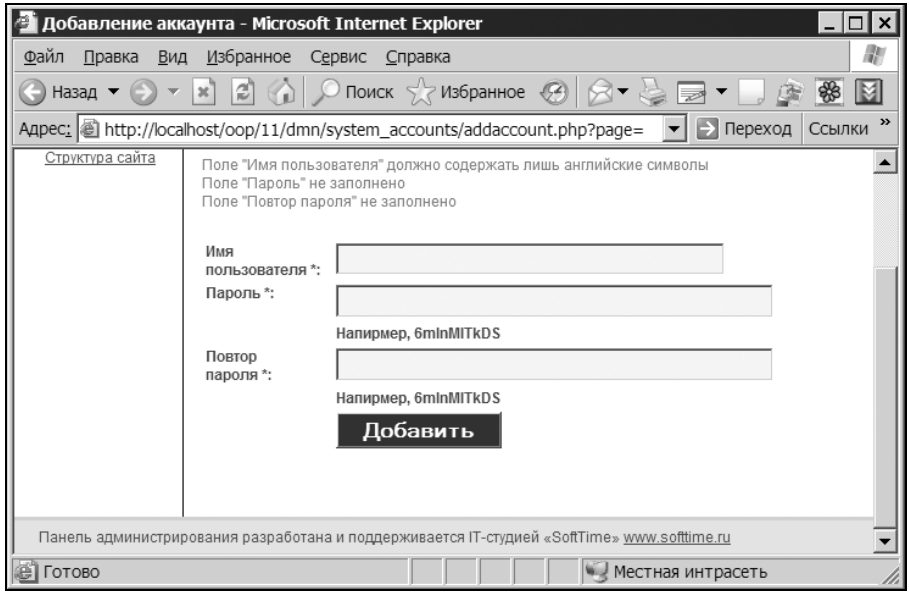


Рис. 11.7. Добавление нового аккаунта

Помимо элементов управления, под паролем и его повтором выводится параграф с примером пароля, который трудно подобрать. Пароль генерируется при помощи специальной функции `generate_password()` из файла utils/utils.password.php. Функция принимает в качестве параметра количество символов, которое должен содержать пароль. В листинге 11.16 приводится содержимое файла utils/utils.password.php.

Листинг 11.16. Функция `generate_password()`

```
<?php
  ////////////////////////////////////////////////
  // Функция генерирует пароль,
  // $number – количество символов в пароле
  ////////////////////////////////////////////////
  function generate_password($number = 10)
  {
    $arr = array('a','b','c','d','e','f',
                 'g','h','i','j','k','l',
```

```
                'm','n','o','q','p','r','s',
                't','u','v','w','x','y','z',
                'A','B','C','D','E','F',
                'G','H','I','J','K','L',
                'M','N','O','Q','P','R','S',
                'T','U','V','W','X','Y','Z',
                '1','2','3','4','5','6',
                '7','8','9','0','_');
    // Генерируем пароль
    $pass = "";
    for($i = 0; $i < $number; $i++)
    {
      // Вычисляем случайный индекс массива
      $index = rand(0, count($arr) — 1);
      $pass .= $arr[$index];
    }
    return $pass;
  }
?>
```

Удаление пользователей из базы данных осуществляется при помощи файла delaccount.php (листинг 11.17).

Листинг 11.17. Удаление пользователей, delaccount.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Проверяем GET-параметр, предотвращая SQL-инъекцию
  $_GET['id_account'] = intval($_GET['id_account']);

  try
  {
    // Проверяем, не удаляется ли последний аккаунт:
    // если его удалить, в систему невозможно будет войти
    $query = "SELECT COUNT(*) FROM $tbl_accounts";
    $acc = mysql_query($query);
    if(!$acc)
```

```
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка удаления
                               пользователя");
    }
    if(mysql_result($acc, 0) > 1)
    {
      $query = "DELETE FROM $tbl_accounts
                WHERE id_account=".$_GET['id_account'];
      if(mysql_query($query))
      {
        header("Location: index.php?page=".$_GET['page']);
      }
      else
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка удаления
                                 пользователя");
      }
    }
    else
    {
      throw new Exception("Нельзя удалить
                           единственный аккаунт");
    }
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
  catch(Exception $exc)
  {
    require("../utils/exception.php");
  }
?>
```

Скрипту передается в качестве GET-параметра `id_account` первичный ключ удаляемого пользователя. Для того чтобы предотвратить SQL-инъекцию, параметр `$_GET['id_account']` приводится к целому значению при помощи функции `intval()`. Далее скрипт проверяет, не является ли удаляемая запись последней в таблице `system_accounts` (`$tbl_accounts`), и при положительном

результате отказывает в удалении, иначе штатно войти в систему администрирования уже не получится.

При возникновении ошибки пользователя генерируется исключение класса `Exception` с единственным параметрам — текстовым сообщением. Для обработки таких исключений вводится специальный скрипт utils/exception.php, содержимое которого приводится в листинге 11.18.

Листинг 11.18. Обработчик исключений типа `Exception`

```
<?php
  // Включаем заголовок страницы
  require_once("../utils/top.php");

  echo "<p class=help>{$exc->getMessage()}</p>";
  echo "<p class=help><a href=#
                        onclick='history.back()'>Вернутся</a></p>";

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
  exit();
?>
```

Главным отличием данного типа исключений от рассмотренных ранее `ExceptionMember`, `ExceptionObject` и `ExceptionMySQL` заключается в том, что исключения типа `Exception` будут применяться не для отладки PHP-кода, а для сообщения об ограничениях или ошибках конечному пользователю системы. Поэтому обработчик исключений типа `Exception` не содержит сообщений

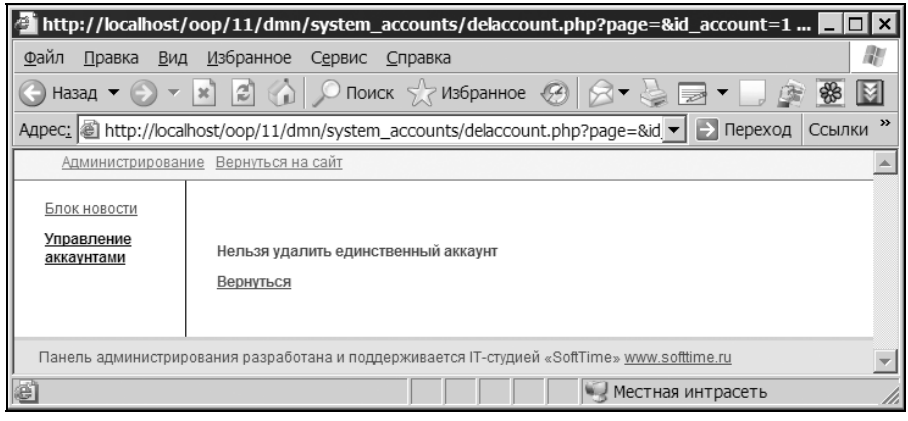


Рис. 11.8. Исключение `Exception` предназначено для пользователя, а не для разработчика

о файле и номере строки, где произошло исключение, он лишь сообщает причину возникновения исключительной ситуации и предоставляет ссылку для возврата на предыдущую страницу (рис. 11.8).

Замечание

Для исключений, предназначенных для информирования конечных пользователей, а не для отладки программы, можно предусмотреть отдельный класс, унаследованный от базового класса `Exception`, однако никакая дополнительная функциональность для него не требуется, а исключение должно обрабатываться в любом случае. Поэтому введение специального типа исключительной ситуации нецелесообразно.

11.4. Блок новости

Блок новости является первым блоком системы управления содержимым сайта (CMS), включающим как систему администрирования, так и блок представления. Очень часто возникает вопрос, с какой системы начинать разработку блока. Обычно первым разрабатывается тот участок кода, который ответственен за наполнение базы данных — это позволяет быстрее получить работоспособный скрипт, начать его отладку и поиск ошибок. Так, для форума, гостевой книги, доски объявлений, чата в первую очередь разрабатывается система представления и лишь затем — система администрирования. Для блока новостей, ответов-вопросов, управления статьями, каталогов продукции, фотогалерей и т. п. в первую очередь разрабатывается система администрирования, которая позволила бы заполнять базу данных, и только потом — система представления, отображающая содержимое базы данных посетителям.

11.4.1. База данных

Остановимся на разработке простейшей новостной системы, где новостные сообщения следуют одно за другим в календарном порядке. Для упрощения задачи не будем разделять новости на категории, что позволит обойтись одной таблицей `system_news`, которая состоит из восьми столбцов:

- `id_news` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`, который обеспечивает автоматическое формирование уникального номера записи;
- `name` — название новостного сообщения;
- `body` — текст новости;
- `putdate` — дата размещения новости в формате "`YYYY-MM-DD hh:mm:ss`";
- `url` — ссылка;

- `urltext` — текст ссылки;
- `urlpict` — путь к изображению, сопровождающему новость;
- `hide` — служебное поле типа `ENUM`, принимающее только два значения: `hide`, если новость скрыта и недоступна для просмотра со страниц сайта, и `show`, если новость следует отображать на страницах сайта.

В листинге 11.19 приводится оператор `CREATE TABLE`, позволяющий создать таблицу `system_news`.

Листинг 11.19. Таблица `system_news` (`$tbl_news`)

```
CREATE TABLE system_news (
  id_news INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  body TEXT NOT NULL,
  putdate DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  url TINYTEXT NOT NULL,
  urltext TINYTEXT NOT NULL,
  urlpict TINYTEXT NOT NULL,
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  PRIMARY KEY (id_news)
);
```

11.4.2. Система администрирования

Система администрирования блока новостей, позволяющая добавлять, редактировать и удалять новостные сообщения, состоит из шести файлов:

- index.php — главная страница, отображающая список новостных сообщений и управляющие ссылки;
- addnews.php — HTML-форма, позволяющая добавлять новостные сообщения;
- editnews.php — HTML-форма, позволяющая редактировать новостные сообщения;
- delnews.php — скрипт для удаления новостного сообщения;
- hide.php — скрипт, скрывающий новостное сообщение;
- show.php — скрипт, отображающий новостное сообщение.

Главный файл index.php отображает список доступных новостных сообщений, а также управляющие ссылки, позволяющие добавлять новостные блоки и подвергать их редактированию или удалению (рис. 11.9).

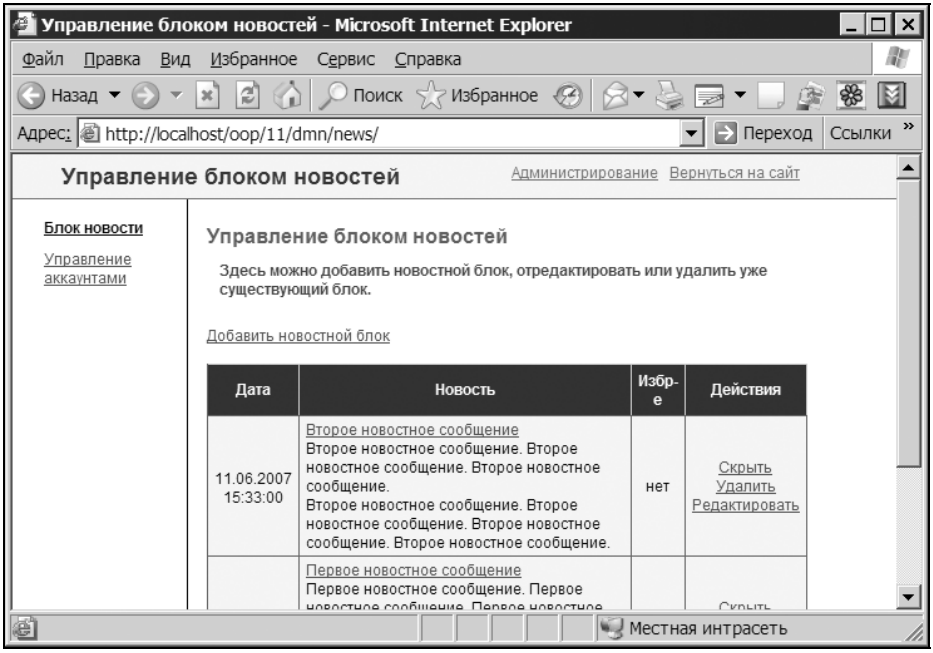


Рис. 11.9. Главная страница системы администрирования блока новостей

Как видно из рис. 11.9, на главной странице блока новостей располагается управляющая ссылка **Добавить новостной блок**, а также таблица, каждая строка которой соответствует одному новостному сообщению. Таблица содержит четыре столбца:

- дату размещения новости;
- содержимое новости (включая название, текст и ссылки);
- изображение (если оно присутствует);
- столбец с управляющими ссылками, позволяющий редактировать, удалять, скрывать/отображать новостное сообщение.

В листинге 11.20 приводится возможная реализация главной страницы системы администрирования блока новостей.

Листинг 11.20. Главная страница блока новостей

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
```

```
// Подключаем SoftTime FrameWork
require_once("../../config/class.config.dmn.php");
// Подключаем блок отображения текста в окне браузера
require_once("../utils/utils.print_page.php");

// Данные переменные определяют название страницы и подсказку.
$title = 'Управление блоком "Блок новостей"';
$pageinfo = '<p class=help>Здесь можно добавить
             новостной блок, отредактировать или
             удалить уже существующий блок.</p>';

// Включаем заголовок страницы
require_once("../utils/top.php");

try
{
  // Количество ссылок в постраничной навигации
  $page_link = 3;
  // Количество позиций на странице
  $pnumber = 10;
  // Объявляем объект постраничной навигации
  $obj = new pager_mysql($tbl_news,
                         "",
                         "ORDER BY putdate DESC",
                         $pnumber,
                         $page_link);

  // Добавить аккаунт
  echo "<a href=addnews.php?page=$_GET[page]
           title='Добавить новостной блок'>
           Добавить новостной блок</a><br><br>";

  // Получаем содержимое текущей страницы
  $news = $obj->get_page();
  // Если имеется хотя бы одна запись — выводим
  if(!empty($news))
  {
    ?>
    <table width="100%"
           class="table"
           border="0"
           cellpadding="0"
           cellspacing="0">
```

```
  <tr class="header" align="center">
    <td width=200>Дата</td>
    <td width=60%>Новость</td>
    <td width=40>Избр-е</td>
    <td>Действия</td>
  </tr>
<?php
for($i = 0; $i < count($news); $i++)
{
  // Если новость отмечена как невидимая (hide='hide'), выводим
  // ссылку "отобразить", если как видимая (hide='show') — "скрыть"
  $colorrow = "";
  $url = "?id_news={$news[$i][id_news]}&page=$page";
  if($news[$i]['hide'] == 'show')
  {
    $showhide = "<a href=hide.php$url
                    title='Скрыть новость в блоке новостей'>
                 Скрыть</a>";
  }
  else
  {
    $showhide = "<a href=show.php$url
                    title='Отобразить новость в блоке новостей'>
                 Отобразить</a>";
    $colorrow = "class='hiddenrow'";
  }
  // Проверяем наличие изображения
  if($news[$i]['urlpict'] != '' &&
     $news[$i]['urlpict'] != '-' &&
     is_file("../../".$news[$i]['urlpict']))
  {
  $url_pict = "<b><a href=../../{$news[$i][urlpict]}>есть</a></b>";
  }
  else $url_pict = "нет";

  $news_url="";
  if (!empty($news[$i]['url']))
  {
    if(!preg_match("|^http://|i",$news[$i]['url']))
    {
      $news[$i]['url'] = "http://{$news[$i][url]}";
    }
```

```
        $news_url = "<br><b>Ссылка:</b>
                     <a href='{$news[$i][url]}'>
                        {$news[$i][urltext]}</a>";
        if(empty($news[$i]['urltext']))
        {
          $news_url = "<br><b>Ссылка:</b>
                       <a href='{$news[$i][url]}'>
                          {$news[$i][url]}</a>";
        }
      }

      // Преобразуем дату из формата MySQL YYYY-MM-DD hh:mm:ss
      // в формат DD.MM.YYYY hh:mm:ss
      list($date, $time) = explode(" ", $news[$i]['putdate']);
      list($year, $month, $day) = explode("-", $date);
      $news[$i]['putdate'] = "$day.$month.$year $time";

      // Выводим новость
      echo "<tr $colorrow >
              <td><p align=center>{$news[$i][putdate]}</td>
              <td>
                <a title='Редактировать текст новости'
                   href=editnews.php$url>{$news[$i][name]}</a><br>
               ".nl2br(print_page($news[$i]['body']))." $news_url </td>
              <td align=center>$url_pict</td>
              <td align=center>$showhide<br>
                 <a href=# onClick=\"delete_news('delnews.php$url');\"
                    title='Удалить новость'>Удалить</a><br>
                 <a href=editnews.php$url}
               title='Редактировать текст новости'>Редактировать</a></td>
             </tr>";
    }
    echo "</table><br>";
  }

  // Выводим ссылки на другие страницы
  echo $obj;
}
catch(ExceptionMySQL $exc)
{
  require("../utils/exception_mysql.php");
}
```

```
  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
<script language="JavaScript">
  function delete_news(url)
  {
    if(confirm("Вы действительно хотите удалить новостное сообщение?"))
    {
      location.href=url;
    }
    return false;
  }
</script>
```

Как видно из листинга 11.20, вывод новостных блоков из таблицы `system_news` (`$tbl_news`) осуществляется при помощи объекта класса `pager_mysql` (см. *главу 4*), который обеспечивает постраничное представление результата SELECT-запроса. Возможные ошибки MySQL-сервера обрабатываются при помощи исключения типа `ExceptionMySQL`.

Текст новости может содержать теги bbCode, предоставляющие пользователю возможность форматирования текста без использования потенциально опасного HTML-кода:

- `[b]...[/b]` — текст, заключенный в эти теги, выделяется жирным шрифтом; их использование эквивалентно `<b>...</b>`;
- `[i]...[/i]` — текст, заключенный в эти теги, выделяется курсивом (наклонным шрифтом); их использование эквивалентно `<i>...</i>`;
- [url]http://www.site.ru[/url] — ссылка, преобразующаяся к виду http://www.site.ru;
- [url=http://www.site.ru]текст[/url] — ссылка, преобразующаяся к виду текст.

Для их автоматической обработки перед выводом в окно браузера используется функция `print_page()` из файла utils/utils.print_page.php (листинг 11.21).

Листинг 11.21. Обработка текста перед выводом в окно браузера

```
<?php
function print_page($postbody)
{
    // Разрезаем слишком длинные слова
    $postbody = preg_replace_callback(
              "|([a-za-я\d!]{35,})|i",
```

```
            "split_text",
            $postbody);
    // Предотвращаем XSS-инъекции
    $postbody = htmlspecialchars($postbody, ENT_QUOTES);
    // Теги
    $pattern = "#\[b\](.+)\[\/b\]#isU";
    $postbody = preg_replace($pattern,
                             '<b>\\1</b>',
                             $postbody);
    $pattern = "#\[i\](.+)\[\/i\]#isU";
    $postbody = preg_replace($pattern,
                             '<i>\\1</i>',
                             $postbody);
   $pattern = "#\[url\][\s]*((?=(http:|mailto:))[\S]*)[\s]*\[\/url\]#si";
    $postbody = preg_replace($pattern,
                             '<a href="\\1" target=_blank>\\1</a>',
                             $postbody);
    $pattern = "#\[url[\s]*=[\s]*((?=(http:|mailto:))".
               "[\S]+)[\s]*\][\s]*([^\[]*)\[/url\]#isU";
    $postbody = preg_replace($pattern,
                             '<a href="\\1" target=_blank>\\3</a>',
                             $postbody);
    return $postbody;
}
function split_text($matches)
{
  return wordwrap($matches[1], 35, ' ',1);
}
?>
```

Функция `print_page()` принимает текст и преобразует все bbCode в их HTML-эквиваленты, предварительно преобразовав текст при помощи функции `htmlspecialchars()` и разбив слишком длинные слова при помощи функции `preg_replace_callback()` и функции обратного вызова `split_text()`.

Добавление новостного блока осуществляется при помощи скрипта addnews.php, представленного в листинге 11.22.

Листинг 11.22. Добавление новостного блока, addnews.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
```

```
// Подключаем классы формы
require_once("../../config/class.config.dmn.php");

if(empty($_POST))
{
  // Отмечаем флажок hide
  $_REQUEST['hide'] = true;
}
$name        = new field_text("name",
                              "Название",
                              true,
                              $_POST['name']);
$body = new field_textarea("body",
                           "Содержимое",
                           true,
                           $_POST['body']);
$url         = new field_text("url",
                              "Ссылка",
                              false,
                              $_POST['url']);
$urltext    = new field_text("urltext",
                             "Текст ссылки",
                              false,
                              $_POST['urltext']);
$date        = new field_datetime("date",
                              "Дата новости",
                              $_POST['date']);
$hide        = new field_checkbox("hide",
                                  "Отображать",
                                  $_REQUEST['hide']);
$urlpict   = new field_file("urlpict",
                            "Изображение",
                            false,
                            $_FILES,
                            "../../files/news/");
$page    = new field_hidden_int("page",
                                "",
                                true,
                                $_REQUEST['page']);

try
{
  $form = new form(array("name" => $name,
                         "body" => $body,
```

```
                        "url" => $url,
                        "urltext" => $urltext,
                        "date" => $date,
                        "hide" => $hide,
                        "urlpict" => $urlpict,
                        "page" => $page),
                    "Добавить",
                    "field");
}
catch(ExceptionObject $exc)
{
  require("../utils/exception_object.php");
}

// Обработчик HTML-формы
if(!empty($_POST))
{
  try
  {
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();
    if(empty($error))
    {
      // Скрытая или открытая директория
      if($form->fields['hide']->value) $showhide = "show";
      else $showhide = "hide";
      // Изображение
      $str = $form->fields['urlpict']->get_filename();
      if(!empty($str))
      {
        $img = "files/news/".$form->fields['urlpict']->get_filename();
      }
      else $img = '';
      // Формируем SQL-запрос на добавление
      // новостного сообщения
      $query = "INSERT INTO $tbl_news
                VALUES (NULL,
                        '{$form->fields[name]->value}',
                        '{$form->fields[body]->value}',
                        '{$form->fields[date]->get_mysql_format()}',
                        '{$form->fields[url]->value}',
                        '{$form->fields[urltext]->value}',
```

```
                        '$img',
                        '$showhide')";
        if(!mysql_query($query))
        {
          throw new ExceptionMySQL(mysql_error(),
                                   $query,
                                  "Ошибка добавления новостного
                                   сообщения");
        }
        // Осуществляем перенаправление
        // на главную страницу администрирования
        header("Location: index.php?page={$form->fields[page]->value}");
        exit();
      }
    }
    catch(ExceptionMySQL $exc)
    {
      require("../utils/exception_mysql.php");
    }
  }
  // Начало страницы
  $title     = 'Добавление новостного сообщения';
  $pageinfo  = '<p class=help></p>';
  // Включаем заголовок страницы
  require_once("../utils/top.php");

  echo "<p><a href=# onClick='history.back()'>Назад</a></p>";
  // Выводим сообщения об ошибках, если они имеются
  if(!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
```

HTML-форма для добавления новостного сообщения содержит три текстовых поля для названия name, ссылки `url` и текста ссылки `urltext`. Помимо этого,

текстовая область `body` для текста новости, блок даты и времени для выбора времени размещения новости (по умолчанию подставляется текущее время), флажок `hide` для выбора статуса новостного сообщения (скрытое или доступное для просмотра) и поле для загрузки изображения на сервер `urlpict`. Помимо этого, через скрытое поле `page` передается номер страницы в системе постраничной навигации для корректного возврата на страницу, с которой посетитель осуществляет добавление новостного сообщения. Внешний вид HTML-формы приведен на рис. 11.10.

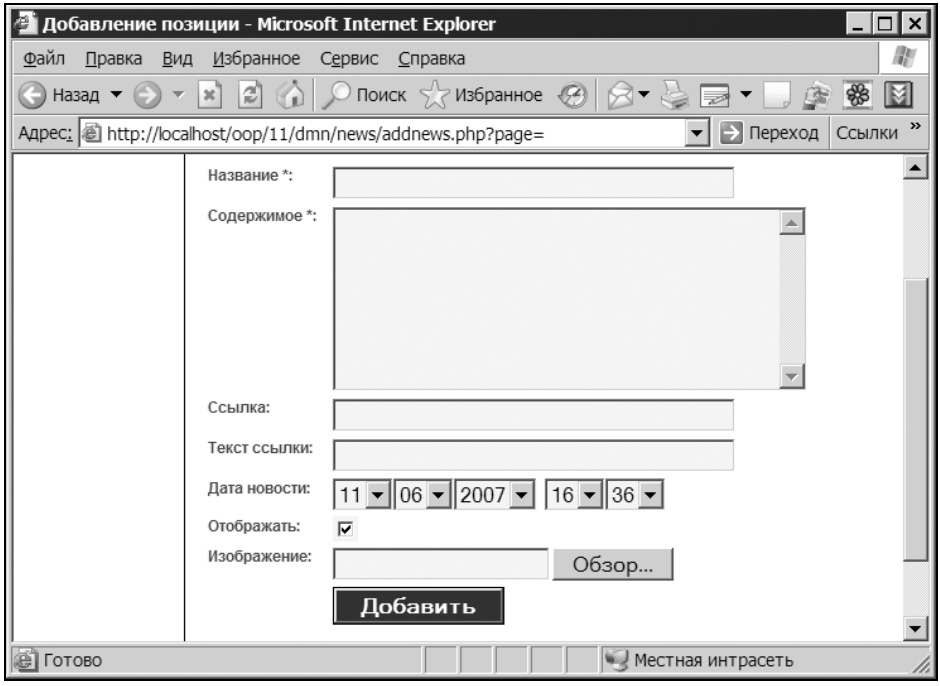


Рис. 11.10. Добавление новостного сообщения

Общий механизм добавления новой записи в базу данных совпадает с ранее рассмотренными HTML-формами. Пожалуй, подробнее стоит остановиться на загрузке файла изображения на сервер. Для хранения файлов в корне сайта обычно выделяют отдельную директорию, например, `/files`, при этом для каждого блока в директории выделяется отдельная поддиректория:

```
/files
  /news
  /article
  /catalog
```

Замечание

В UNIX, который традиционно используется в качестве операционной системы на серверах хост-провайдеров, владелец файлов и пользователь, из-под учетной записи которого запущен Web-сервер, зачастую различны. Более того, иногда они даже не входят в общую группу, поэтому для того чтобы PHP-скрипт, работающий от имени пользователя Apache, мог осуществлять запись в директорию, приходится давать ему максимальные права доступа — 0777. На современных серверах это абсолютно безопасно, т. к. исключается основная опасность — возможность редактирования файлов из скриптов соседнего аккаунта.

Каждый из блоков хранит свои файлы в отдельной поддиректории, однако уровень вложения блока представления и системы администрирования не совпадают, и при оперировании в системе администрирования путь к файлу следует предварять префиксом "../../". В базу данных при этом путь помещается без префикса — это позволяет оперировать путями к файлам из системы представления, не добавляя никаких префиксов.

Редактирование новостного блока осуществляется при помощи скрипта editnews.php, содержимое которого представлено в листинге 11.23. Скрипту передается в качестве GET-параметра `id_news` первичный ключ редактируемого новостного сообщения. Помимо этого скрипт может принимать необязательный GET-параметр `page`, обеспечивающий возврат на текущую страницу в рамках постраничной навигации.

Листинг 11.23. Редактирование новостного блока, editnews.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем классы формы
  require_once("../../config/class.config.dmn.php");

  // Предотвращаем SQL-инъекцию
  $_GET['id_news'] = intval($_GET['id_news']);

  try
  {
    // Извлекаем из таблицы news запись, соответствующую
    // исправляемому новостному сообщению
    $query = "SELECT * FROM $tbl_news
              WHERE id_news=$_GET[id_news]";
```

```
$new = mysql_query($query);
if(!$new)
{
  throw new ExceptionMySQL(mysql_error(),
                           $query,
                          "Ошибка при обращении
                           к таблице новостей");
}
$news = mysql_fetch_array($new);
if(empty($_POST))
{
  // Берем информацию для оставшихся переменных из базы данных
  $_REQUEST = $news;
  $_REQUEST['date']['month']  = substr($news['putdate'],5,2);
  $_REQUEST['date']['day']    = substr($news['putdate'],8,2);
  $_REQUEST['date']['year']   = substr($news['putdate'],0,4);
  $_REQUEST['date']['hour']   = substr($news['putdate'],11,2);
  $_REQUEST['date']['minute'] = substr($news['putdate'],14,2);
  // Определяем, скрыто поле или нет
  if($news['hide'] == 'show') $_REQUEST['hide'] = true;
  else $_REQUEST['hide'] = false;
}

$name        = new field_text("name",
                              "Название",
                              true,
                              $_REQUEST['name']);
$body = new field_textarea("body",
                           "Содержимое",
                           true,
                           $_REQUEST['body']);
$url         = new field_text("url",
                              "Ссылка",
                              false,
                              $_REQUEST['url']);
$urltext    = new field_text("urltext",
                              "Текст ссылки",
                              false,
                              $_REQUEST['urltext']);
$date        = new field_datetime("date",
                              "Дата новости",
                              $_REQUEST['date']);
```

```
$hide        = new field_checkbox("hide",
                          "Отображать",
                          $_REQUEST['hide']);
$filename    = new field_file("filename",
                            "Изображение",
                            false,
                            $_FILES,
                            "../../files/news/");
$id_news     = new field_hidden_int("id_news",
                                    "",
                                    $_REQUEST['id_news']);
$page        = new field_hidden_int("page",
                                    "",
                                    $_REQUEST['page']);
// Инициируем форму массивом из двух элементов
// управления — поля ввода name и текстовой области
// textarea
try
{
  if(empty($news['urlpict']))
  {
    $form = new form(array("name" => $name,
                           "body" => $body,
                           "url" => $url,
                           "urltext" => $urltext,
                           "date" => $date,
                           "hide" => $hide,
                           "filename" => $filename,
                           "id_news" => $id_news,
                           "page" => $page),
                  "Редактировать",
                  "field");
  }
  else
  {
    // Удаление изображения
    $delimg = new field_checkbox("delimg",
                            "Удалить изображение",
                            $_REQUEST['delimg']);
    $form = new form(array("name" => $name,
                           "body" => $body,
                           "url" => $url,
                           "urltext" => $urltext,
```

```
                                "date" => $date,
                                "hide" => $hide,
                                "delimg" => $delimg,
                                "filename" => $filename,
                                "id_news" => $id_news,
                                "page" => $page),
                        "Редактировать",
                        "field");
    }
  }
  catch(ExceptionObject $exc)
  {
    require("../utils/exception_object.php");
  }

  // Обработчик HTML-формы
  if(!empty($_POST))
  {
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();
    if(empty($error))
    {
      // Скрытый или открытый каталог
      if($form->fields['hide']->value) $showhide = "show";
      else $showhide = "hide";
      // Удаляем старые файлы, если они имеются
      $url_pict = "";
      $str = $form->fields['delimg']->value;
      if(!empty($str) || !empty($_FILES['filename']['name']))
      {
        $path = str_replace("//","/","../../".$news['urlpict']);
        if(file_exists($path))
        {
          @unlink($path);
        }
        $url_pict = "urlpict = '',";
      }
      if(!empty($_FILES['filename']['name']))
      {
        $url_pict = "urlpict = 'files/news/".
                    $form->fields['filename']->get_filename()."',";
      }
```

```
      // Формируем SQL-запрос на добавление новости
      $query = "UPDATE $tbl_news
                SET name = '{$form->fields['name']->value}',
                    body = '{$form->fields['body']->value}',
               putdate = '{$form->fields['date']->get_mysql_format()}',
                    url = '{$form->fields['url']->value}',
                    urltext = '{$form->fields['urltext']->value}',
                    $url_pict
                    hide = '{$showhide}'
                WHERE id_news=".$form->fields['id_news']->value;
      if(!mysql_query($query))
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка при редактировании
                                 новостного сообщения");
      }
      // Осуществляем переадресацию на главную страницу
      // администрирования
      header("Location: index.php?page={$form->fields[page]->value}");
      exit();
    }
  }
}
catch(ExceptionMySQL $exc)
{
  require("../utils/exception_mysql.php");
}
// Данные переменные определяют название страницы и подсказку.
$title = "Редактирование новости";
$pageinfo='<p class="help"></p>';
// Включаем заголовок страницы
require_once("../utils/top.php");

echo "<p><a href=# onClick='history.back()'>Назад</a></p>";
// Выводим сообщения об ошибках, если они имеются
if(!empty($error))
{
  foreach($error as $err)
  {
    echo "<span style=\"color:red\">$err</span><br>";
  }
}
```

```
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
```

Скрипт редактирования новостного блока помимо элементов, рассмотренных при описании HTML-формы добавления новости, содержит скрытый элемент управления `id_news`. Кроме того, если новостной блок содержит изображение, то в HTML-форму встраивается флажок `delimg`, позволяющий удалить и его. Кроме того, старое изображение удаляется и в том случае, когда производится загрузка нового изображения. В конечном итоге формируется UPDATE-запрос, который позволяет обновить данные в таблице `system_news` (`$tbl_news`).

