Дмитрий Ляпин
Александр Никитин

Начинаем с видеоуроков

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.068
ББК 32.973.26-018.1
Л97

Ляпин, Д. А.

Л97 PHP — это просто. Начинаем с видеоуроков / Д. А. Ляпин, А. В. Никитин. — СПб.: БХВ-Петербург, 2012. — 176 с.: ил. + Видеокурс (на CD-ROM)

ISBN 978-5-9775-0678-6

В книге есть все, что необходимо начинающему веб-программисту, который собирается научиться создавать веб-сайты, соответствующие современным требованиям веб-разработки. Приведен краткий обзор языка HTML, необходимый для отображения содержимого веб-страниц. Подробно изложены принципы веб-программирования на языке PHP. Рассказано об использовании объектно-ориентированного подхода в программировании на PHP. Раскрыты вопросы работы с файлами, базами данных, обработки пользовательского ввода. Рассмотрен современный шаблон проектирования веб-приложений MVC. Подробно изложены принципы профессионального подхода к проектированию и реализации PHP-сценариев. В книге даны самые современные принципы веб-программирования, сделан упор на профессиональный подход к разработке и полный отказ от устаревших приемов.

Прилагаемый компкт-диск содержит видеоуроки для каждой главы.

Для начинающих веб-программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 29.02.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 14,19.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

ISBN 978-5-9775-0678-6

Оглавление

Предисловие

Учебные материалы книги разделены на 13 глав, каждой из которых соответствует видеоурок. Для того чтобы ваше обучение было максимально эффективным, знакомство с новым материалом следует начинать с видеоурока, где авторы показывают реальные примеры использования излагаемой информации. Не волнуйтесь, если на данном этапе у вас появится какое-либо непонимание или объяснений из видео будет недостаточно. Пока ваша задача — просто повторить приведенный в видеоуроке пример, пускай даже до конца не понимая, как это работает. Все ответы ждут вас в теоретической части урока, в соответствующей главе книги. После ознакомления с теорией читателю предлагается закрепить полученные знания выполнением специального задания.

Таким образом, можно выделить три основных звена при изучении каждой главы:

- ❒ просмотр видеоурока и повторение практических действий, приведенных в нем;
- ❒ ознакомление с теорией, изложенной в главе;
- ❒ выполнение предложенного в конце главы задания.

Учебный курс ориентирован на месяц регулярной работы с данной книгой. Упор на практическое использование излагаемых материалов в совокупности с видеоиллюстрациями позволяет глубоко и качественно освоить знания, заложенные в книге. Данная методика освоения языков программирования активно применяется в центре компьютерного обучения "Школа программирования" (**http://proglive.ru**), который учрежден авторами книги. За годы использования данного подхода к обучению его эффективность прекрасно зарекомендовала себя и была подтверждена результатами нескольких сотен выпускников центра.

По результатам обучения читатели без какого-либо практического опыта веб-программирования получат все необходимые знания и умения для того, чтобы самостоятельно разрабатывать сайты на языке PHP и уверенно чувствовать себя на рынке труда веб-программистов.

Введение

Здравствуйте, дорогие читатели! Рады приветствовать вас на страницах этой книги. Давайте познакомимся. Нас зовут Александр Никитин и Дмитрий Ляпин. Мы являемся основателями центра компьютерного обучения "Школа программирования" (**http://proglive.ru**), по совместительству действующими программистами и авторами-разработчиками ряда других интернет-проектов.

Так сложилось, что практически всю нашу сознательную жизнь мы посвятили одному делу — программированию. На текущий момент мы занимаемся этим уже более 13 лет. За это время нам пришлось столкнуться с логическим и функциональным программированием, объектно-ориентированным и структурным программированием, системным и веб-программированием, параллельным программированием, программированием микроконтроллеров и искусственных интеллектов, программированием драйверов и компиляторов, низкоуровневым программированием в различных операционных системах.

Мы успели сменить не одну работу. Разрабатывали, внедряли, расширяли, делали сайты-визитки и крупные интернет-порталы. Прошли всё — от работы на рекламные агентства в роли HTML-верстальщиков до участия в проектах внедрения автоматизированных систем в крупнейших банках и компаниях федерального масштаба.