За удаление новостного сообщения несет ответственность скрипт delnews.php, который точно так же, как и скрипт editnews.php, может принимать два GET-параметра: `id_news` — первичный ключ удаляемой записи и `page` — номер страницы в постраничной навигации (для корректного возврата назад). Содержимое скрипта delnews.php приводится в листинге 11.24.

Листинг 11.24. Удаление новостного сообщения, delnews.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Проверяем параметр id_news, предотвращая SQL-инъекцию
  $_GET['id_news'] = intval($_GET['id_news']);

  try
  {
    // Если новостное сообщение содержит
    // изображение — удаляем его
    $query = "SELECT * FROM $tbl_news
              WHERE id_news=$_GET[id_news]";
    $new = mysql_query($query);
    if(!$new)
    {
```

```
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка удаления
                               новостного блока");
    }
    if(mysql_num_rows($new) > 0)
    {
      $news = mysql_fetch_array($new);
      if(file_exists("../../".$news['urlpict']))
      {
        @unlink("../../".$news['urlpict']);
      }
    }
    // Формируем и выполняем SQL-запрос
    // на удаление новостного блока из базы данных
    $query = "DELETE FROM $tbl_news
              WHERE id_news=$_GET[id_news]
              LIMIT 1";
    if(mysql_query($query))
    {
      header("Location: index.php?page=$_GET[page]");
    }
    else
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка удаления
                               новостного блока");
    }
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
?>
```

Прежде чем удалять запись, соответствующую первичному ключу `$_GET['id_news']` из таблицы `system_news` (`$tbl_news`), скрипт delnews.php проверяет, не содержит ли новостной блок изображения. Если изображение имеется, оно удаляется при помощи функции `unlink()`. Лишь после этого новостной блок удаляется из базы данных при помощи DELETE-запроса.

Если новостной блок является скрытым от просмотра посетителями (поле `hide` принимает значение `hide`), напротив него выводится управляющая ссыл-

ка **Отобразить**, которая указывает на файл show.php. Если новостной блок доступен для просмотра (поле `hide` принимает значение `show`), напротив него выводится управляющая ссылка **Скрыть**, которая указывает на файл hide.php. Оба файла принимают в качестве GET-параметров первичный ключ новостного сообщения `id_news` и номер страницы в постраничной навигации `page` для корректного возврата. В листинге 11.25 приводится содержимое файла hide.php.

Листинг 11.25. Сокрытие новостного блока, hide.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Проверяем параметр id_news, предотвращая SQL-инъекцию
  $_GET['id_news'] = intval($_GET['id_news']);

  // Скрываем новость
  try
  {
    $query = "UPDATE $tbl_news SET hide='hide'
              WHERE id_news=".$_GET['id_news'];
    if(mysql_query($query))
    {
      header("Location: index.php?page=$_GET[page]");
    }
    else
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка при обращении
                               к блоку новостей");
    }
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
?>
```

Файл show.php полностью эквивалентен файлу hide.php за исключением SQL-запроса, меняющего статус новостного блока (листинг 11.26).

Листинг 11.26. Отображение новостного блока, show.php

```
<?php
    ...
    $query = "UPDATE $tbl_news SET hide='show'
              WHERE id_news=$_GET[id_news]";
    ...
?>
```

11.4.3. Система представления

Инструментарий для заполнения базы данных создан, далее следует заняться выводом новостных блоков на страницы сайта. Как правило, для новостного блока подготавливают два скрипта: первый выводит три последние новости в кратком виде и ссылку на полный список новостей, второй скрипт осуществляет вывод списка новостей и просмотр полных версий. На рис. 11.11 приводится возможный внешний вид новостного блока.

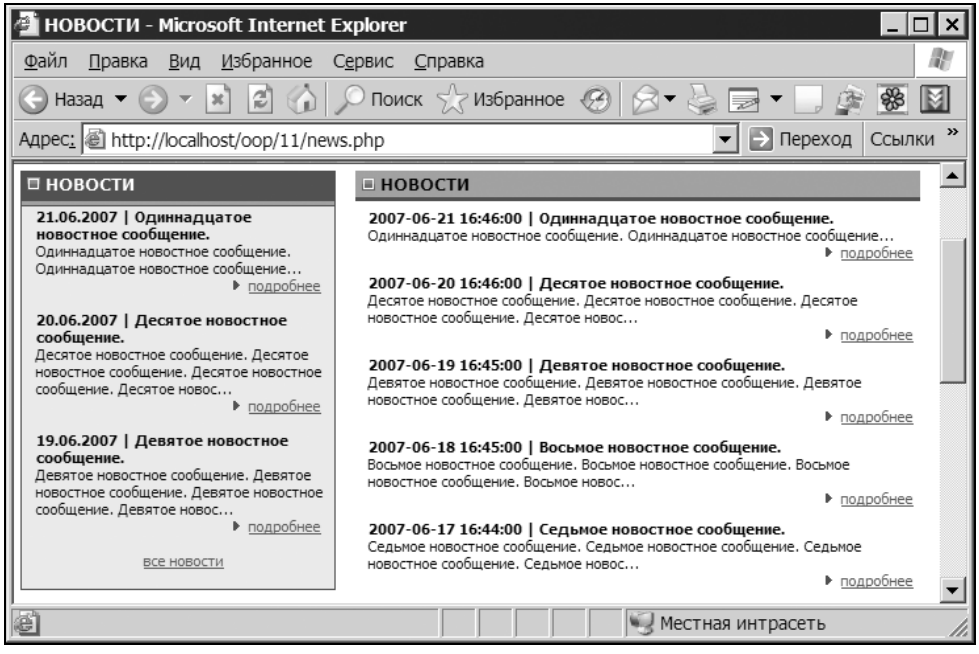


Рис. 11.11. Система представления новостного блока

На рис. 11.11 в левой части страницы выводятся три последние новости, которые отображаются на каждой из страниц сайта; справа отображается текущая информация — в данном случае это полный список новостных блоков. Переход по ссылке **подробнее...** приводит к отображению полной версии новостного блока (рис. 11.12).

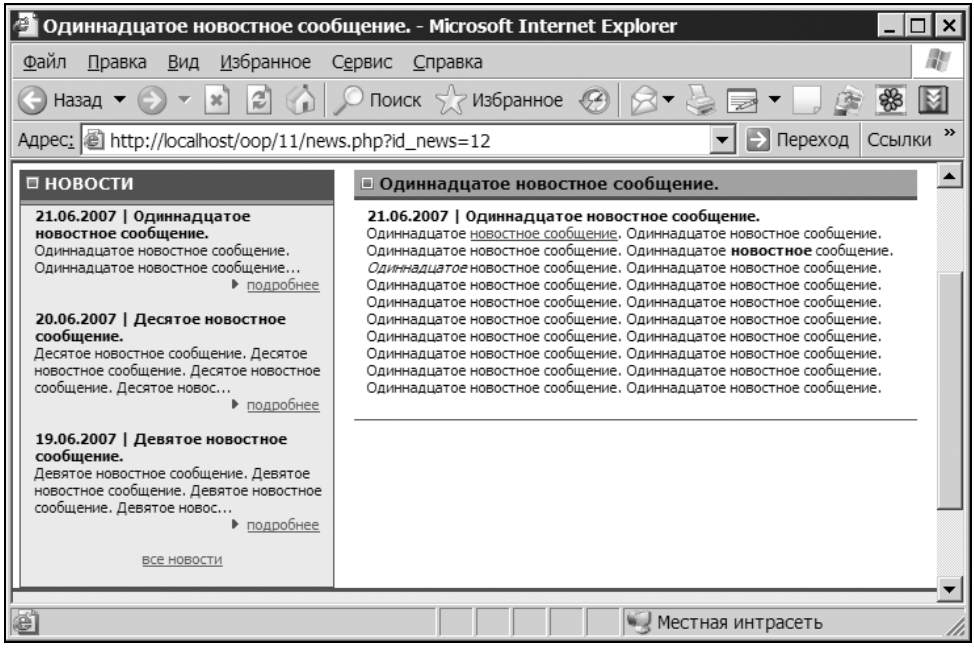


Рис. 11.12. Полный вариант новостного сообщения

В листинге 11.27 приводится скрипт, выводящий три последних новостных сообщения в левой части сайта.

Замечание

Здесь не описывается стилевое оформление страницы представления, включающее, по аналогии со страницами администрирования, шапку top.php и завершение страницы bottom.php. Предполагается, что дизайн будет изменяться для каждого нового сайта, и блок представления служит лишь примером возможной реализации. Полный вариант системы представления можно найти на компакт-диске, поставляемом вместе с книгой.

Листинг 11.27. Вывод последних трех новостных позиций, top.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config/config.php");
```

```
// Подключаем систему классов
require_once("config/class.config.php");
// Подключаем функцию вывода текста с bbCode
require_once("dmn/utils/utils.print_page.php");

$query = "SELECT id_news,
                 name,
                 body,
                 DATE_FORMAT(putdate,'%d.%m.%Y') as putdate_format,
                 url,
                 urltext,
                 urlpict,
                 hide
          FROM $tbl_news
          WHERE hide = 'show'
          ORDER BY putdate DESC
          LIMIT 3";
$new = mysql_query($query);
if (!$new) exit("Ошибка при обращении к блоку новостей");
if(mysql_num_rows($new))
{
  $patt = array("[b]", "[/b]", "[i]", "[/i]");
  $repl = array("", "", "", "");
  $pattern_url = "|\[url[^\]]*\]|";
  $pattern_b_url = "|\[/url[^\]]*\]|";
  while($news_up = mysql_fetch_array($new))
  {
    if(strlen($news_up['body']) > 100)
    {
      $news_up['body'] = substr($news_up['body'], 0, 100)."...";
      $news_up['body'] = str_replace($patt, $repl, $news_up['body']);
      $news_up['body'] = preg_replace($pattern_url,
                                      "", $news_up['body']);
      $news_up['body'] = preg_replace($pattern_b_url,
                                      "", $news_up['body']);
    }

      echo "<b>$news_up[putdate_format] |
          ".print_page($news_up['name'])."</b><br>
          ".print_page($news_up['body'])."
        <div align=\"right\">
          <a href=\"news.php?id_news=$news_up[id_news]\"
             class=\"rightpanel_lnk\">
             подробнее
```

```
            </a>
          </div>
          <br>";
    }
  }
  echo "<div align=\"center\">
          <a href=\"news.php\"
             class=\"rightpanel_lnk\">все новости</a><br>
        </div><br>";
?>
```

Для того чтобы выбрать последние три новостных блока, в SQL-запросе осуществляется обратная сортировка (`ORDER BY ... DESC`) по календарному полю `putdate` при одновременном ограничении количества выбираемых записей с помощью конструкции `LIMIT`.

Текст новости может быть достаточно объемным, поэтому имеет смысл в сокращенном варианте ограничить его сотней символов при помощи функции `substr()`. Так как функция может разрезать текст на части таким образом, что для открывающего тега bbCode не окажется парного закрывающего тега, из краткого варианта новостной позиции удаляются все bbCode-теги. Простые теги, `[i]`, `[/i]`, `[b]`, `[/b]`, удаляются при помощи функции `str_replace()`, в то время как для сложных тегов `[url=]` и `[/url]` необходимо создавать регулярные выражения и использовать функцию `preg_replace()`. Перед выводом в окно браузера текст новостной позиции обрабатывается функцией `print_page()` (см. листинг 11.21) из файла dmn/utils/utils.print_page.php.

В листинге 11.28 приводится содержимое файла news.php, который несет ответственность за формирование списка новостных сообщений и отображение полного варианта новости.

Листинг 11.28. Список новостей, news.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config/config.php");
  // Подключаем SoftTime FrameWork
  require_once("config/class.config.php");

  // Если GET-параметр id_news не передан — выводим
  // список новостных сообщений
  if(empty($_GET['id_news']))
  {
    // Проверяем параметр page, предотвращая SQL-инъекцию
    $_GET['page'] = intval($_GET['page']);
```

```
// Число сообщений на странице
$pnumber = 10;
// Число ссылок в постраничной навигации
$page_link = 3;
// Объявляем объект постраничной навигации
$obj = new pager_mysql($tbl_news,
                       "",
                       "ORDER BY putdate DESC",
                       $pnumber,
                       $page_link);

// Подключаем верхний шаблон
$title = "НОВОСТИ";
$keywords = "новости";
require_once ("templates/top.php");

// Получаем содержимое текущей страницы
$news = $obj->get_page();
// Если имеется хотя бы одна запись — выводим
if(!empty($news))
{
  echo htmlspecialchars($title);

  $patt = array("[b]", "[/b]", "[i]", "[/i]");
  $repl = array("", "", "", "");
  $pattern_url = "|\[url[^\]]*\]|";
  $pattern_b_url = "|\[/url[^\]]*\]|";
  for($i = 0; $i < count($news); $i++)
  {
    if(strlen($news[$i]['body']) > 100)
    {
      $news[$i]['body'] = substr($news[$i]['body'], 0, 100)."...";
      $news[$i]['body'] = str_replace($patt,
                                      $repl, $news[$i]['body']);
      $news[$i]['body'] = preg_replace($pattern_url,
                                       "", $news[$i]['body']);
      $news[$i]['body'] = preg_replace($pattern_b_url,
                                       "", $news[$i]['body']);
    }

    echo "<b>".$news[$i]['putdate']." | ".
          print_page($news[$i]['name'])."</b>
          <br>".print_page($news[$i]['body'])."
```

```
            <a href=\"news.php?id_news=".$news[$i]['id_news']."\" >
               подробнее
             </a>
            <br>";
    }
    // Выводим ссылки на другие страницы
    echo $obj;
  }
}
// Если GET-параметр id_news передан — выводим полную
// версию новостного сообщения
else
{
  // Проверяем, является ли параметр id_news числом
  $_GET['id_news'] = intval($_GET['id_news']);
  // Выводим выбранное новостное сообщение
  $query = "SELECT id_news,
                   name,
                   body,
                   DATE_FORMAT(putdate,'%d.%m.%Y') as putdate_format,
                   url,
                   urltext,
                   urlpict,
                   hide
            FROM $tbl_news
            WHERE hide = 'show' AND
                  id_news = $_GET[id_news]";
  $res = mysql_query($query);
  if(!$res) exit(mysql_error($query));
  $news = mysql_fetch_array($res);

  // Подключаем верхний шаблон
  $title = $news['name'];
  $keywords = "новости";
  require_once ("templates/top.php");

  echo htmlspecialchars($title);

  $url_pict = "";
  if ($news['urlpict'] != '' && $news['urlpict'] != '-')
  {
    $url_pict = "<img src=".print_page($news['urlpict']).">";
  }
```

```
    $news_url = "";
    if (!empty($news['url']))
    {
      if(!preg_match("|^http://|i",$news['url']))
      {
        $news[$i]['url'] = "http://{$news[url]}";
      }
      $news_url = "<br><b>Ссылка:</b>
                   <a href='".print_page($news['url'])."'>".
                             print_page($news['urltext'])."</a>";
      if(empty($news[$i]['urltext']))
      {
        $news_url = "<br><b>Ссылка:</b>
                     <a href='".print_page($news['url']."'>".
                               print_page($news['url'])."</a>";
      }
    }

    echo "<b>".$news['putdate_format']." | ".
          print_page($news['name'])."</b>
          <br>
          $url_pict ".nl2br(print_page($news['body']))."
          <br>$news_url
          <br>";
  }
  // Подключаем нижний шаблон
  require_once ("templates/bottom.php");
?>
```

Для организации постраничной навигации используется класс `pager_mysql` (см. *главу 4*). Если скрипту не передается GET-параметр `id_news` с первичным ключом новостного сообщения, выводится полный список новостей. Переход по ссылке **подробнее...** приводит к тому, что скрипту news.php передается первичный ключ конкретного новостного блока, что приводит к выводу полной версии новостного сообщения.

11.5. Управления статьями и меню

Сердцем любой системы управления содержимым сайта (CMS) является система разделов и статей, позволяющих создавать бесконечно-вложенную структуру сайта, снабжая каждый из уровней статьями. Сами статьи состоят из параграфов, каждый из которых может быть проиллюстрирован одним или

несколькими изображениями. Структура блока традиционно делится на базу данных, систему администрирования и систему представления.

11.5.1. База данных

В отличие от предыдущих блоков, оперировать лишь одной таблицей не удастся. Для полноценного представления результатов потребуется как минимум четыре таблицы:

- `system_menu_catalog` — таблица для хранения разделов, которые могут содержать в своем составе как подразделы, так и статьи (в скриптах имя таблицы хранится в переменной `$tbl_catalog`);
- `system_menu_position` — таблица для хранения статей, которые состоят из параграфов (в скриптах имя таблицы хранится в переменной `$tbl_position`);
- `system_menu_paragraph` — таблица для хранения параграфов, каждый из которых может быть проиллюстрирован несколькими изображениями (в скриптах имя таблицы хранится в переменной `$tbl_paragraph`);
- `system_menu_paragraph_image` — таблица для хранения изображений (в скриптах имя таблицы хранится в переменной `$tbl_paragraph_image`).

Таблица `system_menu_catalog`, предназначенная для хранения разделов сайта, состоит из следующих восьми полей:

- `id_catalog` — первичный ключ таблицы, предназначенный для идентификации раздела и снабженный атрибутом `AUTO_INCREMENT`, позволяющим автоматически генерировать для новых записей уникальный идентификатор;
- `name` — название раздела;
- `description` — описание раздела;
- `keywords` — ключевые слова, помещаемые в META-тег `keywords` и сообщающие поисковым роботам основное содержание страницы;
- `modrewrite` — текстовое поле, предназначенное для формирования URL при помощи модуля Web-сервера Apache mod_rewrite, который позволяет преобразовать GET-параметры вида index.php?year=2006&month=10&day=26 в более компактный и читаемый URL вида /2006/10/26. К сожалению, такой подход требует привязки приложения к корню сайта (что не всегда удобно) и несколько увеличивает нагрузку на Web-сервер Apache;
- `pos` — позиция раздела относительно других разделов: данное поле предназначено для сортировки разделов; разделы можно сортировать по назва-

нию, но это не всегда удобно, т. к. зачастую они должны быть расположены в логическом, а не алфавитном порядке;

- hide — служебное поле типа ENUM, принимающее только два значения: hide, если раздел скрыт и недоступен для просмотра со страниц сайта, и show, если раздел отображается на страницах сайта;
- id_parent — внешний ключ таблицы, содержащий значение id_catalog родительского раздела; для корневого раздела принимает значение 0. Данное поле обеспечивает возможность создания бесконечно вложенной структуры сайта.

В листинге 11.29 приводится оператор CREATE TABLE, создающий таблицу system_menu_catalog, которая в скриптах обозначается при помощи переменной $tbl_catalog.

Листинг 11.29. Таблица разделов system_menu_catalog ($tbl_catalog)

```
CREATE TABLE system_menu_catalog (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  description TEXT NOT NULL,
  keywords TINYTEXT NOT NULL,
  modrewrite TINYTEXT NOT NULL,
  pos INT(11) NOT NULL DEFAULT '0',
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  id_parent INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id_catalog)
);
```

Остановимся подробнее на поле id_parent, которое обеспечивает создание бесконечно вложенной структуры разделов. Если раздел является подразделом, данное поле принимает значение id_catalog родительского каталога. В листинге 11.30 приводится дамп, состоящий из двух корневых каталогов "Материнские платы" и "Жесткие диски", каждый из которых содержит подразделы, соответствующие различным производителям.

Листинг 11.30. Дамп таблицы system_menu_catalog ($tbl_catalog)

```
INSERT INTO system_menu_catalog
VALUES (1, 'Материнские платы', '', '', '', 1, 'show', 0),
       (2, 'Жесткие диски', '', '', '', 2, 'show', 0),
       (3, 'Micro-Star', '', '', '', 1, 'show', 1),
       (4, 'Gigabyte', '', '', '', 2, 'show', 1),
```

```
(5, 'Asustek', '', '', '', 3, 'show', 1),
(6, 'Epox', '', '', '', 4, 'show', 1),
(7, 'Maxtor', '', '', '', 1, 'show', 2),
(8, 'Samsung', '', '', '', 2, 'show', 2),
(9, 'Seagate', '', '', '', 3, 'show', 2);
```

На рис. 11.13 приводится схема соотношения разделов из дампа к листингу 11.30 со значениями полей `id_catalog`, `id_parent` и `pos`. Поле `pos` обеспечивает сортировку подразделов в рамках каждой из групп разделов.

В результате, чтобы получить все подразделы раздела "Материнские платы" (`id_catalog = 1`), достаточно выполнить SQL-запрос вида:

```
SELECT * FROM system_menu_catalog
WHERE id_parent = 1;
```

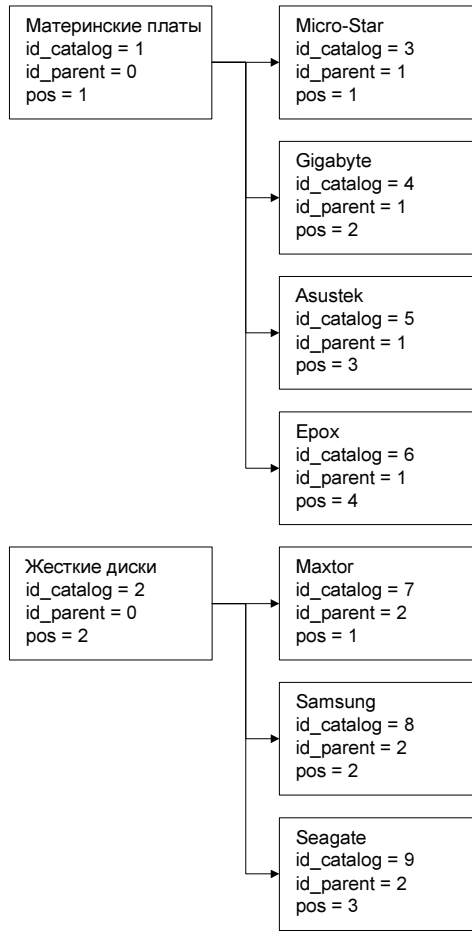


Рис. 11.13. Структура вложенного каталога

Если при этом добавить конструкцию `ORDER BY pos`, то записи будут отсортированы по полю `pos`:

```
SELECT * FROM system_menu_catalog
WHERE id_parent = 1
ORDER BY pos;
```

Впрочем, сортировка по полю `pos` необязательна — иногда удобнее сортировать результаты запроса в алфавитном порядке:

```
SELECT * FROM system_menu_catalog
WHERE id_parent = 1
ORDER BY name;
```

Каждый из разделов, за исключением корневого, может содержать статьи и ссылки на страницы (возможно, даже другого сайта). Для хранения служебной информации об этих элементах предназначена таблица `system_menu_position`, которая содержит восемь полей:

- `id_position` — первичный ключ таблицы, предназначенный для идентификации позиции и снабженный атрибутом `AUTO_INCREMENT`, позволяющим автоматически генерировать для новых записей уникальный идентификатор;
- `name` — название статьи или ссылки;
- `url` — поле предназначено для хранения URL, если текущий элемент является ссылкой; если же он является статьей, данное поле принимает значение `article`;
- `keywords` — ключевые слова, помещаемые в META-тег `keywords` и сообщающие поисковым роботам основное содержание страницы;
- `modrewrite` — текстовое поле, предназначенное для формирования URL при помощи модуля Web-сервера Apache mod_rewrite, который позволяет преобразовать GET-параметры вида index.php?year=2006&month=10&day=26 в более компактный и читаемый URL вида /2006/10/26. К сожалению, такой подход требует привязки приложения к корню сайта (что не всегда удобно) и несколько увеличивает нагрузку на Web-сервер Apache;
- `pos` — поле предназначено для сортировки элементов: чем меньше его значение, тем выше располагается элемент, и наоборот, чем больше значение поля `pos`, тем ниже располагается элемент в общем списке;
- `hide` — служебное поле типа `ENUM`, принимающее только два значения: `hide`, если позиция скрыта и недоступна для просмотра со страниц сайта, и `show`, если позиция отображается на страницах сайта;

- `id_catalog` — внешний ключ для таблицы `system_menu_catalog`, позволяющий привязать статью к конкретному разделу.

Поле `id_catalog` позволяет связать таблицы `system_menu_catalog` и `system_menu_position` связью "один-ко-многим" (рис. 11.14).

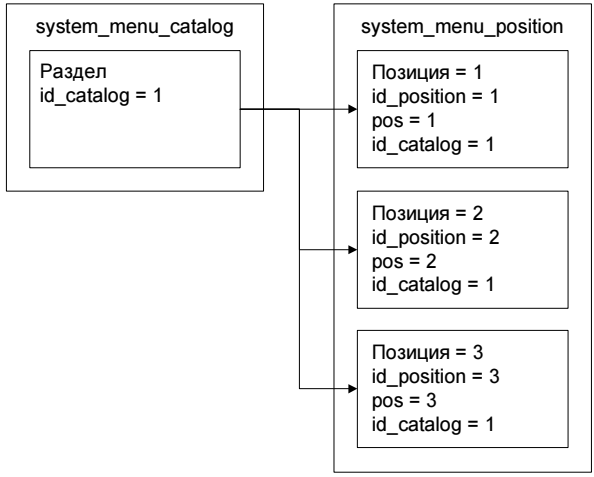


Рис. 11.14. Связь "один-ко-многим" таблиц `system_menu_catalog` и `system_menu_position` осуществляется при помощи внешнего ключа `id_catalog`

В листинге 11.31 приводится оператор `CREATE TABLE`, создающий таблицу `system_menu_position`, которая в скриптах обозначается при помощи переменной `$tbl_position`.