На текущий момент нам есть, что сказать о своей профессии. Мы прошли нелегкий путь, на котором было немало препятствий и ловушек. И нам искренне хочется облегчить эту дорогу для вас, наших читателей.

И это не пустые слова. Дело в том, что больше половины начинающих программистов бросают эту профессию из-за сложностей в обучении и высокой конкуренции на рынке. Начальный энтузиазм и легкость сменяются рутинным чтением справочников, от которых начинают "закипать мозги". Собственные программы, которые вы делаете для тренировки в ходе обучения, становятся все сложнее, и в какой-то момент вы сами перестаете понимать свой код. Все это приводит к потере интереса к обучению. Начинающий разработчик перестает развиваться в области программирования и начинает искать себя в чем-то другом. А ведь какие светлые и амбициозные чувства были у него в начале пути!

Действительно, обучение программированию можно сравнить с постоянным движением в гору. Каждый шаг будет даваться с трудом, каждую минуту вам нужно

прикладывать усилия для преодоления препятствий. Зато в какой-то момент вы окажетесь на вершине этой горы, и тогда для вас откроются невероятные возможности.

В этой книге мы постараемся максимально облегчить вашу дорогу к профессиональному использованию такого мощного языка веб-программирования как PHP. Кстати, почему именно PHP?

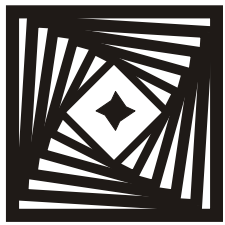
PHP — это самый популярный серверный язык в настоящее время, и это не случайно. Он прост в освоении, гибок и функционален. PHP используют как новички, так и профессионалы. Существует огромное количество готовых библиотек, написанных на PHP и призванных упростить разработку программистам. Можно сказать, что PHP в мире веб-программирования аналогичен английскому языку в языковой среде. Программируя на PHP, вы сможете пользоваться опытом и наработками огромного количества программистов по всему миру, а также ваш собственный код может быть полезным широкому кругу разработчиков.

Эта книга проведет вас от самых азов веб-разработки до приемов и концепций, которые применяются настоящими профессионалами. В первых уроках мы познакомимся с общими принципами функционирования Интернета и разберемся с тем, что же вообще собой представляют веб-страницы и сайты. Только освоив эту информацию, мы будем готовы перейти непосредственно к изучению языка PHP.

На протяжении всей книги мы особое внимание уделяем культуре написания кода. С нашей точки зрения культура кода имеет огромное значение для разработчиков, несмотря на то, что множество программистов даже не понимает этого словосочетания. Неосновательный подход веб-разработчиков к своему делу приводит к тому, что на данный момент для заказчика считается немалой удачей найти хорошего программиста. В этой связи программистам следует учиться как можно более качественно применять современные средства разработки в своей практике. В этом залог успеха нашей профессии. Именно поэтому последние главы книги мы посвятили не обучению правилам языка PHP, а методикам его грамотного использования.

Надеемся, что чтение этой книги и обучение PHP покажется вам интересным, и вы с удовольствием и энтузиазмом будете применять полученные знания на практике. Успехов!

Александр Никитин, Дмитрий Ляпин



УРОК 1

Основы протокола HTTP

Прежде чем перейти к изучению такого популярного и мощного языка веб-программирования, как PHP, важно иметь четкое представление о том, *как* вообще функционирует Интернет. Что собой представляют PHP-сценарии и HTML-код? Как взаимодействуют наши компьютеры с серверами, на которых располагаются веб-страницы? И, наконец, как эти замысловатые сценарии превращаются в те прекрасные интернет-сайты, которые мы каждый день видим на экранах своих мониторов.

К сожалению, огромное количество начинающих PHP-программистов даже не отдает себе отчет в том, где выполняются написанные ими сценарии, и вообще, что собой представляет процесс выполнения PHP-сценария. Однако без этих знаний очень трудно достичь каких-либо существенных успехов в PHP-разработке. Более того, эти знания должны быть получены в первую очередь.

Представьте себе архитектора, который абсолютно не разбирается в строительных материалах. Можно ли назвать его настоящим профессионалом? Скорее всего, такой человек не сможет построить что-нибудь сложнее сарая. Так и веб-программист, не понимающий основных принципов интернет-взаимодействия, просто не способен решать более-менее серьезные задачи.

Поэтому данный урок нашей книги мы посвящаем базовым вопросам взаимодействия веб-серверов, на которых располагаются интернет-страницы, и клиентских программ для просмотра этих самых страниц. Но если вы думаете, что вас ожидает лишь важная, но скучная теория, то ошибаетесь. В конце мы предложим пару практических заданий, которые позволят еще лучше понять принципы клиент-серверного взаимодействия программ при загрузке веб-страниц.

Что происходит между тем, как вы набрали адрес сайта и увидели страницу на экране?

Все мы знакомы с браузерами. Это те самые программы, которые позволяют нам просматривать страницы в Интернете. Но если вы спросите десять программистов, как работает веб-обозреватель, то восемь из них не смогут дать внятного ответа.

Давайте попробуем самостоятельно разобраться, что же происходит в промежутке между запросом какого-либо сайта и отображением страницы на экране.

Рассмотрим адрес какой-нибудь реальной веб-страницы, например **http://prog-school.ru/catalog**. На рис. 1.1 приведено ее отображение в браузере.

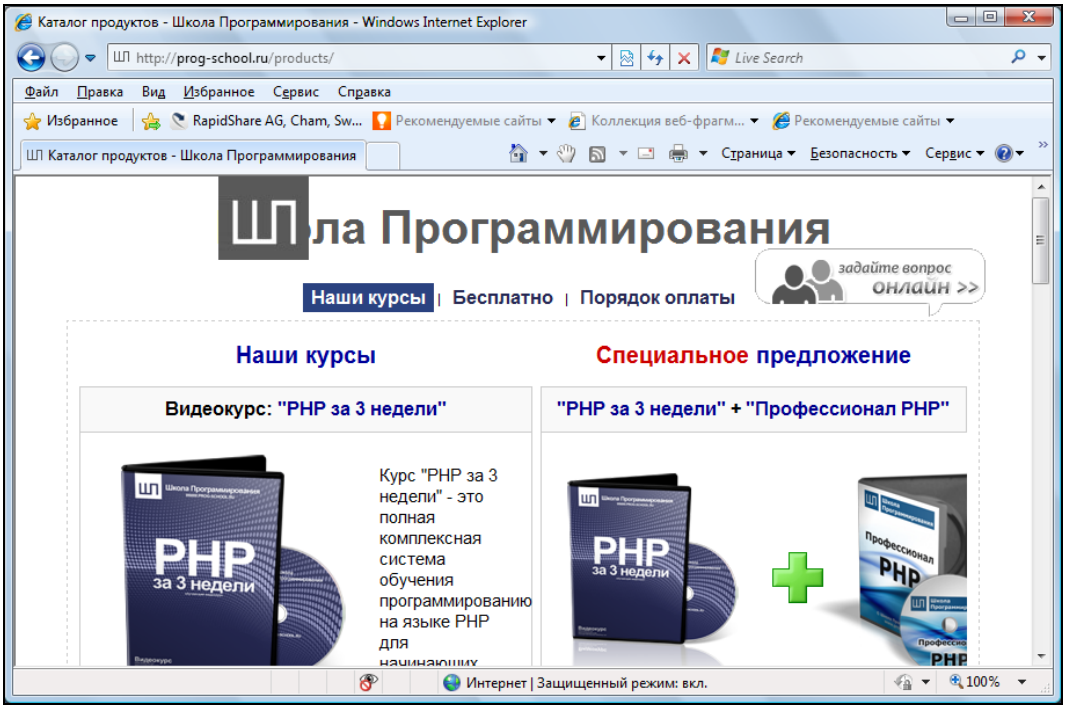


Рис. 1.1. Отображение в браузере страницы **http://prog-school.ru/catalog**

Обратите внимание на строку в адресном поле: **http://prog-school.ru/products/**. Она однозначно идентифицирует веб-страницу, которую должен отобразить для вас браузер. Эту строку также называют URL или URI. Давайте подробнее разберем обе эти аббревиатуры.