Замечание

Предложенная структура базы данных позволяет хранить одну статью только в одном разделе; невозможно включить одну и ту же статью одновременно в несколько разделов. Для такой организации данных необходимо предусмотреть дополнительную таблицу, каждая из записей которой сопоставляла бы первичный ключ элемента `id_position` с разделом `id_catalog`, осуществляя тем самым связь "многие-ко-многим". Количество записей для каждого из элементов `id_position` определяло бы, в скольких разделах располагается элемент, а количество записей для раздела `id_catalog` определяло бы количество элементов в разделе. Здесь такая организация структуры базы данных не рассматривается, т. к. ссылки позволяют сослаться на любую страницу, в том числе и на статью из другого раздела сайта.

Листинг 11.31. Таблица позиций `system_menu_position` (`$tbl_position`)

```
CREATE TABLE system_menu_position (
  id_position INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
```

```
  url TEXT NOT NULL,
  keywords TINYTEXT NOT NULL,
  modrewrite TINYTEXT NOT NULL,
  pos int(11) NOT NULL DEFAULT '0',
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  id_catalog INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id_position)
);
```

Если позиция представляет собой ссылку, то рассмотренных выше таблиц `system_menu_catalog` и `system_menu_position` достаточно для хранения данных. Однако если таблица является статьей, необходимо предусмотреть еще одну таблицу `system_menu_paragraph`, которая бы хранила параграфы, принадлежащие текущей статье. Таблица `system_menu_paragraph` содержит восемь полей:

- `id_paragraph` — первичный ключ таблицы, предназначенный для идентификации параграфа и снабженный атрибутом `AUTO_INCREMENT`, позволяющим автоматически генерировать уникальный идентификатор для новых записей;
- `name` — содержимое параграфа;
- `type` — поле типа `ENUM`, которое может принимать одно из семи значений: `text` — для обычного текстового параграфа, и шесть (от `title_h1` до `title_h6`) — для HTML-заголовков от `<H1>` до `<H6>`;
- `align` — поле типа `ENUM`, определяющее выравнивание параграфа; может принимать одно из трех значений: `left` — выравнивание по левому краю, `center` — выравнивание по центру и `right` — выравнивание по правому краю;
- `hide` — служебное поле типа `ENUM`, принимающее только одно из двух значений: `hide`, если параграф скрыт и недоступен для просмотра со страниц сайта, и `show`, если параграф отображается на страницах сайта;
- `pos` — позиция параграфа относительно других параграфов; данное поле предназначено для их сортировки;
- `id_position` — внешний ключ для таблицы `system_menu_position`, позволяющий привязать параграф к конкретной статье;
- `id_catalog` — внешний ключ для таблицы `system_menu_catalog`, указывающий, к какому разделу принадлежит статья и текущий параграф.

Следует отметить, что таблица `system_menu_paragraph` связана с двумя таблицами `system_menu_position` и `system_menu_catalog`. На первый взгляд может показаться, что связь с таблицей `system_menu_catalog` является излишней, т. к. всегда можно восстановить принадлежность параграфа тому или иному раз-

делу через таблицу позиций `system_menu_position` без расширения таблицы на дополнительное поле, содержащее внешний ключ `id_catalog`. Однако практика показывает, что предложенный подход является более рациональным, т. к. позволяет удалять разделы без перебора всех элементов. Так, для того чтобы удалить элементы и параграфы раздела, достаточно выполнить всего три DELETE-запроса:

```
DELETE FROM system_menu_paragraph WHERE id_catalog = 1;
DELETE FROM system_menu_position WHERE id_catalog = 1;
DELETE FROM system_menu_catalog WHERE id_catalog = 1;
```

Если бы в таблице `system_menu_paragraph` отсутствовал внешний ключ `id_catalog`, пришлось бы выполнять SELECT-запрос к таблице `system_menu_position`:

```
SELECT * FROM system_menu_position WHERE id_catalog = 1;
```

Далее в цикле необходимо было бы выполнять отдельный DELETE-запрос на удаление каждого из параграфов.

Замечание

Тип таблиц InnoDB позволяет осуществлять каскадное удаление данных из связанных таблиц, однако таблицы InnoDB гораздо медленнее (иногда в разы), чем традиционные MyISAM-таблицы. Поэтому многие Web-разработчики ориентируются на менее функциональные, но более быстрые MyISAM-таблицы.

В листинге 11.32 приводится оператор `CREATE TABLE`, создающий таблицу `system_menu_paragraph`, которая в скриптах обозначается при помощи переменной `$tbl_paragraph`.

Замечание

Поле `type` в листинге 11.32 заключено в обратные кавычки. Это связано с тем, что слово `type` является зарезервированным, и чтобы MySQL-сервер мог корректно разобрать SQL-запрос, необходимо при помощи обратных кавычек указать на то, что это название столбца, а не ключевое слово.

Листинг 11.32. Таблица параграфов `system_menu_paragraph` (`$tbl_paragraph`)

```
CREATE TABLE system_menu_paragraph (
  id_paragraph INT(11) NOT NULL AUTO_INCREMENT,
  name TEXT NOT NULL,
  `type` ENUM('text', 'title_h1', 'title_h2', 'title_h3', 'title_h4',
              'title_h5', 'title_h6') NOT NULL DEFAULT 'text',
  align ENUM('left','center','right') NOT NULL DEFAULT 'left',
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  pos INT(11) NOT NULL,
```

```
  id_position INT(11) NOT NULL,
  id_catalog INT(11) NOT NULL,
  PRIMARY KEY (id_paragraph)
);
```

Для формирования полноценных статей необходимо предусмотреть добавление изображений, причем каждый из параграфов может содержать произвольное количество изображений (формируя своеобразную фотогалерею). Хранить фотографии в базе данных MySQL не рационально, т. к. скорость доступа к ним окажется гораздо ниже, чем если бы они хранились на жестком диске. MySQL, как и любая другая СУБД, с большой скоростью оперирует короткими текстовыми строками, однако при возрастании объема данных и таблиц скорость обращения резко падает. Принимая это во внимание, изображения будут храниться в специальной директории, а в таблицу базы данных будет помещаться лишь путь к ним. Для хранения путей к изображениям разработаем таблицу `system_menu_paragraph_image`, которая содержит десять полей:

- `id_image` — первичный ключ таблицы, предназначенный для идентификации изображения и снабженный атрибутом `AUTO_INCREMENT`, позволяющим автоматически генерировать уникальный идентификатор для новых записей;
- `name` — название изображения, выводимое в качестве подписи под изображением;
- `alt` — содержимое ALT-тега, которое выводится у посетителя, если в его браузере отключено отображение изображений;
- `small` — путь к уменьшенному варианту изображения;
- `big` — путь к большому изображению, которое выводится при щелчке мышью по его уменьшенному варианту;
- `hide` — служебное поле типа `ENUM`, принимающее только два значения: `hide`, если изображение скрыто и недоступно для просмотра со страниц сайта, и `show`, если изображение отображается на страницах сайта;
- `pos` — позиция изображения относительно других изображений; данное поле предназначено для их сортировки;
- `id_position` — внешний ключ для таблицы `system_menu_position`, указывающий, к какой статье принадлежит параграф, а следовательно, и изображение;
- `id_catalog` — внешний ключ для таблицы `system_menu_catalog`, указывающий, к какому разделу принадлежит статья, параграф, а следовательно, и изображение;

- `id_paragraph` — внешний ключ для таблицы `system_menu_paragraph`, позволяющий привязать изображение к конкретному параграфу.

В листинге 11.33 приводится оператор `CREATE TABLE`, создающий таблицу `system_menu_paragraph_image`, которая в скриптах обозначается при помощи переменной `$tbl_paragraph_image`.

Листинг 11.33. Таблица изображений `system_menu_paragraph_image` (`$tbl_paragraph_image`)

```
CREATE TABLE system_menu_paragraph_image (
  id_image INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  alt TINYTEXT NOT NULL,
  small TINYTEXT NOT NULL,
  big TINYTEXT NOT NULL,
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  pos INT(11) NOT NULL DEFAULT '0',
  id_position INT(11) NOT NULL DEFAULT '0',
  id_catalog INT(11) NOT NULL DEFAULT '0',
  id_paragraph INT(11) NOT NULL,
  PRIMARY KEY (id_image)
);
```

11.5.2. Система администрирования

Структура системы администрирования напоминает структуру системы администрирования блока новостей (см. *раздел 11.4.2*). Для каждого из объектов, фигурирующих в базе данных (раздел, позиция, параграф и изображение) предусматривается страница со списком объектов одного из типов. На данной странице располагаются управляющие ссылки, позволяющие добавлять, удалять, редактировать и менять порядок следования объектов.

Ниже приводится полный список файлов системы администрирования системы управления содержимым сайта (CMS):

- artadd.php — HTML-форма, позволяющая добавлять статьи;
- artedit.php — HTML-форма, позволяющая редактировать статьи;
- catadd.php — HTML-форма, позволяющая добавлять разделы;
- catdel.php — скрипт, позволяющий удалять разделы;
- catdown.php — скрипт, позволяющий опускать раздел на одну позицию вниз относительно остальных разделов;

- catedit.php — HTML-форма, позволяющая редактировать разделы;
- cathide.php — скрипт, позволяющий скрывать разделы;
- catshow.php — скрипт, позволяющий отображать разделы;
- catup.php — скрипт, позволяющий поднимать раздел на одну позицию вверх относительно остальных разделов;
- image.php — страница, выводящая список изображений текущего параграфа;
- imgadd.php — HTML-форма, позволяющая добавлять изображения;
- imgdel.php — скрипт, позволяющий удалять изображения;
- imgdown.php — скрипт, позволяющий опускать изображение на одну позицию вниз относительно остальных изображений;
- imghide.php — скрипт, позволяющий скрывать изображение;
- imgshow.php — скрипт, позволяющий отображать изображение;
- imgup.php — скрипт, позволяющий поднимать изображение на одну позицию вверх относительно остальных изображений;
- index.php — страница, выводящая список подразделов текущего раздела;
- paradd.php — HTML-форма, позволяющая добавлять параграфы;
- paragraph.php — страница, выводящая список параграфов текущей статьи;
- pardel.php — скрипт, позволяющий удалять параграфы;
- pardown.php — скрипт, позволяющий опускать параграф на одну позицию вниз относительно остальных параграфов;
- paredit.php — HTML-форма, позволяющая редактировать параграфы;
- parhide.php — скрипт, позволяющий скрывать параграф;
- parshow.php — скрипт, позволяющий отображать параграфы;
- parup.php — скрипт, позволяющий поднимать параграф на одну позицию вверх относительно остальных параграфов;
- position.php — страница, выводящая список позиций текущего раздела;
- show.php — страница просмотра увеличенного изображения;
- urladd.php — HTML-форма, позволяющая добавлять ссылки;
- urldel.php — скрипт, позволяющий удалять элементы (ссылки и статьи);
- urldown.php — скрипт, позволяющий опустить элемент (ссылку или статью) на одну позицию вниз относительно остальных элементов (ссылок или статей);

- urledit.php — HTML-форма, позволяющая редактировать ссылки;
- urlhide.php — скрипт, позволяющий скрывать элемент (ссылку или статью);
- urlshow.php — скрипт, позволяющий отображать элемент (ссылку или статью);
- urlup.php — скрипт, позволяющий поднимать элемент (ссылку или статью) на одну позицию вверх относительно остальных элементов (ссылок или статей).

Разработка иерархических систем, включающих несколько связанных таблиц, осуществляется, как правило, сверху вниз. Сначала разрабатываются скрипты и HTML-формы для разделов, потом для элементов (ссылок и статей), затем для параграфов и, наконец, для изображений. Такой порядок разработки диктуется структурой базы данных и необходимостью разработанных инструментов для добавления, редактирования и просмотра более высокого уровня, прежде чем приступать к разработке подчиненного.

Замечание

Создание скриптов, удаляющих разделы, статьи, параграфы и изображения, часто откладывается до тех пор, пока не будет построена вся инфраструктура для добавления и удаления этих объектов. Удаление разделов должно приводить к удалению всех подчиненных статей их параграфов и изображений, удаление статьи должно сопровождаться удалением параграфов и изображений, а удаление параграфа не должно оставлять в системе не принадлежащих никому изображений. Отлаживать такие скрипты удобнее и надежнее, когда имеются инструменты для построения всех уровней системы.

При обращении пользователя к блоку управления содержимым сайта первой страницей, куда он попадает, является страница index.php, отображающая список разделов (рис. 11.15).

Как видно из рис. 11.15, на главной странице выводится таблица, каждая из строк которой отводится для одного из разделов. Столбцы таблицы включают название раздела, его описание, позицию и управляющие ссылки. Название раздела представляет собой ссылку, переход по которой приводит к отображению подразделов и элементов текущего раздела. На рис. 11.16 приводится страница с подразделами раздела "Материнские платы"; важно отметить, что в навигационной строке перед таблицей разделов появляется дополнительный пункт, позволяющий вернуться на один уровень выше. По мере перехода на более низкие подразделы навигационная строка удлиняется, указывая пользователю, в какой части иерархии он находится в настоящий момент.

Содержимое файла index.php представлено в листинге 11.34.

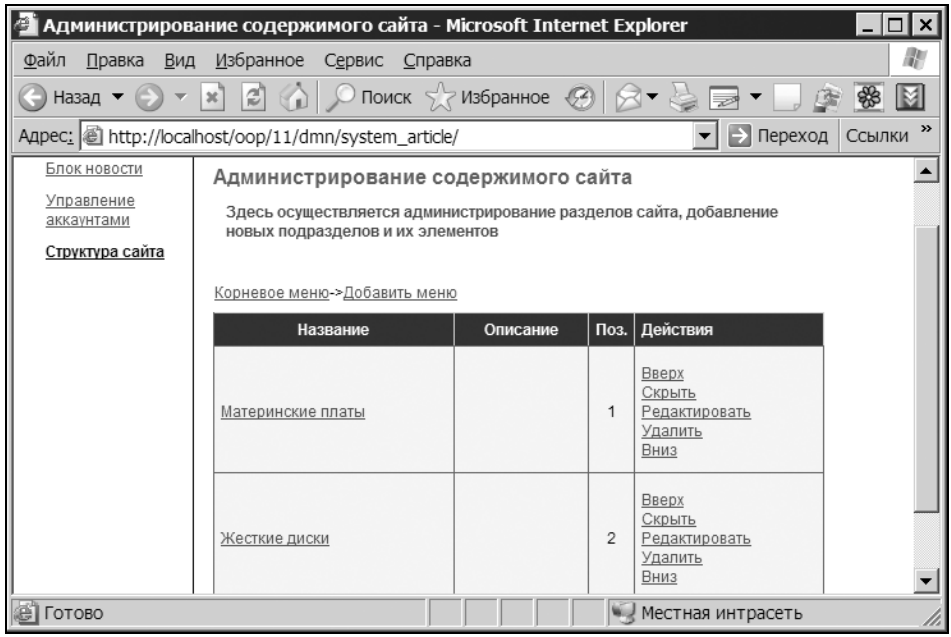


Рис. 11.15. Список разделов системы администрирования содержимого сайта (CMS)

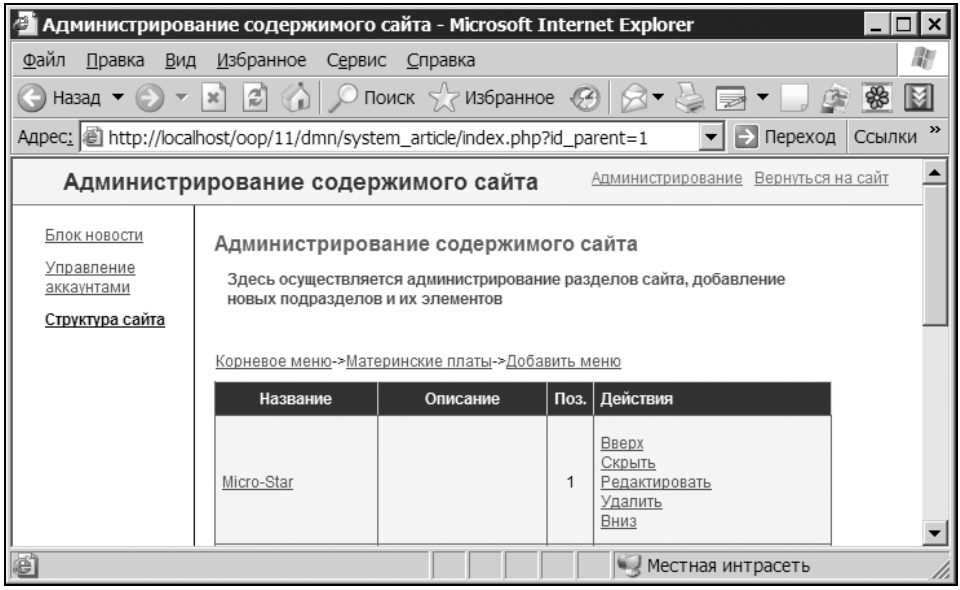


Рис. 11.16. Подразделы раздела "Материнские платы"

Листинг 11.34. Список разделов и подразделов, index.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");
  // Навигационное меню
  require_once("../utils/utils.navigation.php");
  // Подключаем блок отображения текста в окне браузера
  require_once("../utils/utils.print_page.php");

  $title = $titlepage = 'Администрирование меню сайта';
  $pageinfo = '<p class=help>Здесь осуществляется администрирование
                             разделов сайта, добавление новых подразделов
                             и их элементов</p>';

  // Включаем заголовок страницы
  require_once("../utils/top.php");

  $_GET['id_parent'] = intval($_GET['id_parent']);

  // Если это не корневой каталог, выводим ссылки
  // для возврата и для добавления подкаталога
  echo '<table cellspacing="0" cellspacing="0" border=0>
          <tr valign="top"><td height="25"><p>';
  echo "<a class=menu href=index.php?id_parent=0>Корневое меню</a>-&gt;".
          menu_navigation($_GET['id_parent'], "", $tbl_catalog).
       "<a class=menu
href=catadd.php?id_catalog=$_GET[id_parent]&id_parent=$_GET[id_parent]>
          Добавить меню</a>";
  echo "</td></tr></table>";

  // Выводим список каталогов
  $query = "SELECT * FROM $tbl_catalog
            WHERE id_parent=$_GET[id_parent]
            ORDER BY pos";

  $ctg = mysql_query($query);
  if(!$ctg) exit("Ошибка при обращении к каталогу");
  if(mysql_num_rows($ctg)>0)
```

```
{
  // Выводим заголовок таблицы каталогов
  echo '<table width="100%"
              class="table"
              border="0"
              cellpadding="0"
              cellspacing="0">
          <tr class="header" align="center">
            <td align=center>Название</td>
            <td align=center>Описание</td>
            <td width=20 align=center>Поз.</td>
            <td width=50 align=center>Действия</td>
          </tr>';
  while($catalog = mysql_fetch_array($ctg))
  {
    $url = "id_catalog=$catalog[id_catalog]&".
           "id_parent=$catalog[id_parent]";
    // Выясняем, скрыт каталог или нет
    if($catalog['hide'] == 'hide')
    {
      $strhide = "<a href=catshow.php?$url>Отобразить</a>";
      $style=" class=hiddenrow ";
    }
    else
    {
      $strhide = "<a href=cathide.php?$url>Скрыть</a>";
      $style="";
    }

    // Выводим список каталогов
    echo "<tr $style >
          <td><p><a href=index.php?id_parent=$catalog[id_catalog]>
                    $catalog[name]
                </a></td>
          <td><p>".nl2br( print_page(
                              $catalog['description'])). </td>
          <td><p align=center>$catalog[pos]</td>
          <td><p>
          <a href=catup.php?$url>Вверх</a><br>
          $strhide<br>
          <a href=catedit.php?$url>Редактировать</a><br>
          <a href=#
             onClick=\"delete_catalog('catdel.php?$url');\">
```

```
                Удалить</a><br>
            <a href=catdown.php?$url>Вниз</a><br></td>
          </tr>";
    }
    echo "</table>";
  }

  // Выводим элементы текущего раздела
  if(isset($_GET['id_parent']) && $_GET['id_parent'] != 0)
  {
    // Выводим элементы текущего каталога
    include "position.php";
  }

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
<script language='JavaScript1.1' type='text/javascript'>
<!--
  function delete_catalog(url)
  {
    if(confirm("Вы действительно хотите удалить раздел?"))
    {
      location.href=url;
    }
    return false;
  }
//-->
</script>
```

Для формирования навигационной строки, информирующей пользователя о том, где он находится, и позволяющей вернуться в вышерасположенные разделы, применяется рекурсивная функция `menu_navigation()` из файла utils/utils.navigation.php (листинг 11.35).

Листинг 11.35. Функция `menu_navigation()`

```
<?php
  function menu_navigation($id_catalog, $link, $catalog)
  {
    $id_catalog = intval($id_catalog);
    $query = "SELECT * FROM $catalog
              WHERE id_catalog = $id_catalog";
```

```
    $cat = mysql_query($query);
    if(!$cat)
    {
       exit("Ошибка обращения к таблице каталога menu_navigation()");
    }
    if(mysql_num_rows($cat) > 0)
    {
      $catalog_result = mysql_fetch_array($cat);
      $link = "<a class=menu
              href=index.php?id_parent=".$catalog_result['id_catalog']."> 
              ".$catalog_result['name']."</a>-&gt;".$link;
      $link = menu_navigation($catalog_result['id_parent'],
                              $link,
                              $catalog);
    }
    return $link;
  }
?>
```

Функция `menu_navigation()` принимает три аргумента: `$id_catalog` — первичный ключ самого глубокого каталога, `$link` — начальная строка навигационной строки и `$catalog` — название таблицы. Функция вызывает сама себя до тех пор, пока не будет достигнут корневой каталог (`$id_catalog = 0`), а переменная `$link` не получит навигационную строку. Последняя ссылка в навигационной строке является ссылкой на файл catadd.php для добавления нового подраздела в текущий раздел.

Файл index.php принимает в качестве GET-параметра `$_GET['id_parent']` значение первичного ключа текущего раздела. Если текущий раздел не является корневым (`$_GET['id_parent']` отлично от нулевого значения), в скрипт включается файл position.php, выводящий список элементов текущего раздела.

Для добавления нового подраздела предназначен файл catadd.php, содержимое которого представлено в листинге 11.36.

Листинг 11.36. Добавление подраздела, catadd.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем классы формы
  require_once("../../config/class.config.dmn.php");
```

```
// Параметры
$button_name = "Добавить";
$class_name = "field";

if(empty($_POST)) $_REQUEST['hide'] = true;
$name = new field_text("name",
                       "Название",
                       true,
                       $_POST['name']);
$description = new field_textarea("description",
                            "Описание",
                            false,
                            $_POST['description']);
$keywords = new field_text("keywords",
                           "Ключевые слова",
                           false,
                           $_POST['keywords']);
$modrewrite = new field_text_english("modrewrite",
                           "Название для<br>ReWrite",
                           false,
                           $_POST['modrewrite']);
$hide = new field_checkbox("hide",
                           "Отображать",
                           $_REQUEST['hide']);
$id_parent = new field_hidden_int("id_parent",
                            true,
                            $_REQUEST['id_parent']);
$page = new field_hidden_int("page",
                            false,
                            $_REQUEST['page']);
try
{
  // Форма
  $form = new form(array("name" => $name,
                         "description" => $description,
                         "keywords" => $keywords,
                         "modrewrite" => $modrewrite,
                         "hide" => $hide,
                         "modrewrite" => $modrewrite,
                         "id_parent" => $id_parent,
                         "page" => $page),
                         $button_name,
                         $class_name);
}
```

```
catch(ExceptionObject $exc)
{
  require("../utils/exception_object.php");
}

// Обработчик HTML-формы
if(!empty($_POST))
{
  try
  {
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();
    if(empty($error))
    {
      //$form->fields['id_catalog']->value += 0;
      // Извлекаем максимальную из существующих позиций
      $query = "SELECT MAX(pos) FROM $tbl_catalog
                WHERE id_parent = {$form->fields[id_parent]->value}";
      $pos = mysql_query($query);
      if(!$pos)
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка при извлечении
                                 текущей позиции");
      }
      $position = mysql_result($pos, 0) + 1;
      // Скрытый или открытый каталог
      if($form->fields['hide']->value) $showhide = "show";
      else $showhide = "hide";

      // Формируем SQL-запрос на добавление каталога
      $query = "INSERT INTO $tbl_catalog
                VALUES (NULL,
                        '{$form->fields[name]->value}',
                        '{$form->fields[description]->value}',
                        '{$form->fields[keywords]->value}',
                        '{$form->fields[modrewrite]->value}',
                         $position,
                        '$showhide',
                        {$form->fields[id_parent]->value})";
```

```
        if(!mysql_query($query))
        {
          throw new ExceptionMySQL(mysql_error(),
                                   $query,
                                  "Ошибка при добавлении нового
                                   каталога");
        }
        // Осуществляем перенаправление
        // на главную страницу администрирования
        header("Location: index.php?".
               "id_parent={$form->fields[id_parent]->value}&".
               "page={$form->fields[page]->value}");
        exit();
      }
    }
    catch(ExceptionMySQL $exc)
    {
      require("../utils/exception_mysql.php");
    }
  }
  // Начало страницы
  $title     = 'Добавление подраздела';
  $pageinfo  = '<p class=help></p>';
  // Включаем заголовок страницы
  require_once("../utils/top.php");

  echo "<p><a href=# onClick='history.back()'>Назад</a></p>";
  // Выводим сообщения об ошибках, если они имеются
  if(!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
```

HTML-форма для создания нового подраздела состоит из трех текстовых полей для названия каталога `name`, ключевых слов `keywords`, для URL `modrewrite`, текстовой области для описания каталога `description`, флажка `hide`, позволяющего назначить каталогу статус (отображаемый или скрытый), и двух скрытых полей: `id_parent` и `page`, предназначенных для одноименных GET-параметров. Внешний вид HTML-формы представлен на рис. 11.17.

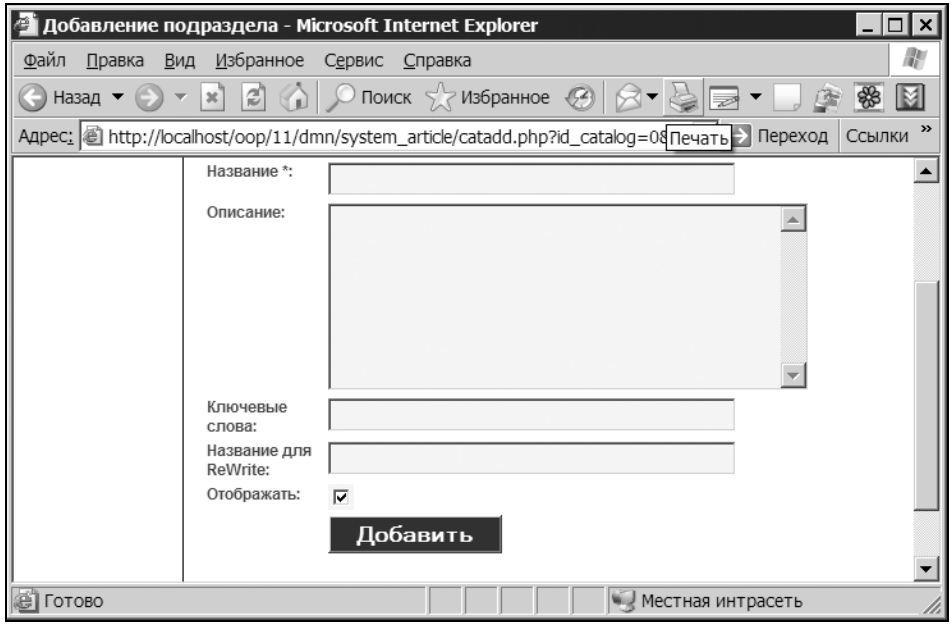


Рис. 11.17. HTML-форма для добавления нового подраздела

Перед добавлением новой записи в таблицу `system_menu_catalog` (`$tbl_catalog`) вычисляется позиция нового раздела (он помещается в конец); для этого при помощи SELECT-запроса с участием MySQL-функции `MAX()` извлекается максимальное значение столбца `pos`.

Редактирование раздела catedit.php внешне повторяет HTML-форму для добавления раздела catadd.php, различие заключается в том, что в поля HTML-формы вставляются значения редактируемой записи. Для того чтобы указать первичный ключ редактируемого раздела, помимо GET-параметров `id_parent` и `page` скрипту передается дополнительный параметр `id_catalog`, который помещается в одноименное скрытое поле. В обработчике HTML-формы для редактирования разделов применяется SQL-оператор `UPDATE`, обновляющий запись таблицы вместо `INSERT`, используемого для добавления новой записи. В листинге 11.37 приводится содержимое файла catedit.php.

Листинг 11.37. Редактирование подраздела, catedit.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем классы формы
  require_once("../../config/class.config.dmn.php");

  // Параметры
  $button_name = "Редактировать";
  $class_name = "field";

  $_GET['id_catalog'] = intval($_GET['id_catalog']);
  if(empty($_POST))
  {
    try
    {
      $query = "SELECT * FROM $tbl_catalog
                WHERE id_catalog=$_GET[id_catalog]
                LIMIT 1";
      $cat = mysql_query($query);
      if(!$cat)
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка при обращении
                                 к каталогу");
      }
      $_REQUEST = mysql_fetch_array($cat);
      if($_REQUEST['hide'] == 'show') $_REQUEST['hide'] = true;
      else $_REQUEST['hide'] = false;
    }
    catch(ExceptionMySQL $exc)
    {
      require("../utils/exception_mysql.php");
    }
  }

  $name = new field_text("name",
                         "Название",
                         true,
                         $_REQUEST['name']);
```

```
$description = new field_textarea("description",
                                  "Описание",
                                  false,
                                  $_REQUEST['description']);
$keywords = new field_text("keywords",
                           "Ключевые слова",
                           false,
                           $_REQUEST['keywords']);
$modrewrite = new field_text_english("modrewrite",
                           "Название для<br>ReWrite",
                           false,
                           $_REQUEST['modrewrite']);
$hide = new field_checkbox("hide",
                           "Отображать",
                           $_REQUEST['hide']);
$id_catalog = new field_hidden_int("id_catalog",
                                  true,
                                  $_REQUEST['id_catalog']);
$id_parent = new field_hidden_int("id_parent",
                                  true,
                                  $_REQUEST['id_parent']);
$page = new field_hidden_int("page",
                                  false,
                                  $_REQUEST['page']);
try
{
  $form = new form(array("name" => $name,
                         "description" => $description,
                         "keywords" => $keywords,
                         "modrewrite" => $modrewrite,
                         "hide" => $hide,
                         "modrewrite" => $modrewrite,
                         "id_catalog" => $id_catalog,
                         "id_parent" => $id_parent,
                         "page" => $page),
                   $button_name,
                   $class_name);
}
catch(ExceptionObject $exc)
{
  require("../utils/exception_object.php");
}
```

```
// Обработчик HTML-формы
if(!empty($_POST))
{
  try
  {
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();
    if(empty($error))
    {
      // Проверяем, скрыт или открыт каталог
      if($form->fields['hide']->value) $showhide = "show";
      else $showhide = "hide";
      // Формируем SQL-запрос на добавление каталога
      $query = "UPDATE $tbl_catalog
                SET name        = '{$form->fields[name]->value}',
                   description = '{$form->fields[description]->value}',
                    keywords    = '{$form->fields[keywords]->value}',
                    modrewrite  = '{$form->fields[modrewrite]->value}',
                    hide        = '$showhide'
                WHERE id_catalog = {$form->fields[id_catalog]->value}";
      if(!mysql_query($query))
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка при редактировании
                                 подкаталога");
      }
      // Осуществляем перенаправление
      // на главную страницу администрирования
      header("Location: index.php?".
             "id_parent={$form->fields[id_parent]->value}&".
             "page={$form->fields[page]->value}");
      exit();
    }
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
}
// Начало страницы
$title     = 'Редактирование подменю';
```

```
  $pageinfo  = '<p class=help></p>';
  // Включаем заголовок страницы
  require_once("../utils/top.php");

  echo "<p><a href=# onClick='history.back()'>Назад</a></p>";
  // Выводим сообщения об ошибках, если они имеются
  if(!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
```

Для удаления подраздела используется скрипт catdel.php (листинг 11.38), который принимает два GET-параметра: `id_catalog` — первичный ключ удаляемого раздела и `page` — номер страницы в системе постраничной навигации (для отображения подразделов постраничная навигация не используется, но она применяется для отображения позиций).

Листинг 11.38. Удаление подраздела, catdel.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Защита от SQL-инъекции
  $_GET['id_catalog'] = intval($_GET['id_catalog']);

  try
  {
    // Извлекаем все изображения, принадлежащие каталогу, и удаляем их
    $query = "SELECT * FROM $tbl_paragraph_image
              WHERE id_catalog=$_GET[id_catalog]";
```

```
$img = mysql_query($query);
if(!$img)
{
  throw new ExceptionMySQL(mysql_error(),
                           $query,
                          "Ошибка при извлечении
                           параметров изображения");
}
if(mysql_num_rows($img))
{
  while($image = mysql_fetch_array($img))
  {
    if(file_exists("../../".$image['big']))
      @unlink("../../".$image['big']);
    if(file_exists("../../".$image['small']))
      @unlink("../../".$image['small']);
  }
}

// Формируем и выполняем SQL-запрос на удаление изображений
$query = "DELETE FROM $tbl_paragraph_image
          WHERE id_catalog=$_GET[id_catalog]";
if(!mysql_query($query))
{
  throw new ExceptionMySQL(mysql_error(),
                           $query,
                          "Ошибка при удалении элемента");
}
// Формируем и выполняем SQL-запрос на удаление параграфов
$query = "DELETE FROM $tbl_paragraph
          WHERE id_catalog=$_GET[id_catalog]";
if(!mysql_query($query))
{
  throw new ExceptionMySQL(mysql_error(),
                           $query,
                          "Ошибка при удалении элемента");
}
// Формируем и выполняем SQL-запрос на удаление элемента каталога
$query = "DELETE FROM $tbl_position
          WHERE id_catalog=$_GET[id_catalog]";
if(!mysql_query($query))
{
```

```
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка при удалении элемента");
    }
    // Формируем и выполняем SQL-запрос на удаление каталога
    $query = "DELETE FROM $tbl_catalog
              WHERE id_catalog=$_GET[id_catalog]
              LIMIT 1";
    if(!mysql_query($query))
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка при удалении каталога");
    }

    header("Location: index.php?id_parent=$_GET[id_parent]".
           "&page=$_GET[page]");
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
?>
```

Так как раздел является самой общей сущностью в системе администрирования и может включать множество элементов, параграфов и изображений, приходится последовательно, при помощи SQL-оператора `DELETE`, очищать все таблицы. Поскольку файлы хранятся отдельно в специальной директории — они удаляются первыми, для этого выполняется SELECT-запрос к таблице `system_menu_paragraph_image` (`$tbl_paragraph_image`). Запрос извлекает все изображения, принадлежащие текущему разделу, и уничтожает их при помощи функции `unlink()`.

Сокрытие и отображение раздела сводится к изменению статуса поля `hide` таблицы `system_menu_catalog` (`$tbl_catalog`) на значение `hide` и `show` соответственно. За сокрытие раздела несет ответственность файл cathide.php, а за отображение — файл catshow.php. Оба скрипта, как и catdel.php, принимают три GET-параметра:

- `id_parent` — первичный ключ текущего раздела, необходимый для корректного возврата на начальную страницу;
- `id_catalog` — первичный ключ раздела, который подвергается сокрытию или отображению;

- `page` — номер страницы в системе постраничной навигации, необходимый для корректного возврата на начальную страницу.

В листинге 11.39 приводится содержимое файла cathide.php, осуществляющего сокрытие раздела.

Замечание

Содержимое файла catshow.php, отображающего раздел, полностью эквивалентно файлу cathide.php за исключением SQL-запроса, который, вместо того чтобы исправлять значение поля `hide` на `hide`, присваивает ему значение `show`.

Листинг 11.39. Сокрытие раздела, cathide.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Защита от SQL-инъекции
  $_GET['id_catalog'] = intval($_GET['id_catalog']);

  try
  {
    // Формируем и выполняем SQL-запрос на сокрытие каталога
    $query = "UPDATE $tbl_catalog SET hide='hide'
              WHERE id_catalog=".$_GET['id_catalog'];
    if(mysql_query($query))
    {
      header("Location: index.php?index.php?".
             "id_parent=$_GET[id_parent]&page=$_GET[page]");
    }
    else
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка при сокрытии
                               каталога");
    }
  }
```

```
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
?>
```

Последней парой скриптов, которые будут рассмотрены применительно к разделам, являются файлы catup.php и catdown.php, которые поднимают и соответственно опускают раздел на одну позицию относительно остальных разделов.

В листинге 11.40 приводится содержимое файла catup.php.

Листинг 11.40. Подъем позиции раздела относительно других разделов, catup.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Проверяем, передано ли в параметре число
  $_GET['id_catalog'] = intval($_GET['id_catalog']);
  $_GET['id_parent']  = intval($_GET['id_parent']);

  try
  {
    // Извлекаем позицию текущего каталога
    $query = "SELECT pos FROM $tbl_catalog
              WHERE id_catalog = $_GET[id_catalog]
              LIMIT 1";
    $cat = mysql_query($query);
    if(!$cat)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка при извлечении
                               позиции каталога");
    }
    if(mysql_num_rows($cat))
    {
      $pos_current = mysql_result($cat, 0);
    }
```

```
    // Извлекаем позицию предыдущего каталога
    $query = "SELECT pos FROM $tbl_catalog
              WHERE id_parent = $_GET[id_parent] AND
                    pos < $pos_current
              ORDER BY pos DESC
              LIMIT 1";
    $cat = mysql_query($query);
    if(!$cat)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка при извлечении
                               позиции каталога");
    }
    if(mysql_num_rows($cat))
    {
      $pos_preview = mysql_result($cat, 0);

      // Меняем местами текущий и предыдущий каталоги
      $query = "UPDATE $tbl_catalog
                SET pos = $pos_next + $pos_preview — pos
                WHERE id_parent = $_GET[id_parent] AND
                      pos IN ($pos_next, $pos_preview)";
      if(!mysql_query($query))
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка изменения
                                 позиции раздела");
      }
    }
    // Если запрос выполнен удачно, осуществляем автоматический переход
    // на главную страницу администрирования
    header("Location: index.php?".
           "id_parent=$_GET[id_parent]&page=$_GET[page]");
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
?>
```

Положение раздела относительно других разделов определяется значением поля `pos`. Подъем позиции раздела сводится к обмену значениями полей `pos` между текущим и предыдущим разделами (рис. 11.18).

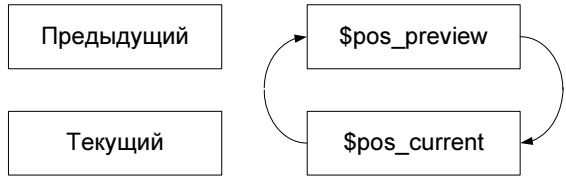


Рис. 11.18. Обмен значениями поля pos между текущей и предыдущей записями для подъема раздела на одну позицию вверх

Скрипт из листинга 11.40 выполняет три SQL-запроса, первый SELECT-запрос извлекает значение поля pos для текущей записи $pos_current, второй SELECT-запрос извлекает значение поля pos для предыдущей записи $pos_preview. Последний UPDATE-запрос осуществляет обмен значениями.

Замечание

Важно отметить, что WHERE-условие запросов имеет дополнительное условие id_parent = $_GET[id_parent], которое предотвращает изменение записей из соседних разделов.

Так как значения поля pos всегда целочисленные, то обмен значениями осуществляется по следующим формулам:

```
pos1 = (pos1 + pos2) – pos1
pos2 = (pos1 + pos2) – pos2
```

Благодаря SQL-оператору IN такой подход позволяет обменять значения при помощи одного UPDATE-запроса, а не двух.

Аналогичным образом работает скрипт catdown.php, опускающий раздел на одну позицию вниз, только вместо предыдущей записи осуществляется поиск следующей записи $pos_next (листинг 11.41).

Листинг 11.41. Опускание позиции раздела относительно остальных разделов, catdown.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");

  // Проверяем, передано ли в параметре число
  $_GET['id_catalog'] = intval($_GET['id_catalog']);
  $_GET['id_parent']  = intval($_GET['id_parent']);
```

```
try
{
  // Извлекаем позицию текущего каталога
  $query = "SELECT pos FROM $tbl_catalog
            WHERE id_catalog = $_GET[id_catalog]
            LIMIT 1";
  $cat = mysql_query($query);
  if(!$cat)
  {
    throw new ExceptionMySQL(mysql_error(),
                             $query,
                            "Ошибка при извлечении
                             позиции каталога");
  }
  if(mysql_num_rows($cat))
  {
    $pos_current = mysql_result($cat, 0);
  }
  // Извлекаем позицию следующего каталога
  $query = "SELECT pos FROM $tbl_catalog
            WHERE id_parent = $_GET[id_parent] AND
                  $sql_fragment pos > $pos_current
            ORDER BY pos
            LIMIT 1";
  $cat = mysql_query($query);
  if(!$cat)
  {
    throw new ExceptionMySQL(mysql_error(),
                             $query,
                            "Ошибка при извлечении
                             позиции каталога");
  }
  if(mysql_num_rows($cat))
  {
    $pos_next = mysql_result($cat, 0);

    // Меняем местами текущий и следующий каталоги
    $query = "UPDATE $tbl_catalog
              SET pos = $pos_next + $pos_current — pos
              WHERE id_parent = $_GET[id_parent] AND
                    pos IN ($pos_next, $pos_current)";
    if(!mysql_query($query))
    {
```

```
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка изменения
                                 позиции раздела");
      }
    }
    // Если запрос выполнен удачно, осуществляем автоматический переход
    // на главную страницу администрирования
    header("Location: index.php?".
           "id_parent=$_GET[id_parent]&page=$_GET[page]");
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
?>
```

За отображение списка позиций (статей и ссылок) несет ответственность файл position.php, который использует в своей работе класс постраничной навигации `pager_mysql` (см. *главу 4*). В листинге 11.42 приводится содержимое файла position.php.

Листинг 11.42. Список позиций (статей и ссылок), position.php

```
<table cellspacing="0" cellpadding="0">
  <tr valign="top">
    <td>
    <?php
      echo "<a class=menu

href=urladd.php?id_parent=$_GET[id_parent]&page=$_GET[page]
title=\"Добавить ссылку на страницу текущего или любого другого сайта\">
               Добавить ссылку
             </a>   
            <a class=menu
              href=artadd.php?id_parent=$_GET[id_parent]&page=$_GET[page]
               title=\"Добавить статью в данный раздел\">
               Добавить статью</a>";
    ?>
    </td>
  </tr>
</table><br>
<?php
```

```
try
{
  // Число ссылок в постраничной навигации
  $page_link = 3;
  // Число элементов на странице
  $pnumber = 10;
  // Объявляем объект постраничной навигации
  $obj = new pager_mysql($tbl_position,
                         "WHERE id_catalog=$_GET[id_parent]",
                         "ORDER BY pos",
                         $pnumber,
                         $page_link,
                         "&id_parent=$_GET[id_parent]");

  // Получаем содержимое текущей страницы
  $position = $obj->get_page();
  // Если имеется хотя бы одна запись — выводим
  if(!empty($position))
  {
    // Выводим заголовок таблицы
    echo '<table width="100%"
                 class="table"
                 border="0"
                 cellpadding="0"
                 cellspacing="0">
            <tr class="header" align="center">
              <td align=center>Название</td>
              <td align=center>URL</td>
              <td width=20 align=center>Поз.</td>
              <td width=50>Действия</td>
            </tr>';
    for($i = 0; $i < count($position); $i++)
    {
      $url = "id_position={$position[$i][id_position]}&".
             "id_catalog=$_GET[id_parent]&page=$_GET[page]";
      // Выясняем, скрыт элемент или нет
      if($position[$i]['hide'] == 'hide')
      {
        $strhide = "<a href=urlshow.php?$url>Отобразить</a>";
        $style = " class=hiddenrow ";
      }
      else
      {
        $strhide = "<a href=urlhide.php?$url>Скрыть</a>";
```

```
    $style = "";
  }
  // Выясняем, является элемент статьей или ссылкой
  if($position[$i]['url'] == 'article')
  {
    $edit = "artedit.php";
    // $url нельзя использовать из-за параметра page
    $name  = "<td>
                <p class=small>
                  <a href=paragraph.php?".
                    "id_position={$position[$i][id_position]}&".
                    "id_catalog=$_GET[id_parent]>".
                    print_page($position[$i]['name'])."</a>
                </p>
              </td>";
  }
  else
  {
    $edit = "urledit.php";
    $name  = "<td><p class=small>".
                print_page($position[$i]['name']).
             "</p></td>";
  }

  // Выводим позиции
  echo "<tr $style>
          $name
          <td><p class=small>".
            print_page($position[$i]['url'])."</td>
          <td align=center><p class=small>".
             print_page($position[$i]['pos']).
          "</p></td>
          <td>
            <p class=small><a href=urlup.php?$url>Вверх</a><br>
            $strhide<br>
            <a href=$edit?$url>Редактировать</a><br>
            <a href=#
               onClick=\"delete_position('urldel.php?$url');\">
               Удалить</a><br>
            <a href=urldown.php?$url>Вниз</a></p>
          </td>
      </tr>";
}
```

```
      echo "</table><br>";
    }
?>
<table cellspacing="0" cellpadding="0">
  <tr valign="top">
    <td>
      <?php
        // Выводим ссылки на другие страницы
        echo $obj;
      ?>
    </td>
  </tr>
</table><br>
<?php
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }
?>
<script language='JavaScript1.1' type='text/javascript'>
<!--
  function delete_position(url)
  {
    if(confirm("Вы действительно хотите удалить элемент?"))
    {
      location.href=url;
    }
    return false;
  }
//-->
</script>
```

Напомним, что в качестве элемента могут выступать либо ссылки, либо статьи. Внешний вид HTML-формы для добавления ссылок urladd.php представлен на рис. 11.19.

Для добавления статей используется HTML-форма artadd.php, внешний вид которой представлен на рис. 11.20.

В листинге 11.43 приводится содержимое файла artadd.php.

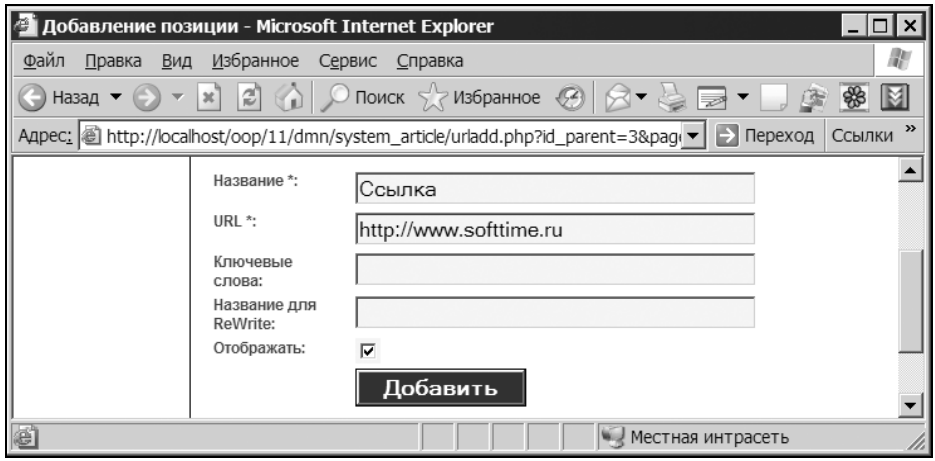


Рис. 11.19. HTML-форма для добавления ссылок

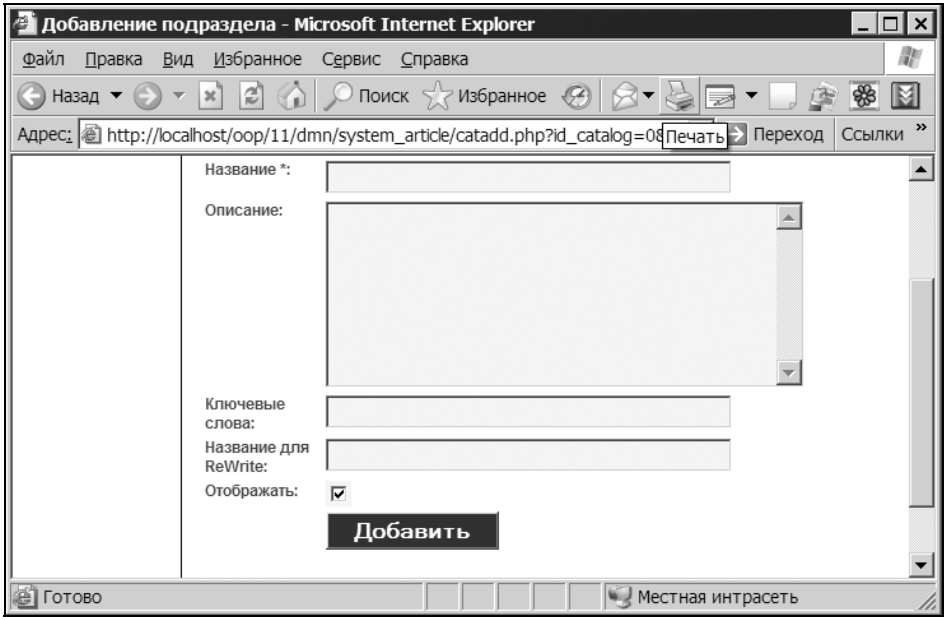


Рис. 11.20. HTML-форма для добавления статей

Листинг 11.43. Добавление статьи, artadd.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
```

```
// Подключаем блок авторизации
require_once("../utils/security_mod.php");
// Подключаем классы формы
require_once("../../config/class.config.dmn.php");

// Защита от SQL-инъекции
$_GET['id_parent'] = intval($_GET['id_parent']);
// Параметры
$button_name = "Добавить";
$class_name = "field";

if(empty($_POST)) $_REQUEST['hide'] = true;
$name = new field_text("name",
                       "Название",
                       true,
                       $_POST['name']);
$description = new field_textarea("description",
                            "Содержимое статьи",
                            true,
                            $_POST['description']);
$keywords = new field_text("keywords",
                           "Ключевые слова",
                           false,
                           $_POST['keywords']);
$modrewrite = new field_text_english("modrewrite",
                           "Название для<br>ReWrite",
                           false,
                           $_POST['modrewrite']);
$hide = new field_checkbox("hide",
                           "Отображать",
                           $_REQUEST['hide']);
$id_parent = new field_hidden_int("id_parent",
                            true,
                            $_REQUEST['id_parent']);
$page = new field_hidden_int("page",
                            false,
                            $_REQUEST['page']);
try
{
  $form = new form(array("name" => $name,
                         "description" => $description,
                         "keywords" => $keywords,
                         "modrewrite" => $modrewrite,
```

```
                            "hide" => $hide,
                            "modrewrite" => $modrewrite,
                            "id_parent" => $id_parent,
                            "page" => $page),
                            $button_name,
                            $class_name);
}
catch(ExceptionObject $exc)
{
  require("../utils/exception_object.php");
}

// Обработчик HTML-формы
if(!empty($_POST))
{
  try
  {
    // Проверяем корректность заполнения HTML-формы
    // и обрабатываем текстовые поля
    $error = $form->check();
    if(empty($error))
    {
      // Извлекаем текущую максимальную позицию
      $query = "SELECT MAX(pos) FROM $tbl_position
                WHERE id_catalog = {$form->fields[id_parent]->value}";
      $pos = mysql_query($query);
      if(!$pos)
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка при извлечении
                                 текущей позиции");
      }
      $pos = mysql_result($pos, 0) + 1;
      // Скрыт или открыт элемент
      if($form->fields['hide']->value) $showhide = "show";
      else $showhide = "hide";
      // Формируем SQL-запрос на добавление элемента
      $query = "INSERT INTO $tbl_position
                VALUES (NULL,
                        '{$form->fields[name]->value}',
                        'article',
                        '{$form->fields[keywords]->value}',
```

```
                    '{$form->fields[modrewrite]->value}',
                     $pos,
                    '$showhide',
                    {$form->fields[id_parent]->value})";
if(!mysql_query($query))
{
  throw new ExceptionMySQL(mysql_error(),
                           $query,
                          "Ошибка при добавлении
                           нового элемента");
}
// Извлекаем значение автоматически сформированного
// первичного ключа только что вставленной записи
// AUTO_INCREMENT
$id_position = mysql_insert_id();
// Разбиваем текст на параграфы
$par = preg_split("|\r\n|",
                  $form->fields['description']->value);
if(!empty($par))
{
  $i = 0;
  foreach($par as $parag)
  {
    $i++;
    $sql[] = "(NULL,
                '$parag',
                'text',
                'left',
                'show',
                $i,
                $id_position,
                {$form->fields[id_parent]->value})";
  }
  $query = "INSERT INTO $tbl_paragraph
            VALUES ".implode(",",$sql);
  if(!mysql_query($query))
  {
    throw new ExceptionMySQL(mysql_error(),
                           $query,
                          "Ошибка при добавлении
                           нового элемента");
  }
}
```

```
        // Осуществляем перенаправление на главную
        // страницу администрирования
        header("Location: index.php?".
               "id_parent={$form->fields[id_parent]->value}&".
               "page={$form->fields[page]->value}");
        exit();
      }
    }
    catch(ExceptionMySQL $exc)
    {
      require("../utils/exception_mysql.php");
    }
  }
  // Начало страницы
  $title     = 'Добавление позиции';
  $pageinfo  = '<p class=help></p>';
  // Включаем заголовок страницы
  require_once("../utils/top.php");

  echo "<p><a href=# onClick='history.back()'>Назад</a></p>";
  // Выводим сообщения об ошибках, если они имеются
  if(!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
  // Выводим HTML-форму
  $form->print_form();

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
```

Главная особенность скрипта из листинга 11.43 заключается в том, что содержимое текстовой области `description` разбивается на параграфы при помощи функции `preg_split()` и регулярного выражения, соответствующего переводу строки. При этом параграфы при помощи многострочного оператора `INSERT` вставляются в таблицу `system_paragraph` (`$tbl_paragraph`), в то время как все остальные данные из HTML-формы artadd.php помещаются в таблицу `system_position` (`$tbl_position`). Значение первичного ключа

`id_position` только что вставленной в таблицу `system_position` записи, которое необходимо для записей таблицы `system_paragraph`, извлекается при помощи функции `mysql_insert_id()`.

Замечание

Скрипты, осуществляющие редактирование, удаление, скрытие/отображение и перемещение элементов повторяют соответствующие скрипты, ранее рассмотренные для разделов. Их листинги не приводятся в данной книге, но присутствуют на компакт-диске, поставляемом вместе с ней.

В списке элементов ссылки отображаются не подсвеченными, тогда как названия статей представляют собой гиперссылки, ведущие на страницу paragraph.php, которая позволяет добавлять, редактировать, удалять, скрывать/отображать и менять позиции отдельных параграфов статьи (рис. 11.21).

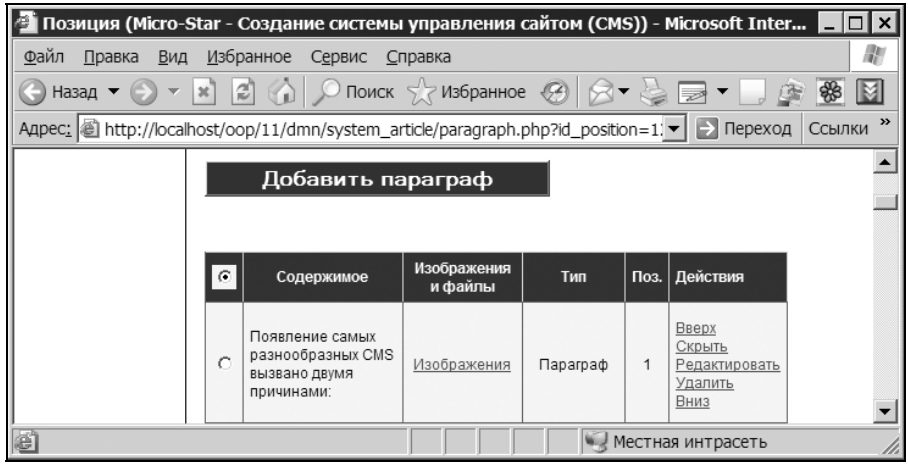


Рис. 11.21. Страница управления параграфами статьи

Как видно из рис. 11.21, страница управления параграфами представляет собой таблицу, каждая строка в которой соответствует одному параграфу. При этом каждый параграф и шапка снабжаются переключателями, позволяющими указать место вставки нового параграфа. В разделе "Содержимое" приводится текст параграфа. Следующий раздел "Изображения и файлы" предоставляет ссылку для управления изображениями, причем если параграф имеет хотя бы одно изображение, то рядом со ссылкой приводится количество изображений в круглых скобках. Столбец "Тип" сообщает, является ли текущий элемент параграфом или заголовком (от H1 до H6). Столбец "Поз." определяет номер параграфа, а столбец "Действия" предоставляет управляющие ссылки. В листинге 11.44 приводится содержимое файла paragraph.php.