URL и URI

URL (Uniform Resource Locator) — единообразный локатор (определитель местонахождения) ресурса. URL — это стандартизированный способ записи адреса ресурса в Интернете.

URI (Uniform Resource Identifier) — унифицированный (единообразный) идентификатор ресурса. URI — это последовательность символов, идентифицирующая абстрактный или физический ресурс.

URI — это более общее понятие, нежели URL. URI не всегда указывает то, как получить ресурс, в отличие от URL, а только идентифицирует его. URL — это URI,

который, помимо идентификации ресурса, предоставляет еще и информацию о местонахождении этого ресурса. Действительно, любой URL-адрес несет достаточно информации для точного нахождения страницы. Далее в этой книге при использовании адресов сайтов будем придерживать аббревиатуры URL.

Структура адреса сайта

Вернемся к URL-адресу **http://prog-school.ru/catalog**. Его можно разделить на 3 части:

- **http://**;
- **prog-school.ru**;
- **/catalog**.

Первая часть адреса (**http://**) определяет протокол взаимодействия браузера с сервером. В нашем случае это протокол HTTP, о нем речь пойдет далее.

Вторая часть адресной строки называется *доменом* и служит для идентификации конкретного сайта с помощью службы DNS. DNS (Domain Name System, система доменных имен) — компьютерная распределенная система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства). В сети существует большое количество DNS-серверов, которые по доменному имени ресурса могут "подсказать" его реальное местоположение, определяемое IP-адресом.

Третья часть адресной строки определяет подзапрос к сайту. Это может быть путь до определенного файла или каталога на сервере. Здесь же могут быть заданы различные параметры HTTP-запроса.

Исходный код HTML-страницы

Теперь давайте разберем, что же получает браузер в ответ на сформированный HTTP-запрос. Для этого еще раз взгляните на рис. 1.1. Как видно, страница может состоять из текста, картинок, гиперссылок, полей ввода, кнопок и других элементов. Информация обо всем этом была передана от веб-сервера браузеру, который и сгенерировал конечный внешний вид страницы. Передаваемые данные описываются с помощью протокола HTML.

HTML (HyperText Markup Language, язык разметки гипертекста) — стандартный язык разметки документов в Интернете. Язык HTML интерпретируется браузером и отображается в виде документа в удобной для человека форме.

Можно сказать, что браузеры выполняют две основные функции — это взаимодействие с веб-серверами посредством HTTP-запросов, а также преобразование полученного от сервера HTML-кода в визуальное представление.

Схема клиент-серверного взаимодействия при работе браузера

Итак, давайте теперь ответим на главный вопрос этого урока: что же происходит в промежутке между тем, как мы набрали адрес сайта в браузере, и моментом, когда его содержимое отобразилось на экране.

После того как URL сайта введен в адресную строку и нажата клавиша <Enter>, браузер формирует пакет данных, который посылает по сети. Этот пакет содержит URL запрашиваемого сайта, а также прочие данные запроса, оформленные согласно протоколу HTTP. Содержание пакета нас сейчас не интересует, важно то, что переданный URL позволяет промежуточным узлам в Интернете доставить наш пакет с HTTP-запросом по адресу до нужного сервера.

На физическом веб-сервере (т. е. некоем реальном компьютере) должна быть запущена соответствующая программа, которая также называется *веб-сервером* и служит для обработки входящих HTTP-запросов. Самым популярным веб-сервером на данный момент является программа Apache, логотип которой приведен на рис. 1.2.

После получения пакета с HTTP-запросом веб-сервер определяет, какие действия необходимы для его обработки. Если HTTP-запрос осуществляется к обычной HTML-странице, то веб-сервер просто передает ее содержимое браузеру. Если же HTTP-запрос осуществляется к какому-либо сценарию (например, PHP-сценарию), веб-сервер передает запрос на обработку соответствующей программе, отвечающей за обработку этого типа сценариев. Обработчик сценария в свою очередь может вызывать другие программы в ходе своей работы, например СУБД MySQL.