Листинг 11.44. Список параграфов paragraph.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Подключаем блок авторизации
  require_once("../utils/security_mod.php");
  // Подключаем SoftTime FrameWork
  require_once("../../config/class.config.dmn.php");
  // Навигационное меню
  require_once("../utils/utils.navigation.php");
  // Обработка текста перед выводом
  require_once("../utils/utils.print_page.php");

  // Защита от SQL-инъекции
  $_GET['id_position'] = intval($_GET['id_position']);
  $_GET['id_catalog']  = intval($_GET['id_catalog']);

  try
  {
    // Извлекаем информацию о каталоге
    $query = "SELECT * FROM $tbl_catalog
              WHERE id_catalog = $_GET[id_catalog]
              LIMIT 1";
    $cat = mysql_query($query);
    if(!$cat)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка при обращении
                               к каталогу");
    }
    $catalog = mysql_fetch_array($cat);

    // Извлекаем информацию об элементе
    $query = "SELECT * FROM $tbl_position
              WHERE id_position = $_GET[id_position]";
    $pos = mysql_query($query);
    if(!$pos) exit("Ошибка при обращении к элементу");
    $position = mysql_fetch_array($pos);

    $title = $titlepage = 'Элемент ('.$catalog['name'].
                          ' — '.$position['name'].')';
```

```
$pageinfo = '<p class=help>Здесь осуществляется
             администрирование элемента ('.$catalog['name'].
             ' — '.$position['name'].').
             Параграф может представлять собой как обычный
             текстовый абзац, так и заголовок. Возможно
             использование шести заголовков
             (H1, H2, H3, H4, H5, H6), H1 — самый крупный
             заголовок, применяемый обычно для названия
             страниц, далее заголовки уменьшаются в
             размере, так что H6 — это самый мелкий заголовок.</p>';


// Число ссылок в постраничной навигации
$page_link = 3;
// Число элементов на странице
$pnumber = 10;
// Объявляем объект постраничной навигации
$obj = new pager_mysql($tbl_paragraph,
                       "WHERE id_position = $_GET[id_position] AND
                              id_catalog=$_GET[id_catalog]",
                       "ORDER BY pos",
                       $pnumber,
                       $page_link,
                       "&id_position=$_GET[id_position]&".
                       "id_catalog=$_GET[id_catalog]");

// Получаем содержимое текущей страницы
$paragraph = $obj->get_page();

// Включаем заголовок страницы
require_once("../utils/top.php");

// Если это не корневой каталог, выводим ссылки для возврата
// и для добавления подкаталога
if($_GET['id_catalog'] != 0)
{
  echo '<table cellspacing="0" cellspacing="0" border=0><tr>
  <tr valign="top"><td height="25"><p>';
  echo "<a class=menu href=index.php?id_parent=0>
           Корневой каталог</a>-&gt;".
           menu_navigation($_GET['id_catalog'],
                           "",
                           $tbl_catalog).$position['name'];
```

```
  echo "</td></tr></table>";
}

// Добавить параграф
echo "<form action=paradd.php>";
echo "<input class=button
             type=submit
             value='Добавить параграф'><br><br>";
echo "<input type=hidden name=page value='$_GET[page]'><br><br>";

if(!empty($paragraph))
{
  // Выводим заголовок таблицы каталогов
  echo "<input type=hidden
               name=id_catalog
               value=$_GET[id_catalog]>";
  echo "<input type=hidden
               name=id_position
               value=$_GET[id_position]>";
  echo '<table width="100%"
               class="table"
               border="0"
               cellpadding="0"
               cellspacing="0">
          <tr class="header" align="center">
            <td width=20 align=center>
               <input type=radio name=pos value=-1 checked></td>
            <td align=center>Содержимое</td>
            <td width=100 align=center>Изображения<br> и файлы</td>
            <td width=100 align=center>Тип</td>
            <td width=20 align=center>Поз.</td>
            <td width=50>Действия</td>
          </tr>';
  for($i = 0; $i < count($paragraph); $i++)
  {
    $url = "id_paragraph={$paragraph[$i][id_paragraph]}".
           "&id_position=$_GET[id_position]&".
           "id_catalog=$_GET[id_catalog]".
           "&page=$_GET[page]";
    // Выясняем тип параграфа
    $type = "Параграф";
    switch($paragraph[$i]['type'])
    {
      case 'text':
```

```
      $type = "Параграф";
      break;
    case 'title_h1':
      $type = "Заголовок H1";
      break;
    case 'title_h2':
      $type = "Заголовок H2";
      break;
    case 'title_h3':
      $type = "Заголовок H3";
      break;
    case 'title_h4':
      $type = "Заголовок H4";
      break;
    case 'title_h5':
      $type = "Заголовок H5";
      break;
    case 'title_h6':
      $type = "Заголовок H6";
      break;
  }
  // Выясняем тип выравнивания параграфа
  $align = "";
  switch($paragraph[$i]['align'])
  {
    case 'left':
      $align = "align=left";
      break;
    case 'center':
      $align = "align=center";
      break;
    case 'right':
      $align = "align=right";
      break;
  }
  // Выясняем, скрыт элемент или нет
  if($paragraph[$i]['hide'] == 'hide')
  {
    $strhide = "<a href=parshow.php?$url>Отобразить</a>";
    $style=" class=hiddenrow ";
  }
  else
  {
    $strhide = "<a href=parhide.php?$url>Скрыть</a>";
```

```
        $style="";
      }
      // Вычисляем, сколько изображений у данного элемента
      $query = "SELECT COUNT(*) FROM $tbl_paragraph_image
                WHERE id_paragraph = {$paragraph[$i][id_paragraph]} AND
                      id_position = $_GET[id_position] AND
                      id_catalog = $_GET[id_catalog]";
      $tot = mysql_query($query);
      if(!$tot)
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка при подсчете
                                 количества изображений");
      }
      $total_image = mysql_result($tot, 0);
      if($total_image) $print_image = " ($total_image)";
      else $print_image = "";

      echo "<tr $style>
            <td><p align=center>
            <input type=radio name=pos value=".$paragraph[$i]['pos']."
            </p></td>
            <td>
              <p $align>".print_page($paragraph[$i]['name'])."</p>
            </td>
            <td><p align=center>
              <a href=image.php?$url>Изображения$print_image</a>
            </p></td>
            <td><p align=center>".print_page($type)."</p></td>
            <td><p align=center>".$paragraph[$i]['pos']."</p></td>
            <td><p>
            <a href=parup.php?$url>Вверх</a><br>
            $strhide<br>
            <a href=paredit.php?$url>Редактировать</a><br>
            <a href=#
             onClick=\"delete_par('pardel.php?$url');\">Удалить</a><br>
            <a href=pardown.php?$url>Вниз</a><br></td>
          </tr>";
    }
    echo "</table><br><br>";
  }
  echo "</form>";
?>
```

```
<table cellspacing="0" cellpadding="0">
  <tr valign="top">
    <td>
      <?php
        // Выводим ссылки на другие страницы
        echo $obj;
      ?>
    </td>
  </tr>
</table><br>
<?php
  }
  catch(ExceptionMySQL $exc)
  {
    require("../utils/exception_mysql.php");
  }

  // Включаем завершение страницы
  require_once("../utils/bottom.php");
?>
<script language='JavaScript' type='text/javascript'>
<!--
  function delete_par(url)
  {
    if(confirm("Вы действительно хотите удалить параграф?"))
    {
      location.href=url;
    }
    return false;
  }
//-->
</script>
```

Замечание

Скрипты, осуществляющие редактирование, удаление, скрытие/отображение и перемещение параграфов, повторяют аналогичные скрипты, ранее рассмотренные для разделов. Их листинги не приводятся в книге, но с ними можно ознакомиться на компакт-диске, поставляемом вместе с книгой.

Переход по ссылке "Изображения" приводит к странице управления изображениями параграфа image.php. Содержимое файла image.php повторяет ранее рассмотренные страницы index.php, position.php и paragraph.php, обеспечивая постраничный вывод таблицы `system_paragraph_image` (`$tbl_paragraph_image`).

Интересной особенностью данной системы управления является отображение изображения в отдельном окне (рис. 11.22).

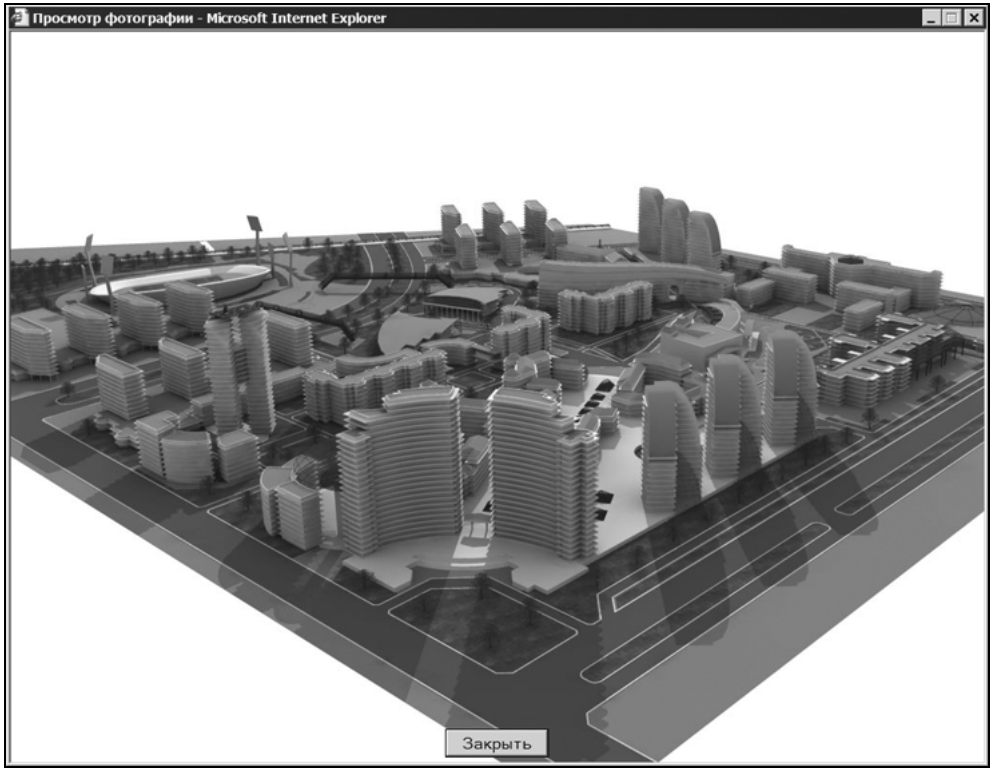


Рис. 11.22. Вывод изображения во всплывающем окне

За вывод изображения в отдельном окне несет ответственность файл show.php, содержимое которого представлено в листинге 11.45.

Листинг 11.45. Окно для отображения изображения

```
<?php
  $filename = $_GET['img'];
  $size = getimagesize($filename);
?>
<html>
<head>
<title>Просмотр фотографии</title>
<meta http-equiv="imagetoolbar" content="no">
```

```
<style>
 table{font-size: 12px; font-family: Arial, Helvetica, sans-serif;
background-color: #F3F3F3;}
</style>
</head>
<body marginheight="0"
      marginwidth="0"
      rightmargin="0"
      bottommargin="0"
      leftmargin="0"
      topmargin="0">
<table height="100%"
       cellpadding="0"
       cellspacing="0"
       width="100%"
       border="1">
  <tr>
    <td height="100%" valign="middle" align="center">
      Дождитесь загрузки изображения
      <div  style="position: absolute; top: 0px; left: 0px">
        <img src="<? echo $filename;?>" border="0" <?= $size[3] ?>>
      </div>
    </td>
  </tr>
</table>
<div style="position: absolute; z-index: 2; width: 100%; bottom: 5px"
     align="center">
<input class=button type="submit" value="Закрыть"
      onclick="window.close();"></div>
</body>
</html>
```

11.5.3. Система представления

В первую очередь необходимо организовать меню для доступа к разделам и статьям из любой точки сайта. Для этого модернизируем рассмотренный в *разделе 11.4.3* файл top.php, добавив в него, помимо блока для вывода трех последних новостных сообщений, вывод корневых разделов системы администрирования содержимого сайта (рис. 11.23).

В листинге 11.46 приводится фрагмент файла top.php, который несет ответственность за формирование блока "Разделы".

Рис. 11.23. Блок доступа к разделам сайта

Листинг 11.46. Вывод корневых разделов сайта

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config/config.php");
  // Подключаем систему классов
  require_once("config/class.config.php");
  ...
  $query = "SELECT * FROM $tbl_catalog
            WHERE hide = 'show' AND id_parent = 0
            ORDER BY pos";
  $cat = mysql_query($query);
  if (!$cat) exit("Ошибка при обращении к блоку статей");
  if(mysql_num_rows($cat))
  {
    while($catalog = mysql_fetch_array($cat))
    {
      echo "<b><a href=index.php?id_catalog=$catalog[id_catalog]
                  class=\"rightpanel_lnk\">
                  $catalog[name]</a></b><br><br>";
    }
  }
  ...
?>
```

Каждая из ссылок ведет на файл index.php, который, в зависимости от значения передаваемого ему GET-параметра `id_catalog`, выводит значение того

или иного раздела. Раздел может содержать как другие подразделы, так и статьи (рис. 11.24).

Рис. 11.24. Список подразделов раздела

Каждый раздел может содержать статьи, которые выводятся в виде списка (рис. 11.25). Причем, если раздел содержит только одну статью, она выводится без списка подразделов и статей. В листинге 11.47 приводится содержимое файла index.php.

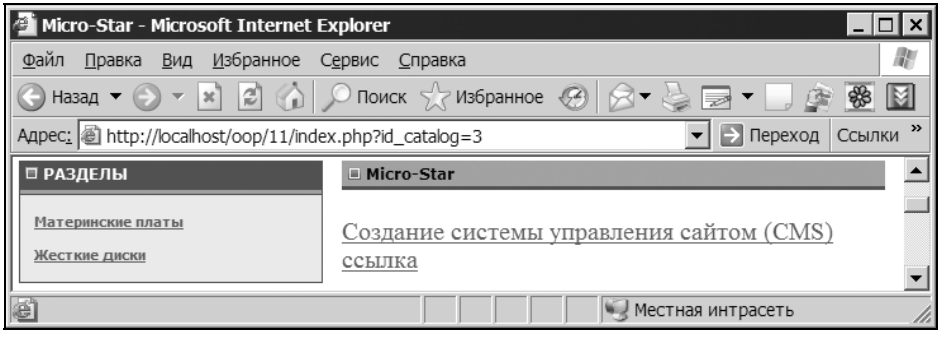


Рис. 11.25. Список статей

Листинг 11.47. Вывод содержимого подразделов, index.php

```
<?php
  // Определяем параметр для статей
  define("ARTICLE", 1);
```

```
// Если не передан параметр id_position — выводим список статей
if(empty($_GET['id_position']))
{
  // Проверяем GET-параметры, предотвращая SQL-инъекцию
  $_GET['page']       = intval($_GET['page']);
  $_GET['id_catalog'] = intval($_GET['id_catalog']);

  // Запрашиваем параметры текущего раздела
  $query = "SELECT * FROM $tbl_catalog
            WHERE hide = 'show' AND
                  id_catalog = ".$_GET['id_catalog'];
  $cat = mysql_query($query);
  if(!$cat) exit("Ошибка при извлечении параметров текущего раздела");
  $catalog = mysql_fetch_array($cat);

  // Подключаем верхний шаблон
  $title = $catalog['name'];
  $keywords = $catalog['keywords'];
  require_once ("templates/top.php");

  // Название
  ?>
    <!-- s rightpanel -->
    <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
      <td height="20" bgcolor="#82A6DE" class="rightpanel_ttl">
        <img src="dataimg/dot_ttl.gif" align="absmiddle">
        <?php echo htmlspecialchars($title); ?>
      </td>
    </tr>
    <tr>
      <td height="3" nowrap bgcolor="#004BBC"></td>
    </tr>
    <tr>
      <td> </td>
    </tr>
  <?php
  // Запрашиваем подразделы текущего раздела
  $query = "SELECT * FROM $tbl_catalog
            WHERE hide = 'show' AND id_parent = $_GET[id_catalog]
            ORDER BY pos";
  $sub = mysql_query($query);
  if (!$sub) exit("Ошибка при обращении к блоку статей");
```

```
if(mysql_num_rows($sub))
{
  echo "<tr><td class=\"main_txt\">";
  while($subcatalog = mysql_fetch_array($sub))
  {
    echo "<h4><a href=\"$_SERVER[PHP_SELF]?".
         "id_catalog=$subcatalog[id_catalog]\"
          class=\"menu_lnk\">".
          htmlspecialchars($subcatalog['name'])."</a>
          </h4>";
  }
  echo "</td></tr>";
}

// Запрашиваем статьи текущего раздела
$query = "SELECT * FROM $tbl_position
          WHERE hide = 'show' AND
                id_catalog = $_GET[id_catalog]
          ORDER BY pos";
$pos = mysql_query($query);
if (!$pos) exit("Ошибка при обращении к блоку статей");
if(mysql_num_rows($pos) > 0)
{
  // Статья одна и подразделов нет
  if(mysql_num_rows($pos) == 1 && !mysql_num_rows($sub))
  {
    // Получаем параметры текущей статьи
    $position = mysql_fetch_array($pos);
    // Если статья представляет собой
    // ссылку — осуществляем переадресацию
    if($position['url'] != 'article')
    {
      echo "<HTML><HEAD>
         <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$position[url]'>
            </HEAD></HTML>";
      exit();
    }
    // Статья одна и нет подразделов — выводим содержимое статьи
    $_GET['id_position'] = $position['id_position'];
    require_once("article_print.php");
  }
  // Статей несколько или имеются подразделы
  else
  {
```

```
      echo "<tr><td class=\"main_txt\">";
      while($position = mysql_fetch_array($pos))
      {
        if($position['url'] != 'article')
        {
          echo "<a href=\"".htmlspecialchars($position['url'])."\"
                   class=\"rightpanel_lnk\">
                 ".htmlspecialchars($position['name'])."</a><br>";
        }
        else
        {
          echo "<a href=\"$_SERVER[PHP_SELF]?".
               "id_catalog=$_GET[id_catalog]&".
               "id_position=$position[id_position]\"
                class=\"rightpanel_lnk\">".
                htmlspecialchars($position['name'])."</a><br>";
        }
      }
      echo "</td></tr>";
    }
  }
  echo "</table>";
}
else
{
  // Проверяем GET-параметры, предотвращая SQL-инъекцию
  $_GET['id_position'] = intval($_GET['id_position']);
  // Получаем параметры текущей статьи
  $query = "SELECT * FROM $tbl_position
            WHERE hide = 'show' AND
                  id_position = $_GET[id_position]";
  $pos = mysql_query($query);
  if (!$pos) exit("Ошибка при обращении к блоку статей");
  $position = mysql_fetch_array($pos);
  // Если статья представляет собой ссылку -
  // осуществляем переадресацию
  if($position['url'] != 'article')
  {
    echo "<HTML><HEAD>
          <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$position[url]'>
          </HEAD></HTML>";
    exit();
  }
```

```
    // Подключаем верхний шаблон
    $title = $position['name'];
    $_GET['id_catalog'] = $position['id_catalog'];
    $keywords = $position['keywords'];
    require_once ("templates/top.php");

    // Название
    ?>
      <!-- s rightpanel -->
      <table width="100%" border="0" cellpadding="0" cellspacing="0">
      <tr>
        <td height="20" bgcolor="#82A6DE" class="rightpanel_ttl">
          <img src="dataimg/dot_ttl.gif" align="absmiddle">
          <?php echo htmlspecialchars($title); ?>
        </td>
      </tr>
      <tr>
        <td height="3" nowrap bgcolor="#004BBC"></td>
      </tr>
      <tr>
        <td> </td>
      </tr>
    <?php
    // Выводим статью
    require_once("article_print.php");

    echo "</table>";
  }
  // Подключаем нижний шаблон
  require_once ("templates/bottom.php");
?>
```

Файл index.php формирует только списки подразделов и статей; передача файлу index.php GET-параметра `id_position` приводит к подключению файла article_print.php, который, в свою очередь, формирует статью (рис. 11.26). В листинге 11.48 приводится содержимое файла article_print.php.

Листинг 11.48. Блок формирования статьи, article_print.php

```
<?php
  if(!defined("ARTICLE")) return;
  if(!preg_match("|^[\d]+$|",$_GET['id_position'])) return;
  if(!preg_match("|^[\d]+$|",$_GET['id_catalog'])) return;
```

```
// Обработка текста перед выводом
require_once("dmn/utils/utils.print_page.php");

// Выводим список каталогов
$query = "SELECT * FROM $tbl_paragraph
          WHERE id_position = $_GET[id_position] AND
                id_catalog = $_GET[id_catalog] AND
                hide = 'show'
          ORDER BY pos";

$par = mysql_query($query);
if(!$par) exit("Ошибка при обращении к параграфам элемента");
$type_catalog = "";
if(mysql_num_rows($par)>0)
{
  while($paragraph = mysql_fetch_array($par))
  {
    // Выясняем тип выравнивания параграфа
    $align = "";
    switch($paragraph['align'])
    {
      case 'left':
        //$type .= " (слева)";
        $align = "left";
        break;
      case 'center':
        //$type .= " (по центру)";
        $align = "center";
        break;
      case 'right':
        //$type .= " (справа)";
        $align = "right";
        break;
    }

    // Изображения элемента
    $image_print = "";
    $query = "SELECT * FROM $tbl_paragraph_image
              WHERE id_paragraph = $paragraph[id_paragraph] AND
                    id_position = $_GET[id_position] AND
                    id_catalog = $_GET[id_catalog] AND
                    hide = 'show'";
    $img = mysql_query($query);
```

```
      if(!$img) exit("Ошибка при извлечении изображений");
      if(mysql_num_rows($img))
      {
        // Извлекаем изображения
        unset($img_arr);
        while($image = mysql_fetch_array($img))
        {
          // ALT-тег
          if(!empty($image['alt'])) $alt = "alt='$image[alt]'";
          else $alt = "";
          // Размер уменьшенного варианта изображения
          $size_small = @getimagesize($image['small']);
          // Название изображения
          if(!empty($image['name']))
          {
            $name = "<br><br><b>".$image['name']."</b>";
          }
          else $name = "";
          // Полный вариант изображения
          if(empty($image['big']))
          {
            $img_arr[] = "<img $alt src='$image[small]'
                              width=$size_small[0]
                              height=$size_small[1]>$name";
          }
          else
          {
            $size = @getimagesize($image['big']);
            $img_arr[] = "<a href=#

onclick=\"show_img('$image[id_image]',".$size[0].",".$size[1]."); return
false \">
                          <img $alt src='$image[small]'
                              border=0
                              width=$size_small[0]
                              height=$size_small[1]></a>$name";
          }
        }
        for($i = 0; $i < count($img_arr)%3; $i++) $img_arr[] = "";
        // Выводим изображения
        for($i = 0, $k = 0; $i < count($img_arr); $i++, $k++)
        {
          if($k == 0)
             $image_print .= "<table cellpadding=5><tr valign=top>";
```

```
      $image_print .= "<td class=\"main_txt\">".$img_arr[$i]."</td>";
      if($k == 2)
      {
        $k = -1;
        $image_print .= "</tr></table>";
      }
    }
  }

  // Выясняем тип параграфа
  $class = "rightpanel_txt";
  switch($paragraph['type'])
  {
    case 'text':
      $class = "rightpanel_txt";
      echo "<tr>
              <td class=\"$class\">
                <p align=$align>".print_page($paragraph['name']).
               "<br>$image_print</p>
              </td>
            </tr>";
      break;
    case 'title_h1':
      $class = "rightpanel_ttl";
      echo "<tr>
              <td class=\"$class\">
                <h1 align=$align>".print_page($paragraph['name']).
               "<br>$image_print</h1>
              </td>
            </tr>";
      break;
    case 'title_h2':
      $class = "rightpanel_ttl";
      echo "<tr>
              <td class=\"$class\">
                <h2 align=$align>".print_page($paragraph['name']).
               "<br>$image_print</h2>
              </td>
            </tr>";
      break;
    case 'title_h3':
      $class = "rightpanel_ttl";
      echo "<tr>
```

```
                <td class=\"$class\">
                  <h3 align=$align>".print_page($paragraph['name']).
                  "<br>$image_print</h3>
                </td>
              </tr>";
          break;
        case 'title_h4':
          $class = "rightpanel_ttl";
          echo "<tr>
                  <td class=\"$class\">
                    <h4 align=$align>".print_page($paragraph['name']).
                    "<br>$image_print</h4>
                  </td>
                </tr>";
          break;
        case 'title_h5':
          $class = "rightpanel_ttl";
          echo "<tr>
                  <td class=\"$class\">
                    <h5 align=$align>".print_page($paragraph['name']).
                    "<br>$image_print</h5>
                  </td>
                </tr>";
          break;
        case 'title_h6':
          $class = "rightpanel_ttl";
          echo "<tr>
                  <td class=\"$class\">
                    <h6 align=$align>".print_page($paragraph['name']).
                    "<br>$image_print</h6>
                  </td>
                </tr>";
          break;
      }
    }
  }
?>
<script language='JavaScript1.1' type='text/javascript'>
<!--
  function show_img(id_image,width,height,adm)
  {
    var a;
    var b;
    var url;
```

```
    vidWindowWidth=width;
    vidWindowHeight=height;
    a=(screen.height-vidWindowHeight)/5;
    b=(screen.width-vidWindowWidth)/2;
    features = "top=" + a +
               ",left=" + b +
               ",width=" + vidWindowWidth +
               ",height=" + vidWindowHeight +

",toolbar=no,menubar=no,location=no,directories=no,scrollbars=no,resizabl
e=no";
    url="show.php?id_image=" + id_image;
    window.open(url,'',features,true);
  }
//-->
</script>
```

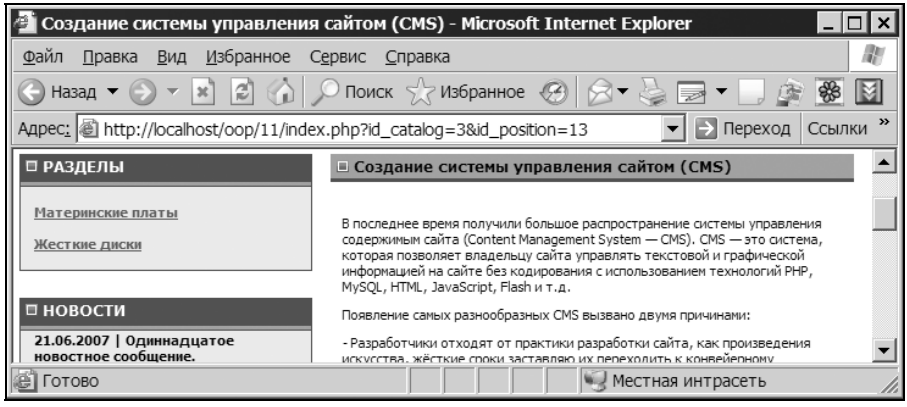


Рис. 11.26. Вывод статьи

JavaScript-скрипт `show_img()` несет ответственность за вывод большого изображения в отдельном окне при помощи скрипта show.php из листинга 11.49.

Листинг 11.49. Отображение увеличенного изображения, show.php

```
<?php
  ////////////////////////////////////////////////////////
  // 2006-2007 (C) IT-студия SoftTime http://www.softtime.ru
  ////////////////////////////////////////////////////////
  // Устанавливаем соединение с базой данных
  require_once("config/config.php");
```

```
  // Предотвращаем SQL-инъекцию
  $_GET['id_image'] = intval($_GET['id_image']);

  $query = "SELECT * FROM $tbl_paragraph_image
            WHERE hide = 'show' AND
                  id_image = $_GET[id_image]";
  $pos = mysql_query($query);
  if($pos)
  {
    if(mysql_num_rows($pos))
    {
      $position = mysql_fetch_array($pos);
    }
    else exit($query);
  }
  else exit(mysql_error());
?>
<html>
<head>
<title>Просмотр фотографии</title>
<meta http-equiv="imagetoolbar" content="no">
<style>
 table{font-size: 12px; font-family: Arial, Helvetica, sans-serif;
background-color: #F3F3F3;}
</style>
</head>
<body marginheight="0"
      marginwidth="0"
      rightmargin="0"
      bottommargin="0"
      leftmargin="0"
      topmargin="0">
<table height="100%"
       cellpadding="0"
       cellspacing="0"
       width="100%"
       border="1">
  <tr>
    <td height="100%" valign="middle" align="center">
     Дождитесь загрузки изображения
     <div  style="position: absolute; top: 0px; left: 0px">
       <img src="<?= $position['big'];?>" border="0">
    </div>
```

```
    </td>
  </tr>
</table>
<div style="position: absolute; z-index: 2; width: 100%; bottom: 5px"
align="center">
<input class=button
       type="submit"
       value="Закрыть"
       onclick="window.close();"></div>
</body>
</html>
```

Заключение

Чтение книги — это всегда выслушивание монолога авторов. Что, в общем-то, не очень хорошо, поскольку быстрая обратная связь в равной мере необходима как читателям, так и пишущим книгу. Чтобы превратить авторский монолог в полноценный диалог с читателем, мы создали на своем сайте форум, на котором можно задать свои вопросы. Адрес форума: **http://www.softtime.ru/forum/**. Авторы искренне надеются, что после того как вы перелистнете эту страницу, мы с вами не расстанемся, а встретимся на нашем форуме.

Успехов вам и всего доброго!

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Предопределенные классы

В PHP имеются готовые классы, которые не требуют объявления и могут использоваться непосредственно. Типичным примером предопределенного класса является класс исключения `Exception` (см. *главу 8*) или набор классов отражений (см. *главу 9*). Предопределенные классы могут быть встроены в ядро или подключаться при помощи одного из расширений.

В этом приложении будут рассмотрены три набора предопределенных классов:

- библиотека mysqli;
- класс `dir`;
- библиотека SPL.

П1.1. Библиотека php_mysqli

Расширение php_mysqli является новым объектно-ориентированным интерфейсом для доступа к СУБД MySQL из PHP-кода и доступен при работе с СУБД MySQL версии выше 4.1. Интерфейс включает в себя классы для доступа к СУБД MySQL, отложенных запросов и класса результирующей таблицы. Библиотека не заменяет собой функции из расширения php_mysql — наряду с объектно-ориентированным расширением доступен процедурный стиль программирования.

Замечание

В отличие от старого расширения `php_mysql`, в новом расширении php_mysqli функции начинаются с префикса `mysqli`, а не с `mysql`.

Главным классом расширения является `mysqli`, при помощи которого осуществляется установка соединения с сервером и выполнения запросов. В лис-

тинге П1.1 приводится пример, в котором устанавливается соединение с сервером MySQL и выводятся сведения о его версии.

Листинг П1.1. Версия MySQL-сервера

```
<?php
  // Объявляем объект класса mysqli, который устанавливает
  // соединение с сервером
  $mysqli = new mysqli("localhost", "root", "", "test");
  if (mysqli_connect_errno())  exit("Ошибка установки соединения");

  // Выводим версию сервера
  echo $mysqli->server_info;

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

Методы класса `mysqli` представлены в табл. П1.1.

Таблица П1.1. *Методы класса* `mysqli`

Метод	Описание
`autocommit(`*mode*`)`	Включает или отключает режим автозавершения транзакции. По умолчанию автозавершение включено, и каждый оператор рассматривается как отдельная транзакция. Отключение режима автозавершения требует явного завершения транзакции при помощи методов `commit()` и `rollback()`. Параметр `mode` принимает значение `true` или `false`
`change_user(`*user*`, `*password*`, `*database*`)`	Изменяет параметры соединения для текущего объекта `mysqli`, устанавливая нового пользователя `user`, его пароль `password` и текущую базу данных `database`
`character_set_name()`	Возвращает текущую кодировку соединения, которую извлекает из системной переменной MySQL — `character_set`
`close()`	Закрывает соединение с сервером MySQL
`commit()`	Подтверждает, что транзакция успешно завершена и все изменения записаны на жесткий диск
`get_client_info()`	Возвращает версию клиентской библиотеки
`kill(`*processid*`)`	Уничтожает процесс с идентификатором `processid`

Таблица П1.1 *(продолжение)*

Метод	Описание
multi_query(*query*)	Выполняет SQL-запросы в строке query, в отличие от метода query() в строке query допускается несколько SQL-запросов, разделенных точкой с запятой
more_results()	Проверяет, имеются ли записи в таблице, полученной в результате выполнения метода multi_query(), и возвращает true, если записи имеются, и false — в противном случае
next_result()	Получает следующую запись из таблицы, возвращенной в результате выполнения метода multi_query()
options(*option*, *value*)	Устанавливает для параметра option значение value
ping()	Проверяет соединение с сервером, и, если оно отсутствует, выполняет повторную попытку соединения
prepare(*query*)	Создает отложенный запрос query, возвращая дескриптор отложенного запроса. При помощи метода execute() отложенного запроса его можно выполнить в любой удобный момент
query(*query*)	Выполняет SQL-запрос в строке query, в отличие от метода multi_query() возможно выполнение только одного запроса
real_connect([*hostname*[, *username*[, *passwd*[, *dbname*[, *port*[, *socket*[, *flags*]]]]]]])	Устанавливает соединение с сервером, если оно не было установлено заранее конструктором класса mysqli, а объект был получен при помощи функции mysqli_init()
mysqli_real_escape_string (*escapestr*)	Экранирует специальные символы для SQL-запроса, который передается методу query() или multi_query()
rollback()	Выполняет откат транзакции, возвращая СУБД в состояние, которое было до начала транзакции
select_db(*dbname*)	Позволяет выбрать текущую базу данных dbname
set_charset(*charset*)	Устанавливает текущую кодировку соединения charset
ssl_set(*key*, *cert*, *ca*, *capath*, *cipher*)	Устанавливает безопасное SSL-соединение с MySQL-сервером
stat()	Возвращает строку, содержащую информацию о состоянии MySQL-сервера

Таблица П1.1 *(окончание)*

Метод	Описание
stmt_init()	Возвращает объект отложенного запроса; является альтернативой методу prepare()
store_result()	Возвращает дескриптор результирующей таблицы последнего запроса
use_result()	Инициирует результирующую таблицу последнего запроса
thread_safe()	Возвращает true, если клиентская библиотека скомпилирована в безопасном режиме, и false — в противном случае

Помимо методов, класс mysqli имеет ряд открытых членов, список которых представлен в табл. П1.2.

Таблица П1.2. *Члены класса* mysqli

Член	Описание
affected_rows	Возвращает количество записей, обработанных операторами DELETE, UPDATE и REPLACE
client_info	Возвращает версию клиентской библиотеки в виде строки
client_version	Возвращает версию клиентской библиотеки в виде числа
errno	Возвращает номер ошибки для последней операции
error	Возвращает текстовое сообщение об ошибке для последней операции
field_count	Возвращает количество полей результирующей таблицы из последнего запроса
host_info	Возвращает адрес MySQL-сервера и тип соединения
info	Возвращает информацию о последнем запросе, количество затронутых столбцов, предупреждений и т. п.
insert_id	Возвращает первичный ключ, сгенерированный по механизму AUTO_INCREMENT при последнем INSERT-запросе
protocol_version	Возвращает версию протокола взаимодействия клиента и MySQL-сервера
server_info	Возвращает версию MySQL-сервера в виде строки
server_version	Возвращает версию MySQL-сервера в виде числа
sqlstate	Возвращает номер и текстовое сообщение о последней ошибке
thread_id	Возвращает уникальный идентификатор текущего потока

***Таблица П1.2** (окончание)*

Член	Описание
`warning_count`	Определяет количество предупреждений, которое возвратил MySQL-сервер для последнего SQL-запроса

Второй класс — `mysqli_stmt` — применяется для формирования и обработки отложенных запросов. В листинге П1.2 приводится типичный вариант использования отложенного запроса.

Листинг П1.2. Использование класса `mysqli_stmt`

```
<?php
  // Устанавливаем соединение с сервером MySQL
  $mysqli = new mysqli("localhost", "root", "", "test");
  if (mysqli_connect_errno())  exit("Ошибка установки соединения");

  // Формируем подготовленный запрос
  $query = "SELECT name, family FROM users ORDER BY name";
  if ($stmt = $mysqli->prepare($query))
  {
    // Выполняем запрос
    $stmt->execute();

    // Связываем переменные $name и $family со столбцами
    // name и family таблицы users
    $stmt->bind_result('ss', $name, $family);

    // Сохраняем результирующую таблицу
    $stmt->store_result();

    // Перемещаемся на вторую строку (нумерация начинается с 0)
    $stmt->data_seek(1);

    // Помещаем данные в переменные $name и $family
    $stmt->fetch();

    // Выводим результат
    echo "Имя : $name<br>";
    echo "Фамилия : $family<br>";

    // Освобождаем дескриптор подготовленного запроса
    $stmt->close();
  }
```

```
  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

В табл. П1.3 приводится список методов класса `mysqli_stmt`.

Таблица П1.3. *Методы класса* `mysqli_stmt`

Метод	Описание
`bind_param(`*types*`, &`*var1* `[, &...])`	Связывает переменные, передаваемые в качестве аргумента, с переменными в SQL-запросе, которые обозначаются символом `?`. Аргумент `types` задает тип данных и может принимать значения `'i'`, `'d'`, `'s'` и `'b'` для целочисленных, вещественных, строковых и BLOB-полей
`bind_result(&`*var1* `[, &...] )`	Связывает переменные, передаваемые в качестве аргумента, со столбцами результирующей таблицы
`close()`	Освобождает дескриптор отложенного запроса
`data_seek(`*offset*`)`	Перемещается на строку `offset` в результирующей таблице (нумерация строк начинается с 0)
`execute()`	Выполняет отложенный запрос
`fetch()`	Помещает данные из текущей строки результирующей таблицы в связанные переменные, назначенные ранее при помощи метода `bind_result()`
`free_result()`	Освобождает память, выделенную под результирующую таблицу; после вызова метода данные результирующей таблицы становятся недоступны
`result_metadata()`	Возвращает метаданные отложенного запроса
`prepare(`*query*`)`	Создает отложенный запрос
`send_long_data(`*param_nr*`,` *data*`)`	Параметру `param_nr` присваивается значение `data`. Функция используется для передачи объемных BLOB-значений, которые превышают `max_allowed_packet`. Параметр, назначенный данной функцией, будет передан на сервер в несколько приемов
`reset()`	Сбрасывает отложенный запрос
`store_result()`	Сохраняет результирующую таблицу

Помимо методов, класс `mysqli_stmt` имеет ряд открытых членов, список которых представлен в табл. П1.4.

Последний класс `mysqli_result` предназначен для работы с результирующей таблицей. Методы класса представлены в табл. П1.5.

Таблица П1.4. *Члены класса* `mysqli_stmt`

Член	Описание
`affected_rows`	Возвращает количество записей, обработанных операторами `DELETE`, `UPDATE` и `REPLACE`
`errno`	Возвращает номер ошибки для последней операции
`error`	Возвращает текстовое сообщение об ошибке для последней операции
`field_count`	Возвращает количество полей результирующей таблицы из последнего запроса
`id`	Возвращает идентификатор запроса
`insert_id`	Возвращает первичный ключ, сгенерированный по механизму `AUTO_INCREMENT` при последнем `INSERT`-запросе
`num_rows`	Возвращает количество записей в результирующей таблице
`param_count`	Возвращает количество параметров в отложенном запросе
`sqlstate`	Возвращает номер и текстовое сообщение о последней ошибке

Таблица П1.5. *Методы класса* `mysqli_result`

Метод	Описание
`close()`	Освобождает память, выделенную под результирующую таблицу
`data_seek(`*`offset`*`)`	Отыскивает в результирующей таблице строку с номером `offset` (нумерация строк начинается с 0)
`fetch_field()`	Возвращает информацию о столбцах результирующей таблицы в виде объекта
`fetch_fields()`	Возвращает массив объектов с информацией о столбцах результирующей таблицы
`fetch_field_direct (`*`fieldnr`*`)`	Возвращает объект с информацией о столбце с номером `fieldnr`. Нумерация столбцов в результирующей таблице начинается с 0
`fetch_array(`*`resulttype`*`)`	Возвращает текущую запись результирующей таблицы в виде массива, элементы которого соответствуют полям записи. Параметр `resulttype` определяет тип возвращаемого массива: `MYSQLI_NUM` — числовой массив, `MYSQLI_ASSOC` — ассоциативный массив, `MYSQLI_BOTH` — возвращает и числовой, и ассоциативный массив
`fetch_assoc()`	Возвращает текущую запись результирующей таблицы в виде ассоциативного массива. Элементы массива соответствуют полям записи

Таблица П1.5 *(окончание)*

Метод	Описание
`fetch_object()`	Возвращает текущую запись результирующей таблицы в виде объекта
`fetch_row()`	Возвращает текущую запись в виде обычного массива
`free_result()`	Освобождает память, выделенную под результирующую таблицу
`field_seek(`*`fieldnr`*`)`	Устанавливает курсор результирующей таблицы на запись номер `fieldnr` (нумерация записей начинается с 0)

Помимо методов, класс `mysqli_result` имеет ряд открытых членов, список которых представлен в табл. П1.6.

Таблица П1.6. *Члены класса* `mysqli_result`

Член	Описание
`current_field`	Возвращает номер текущего столбца (нумерация столбцов начинается с 0) при извлечении информации методом `fetch_field()`
`field_count`	Возвращает количество столбцов в результирующей таблице
`lengths`	Возвращает количество символов в текущем столбце при извлечении информации методом `fetch_field()`
`num_rows`	Возвращает количество записей в результирующей таблице
`type`	Возвращает одну из констант: `MYSQLI_STORE_RESULT` или `MYSQLI_USE_RESULT`, в зависимости от того, сохранен результат или нет

В листинге П1.3 представлен файл config.php, устанавливающий соединение с MySQL-сервером. Такой файл включается при помощи конструкции `require_once` во все скрипты, где требуется наличие класса объекта класса `mysqli`.

Листинг П1.3. Установка соединения с СУБД MySQL config.php

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
```

```
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с сервером MySQL
  $mysqli = new mysqli($dblocation, $dbuser, $dbpasswd, $dbname);
  if (mysqli_connect_errno()) exit("Ошибка установки соединения");

  // Устанавливаем кодировку соединения. Следует выбрать ту кодировку,
  // в которой данные будут отправляться MySQL-серверу
  $mysqli->query("SET NAMES 'cp1251'");
?>
```

Объявление объекта класса `mysqli` приводит к установке соединения с сервером, расположенным по адресу `$dblocation`, и выбору базы данных `$dbname` от имени пользователя `$dbuser`, имеющего пароль `$dbpasswd`.

MySQL-серверу отправляется запрос `"SET NAMES 'cp1251'"`. Данный запрос сообщает, что все данные будут передаваться на сервер кодировке Windows-1251, и MySQL-сервер, в свою очередь, в какой бы кодировке не хранились данные в таблицах, должен предоставлять результат в кодировке Windows-1251.

В следующих разделах будут рассмотрены наиболее типичные задачи, которые возникают при взаимодействии PHP-скриптов и СУБД MySQL.

П1.1.1. Создание базы данных

Создать новую базу данных можно путем выполнения оператора `CREATE DATABASE`. В листинге П1.4 представлен скрипт, создающий базу данных `dbase`.

Листинг П1.4. Создание базы данных

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем и выполняем запрос
  $query = "CREATE DATABASE dbase";
  if(!$mysqli->query($query))
  {
    exit("База данных не создана ".$mysqli->error);
  }
?>
```

Метод query() объекта класса mysqli в случае успешного завершения возвращает либо TRUE, если запрос не возвращает результирующей таблицы, либо объект результирующей таблицы, который в контексте оператора if также воспринимается как истинное значение (TRUE). Если запрос заканчивается неудачей, возвращается FALSE.

Следует обязательно проверить, не возвращает ли СУБД MySQL ошибку выполнения последнего запроса, поскольку интерпретатор PHP никак не сигнализирует об ошибках синтаксиса MySQL: PHP-скрипт и SQL-запросы выполняются в разных потоках и PHP-интерпретатор не имеет сведений об успешности или неудаче выполнения SQL-запроса.

П1.1.2. Создание и заполнение таблицы

Процесс создания и заполнения таблицы схож с рассмотренным в предыдущем разделе процессом создания базы данных. Отличие заключается лишь в том, что потребуется выполнить несколько SQL-запросов.

Создадим таблицу catalogs, состоящую из трех полей:

- id_catalog — первичный ключ таблицы, снабженный атрибутом AUTO_INCREMENT;
- name — название элемента каталога;
- putdate — дата последней модификации каталога.

Для выполнения нескольких запросов удобно воспользоваться методом multi_query(), который позволяет за один раз выполнить сразу несколько запросов, отделенных друг от друга точкой с запятой (листинг П1.5).

Листинг П1.5. Создание таблицы catalogs

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем запросы
  $query = "CREATE TABLE catalogs (
            id_catalog int(11) NOT NULL AUTO_INCREMENT,
            name tinytext NOT NULL,
            putdate datetime NOT NULL,
            PRIMARY KEY  (id_catalog)
            ) ENGINE=MyISAM;
            INSERT INTO catalogs
            VALUES (NULL, 'Процессоры', '2007-01-10 12:47:00');
            INSERT INTO catalogs
```

```
            VALUES (NULL, 'Материнские платы', '2006-12-28 18:32:41');
            INSERT INTO catalogs
            VALUES (NULL, 'Видеоадаптеры', '2007-01-10 16:48:05');
            INSERT INTO catalogs
            VALUES (NULL, 'Жесткие диски', '2007-01-05 20:01:58');
            INSERT INTO catalogs
            VALUES (NULL, 'Оперативная память', '2006-12-20 10:00:13')";
  if(!$mysqli->multi_query($query))
  {
    exit("Таблицу не удалось развернуть ".$mysqli->error);
  }
?>
```

Замечание

В процедурной библиотеке php_mysql отсутствует аналог, позволяющий выполнить одновременно несколько SQL-запросов — обычно SQL-дамп разбивается на отдельные запросы, которые выполняются в цикле.

П1.1.3. Заполнение связанных таблиц

Еще один интересный случай представляет собой заполнение связанных таблиц. Пусть имеется таблица `news`, предназначенная для хранения новостных сообщений и состоящая из трех полей:

- `id_news` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — название новостной позиции;
- `putdate` — дата размещения новости.

С таблицей `news` связана таблица `news_contents`, предназначенная для хранения текста новости и также состоящая из трех полей:

- `id_content` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `content` — содержимое новостного сообщения;
- `id_news` — внешний ключ, содержащий значения полей `id_news` из таблицы `news`, связывая новостной блок и текст новости.

В листинге П1.6 представлены операторы `CREATE TABLE`, которые создают таблицы `news` и `news_contents`.

Листинг П1.6. Таблицы news и news_contents

```
CREATE TABLE news (
  id_news INT(11) NOT NULL AUTO_INCREMENT,
```

```
  name TINYTEXT NOT NULL,
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_news)
);
CREATE TABLE news_contents (
  id_content INT(11) NOT NULL AUTO_INCREMENT,
  content TEXT NOT NULL,
  id_news INT(11) NOT NULL,
  PRIMARY KEY (id_content)
);
```

На рис. П1.1 представлена HTML-форма, которая позволяет добавить название и текст новостного сообщения в базу данных.

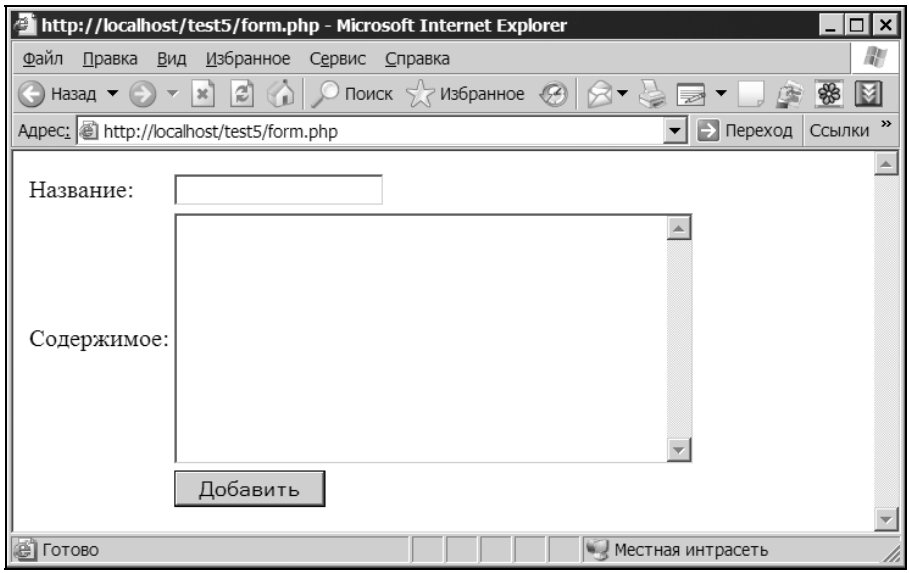


Рис. П1.1. HTML-форма для добавления новостного сообщения в базу данных

HTML-код, воссоздающий форму, представленную на рис. П1.1, приведен в листинге П1.7.

Листинг П1.7. HTML-форма для добавления новостного сообщения в базу данных

```
<table>
<form action=addnews.php method=POST>
  <tr>
    <td>Название:</td>
```

```
    <td><input type=text name=name></td>
  </tr>
  <tr>
    <td>Содержимое:</td>
    <td><textarea name=content rows=10 cols=40></textarea></td>
  </tr>
  <tr>
    <td></td>
    <td><input type=submit value=Добавить></td>
  </tr>
</form>
</table>
```

Как видно из листинга П1.7, обработчиком HTML-формы назначен файл addnews.php. В листинге П1.8 представлен обработчик HTML-формы, который осуществляет вставку новостного сообщения в таблицы `news` и `news_contents`.

Листинг П1.8. Добавление новостного сообщения в базу данных

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Проверяем, заполнены ли поля HTML-формы
  if(empty($_POST['name'])) exit('Не заполнено поле "Название"');
  if(empty($_POST['content'])) exit('Не заполнено поле "Содержимое"');

  // Экранируем специальные символы
  if (!get_magic_quotes_gpc())
  {
    $_POST['name'] = $mysqli->real_escape_string($_POST['name']);
    $_POST['content'] = $mysqli->real_escape_string($_POST['content']);
  }

  // Добавляем новостное сообщение в таблицу news
  $query = "INSERT INTO news VALUES (NULL, '$_POST[name]', NOW())";
  if(!$mysqli->query($query))
  {
    exit("Ошибка добавления новостного сообщения".$mysqli->error);
  }

  // Получаем только что сгенерированный идентификатор id_news
  $id_news = $mysqli->insert_id;
```

```
  // Вставляем содержимое новостного сообщения в таблицу news_contents.
  // Формируем запросы
  $query = "INSERT INTO news_contents
            VALUES (NULL, '$_POST[content]', $id_news)";
  if(!$mysqli->query($query))
  {
    exit("Ошибка добавления новостного сообщения ".$mysqli->error);
  }

  // Осуществляем переадресацию на главную страницу
  header("Location: form.php");
?>
```

Так как данные передаются методом `POST`, то содержимое полей `name` и `content` попадает в элементы суперглобального массива `$_POST['name']` и `$_POST['name']` соответственно. В начале обработчика производится проверка правильности заполнения полей формы — если хоть одно из полей остается незаполненным, процесс добавления информации приостанавливается с выдачей соответствующего предупреждения. Наличие символов в строке можно проверить при помощи функции `empty()`, которая возвращает `TRUE`, если строка пустая, и `FALSE`, если строка содержит хотя бы один символ.

После этого при помощи метода `real_escape_string()` экранируются специальные символы в строковых переменных. Если строка содержит символ `'`, то синтаксис SQL-оператора может быть нарушен. Такой символ следует заменить на последовательность `\'`. Именно для этого и предназначен метод `real_escape_string()` класса `mysqli`.

Замечание

На ряде серверов включен режим "магических кавычек". В этом режиме экранирование специальных символов в `POST`-, `GET`- и `COOKIE`-данных осуществляется автоматически, и повторно экранировать данные не требуется. Выяснить, включен режим "магических кавычек" или нет, можно при помощи функции `get_magic_quotes_gpc()`, которая возвращает `TRUE`, если режим включен, и `FALSE` — в противном случае.

Основной особенностью добавления данных в связанные таблицы заключается в том, что для добавления записи во вторую таблицу `news_content` необходимо знать первичный ключ `id_news`, который был сгенерирован по механизму `AUTO_INCREMENT` для только что созданной записи в таблице `news`. Получить его можно, обратившись к члену `insert_id` класса `mysqli`.

После того как записи добавлены, следует вернуться на главную страницу HTML-формы или страницу, где выводятся новостные сообщения. Для этого

браузеру оправляется HTTP-заголовок с адресом страницы, на которую следует осуществить переход.

П1.1.4. Вывод данных

Еще одной часто встречающейся задачей является вывод содержимого таблицы из базы данных. Пусть имеется таблица `catalogs`, содержимое которой представлено в листинге П1.9.

Листинг П1.9. Таблица `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (NULL, 'Процессоры'),
                            (NULL, 'Материнские платы'),
                            (NULL, 'Видеоадаптеры'),
                            (NULL, 'Жесткие диски'),
                            (NULL, 'Оперативная память');
```

В листинге П1.10 представлен скрипт, выводящий содержимое таблицы в окно браузера.

Листинг П1.10. Вывод содержимого таблицы `catalogs`

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";
  $cat = $mysqli->query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");

  // Если имеется хотя бы одна запись, выводим список
  if($cat->num_rows)
  {
    while($catalog = $cat->fetch_array())
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Как видно из листинга П1.10, метод query() объекта mysqli возвращает объект результирующей таблицы $cat. Член num_rows объекта $cat возвращает количество записей в результирующей таблице. Если это значение больше нуля, содержимое таблицы извлекается при помощи метода fetch_array(). За один подход функция извлекает из результирующей таблицы одну запись, которая представляется в виде ассоциативного массива $catalog. В качестве ключей массива выступают имена столбцов таблицы catalogs, а в качестве значений — элементы результирующей таблицы. Повторный вызов метода fetch_array() приводит к извлечению следующего столбца и т. д. Поэтому вызов функции в цикле while() приводит к последовательному извлечению всех строк результирующей таблицы.

П1.1.5. Повторное чтение результирующей таблицы

Следует отметить, что вызов метода fetch_array() объекта класса mysqli_stmt в цикле while приводит к тому, что курсор останавливается на последней записи результирующей таблицы и повторная попытка извлечения записей заканчивается неудачей. Для повторного чтения результирующей таблицы без отправки серверу MySQL SELECT-запроса необходимо вернуть курсор в начало результирующей таблицы. Для этого предназначен метода data_seek(), который имеет следующий синтаксис:

```
bool mysql_data_seek(row_number)
```

Метод устанавливает внутренний курсор на строку row_number (нумерация строк начинается с 0). В случае успеха метод возвращает true, в случае неудачи — false.

Вернемся к задаче вывода содержимого таблицы catalogs, которая была рассмотрена в предыдущем разделе. Пусть требуется подсчитать количество символов в выводимых строках, после чего вывести содержимое результирующей таблицы (листинг П1.11).

Листинг П1.11. Повторное считывание результирующей таблицы

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";

  $cat = $mysqli->query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");
```

```
  // Если имеется хотя бы одна запись, выводим список
  if($cat->num_rows)
  {
    // Подсчитываем количество символов
    // во всех строках таблицы
    $count = 0;
    while($catalog = $cat->fetch_array())
    {
      $count += strlen($catalog['name']);
    }
    echo "Общая длина всех строк ".$count."<br>";

    // Устанавливаем внутренний курсор
    // в начало результирующей таблицы
    $cat->data_seek(0);

    // Повторно считываем результирующую таблицу
    while($catalog = $cat->fetch_array())
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Как видно из листинга П1.11, для того чтобы осуществить повторное считывание результирующей таблицы, необходимо установить курсор на нулевую запись при помощи метода `data_seek()`. Результатом работы скрипта из листинга П1.11 будут следующие строки:

```
Общая длина всех строк 71
Процессоры
Материнские платы
Видеоадаптеры
Жесткие диски
Оперативная память
```

П1.1.6. Количество строк в таблице

В предыдущих разделах для определения количества строк в результирующей таблице использовалось свойство `num_rows` объекта результирующей таблицы. Если `SELECT`-выборка не имеет ограничений и таблица выбирается полностью, можно использовать данные способы для оценки количества строк в таблице. Однако зачастую на одну страницу выводится лишь часть

таблицы, в то время как для организации навигации требуется знать общее количество строк во всей таблице. Если таблица объемная, полная выборка таблицы не приемлема ни по скорости, ни по объему памяти, которая потребуется для ее хранения. В этом случае разумнее прибегнуть к MySQL-функции `COUNT(*)`, которая подсчитывает количество строк в таблице. В листинге П1.12 представлен скрипт, подсчитывающий количество строк в таблице `catalogs` (см. листинг П1.9).

Листинг П1.12. Подсчет количества строк в таблице

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT COUNT(*) AS total FROM catalogs";

  $cat = $mysqli->query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");

  $result = $cat->fetch_object();
  echo $result->total;
?>
```

В листинге П1.12 при помощи метода `fetch_object()` строка результирующей таблицы возвращается в виде объекта, имена членов которого совпадают с именами полей в результирующей таблице. Для удобства единственный столбец в результирующей таблице переименовывается в `total` при помощи оператора `AS`.

П1.1.7. Удаление данных

При удалении данных часто прибегают к флажкам — можно отметить сразу несколько позиций, которые подвергаются удалению (рис. П1.2).

В листинге П1.13 представлен скрипт, позволяющий удалять сразу несколько записей, которые пользователь отмечает флажками.