Результатом работы обработчика сценариев является HTML-код, который веб-сервер посылает обратно на компьютер пользователя. Сгенерированный HTML-код веб-сервер упаковывает в HTTP-пакет, который и передается по сети обратно клиенту. Полученный HTTP-ответ попадает в браузер клиента, который извлекает из него HTML-код и генерирует на его основе графическое представление запрошенной страницы.

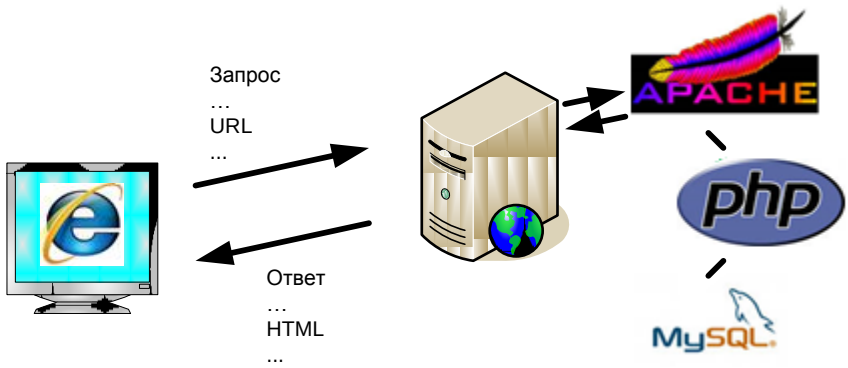


Рис 1.2. Схема взаимодействия браузера и веб-сервера

Протокол HTTP

Давайте дадим более четкое определение протоколу HTTP и разберемся, зачем он нужен.

HTTP (HyperText Transfer Protocol, протокол передачи гипертекста) — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов). Основой HTTP является технология "клиент-сервер", т. е. предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом. HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов.

Иными словами, HTTP — это язык, на котором общаются браузер и веб-сервер при обмене пакетами с данными. Обмен сообщениями в ходе HTTP-соединения идет по обыкновенной схеме "запрос—ответ"

Каждое HTTP-сообщение состоит из трех частей, которые передаются в указанном порядке:

1. Стартовая строка — определяет тип сообщения.
2. Заголовки — характеризуют тело сообщения, параметры передачи и прочие сведения.
3. Тело сообщения — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, т. к. указывает на тип запроса/ответа.

Стартовые строки HTTP-запросов и HTTP-ответов различны. Для HTTP-запроса стартовая строка имеет следующий вид:

```
МЕТОД URI HTTP/Версия
```

Здесь:

- `МЕТОД` — название запроса, одно слово прописными буквами. В версии HTTP 0.9 использовался только метод GET, список запросов для версии 1.1 представлен далее;
- `URI` определяет путь к запрашиваемому документу;
- `Версия` — пара разделенных точкой арабских цифр. Например: `1.0`. Определяет версию используемого HTTP-протокола. На данный момент наиболее актуальной является версия 1.1.

Для HTTP-ответа стартовая строка выглядит следующим образом:

```
HTTP/Версия КодСостояния Пояснение
```

Здесь:

- `Версия` — пара разделенных точкой арабских цифр как в запросе;
- `КодСостояния` — три цифры. По коду состояния определяется дальнейшее содержимое сообщения и поведение клиента;

- `Пояснение` — текстовое короткое пояснение к коду ответа для пользователя. Никак не влияет на сообщение и является необязательным.

Заголовки HTTP

Заголовки HTTP (HTTP Headers) — это строки в HTTP-сообщении, содержащие разделенную двоеточием пару "параметр: значение". Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Примеры заголовков:

```
Server: Apache/2.2.11 (Win32) PHP/5.3.0
Last-Modified: Sat, 16 Jan 2010 21:16:42 GMT
Content-Type: text/plain; charset=windows-1251
Content-Language: ru
```

В приведенном примере каждая строка представляет собой один заголовок. При этом то, что находится до первого двоеточия, называется *именем* (name), а что после нее — *значением* (value).