Листинг П1.13. Удаление нескольких записей таблицы

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
```

```
// Если суперглобальный массив $_POST не пуст,
// производим обработку запроса
if(!empty($_POST))
{
  // Для удаления позиций необходимо сформировать запрос вида
  // DELETE FROM catalogs WHERE id_catalog IN (4, 6, 8, ..., 20),
  // где цифры в скобках являются элементами
  // массива $_POST['catalog'][]
  $temp = array();
  foreach($_POST['catalog'] as $id_catalog)
  {
    // Проверяем, является ли переменная $id_catalog числом
    if(preg_match("|^[\d]+$|",$id_catalog))
    {
      $temp[] = $id_catalog;
    }
  }
  // Формируем и выполняем запрос на удаление
  // нескольких элементов
  $query = "DELETE FROM catalogs
            WHERE id_catalog IN (".implode(",",$temp).")";
  if($mysqli->query($query))
  {
    echo "<HTML><HEAD>
        <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
          </HEAD></HTML>";
    exit();
  }
}

// Выводим список элементов каталога
$query = "SELECT * FROM catalogs
          ORDER BY id_catalog";
$cat = $mysqli->query($query);
if(!$cat) exit($mysqli->error);
// Если имеется хотя бы одна позиция — выводим ее
if($cat->num_rows)
{
  echo "<form method=post>";
  echo "<table border=1>
          <tr>
            <td> </td>
            <td>Номер</td>
```

```
            <td>Название</td>
          </tr>";
    $i = 0;
    while($catalog = $cat->fetch_array())
    {
      echo "<tr>
            <td><input type=checkbox name=catalog[]
                       value=$catalog[id_catalog]></td>
            <td>$catalog[id_catalog]</td>
            <td>$catalog[name]</td>
          </tr>";
      $i++;
    }
    echo "</table>";
    echo "</br><input type=submit name=send value=Удалить>";
    echo "</form>";
  }
?>
```

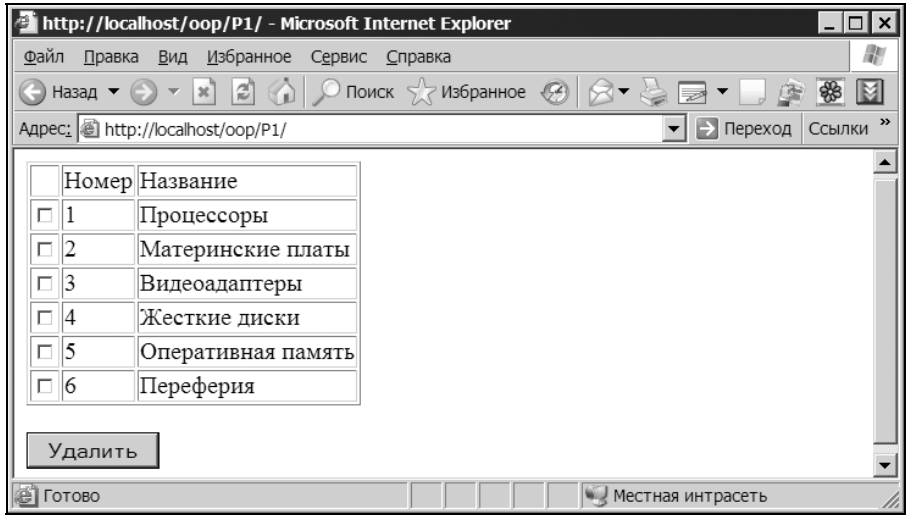


Рис. П1.2. Web-интерфейс для удаления нескольких записей таблицы

П1.1.8. Сортировка

При выводе содержимого таблицы для удобства восприятия информации часто требуется организовывать сортировку по столбцам. Выведем содержимое таблицы `catalogs` (см. листинг П1.9) в таблицу, заголовки которой будут

представлять собой гиперссылки, переход по которым осуществлял бы сортировку выбранного столбца. Повторный переход по гиперссылке должен приводить к сортировке данных в обратном порядке (рис. П1.3).

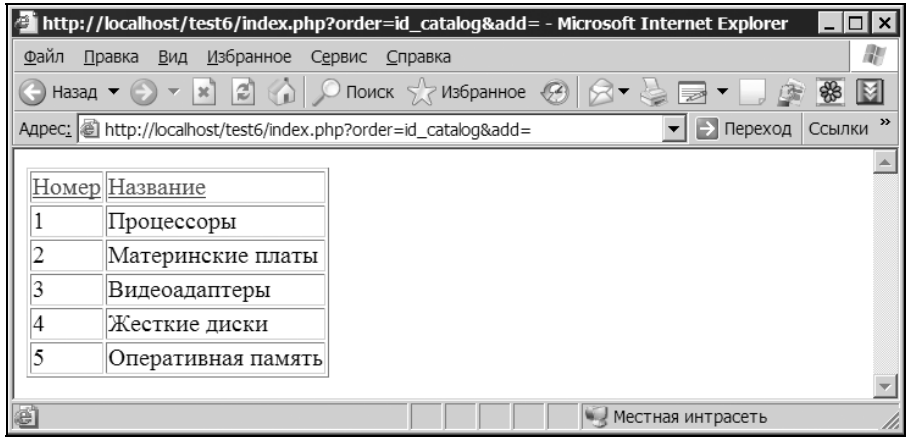


Рис. П1.3. Таблица с возможностью сортировки столбцов

Сортировка результатов SQL-запроса производится при помощи конструкции `ORDER BY`, после которой указывается имя столбца, подвергающегося сортировке. Чтобы сортировка выполнялась в обратном порядке, следует указать необязательное ключевое слово `DESC`. Таким образом, задача сводится к формированию динамического SQL-запроса, в конструкцию `ORDER BY` которого подставлялись бы имена выбранных столбцов. Для этого будем использовать запрос вида:

```
SELECT * FROM catalogs
         ORDER BY $order $desc
```

Переменная `$order` будет содержать имя столбца, а `$desc` — принимать либо пустое значение, либо ключевое слово `DESC` в зависимости от того, какой вид сортировки выбран (листинг П1.14).

Листинг П1.14. Сортировка столбцов

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем параметры, переданные скрипту
  $order = "name";
  if($_GET['order'] == 'name')  $order = "name";
```

```
  if($_GET['order'] == 'id_catalog') $order = "id_catalog";
  if($_GET['add']   == 'desc')
  {
    $desc = "DESC";
    $add  = "";
  }
  else
  {
    $desc = "";
    $add  = "desc";
  }

  // Выводим элементы каталога
  $query = "SELECT * FROM catalogs
            ORDER BY $order $desc";
  $cat = $mysqli->query($query);
  if(!$cat) exit($mysqli->error);
  if($cat->num_rows)
  {
    echo "<table border=1>
            <tr>
  <td><a href=$_SERVER[PHP_SELF]?order=id_catalog&add=$add>Номер</a></td>
  <td><a href=$_SERVER[PHP_SELF]?order=name&add=$add>Название</a></td>
            </tr>";
    while($catalog = $cat->fetch_array())
    {
      echo "<tr>
              <td>$catalog[id_catalog]</td>
              <td>$catalog[name]</td>
            </tr>";
    }
    echo "</table>";
  }
?>
```

В начале скрипта `GET`-параметр `order` подвергается проверке, и если его значение равно одному из столбцов, присутствующих в таблице, в переменную `$order` помещается имя столбца. В противном случае переменная `$order` принимает значение по умолчанию, равное `"name"`. Такая громоздкая проверка необходима для того, чтобы избежать SQL-инъекции. При включенном режиме `register_globals` интерпретатор PHP автоматически создает для `GET`-параметров переменные. То есть при обращении к скрипту появляется возможность передать через параметр `order` SQL-инъекцию. Поэтому при работе с методами `GET`, `POST` или `COOKIE` необходимо всегда явно проверять перемен-

ные, которые передаются от одной страницы к другой, т. к. в этих методах передачи одним из посредников выступает машина клиента, где передаваемые данные могут подвергнуться подделке.

П1.1.9. Параметризация SQL-запросов

В листинге П1.2 приводится пример работы с классом `mysqli_stmt`, реализующим отложенные запросы. Результирующая таблица может быть импортирована непосредственно в переменные PHP. Другой интересной особенностью класса `mysqli_stmt` является создание параметризованных запросов. Неизвестные переменные в таких запросах обозначаются при помощи знака вопроса, а значения могут подставлены при помощи метода `bind_param()`. В листинге П1.15 демонстрируется создание отложенного `INSERT`-запроса вставки данных в таблицу `catalogs`.

Листинг П1.15. Параметризация SQL-запроса

```
<?php
  // Устанавливаем соединение с сервером MySQL
  require_once("config.php");

  // Создаем отложенный запрос
  $stmt =  $mysqli->stmt_init();

  // Формируем подготовленный запрос
  $query = "INSERT INTO catalogs
            VALUES (NULL, ?)";
  if ($stmt = $mysqli->prepare($query))
  {
    // Подставляем два строковых параметра в SQL-запрос
    // $query
    $stmt->bind_param('s', $name);
    $name = "Периферия";

    // Выполняем запрос
    if(!$stmt->execute()) exit($stmt->error);

    // Освобождаем дескриптор подготовленного запроса
    $stmt->close();
  }

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

В качестве параметра выступает строка "Периферия", которая передается через переменную $name. Переменная $name передается в качестве второго параметра метода bind_param(). В качестве первого параметра передается строка, каждый символ которой обозначает тип последующих параметров:

- i — для целочисленного типа;
- d — для типа с плавающей точкой;
- s — для строки;
- b — для поля типа BLOB.

Таким образом, для запроса с двумя параметрами получаем:

```
INSERT INTO catalogs VALUES (?, ?)
```

Вызов метода bind_param() мог бы выглядеть следующим образом:

```
$stmt->bind_param('is', $id_catalog, $name);
```

Переменная $id_catalog при этом могла бы принимать целочисленное значение, например, 0, а переменная $name — строку с именем каталога. Важно помнить, что параметры передаются по ссылке, а не по значению, т. е. следующий вызов метода bind_param() вызовет ошибку.

```
$stmt->bind_param('is', 0, "Переменная");
```

Метод принимает только переменные, значения которых используются в момент вызова метода execute(). Это позволяет определять параметры в переменных даже после вызова метода bind_param().

П1.2. Класс *dir*

Класс dir является предопределенным классом, позволяющим получать доступ к директории. Объявление объекта класса dir позволяет открыть директорию, а его методы — осуществить чтение содержимого директории, заменяя набор функций opendir(), rewinddir(), readdir() и closedir(). Методы и члены класса dir перечислены в табл. П1.7.

Таблица П1.7. *Методы и члены класса* dir

Элемент класса	Описание
path	Путь к директории
handle	Дескриптор открытой директории
read()	Читает очередной элемент директории, передвигая указатель каталога на одну позицию

Таблица П1.7 *(окончание)*

Элемент класса	Описание
`rewind()`	Сбрасывает указатель директории в исходное состояние; применяется, когда в результате чтения директории в цикле указателя устанавливается на конец директории и его следует переместить в начало для повторного чтения
`close()`	Закрывает директорию

Объявление объекта класса `dir` приводит к автоматическому открытию директории, после чего становится доступно использование методов классов. Метод `read()` за один раз читает один элемент директории, перемещая указатель каталога на одну позицию. Последовательный вызов метода `read()` позволяет обойти таким образом всю директорию до конца. В листинге П1.16 приводится скрипт, который выводит содержимое директории `oop`.

Листинг П1.16. Чтение содержимого каталога

```
<?php
  // Открываем директорию
  $cat = dir("oop");
  // Читаем содержимое директории
  while(($file = $cat->read()) !== false)
  {
    echo $file."<br>";
  }
  // Закрываем каталог
  $cat->close();
?>
```

Важно подчеркнуть, что дескриптор открытой директории `$cat->handle` полностью совместим с классическими функциями для работы с директорией. Скрипт из листинга П1.17 по результату полностью эквивалентен скрипту из листинга П1.16.

Листинг П1.17. Альтернативный способ чтения содержимого директории

```
<?php
  // Открываем каталог
  $cat = dir("oop");
  // Читаем содержимое директории
  while(($file = readdir($cat->handle)) !== false)
```

```
  {
    echo $file."<br>";
  }
  // Закрываем директорию
  closedir($cat->handle);
?>
```

Если в этом случае необходимо установить указатель директории на ее начало для повторного чтения, следует вызывать метод `rewind()`. В листинге П1.18 приводится скрипт, который подсчитывает количество файлов и поддиректорий, после чего осуществляет повторное чтение директории для вывода списка его элементов.

Замечание

Из количества поддиректорий вычитается цифра 2, чтобы предотвратить учет двух скрытых служебных поддиректорий: "." — текущая директория и ".." — родительская директория.

Листинг П1.18. Использование метода `rewind()`

```
<?php
  // Открываем директорию
  $dirname = "oop/";
  $cat = dir($dirname);

  // Устанавливаем счетчики файлов и поддиректорий
  // в нулевое значение
  $file_count = 0;
  $dir_count = 0;

  // Подсчитываем количество файлов и поддиректорий
  while(($file = $cat->read()) !== false)
  {
    if(is_file($dirname.$file)) $file_count++;
    else $dir_count++;
  }
  // Не учитываем
  $dir_count = $dir_count - 2;
  // Выводим количество файлов и поддиректорий
  echo "Директория $dirname содержит $file_count файлов
        и $dir_count поддиректорий<br>";

  // Устанавливаем указатель директории в начало
  $cat->rewind();
```

```
  // Читаем содержимое директории
  while(($file = $cat->read()) !== false)
  {
    if($file != "." && $file != "..")
    {
      echo $file."<br>";
    }
  }
  // Закрываем директорию
  $cat->close();
?>
```

П1.3. Библиотека SPL

Библиотека SPL — это сокращение от Standard PHP Library (Стандартная Библиотека PHP"). Программистов, знакомых с C++, придется разочаровать: несмотря на то, что данная библиотека претендует на лавры STL из стандартной библиотеки C++, до законченного варианта SPL еще очень далеко. Библиотека достаточно абстрактна, не содержит алгоритмов и, по сути, поддерживает только итераторы.

Замечание

Слово "стандартный" в компьютерной области имеет огромное значение. Создано огромное количество библиотек, как платформонезависимых, так и ориентирующихся на конкретную аппаратную или операционную реализацию. Прилагательное "стандартный" означает, что любая реализация компилятора или интерпретатора обязана содержать "стандартную" библиотеку, а любой код, созданный с использованием исключительно стандартных средств, может быть откомпилирован под любой платформой, будь то Windows, один из вариантов Linux, MacOS или любая другая операционная система. PHP не стандартизирован — его выпускает одна группа разработчиков, что обеспечивает ему высокую переносимость. Однако международной стандартизации язык не подвергался, поэтому лейбл "стандартный" является скорее маркетинговым ходом разработчика библиотеки.

Библиотека STL из стандартной библиотеки C++ имеет как набор предопределенных контейнеров, итераторов, способных работать с контейнерами, и алгоритмов, оперирующих итераторами. Разработчикам остается лишь заполнять контейнеры и применять к ним алгоритмы поиска, сортировки, объединения, вычитания и т. п. Появление такой библиотеки вызвано тем, что C++ является достаточно суровым языком программирования — сложен и очень требователен к программистам. Например, отсутствуют массивы, как и строки в классическом смысле этого слова: имеются лишь участки памяти с условными границами. Об ассоциативных массивах и регулярных выражени-

ях вообще речи не идет. Программист должен постоянно помнить о необходимости выделения и освобождения памяти. Язык является строго типизированным и, как следствие, поддерживает шаблоны функций и классов, что еще более его усложняет. Поэтому вскоре возникла идея о создании универсальной библиотеки, которая возьмет на себя рутинные, но часто встречающиеся операции.

PHP является интерпретируемым языком программирования и обладает широким набором возможностей, однако при его использовании возникает ряд проблем: отслеживание памяти, строгая типизация, шаблоны и ассоциативные массивы отсутствуют или решены на уровне языка программирования. К достоинствам языка можно отнести огромный массив функций, а простота инициализации массивов и отсутствие необходимости выделения под них памяти вообще сводит большинство задач к созданию массивов.

Все это приводит к тому, что библиотека SPL является несколько искусственным образованием. Если STL является стандартом де-факто в мире C++, то о библиотеке SPL в PHP практически никто не слышал.

Замечание

Данный раздел является введением в библиотеку SPL и не претендует на полноту (тем более, что библиотека находится в стадии разработки). Здесь будут описаны лишь часть классов и интерфейсов, поддерживаемых библиотекой SPL. За детальным описанием библиотеки следует обратиться к официальной документации.

П1.3.1. Итераторы

Итератор — это объект, позволяющий обходить коллекцию способом, не зависящим от внутреннего устройства коллекции. Независимо от того, какую коллекцию обходит итератор, он ведет себя одинако и поддерживает одни и те же методы.

Это позволяет построить с использованием итераторов универсальные алгоритмы (например, сортировки). Далее для каждой новой коллекции потребуется только создать итераторы, алгоритм может быть к нему применен автоматически и его не потребуется создавать заново. Как уже упоминалось выше, алгоритмы в SPL как таковые отсутствуют, а все операции с итераторами сводятся к тому, что появляется возможность использовать итератор в цикле `foreach`. Зачем это нужно?

Многие операции в PHP осуществляются по следующему алгоритму: открыть ресурс, обойти его в цикле, закрыть ресурс. Например, в листинге П1.19 приводится пример обхода в цикле `while()` таблицы, полученной в результате `SELECT`-запроса к таблице `catalogs`.

Листинг П1.19. Обход в цикле результирующей таблицы

```
<?php
  ...
  // Получение результирующей таблицы
  $query = "SELECT * FROM catalogs"
  $cat = mysql_query();
  if(!$cat) exit(mysql_error());

  // Обход результирующей таблицы
  while($catalog = mysql_fetch_array($cat))
  {
    // Работа с массивом $catalog
  }
  ...
?>
```

В листинге П1.20 приводится пример обхода открытой директории при помощи функций `opendir()` и `readdir()`.

Листинг П1.20. Обход директории в цикле

```
<?php
  ...
  // Открытие директории
  $dir = opendir('oop/');

  // Обход директории
  while(($file = readdir($dh)) !== false)
  {
    // Работа с элементами директории
  }
  ...
?>
```

В листинге П1.21 приводится пример построчного обхода файла, содержимое которого преобразуется в массив `$arr`.

Листинг П1.21. Построчный обход содержимого файла

```
<?php
  ...
  // Получение содержимого файла в виде массива
  $arr = file("text.txt");
```

```
  // Обход файла
  for($i = 0; $i < count($arr); $i++)
  {
     // Работа со строками файла
  }
  ...
?>
```

Каждый из примеров в листингах П1.19 и П1.20 демонстрирует обход коллекции (записей, имен файлов и поддиректорий, строк файлов), однако в каждом случае используется свой вариант цикла. Итераторы позволяют использовать в каждом из этих случаев единый метод обхода коллекции при помощи цикла `foreach()` (листинг П1.22).

Листинг П1.22. Использование итератора

```
<?php
  ...
  // Получение коллекции
  $iterator = new InteratorClass();

  // Обход коллекции
  foreach($iterator as $element)
  {
    // Работа с элементом коллекции $element
  }
  ...
?>
```

Если листинг П1.22 реализует какой-либо алгоритм — сортировка, постраничный вывод и т. п., — то для его применения к различным коллекциям достаточно объявить соответствующий коллекции итератор `$iterator`.

П1.3.2. Интерфейс *Iterator*

Каждый итератор реализует интерфейс `Iterator`, который требует реализации следующих методов:

- `current()` — вернуть текущий элемент;
- `key()` — вернуть ключ текущего элемента;
- `next()` — переместить итератор к следующему элементу;
- `rewind()` — поместить итератор в начало коллекции;

- `valid()` — проверить текущий элемент после вызова методов `rewind()` и `next()`; метод возвращает `false`, если итератор находится за пределами коллекции (закончились элементы).

В листинге П1.23 приводится пример создания класса собственного итератора `IteratorDir`, реализующий интерфейс `Iterator`. Класс `IteratorDir` реализует итератор для доступа к элементам директории.

Листинг П1.23. Класс `IteratorDir`

```
<?php
  class IteratorDir implements Iterator
  {
    // Путь к директории
    protected $dir;

    // Номер файла или поддиректории в директории
    protected $key;
    // Имя файла или поддиректории в директории
    protected $file;
    // Флаг, принимающий значение false или true, в зависимости
    // от того, закончились файлы или нет
    protected $valid;

    // Конструктор
    public function __construct($path)
    {
      $this->dir = opendir($path);
    }

    // Деструктор
    public function __destruct()
    {
      closedir($this->dir);
    }

    // Установка итератора в начало
    public function rewind()
    {
      $this->key = 0;
      rewinddir($this->dir);
      $this->next();
    }
```

```
    // Чтение следующего элемента
    public function next()
    {
      $this->key++;
      $this->valid = false !== ($this->file = readdir($this->dir));
    }

    // Получение ключа
    public function key()
    {
      return $this->key;
    }

    // Получение значения
    public function current()
    {
      return $this->file;
    }

    // Возвращает флаг, закончились элементы в коллекции или нет
    public function valid()
    {
      return $this->valid;
    }
  }
?>
```

Теперь для того чтобы обойти директорию в цикле, достаточно объявить объект класса `IteratorDir`, передав конструктору путь к ней в качестве параметра (листинг П1.24).

Листинг П1.24. Использование объекта класса `IteratorDir`

```
<?php
  require_once("class.iteratordir.php");

  $dir = new IteratorDir('oop');
  foreach($dir as $file)
  {
    echo $file."<br>";
  }
?>
```

Результатом работы скрипта будет список файлов и поддиректорий директории "`oop`".

П1.3.3. Класс *DirectoryIterator*

Библиотека SPL содержит уже готовые классы, реализующие интерфейс `Iterator` (объекты которых могут использоваться в цикле `foreach`). Одним из таких классов является `DirectoryIterator`, предоставляющий доступ к содержимому директории. В листинге П1.25 приводится пример использования итератора.

Замечание

По результату выполнения скрипты из листингов П1.24 и П1.25 эквивалентны.

Листинг П1.25. Использование класса `DirectoryIterator`

```
<?php
  $dir = new DirectoryIterator('oop');
  foreach($dir as $file)
  {
    echo $file."<br>";
  }
?>
```

Объект `$file` здесь выступает не как строка, а как объект, реализующий методы, представленные в таблице П1.8.

Таблица П1.8. *Методы класса* `DirectoryIterator`

Метод	Описание
`current()`	Возвращает ссылку на объект `$this`
`getATime()`	Возвращает время последнего доступа к файлу или поддиректории (в секундах, прошедших с полуночи первого января 1970 года)
`getCTime()`	Возвращает время последнего изменения файла или директории (в секундах, прошедших с полуночи первого января 1970 года)
`getFilename()`	Возвращает имя файла или поддиректории
`getGroup()`	Возвращает имя группы, в которую входит файл (только для UNIX-подобных операционных систем)
`getInode()`	Возвращает номер узла в файловой системе, соответствующий файлу или поддиректории (только для UNIX-подобных операционных систем)
`getMTime()`	Возвращает время последней модификации файла или директории (в секундах, прошедших с полуночи первого января 1970 года)
`getOwner()`	Возвращает имя владельца файла (только для UNIX-подобных операционных систем)

Таблица П1.8 *(окончание)*

Метод	Описание
`getPath()`	Возвращает имя директории (без имени файла и поддиректории)
`getPathname()`	Возвращает путь к файлу, включая название директории, а также название файла или поддиректории
`getPerms()`	Возвращает права доступа, назначенные файлу или подкаталогу (только для UNIX-подобных операционных систем)
`getSize()`	Возвращает объем файла (в байтах); для поддиректорий всегда возвращается значение 0
`getType()`	Возвращается тип текущего элемента каталога: `dir` для директории и `file` — для файла
`isDir()`	Возвращает `true`, если текущий элемент является директорией, и `false` — в противном случае
`isExecutable()`	Возвращает `true`, если текущий файл является исполняемым, а директория допускает переход в него, и `false` — в противном случае
`isFile()`	Возвращает `true`, если текущий элемент является файлом, и `false` — в противном случае
`isLink()`	Возвращает `true`, если текущий элемент является ссылкой, и `false` — в противном случае
`isReadable()`	Возвращает `true`, если файл или поддиректория доступны для чтения, и `false` — в противном случае
`isWritable()`	Возвращает `true`, если файл доступен для записи, а директория допускает создание внутри себя файлов, и `false` — в противном случае

Например, для того чтобы после имени файла вывести его размер, достаточно воспользоваться методом `getSize()` (листинг П1.26).

Листинг П1.26. Использование методов класса `DirectoryIterator`

```
<?php
  $dir = new DirectoryIterator('oop');
  foreach($dir as $file)
  {
    // Выводим только файлы
    if($file->isFile())
    {
      // Имя файла и его размер
      echo $file." ".$file->getSize()."<br>";
    }
  }
?>
```

П1.3.4. Класс *FilterIterator*

Элементы коллекции могут быть отфильтрованы при помощи итератора, производного от класса `FilterIterator`. В листинге П1.27 приводится пример создания фильтра `ExtensionFilter` для класса `DirectoryIterator`, который фильтрует все файлы, обладающие расширением ".php".

Листинг П1.28. Создание фильтра `ExtensionFilter`

```
<?php
  class ExtensionFilter extends FilterIterator
  {
    // Фильтруемое расширение
    private $ext;
    // Итератор DirectoryIterator
    private $it;

    // Конструктор
    public function __construct(DirectoryIterator $it, $ext)
    {
       parent::__construct($it);
       $this->it = $it;
       $this->ext = $ext;
    }

    // Метод, определяющий, удовлетворяет текущий элемент
    // фильтру или нет
    public function accept()
    {
      if(!$this->it->isDir())
      {
         $ext = array_pop(explode('.', $this->current()));
         return $ext != $this->ext;
      }
      return true;
    }
  }
?>
```

Использование класса `ExtensionFilter` совместно с классом `DirectoryIterator` приведет к тому, что из результирующего списка файлов будут исключены все файлы с расширением .php (листинг П1.29).

Листинг П1.29. Вывод списка файлов и директорий за исключением PHP-файлов

```
<?php
  require_once("class.extension.filter.php");

  $filter = new ExtensionFilter(
               new DirectoryIterator('oop'), 'php');

  foreach($filter as $file)
  {
    echo $file."<br>";
  }
?>
```

П1.3.5. Класс *LimitIterator*

Класс `LimitIterator` и его производные позволяют осуществить постраничный вывод. Конструктор класса принимает в качестве первого параметра итератор, в качестве второго параметра — начальную позицию (по умолчанию равную 0), а в качестве третьего — смещение от позиции. При этом итератор работает с участком коллекции, определяемым вторым и третьим параметрами. В листинге П1.30 приводится пример вывода первых пяти элементов директории "oop" с исключением PHP-файлов.

Листинг П1.30. Использование класса `LimitIterator`

```
<?php
  require_once("class.extension.filter.php");

  $limit =  new LimitIterator(
              new ExtensionFilter(
                new DirectoryIterator('oop'),
              "php"),
            0, 5);

  foreach($limit as $file)
  {
    echo $file."<br>";
  }
?>
```

П1.3.6. Рекурсивные итераторы

Рекурсивной называется функция, которая вызывает сама себя. Подобные конструкции часто используются для обхода древовидных структур. Например, директории могут быть вложены друг в друга, и для вывода содержимого директории, включая все вложенные поддиректории, может потребоваться рекурсивная функция. Типичный пример рекурсивной функции приводится в листинге П1.31.

Листинг П1.31. Рекурсивная функция для вывода содержимого директории

```
<?php
  function recurse_dir($path)
  {
    static $depth = 0;

    $dir = opendir($path);
    while(($file = readdir($dir)) !== false)
    {
      if ($file == '.' || $file == '..' ) continue;
      echo str_repeat("-", $depth)." $file<br>";

      if(is_dir("$path/$file"))
      {
        $depth++;
        recurse_dir("$path/$file");
        $depth--;
      }
    }
    closedir($dir);
  }

  recurse_dir('oop');
?>
```

Скрипт из листинга П1.33 выводит список файлов и поддиректорий директории "oop", определяя степень вложенности при помощи статической переменной `$depth`. Чтение содержимого директории выполняется в цикле; если текущий элемент является файлом — его название выводится в окно браузера, если поддиректорией — для него вызывается функция `recurse_dir()`. При спуске на один уровень значение переменной `$depth` увеличивается на единицу, при возвращении — уменьшается. Это позволяет выводить перед именем файла то количество символов "-", которое соответствует уровню "залегания" файла.

Решение проблемы вывода содержимого вложенной директории можно решить при помощи итераторов (листинг П1.32).

Листинг П1.32. Рекурсивный обход директории при помощи итераторов

```
<?php
  $dir = new RecursiveIteratorIterator(
           new RecursiveDirectoryIterator('oop'),
         true);

  foreach ($dir as $file)
  {
    echo str_repeat("-", $dir->getDepth())." $file<br>";
  }
?>
```

Метод `getDepth()` итератора `RecursiveIteratorIterator` возвращает глубину вложения элемента.

ПРИЛОЖЕНИЕ 2

Список функций для работы с классами и объектами

На протяжении всей книги используются специальные функции для работы с классами и их объектами. В табл. П2.1 приводится полный список функций, их назначение и ссылки на примеры в книге.

Таблица П2.1. *Список функций для работы с классами и объектами*

Функция	Описание
`call_user_method_array ($method_name, $obj [, $par])`	Осуществляет вызов метода `$method_name` объекта `$obj` с массивом параметров `$par`. Функция признана устаревшей и будет исключена из последующих версий языка PHP; вместо нее рекомендуется использовать функцию `call_user_func_array()`
`call_user_func_array ($methor_name, $par)`	Осуществляет вызов метода или функции `$method_name` с массивом параметров `$par`
`call_user_method ($method_name, $obj [, $par [, $par1, ...]] )`	Осуществляет вызов метода `$method_name` объекта `$obj` с параметрами `$par`, `$par1` и т. д. Функция признана устаревшей и будет исключена из последующих версий языка PHP; вместо нее рекомендуется использовать функцию `call_user_func()`
`call_user_func ($method_name [, $par [, $par1, ...]] )`	Осуществляет вызов метода или функции `$method_name` с параметрами `$par`, `$par1` и т. д.
`class_exists($class_name)`	Возвращает `true`, если класс с именем `$class_name` объявлен, и `false` — в противном случае. Подробнее функция описывается в *разделе 3.7*
`get_class_methods ($class_name)`	Возвращает массив с именами методов класса `$class_name`. Подробнее функция описывается в *разделе 3.13*

Таблица П2.1 *(продолжение)*

Функция	Описание
`get_class_vars ($class_name)`	Возвращает массив с именами и значения членов класса `$class_name`. Подробнее функция описывается в *разделе 3.10*
`get_class ($obj)`	Возвращает имя класса, которому принадлежит объект `$obj`. Подробнее функция описывается в *разделе 3.8*
`get_declared_classes()`	Возвращает массив с именами объявленных классов. Подробнее функция описывается в *разделе 3.7*
`get_declared_interfaces()`	Возвращает массив с именами объявленных интерфейсов. Подробнее функция описывается в *разделе 5.4*
`get_object_vars ($obj)`	Возвращает массив с именами и значениями членов объекта `$obj`. Подробнее функция описывается в *разделе 3.10*
`get_parent_class ($obj)`	Возвращает имя базового класса для объекта или класса `$obj`. Подробнее функция описывается в *разделе 4.6*
`interface_exists ($interface_name [, $autoload])`	Возвращает `true`, если интерфейс с именем `$interface_name` существует, и `false` — в противном случае. Если необязательный параметр `$autoload` равен `true`, функция пытается загрузить интерфейс при помощи функции `__autoload()` (см. *раздел 3.6*). Подробнее функция `interface_exists()` описывается в *разделе 5.4*
`is_a ($obj, $class_name)`	Принимает в качестве первого параметра объект `$obj`, а в качестве второго — имя базового класса `$class_name` и возвращает `true`, если объект является экземпляром класса `$class_name` или экземпляром его потомка, и `false` — в противном случае. Подробнее функция описывается в *разделе 4.6*
`is_callable ($arr [, $syntax_only [, $callable_name]])`	Позволяет выяснить, может ли быть вызван метод класса
`is_object ($obj)`	Возвращает `true`, если переменная `$obj` является объектом, и `false` — в противном случае. Подробнее функция описывается в *разделе 2.7*

Таблица П2.1 *(окончание)*

Функция	Описание
`is_subclass_of ($obj, $class_name)`	Принимает в качестве первого параметра объект `$obj`, а в качестве второго параметра имя базового класса `$class_name` и возвращает `true`, если объект является экземпляром потомка класса `$class_name`, и `false` — в противном случае. Подробнее функция описывается в *разделе 4.6*
`method_exists ($obj, $method_name)`	Возвращает `true`, если объект `$obj` обладает методом `$method_name`, и `false` — в противном случае. Подробнее функция описывается в *разделе 3.13*
`print_r ($obj [, $return])`	Возвращает дамп объекта `$obj`. Если переменная `$return` принимает значение `true`, функция возвращает результат в виде строки, в противном случае результат выводится непосредственно в окно браузера. Подробнее функция описывается в *разделе 2.9*
`property_exists ($class_name, $var)`	Возвращает `true`, если переменная `$var` является членом класса `$class_name`, в противном случае возвращается `false`. Подробнее функция описывается в *разделе 3.10*

ПРИЛОЖЕНИЕ 3

Описание компакт-диска

Компакт-диск, прикладываемый к данной книге, содержит исходные коды всех рассматриваемых Web-приложений (табл. П3.1).

Таблица П3.1. *Содержимое компакт-диска*

Папка	Описание	Главы
\2	Исходный код листингов для главы 2	2
\3	Исходный код листингов для главы 3	3
\4	Исходный код листингов для главы 4	4
\5	Исходный код листингов для главы 5	5
\6	Исходный код листингов для главы 6	6
\7	Исходный код листингов для главы 7	7
\8	Исходный код листингов для главы 8	8
\9	Исходный код листингов для главы 9	9
\10	Исходный код листингов для главы 10	10
\11	Исходный код листингов для главы 11	11

Рекомендуемая литература

Технология Web-разработки включает в себя множество дисциплин, детальное изучение каждой из которых не под силу одному человеку. Однако знакомство только с одной областью, например, HTML или PHP, даже на уровне эксперта, не позволяет разработчику создавать профессиональные сайты высокой сложности. Ценность имеет не отдельный язык программирования, а их совокупность — технология. Чем большим количеством языков программирования и инструментов владеет разработчик, тем он ценнее. Для эффективной работы в Web-области придется приложить значительные усилия, т. к. Web-разработчик высокого уровня должен владеть следующими языками программирования и технологиями:

- язык разметки HTML;
- каскадные таблицы стилей CSS;
- основы дизайна (включая инструменты работы с векторной и растровой графикой);
- клиентский язык JavaScript;
- XML;
- серверный язык PHP;
- серверный язык Perl;
- Web-сервер Apache;
- устройство сети Интернет и основные интернет-протоколы (хотя бы HTTP, IP и DNS);
- сокеты и CURL;
- основы работы с электронной почтой;
- регулярные выражения;

- MySQL и язык SQL, желательно на уровне стандартов и различий между его диалектами для различных систем управления базами данных;
- Flash;
- основы операционной системы UNIX.

Замечание

Перечисленные выше пункты касаются только технологии разработки сайтов для связки Apache, PHP и MySQL, помимо которых существуют технологии ASP.NET, Java и CouldFusion, которые в данной книге не затрагиваются.

Как уже упоминалось, охватить все области Web-разработки одному человеку невозможно, и, как правило, профессиональный Web-разработчик специализируется в нескольких наиболее интересных ему областях. Для создания сайта объединяется несколько разработчиков или даже организаций:

- хост-провайдеры, сдающие в аренду свои серверы, берут на себя организацию хост-площадки, требующую глубоких знаний операционной системы UNIX, администрирование Web-сервера Apache, DNS-сервера bind и MySQL-сервера;
- разработчики, специализирующиеся на Web-дизайне, уделяют большое внимание языку разметки HTML, XML, каскадным таблицам стилей, дизайну, инструментам работы с растровой (Photoshop) и векторной (3D Max, CorelDRAW) графики, технологиям JavaScript и Flash;
- разработчики, обычно специализирующиеся на одном или нескольких серверных языках (PHP и Perl), Web-протоколах (HTTP, IMAP, SMTP, FTP и т. п.), базах данных (MySQL) и регулярных выражениях, занимаются созданием бизнес-логики сайта.

Такое условное разделение вовсе не значит, что, специализируясь в одной области, разработчик может даже не знакомиться с соседними. Ниже приводится список литературы, который, не претендуя на универсальность и объективность, позволит сориентироваться в порядке и объеме знаний, необходимых для профессиональной работы.

Замечание

Охватить весь спектр представленной на рынке литературы мы не можем, более того, выпуск ряда книг со временем может быть прекращен, и читателю потребуется самостоятельно подбирать аналоги. Однако список позволит составить представление об объеме и характере знаний, которыми должен обладать Web-разработчик, специализирующийся на серверных языках программирования, т. е. не занимающийся самостоятельной разработкой Web-дизайна и администрированием серверов, однако представляющий соседние области и способный к переквалификации.

Книги в области программирования делятся на два типа: последовательные, излагающие материал от простого к сложному, и рецептурные, предлагающие ценные советы, разделы в которых, как правило, не зависят друг от друга и читать которые можно в произвольном порядке. В приводимом ниже списке такие книги будут помечены отдельно.

Совет

Если вы только начинаете изучать технологию, ориентируйтесь на книгу объемом 400—600 страниц; книга, содержащая 1000—1200 страниц, больше подходит в том случае, когда вы уже знакомы с технологией и хотите детально в ней разобраться, в противном случае велика вероятность "захлебнуться", не изучив книгу до конца.

Важно понимать, что техническая литература, в отличие от художественной, более насыщена информацией. Поэтому нельзя просто пропустить несколько страниц текста или приступать к следующей главе не до конца понимая ранее изложенный материал. Возможно, одну и ту же главу придется прочитать несколько раз, а саму книгу отложить на полгода. Если разработчик утверждает, что на изучение технологии или языка программирования у него ушло пару недель — он либо знает язык очень поверхностно, либо изучил большое количество языков до этого, поэтому многие сложные моменты оказались проработаны заранее. Чудес не бывает: любое искусство требует упорной работы и длительной практики. Об этом не любят вспоминать, когда цель достигнута, поэтому у начинающего программиста может сложиться ошибочное впечатление, что он топчется на месте, в то время как все вокруг буквально за пару недель способны к освоению приведенных выше технологий.

HTML, XML, CSS, JavaScript и Flash

Для разработчика бизнес-логики не требуется экспертное знание языка разметки HTML, XML, каскадных таблиц стилей и клиентского языка JavaScript. Однако знакомство с данными технологиями крайне желательно, поскольку они описывают основы предметной области: сайт можно построить без использования PHP, однако без HTML и CSS создать более или менее приличный сайт просто не удастся.

В. В. Мержевич. HTML и CSS на примерах. — СПб.: БХВ-Петербург, 2005. — 488 с.

Изучение языка разметки HTML в отрыве от каскадных таблиц стилей CSS в современных условиях не рационально. Представленная книга является идеальным введением в HTML, CSS, а также описывает их взаимодействие. Книга прекрасно сочетает в себе последовательное изложение от простого к

сложному и множество советов, которые можно сразу применить на практике.

Электронная документация по HTML

Книги являются своеобразным комментарием к официальной документации (как комментарии к уголовному или гражданскому кодексам). Как и любой комментарий, книги позволяют читателям разобраться в технологии быстрее, чем если бы они пользовались электронной документацией на английском языке, т. е., приобретая книгу, читатель экономит время в обмен на деньги. Тем не менее электронной документаций не следует пренебрегать, особенно если вы знакомы с технологией, но вам требуется справочник для быстрого поиска.

Замечание

Знание английского языка не указывается в качестве необходимого для Web-разработчика, однако позволяет значительно повысить конкурентоспособность. Документация появляется в первую очередь на английском языке, часть ее вообще не переводится на национальные языки, переводится с опозданием или не в полном объеме.

Если у вас установлена одна из версий Microsoft Office, вы можете найти CHM-справочник (в его имени будет присутствовать подстрока html) по HTML и DHTML в его справочных файлах в директории Program Files. Например, на компьютере авторов установлен Microsoft Office XP и файл называется htmltag.chm.

Э. Мейер. CSS — каскадные таблицы стилей. Подробное руководство, 2-е изд. — Пер. с англ. — СПб.: Символ-Плюс, 2006. — 576 с.

Книга представляет полное изложение каскадных таблиц стилей вплоть до спецификации CSS2.1 с комментариями, позволяющими составить представление о возможностях, реализованных в современных браузерах. Книга читается легко, однако требует предварительного знакомства с HTML.

Ч. Валентайн, К. Минник. XHTML: Пер. с англ. — М.: Издательский дом "Вильямс", 2001. — 480 с.

На смену HTML должен прийти язык разметки XHTML, так Web-приложения для браузеров мобильных устройств (сотовые телефоны, КПК) уже сегодня разрабатываются с использованием этого языка разметки.

Так получилось, что помимо функций структурирования документов HTML стал выполнять и оформительские задачи. Для того чтобы разделить структуру и оформление документов, консорциумом w3c был введен язык разметки XML и каскадные таблицы стилей CSS, при помощи которых можно оформить XML-код. Язык XML настолько гибок, что позволяет самостоятельно

определить нужный XML-формат путем разработки своего собственного словаря. В настоящий момент уже разработано огромное число словарей XML, позволяющих описывать любую информацию — от химических реакций (CML, Chemical Markup Language) до финансовых (OFX, Open Financial Exchange). По сравнению с HTML XML является более гибким форматом; повсеместного применения XML еще не получил, но это лишь вопрос времени.

Н. Питц-Моултис, Ч. Кирк. XML: Пер. с англ. — СПб.: БХВ-Петербург, 2001. — 736 с.

Полное и последовательное изложение языка разметки XML.

Д. Гудман. JavaScript и DHTML. Сборник рецептов. Для профессионалов. — СПб.: Питер, 2004. — 523 с.

Книга представляет множество рецептов для создания динамических сайтов с использованием клиентских технологий. Последовательное изложение языка программирования JavaScript в книге отсутствует, однако охватываются практически все области его применения.

Д. Гудман, М. Моррисон. JavaScript. Библия пользователя, 5-е изд. — Пер. с англ. — М.: ООО "И. Д. Вильямс", 2006. — 1184 с.

Приводится полное и последовательное изложение JavaScript и DHTML. Большое внимание уделяется различиям в реализации JavaScript в современных браузерах.

Электронная документация по JavaScript

Точно так же, как и в случае HTML, в документации к Microsoft Office можно обнаружить CHM-описание для версии JavaScript, реализованной в Internet Explorer. У авторов данный файл называется jscript5.chm. За документацией по JavaScript, реализованному в движке Mozilla, следует обратиться к сайту **http://www.mozilla.org**.

Р. Рейнхардт, С. Дауд. Macromedia Flash 8. Библия пользователя: Пер. с англ. — М.: "И. Д. Вильямс", 2006. — 1328 с.

В книге приводится подробное и последовательное изложение основ работы в среде Macromedia Flash и создание Flash-роликов.

К. Мук. ActionScript для Flash MX. Подробное руководство. — Пер. с англ. — СПб.: Символ-Плюс, 2004. — 1120 с.

В технологии программирования Flash используется язык программирования ActionScript. Данная книга представляет одно из полных его описаний.

PHP и Perl

Для работы приложения на стороне сервера используются серверные языки программирования, такие как PHP и Perl. Perl здесь рассматривается не случайно: разработчики PHP заимствовали из него множество концепций, и для более полного изучения PHP зачастую неплохо познакомиться с Perl, т. к. многие положения в документации по PHP описаны туманно или не описаны вообще.

PHP разрабатывался как более удобное средство для создания Web-приложений, и это удалось на славу — он стал одним из самых быстрых средств разработки. Разработчики PHP преследовали цель создать не контекстный язык с элементами лингвистического языка, как в случае Perl, а язык для быстрой разработки со строгим синтаксисом. Приложение, созданное одним разработчиком, должно было без проблем подхватываться другим, т. е. он планировался как прагматический язык. В свою очередь Perl создавался как произведение искусства — это целая философия. Его последователи зачастую фанатичны (дай им волю, они и операционную систему разработают на Perl), пишут стихи на Perl, устраивают конкурсы на самую непонятную программу, т. е. это достаточно романтический язык — в нем множество соблазнов для программиста, которые мешают созданию промышленного кода. Это язык для души. Perl более красив, чем PHP, но менее удобен для бизнеса и Web.

М. Кузнецов, И. Симдянов. Самоучитель PHP 5. — 2-е изд. перераб. и доп. — СПб.: БХВ-Петербург, 2006. — 608 с.

Приводится последовательное изложение языка, идеально подходящее для знакомства с PHP и его расширениями. Она начинает нашу серию, посвященную Web-разработке.

Д. В. Котеров, А. Ф. Костарев. PHP 5. — СПб.: БХВ-Петербург, 2005. — 1120 с.

Последовательное и полное изложение языка, достойное внимания. Подробно рассматривается технология "клиент-сервер" и взаимодействие с XML. К сожалению, часть книги, посвященная объектно-ориентированному программированию в PHP, несколько устарела.

М. Кузнецов, И. Симдянов, С. Голышев. PHP 5. Практика создания Web-сайта. — СПб.: БХВ-Петербург, 2005. — 960 с.

Рассматривается структура и методика создания современного Web-сайта и его элементов: CMS, форума, системы сбора статистики, рассылки, FTP-менеджера, каталогов продукции и т. п. Книга ориентирована на читателя, уже знакомого с основами HTML и PHP.

М. Кузнецов, И. Симдянов, С. Голышев. PHP 5 на примерах. — СПб.: БХВ-Петербург, 2005. — 576 с.

В отличие от предыдущей книги, рассматриваются не готовые примеры, а сборник коротких и эффективных приемов, взятых из реальной практики: загрузка курса валют, создание динамического изображения, вывод случайного изображения из массива, создание PDF-документа и т. п. Книга носит рецептурный характер, и ее главы могут читаться в произвольном порядке. Несмотря на то, что книга не является последовательным изложением языка программирования PHP, в ней затрагиваются все основные моменты его использования.

Д. Скляр, А. Трахтенберг. PHP. Сборник рецептов. 2-е изд. — Пер. с англ. — БХВ-Петербург, 2007. — 736 с.

Книга с изложением полезных рецептов, главы можно читать в произвольном порядке.

М. Кузнецов, И. Симдянов. Головоломки на PHP для хакера. — СПб.: БХВ-Петербург, 2006. — 464 с.

Книга представляет собой задачник по Web-технологиям с уклоном в защиту Web-приложений от злоумышленников. Цель книги — помочь Web-разработчику научиться самостоятельно обнаруживать и устранять уязвимости в своем коде.

М. Ф. Низамутдинов. Тактика защиты и нападения на Web-приложения. — СПб.: БХВ-Петербург, 2005. — 432 с.

Введение в проблему безопасности Web-приложений, созданных с использованием PHP и MySQL.

Электронная документация по PHP

Электронную документацию по PHP можно загрузить с официального сайта **http://www.php.net**; здесь же находится перевод части документации на русский язык. Однако последним следует пользоваться крайне осторожно, т. к. информация на русском языке сильно устарела.

Э. Леки-Томпсон, А. Коув, С. Новицки, Х. Айде-Гудман. PHP 5 для профессионалов: Пер. с англ. — М.: ООО "И. Д. Вильямс", 2006. — 608 с.

Книга содержит описание паттернов объектно-ориентированного программирования и применение их к PHP. К сожалению, в ней приведено описание объектно-ориентированного подхода для PHP 4, затрагивая ООП PHP 5 лишь на двух страницах, посвященных описанию нововведений. Книга читается тяжело.

Л. Уолл, Т. Кристиансен, Д. Орвант. Программирование на Perl: Пер. с англ. — СПб.: Символ-Плюс, 2004. — 1152 с.

Знаменитая Camel-book ("Верблюжья книга") от создателя языка Ларри Уолла — замечательное введение в язык, с юмором раскрывающее тонкости и философию языка Perl. Книга представляет собой последовательное изложение языка от простого к сложному.

Т. Кристиансен, Н. Торкингтон. Perl. Сборник рецептов. Для профессионалов. 2-е изд. — Пер. с англ. — СПб.: Питер, 2004. — 928 с.

Книга носит рецептурный характер и является своеобразным дополнением книги Ларри Уолла "Программирование на Perl".

СУБД *MySQL*

Ни один современный сайт не может обойтись без использования баз данных. В Российской Федерации широкое распространение получила СУБД MySQL, выступающая стандартом де-факто для российских хост-провайдеров и Web-разработчиков.

Замечание

Несмотря на то, что существует стандарт языка запросов SQL, каждая СУБД реализует свой собственный диалект, иногда значительно отличающийся от диалектов других СУБД. Поэтому процесс изучения SQL сводится сначала к изучению общих основ SQL, а затем диалекта конкретной базы данных.

Дж. Грофф, П. Вайнберг. Энциклопедия SQL, 3-е изд. — СПб.: Питер, 2003. — 896 с.

Добротное и последовательное изложение языка запросов SQL с указанием особенностей отдельных диалектов. Очень ясное и прозрачное изложение, снабженное большим количеством поясняющих схем и рисунков.

М. Кузнецов, И. Симдянов. Самоучитель MySQL 5. — СПб.: БХВ-Петербург, 2005. — 560 с.

Введение в диалект SQL для СУБД MySQL 5.0. Изложение ведется последовательно от простого к сложному. Затрагиваются нововведения MySQL 5.0: хранимые процедуры и функции, представления, триггеры, курсоры и информационная схема.

М. Кузнецов, И. Симдянов. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024 с.

Последовательное и полное изложение MySQL 5.0. В отличие от книги "Самоучитель MySQL 5" подробно рассматривается администрирование MySQL

и взаимодействие с языками программирования C++, Perl и PHP под операционными системами Windows и UNIX.

П. Дюбуа. MySQL, 2-е изд. — М.: Вильямс, 2007. — 1168 с.

Изложение MySQL, на которое стоит обратить внимание. Очень подробно описывается администрирование СУБД в условиях UNIX-подобной операционной системы. К сожалению, книга лишь вскользь описывает нововведения MySQL 4.1 и 5.0.

М. Кузнецов, И. Симдянов. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592 с.

Книга посвящена MySQL версии 5.1 и помимо указанных выше нововведений затрагивает планировщик заданий и сегментирование. Большое внимание уделяется объектно-ориентированной библиотеке php_mysqli. Книга носит рецептурный характер, однако может использоваться в том числе и для последовательного изучения материала.

П. Дюбуа. MySQL. Сборник рецептов: Пер. с англ. — СПб.: Символ-Плюс, 2004. — 1056 с.

Книга носит рецептурный характер и может читаться с любой главы, однако, как и указанная ранее книга этого же автора, описывает лишь MySQL версии 4.0.

Электронная документация по MySQL

В разделе официального сайта MySQL, посвященного разработчикам **http://dev.mysql.com**, можно найти ссылки на документацию в электронном формате, которая обновляется ежедневно. На этом же сайте присутствует и русскоязычный вариант документации, но только для версии MySQL 4.0; для более поздних версий она доступна только на английском языке.

Замечание

Недавно компания MySQL AB выпустила перевод документации на русский язык в виде книг "MySQL. Руководство администратора: Пер. с англ. — М.: "И. Д. Вильямс", 2005. — 624 с." и "MySQL. Справочник по языку: Пер. с англ. — М.: "И. Д. Вильямс", 2005. — 432 с". Однако следует иметь в виду, что официальная документация, подготавливаемая разработчиками, зачастую просто не читаема. Книги содержат зачастую совершенно несвязанные участки текста, понять смысл которых можно только в результате длительных экспериментов. Их можно рекомендовать, если чтение английской документации у вас вызывает затруднение.

Интернет и Web-сервер *Apache*

Web-приложения — это всегда распределенные приложения, в работе которых принимают участие множество серверов и клиентов. Для того чтобы соз-

давать эффективные Web-приложения, необходимо очень четко представлять себе принципы работы сети Интернет и его протоколов.

Д. Э. Камер. Сети TCP/IP. Том 1. Принципы, протоколы и структура, 4-е изд. — Пер. с англ. — М.: "И. Д. Вильямс", 2003. — 880 с.

Прекрасное введение в историю и современное устройство Интернет. Подробно описывается физическая и логическая архитектура, протоколы IP, ICMP, UDP и TCP. К сожалению, мало внимания уделяется протоколам прикладного уровня (HTTP, SMTP, FTP, IMAP и т. п.), которые наиболее интересны для Web-разработчика.

Б. Кришнамурти, Дж. Рексфорд. Web-протоколы. Теория и практика. — М.: ЗАО "Издательство БИНОМ", 2002. — 592 с.

Описание протокола HTTP и приемов работы с ним. Книга не очень эффективная и содержит мало примеров, однако из бумажных книг по протоколу HTTP посоветовать практически нечего. Книга читается тяжело.

Электронная документация по протоколам Интернет

Все протоколы и спецификации сети Интернет описываются в так называемых RFC-документах, получить которые можно с большого количества сайтов, стоит лишь ввести в поисковую систему аббревиатуру RFC. Практически все RFC-документы на английском языке, и только часть из них переведена на русский.

П. Альбитц, К. Ли. DNS и BIND: Пер. с англ. — СПб.: Символ-Плюс, 2004. — 688 с.

Подробно описывается система доменных имен сети Интернет и конфигурирование сервера BIND.

М. Дж. Кабир. Сервер Apache 2. Библия пользователя: Пер с англ. — М.: "И. Д. Вильямс", 2002. — 672 с.

Прекрасное изложение конфигурирования Web-сервера Apache. К сожалению, не затрагивает последнюю версию Apache 2.2, а также ряд новых модулей, позволяющих более эффективно управлять сервером в условиях виртуального хостинга. Однако, несмотря на все недостатки, представляет собой одно из лучших описаний Web-сервера Apache на сегодняшний день.

Регулярные выражения

Регулярные выражения являются специализированным языком программирования, позволяющим манипулировать текстом. В свободном виде регулярные выражения практически не используются и применяются совместно с другими языками программирования. Регулярные выражения можно найти в

JavaScript, PHP, Perl, MySQL, Apache и многих других языках программирования и технологиях. Будучи специализированным языком программирования, регулярные выражения имеют множество диалектов, поэтому в каждом конкретном случае потребуется исследование возможностей данной реализации регулярных выражений.

Замечание

Существует две развитых реализации регулярных выражений: регулярные выражения Perl, реализованные и развиваемые в рамках языка Perl, и регулярные выражения POSIX, реализуемые в рамках стандарта переносимых операционных систем. Более удобными и распространенными являются регулярные выражения Perl.

Дж. Фридл. Регулярные выражения, 2-е изд. — СПб.: Питер, 2003. — 464 с.

Книга представляет собой единственное наиболее полное издание, посвященное регулярным выражениям. Несмотря на то, что в ней не описываются регулярные выражения применительно к PHP, это не мешает восприятию и эффективному применению полученных знаний на практике. Рассматриваются все распространенные диалекты регулярных выражений. Издательство "Питер" приняло решение не переиздавать книгу, однако, к радости ее поклонников, она выложена издательством в свободный доступ. Ее также можно загрузить с нашего сайта по ссылке **http://www.softtime.ru/info/fridl.php**.

UNIX-подобные операционные системы

UNIX-подобные операционные системы, в частности Linux и FreeBSD, интенсивно используются для построения сети Интернет. Только 20% серверов работают под управлением операционных систем, не являющихся UNIX-подобными. С другой стороны, клиентские машины в основном работают под управлением операционной системы Windows. В результате часто возникают конфликты, когда Web-приложение разработанное в условиях Windows, перестает работать в UNIX. Насколько досконально следует изучать UNIX-подобные операционные системы, является делом Web-разработчика, однако иметь представление о них просто необходимо.

Д. Тейнсли. Linux и UNIX: программирование в shell. Руководство разработчика: Пер. с англ. — К.: Издательская группа BHV, 2001. — 464 с.

Добротное изложение стандартного командного интерпретатора UNIX — bash. Подробно рассматривается система прав доступа UNIX и создание собственных скриптов.

Э. Немеет, Г. Снайдер, С. Сибасс, Т. Хейн. UNIX: руководство системного администратора. Для профессионалов. — СПб.: Питер. К.: Издательская группа BHV, 2003. — 925 с.

Данная книга является настоящей энциклопедией UNIX, рассказывающая о тонкостях работы системного администратора этой операционной системы. Повествование опирается на операционные системы Solaris, HP-UX, Red Hat и FreeBSD. Книга прекрасно подойдет как профессионалу, так и новичку, который захочет окунуться в мир UNIX. Изложение материала идет от простого к сложному, рассматриваются процессы пуска и останова системы, привилегии, управление процессами, файловая система и т. д. Не обделены вниманием и сетевые настройки: DNS, BIND, sendmail, NIS, Apache, Usenet, FTP-сервер. Книга читается легко и интересно, приводится множество реальных примеров из практики системного администрирования.

Э. С. Реймонд. Искусство программирования для UNIX: Пер. с англ. — М.: "И. Д. Вильямс", 2005. — 544 с.

Данная книга описывает философию программирования под UNIX. В ней вы не найдете листингов, описания системных вызовов, но зато сможете понять дух операционной системы, принципы построения интерфейсов и архитектуры UNIX-приложений.

В книге затрагиваются вопросы истории развития легендарной операционной системы и раскрываются секреты ее долгожительства. Вы узнаете, почему ООП более популярен в Windows, чем в UNIX, а также сможете познакомиться с культурой и фольклором UNIX.

Если вы никогда не сталкивались с операционной системой UNIX — это именно та книга, с которой нужно начать знакомство. Вы получите ясное представление о том, куда следует идти и что делать, и вообще нужен вам UNIX или нет. Если вы хорошо знаете данную операционную систему, иметь эту книгу вам просто необходимо: создававшаяся в течение 5 лет, она содержит мудрость проектирования под UNIX, накопленную UNIX-сообществом на протяжении 30 лет.

В. Костромин. Самоучитель Linux для пользователя. — СПб.: БХВ-Петербург, 2003. — 672 с.

Добротный самоучитель по операционной системе Linux, последовательно вводящий читателя в обсуждаемую предметную область.

А. Стаханов. Linux. — СПб.: БХВ-Петербург, 2003. — 912 с.

Полное и последовательное изложение основ операционной системы Linux. Подробно описывается установка, конфигурирование и работа в операционной системе Linux.

С. Л. Скловская. Команды Linux. Справочник, 3-е изд., перераб. и доп. — СПб.: ООО "ДиаСофтЮП", 2004. — 848 с.

Операционная система Linux имеет подробную электронную документацию, вызвать которую можно при помощи команды man. Однако зачастую удобнее воспользоваться бумажным справочником, особенно если у вас вызывает затруднение английский язык.

Методология программирования

Программирование — это не только код, спецификации, операционные системы, это еще и методология. Существуют книги, которые не связаны непосредственно ни с одним языком программирования, но которые помогают чувствовать направление и основные тенденции развития индустрии.

М. Кузнецов, И. Симдянов. Программирование: ступени успешной карьеры. — СПб.: БХВ-Петербург, 2006. — 320 с.

Процесс программирования очень увлекателен сам по себе. Однако начинающим программистам следует иметь в виду, что если создаваемый ими код не выдерживает конкуренции или его не удается продать, рано или поздно придется уходить из любимой области. Данная книга посвящена тому, как утвердиться и успешно развиваться в сфере программирования.

С. В. Жарков. Shareware: профессиональная разработка и продвижение программ. — СПб.: БХВ-Петербург, 2002. — 320 с.

Прекрасное руководство продвижения программных продуктов в условиях российской действительности и современного Запада. Книга написана с юмором и читается очень легко.

С. Макконнелл. Совершенный код. Мастер-класс: Пер. с англ. — М.: Издательско-торговый дом "Русская Редакция"; СПб.: Питер, 2005. — 896 с.

Это настоящая библия разработчика, содержащая самое полное собрание эффективных методик современного программирования, от правил именования переменных и функций и принципов структурного программирования до объектно-ориентированного программирования, которая поможет вам достичь высшего мастерства в создании эффективного, читабельного и легкосопровождаемого кода. Книга не содержит воды и недомолвок — вы получите настолько подробные сведения, что никогда не сможете использовать их в полном объеме. Автору удалось добиться практически невозможного: не привязываясь к конкретному языку программирования или платформе, донести до читателей опыт, накопленный за 40 лет существования программирования. Приводимые им рекомендации не голословны, а подтверждаются ис-

следованиями специалистов в области Computer Science, а также отчетами таких известных корпораций, как IBM и Microsoft. Более того, в отличие от всех подобных книг, вы не найдете философских рассуждений, которые невозможно реализовать на практике: все описываемые в книге приемы взяты из реальных проектов и реального кода.

Ф. Брукс. Мифический человеко-месяц или как создаются программные системы: Пер. с англ. — СПб.: Символ-Плюс, 1999. — 304 с.

Классическое произведение по проектированию больших программных продуктов, прочитать которое — долг любого программиста. Пожалуй, ни на одну книгу не ссылаются так часто и, пожалуй, ни одна книга не описывает более честно реальные сложности управления программными проектами.

Предметный указатель

А

Д

И

К

М

Н

О

П

С

Т

Ф

Ч