Все заголовки разделяются на четыре основных группы:

- General Headers (Основные заголовки) — должны включаться в любое сообщение клиента и сервера;
- Request Headers (Заголовки запроса) — используются только в запросах клиента;
- Response Headers (Заголовки ответа) — только для ответов от сервера;
- Entity Headers (Заголовки сущности) — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посылать заголовки получателю. Заголовки запроса и ответа, как и основные заголовки, описывают все сообщение в целом и размещаются только в начальном блоке заголовков, в то время как заголовки сущности характеризуют содержимое каждой части в отдельности, располагаясь непосредственно перед ее телом.

Мы не будем углубляться в изучение форматов HTTP-сообщений. На данном этапе достаточно иметь общее представление о функционировании механизмов, обеспечивающих клиент-серверное взаимодействие между браузерами и веб-серверами. Дополнительную информацию о HTTP-заголовках вы сможете найти в Википедии: **http://ru.wikipedia.org/wiki/Заголовки_HTTP**.

Как посмотреть данные HTTP-запроса и HTTP-ответа в браузере

Давайте подробнее рассмотрим содержимое пакетов, которыми обмениваются браузер и веб-сервер. Для этого нам понадобится специальная программа, которая позволит просматривать передаваемые данные. Для удобства использования будем применять к браузеру FireFox плагин, который называется Firebug. Его можно скачать по адресу **http://firebug.ru**. Если же вы пользуетесь другим веб-обозревателем,

то вам подойдет другой продукт той же компании под названием Firebug Lite (**http://getfirebug.com/firebuglite**).

После установки плагина в браузере в разделе **Инструменты** главного меню появится новый пункт **Firebug**. Выберем его, а затем в раскрывшемся подменю — пункт **Открыть Firebug**, после чего увидим окно плагина (рис. 1.3).

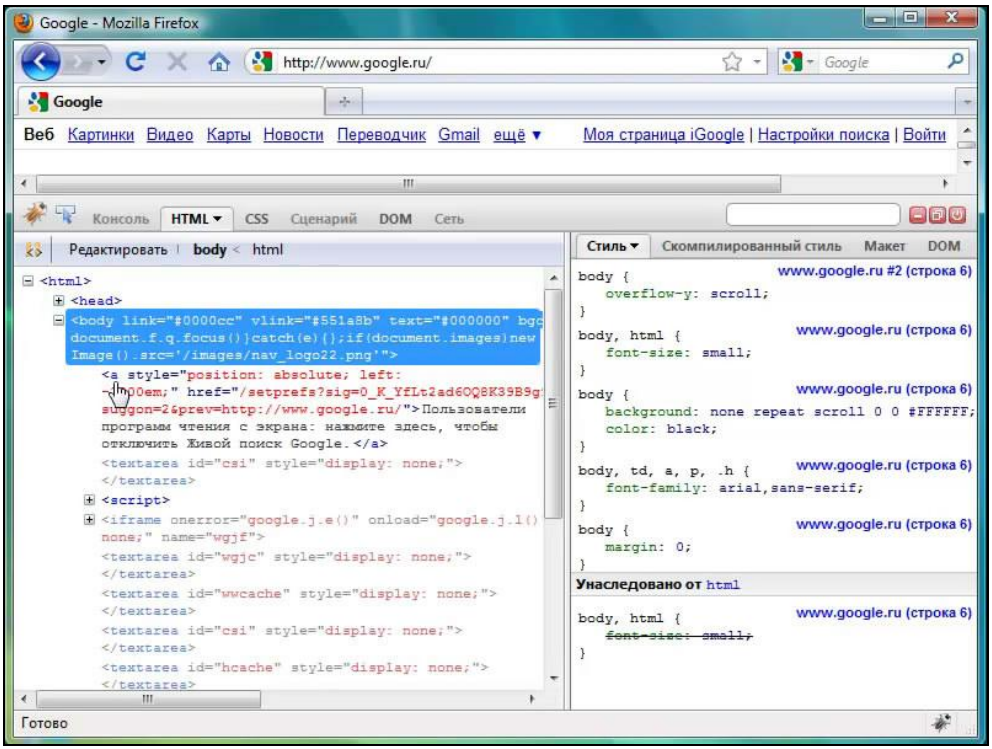


Рис. 1.3. Плагин Firebug

На вкладке **HTML** вы сможете увидеть HTML-код загруженной страницы. Обратите внимание, Firebug позволяет скрывать и раскрывать содержание отдельных элементов, из которых состоит HTML-страница.

Но нас в данный момент больше интересует содержание всего HTTP-пакета, нежели его HTML-составляющей. Чтобы увидеть непосредственное содержимое HTTP-запросов и ответов перейдем на вкладку **Сеть** (рис. 1.4). По умолчанию эта функциональность отключена. Для того чтобы ее включить, нажмите на стрелку рядом с надписью "Сеть" и выберите пункт **Панель включена**. После чего давайте откроем любую интернет-страницу и посмотрим на окно плагина Firebug.

Теперь в окне Firebug мы видим список запросов, которые совершил наш браузер. Обратите внимание, что инициатором взаимодействия между браузером и веб-сервером всегда является именно браузер. Веб-сервер же просто формирует ответы на запросы браузеров клиентов. Также обратите внимание: несмотря на то, что мы запросили всего одну веб-страницу, браузер выполнил несколько HTTP-запросов. Это связано с тем, что некоторые элементы страницы, такие как картинки, таблицы

стилей (css), JavaScript-файлы, подгружаются в отдельных запросах. Иногда для загрузки одной веб-страницы браузер выполняет более сотни запросов.

Если раскрыть любой из пунктов, щелкнув по значку +, мы увидим данные соответствующего HTTP-запроса. Они поделены на четыре категории: **Параметры**, **Заголовки**, **Ответ**, **Кэш**. Параметры у запроса могут отсутствовать, поэтому одноименной вкладки вы можете не увидеть. На закладке **Заголовки** представлены HTTP-заголовки запроса браузера и ответа сервера. Вкладка **Ответ** содержит HTML-код, полученный браузером от сервера. А на вкладке **Кэш** можно найти служебную информацию по кэшированию страницы.

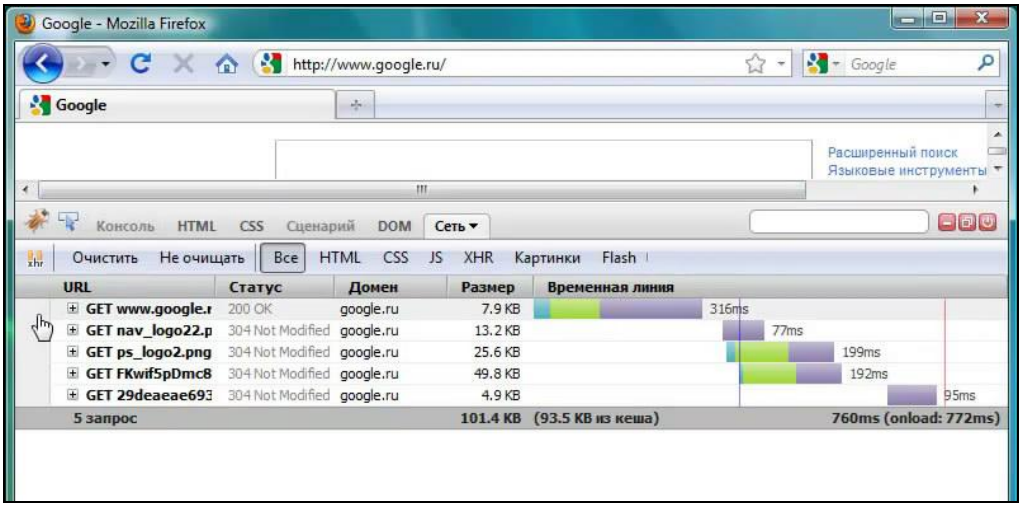


Рис. 1.4. Плагин Firebug, вкладка **Сеть**

Самостоятельно откройте несколько интернет-страниц и просмотрите отправленные и полученные данные. Скорее всего, большинство из них вам сейчас незнакомо, однако в этом нет ничего страшного. На данном этапе достаточно просто понимать, что происходит в процессе информационного обмена между браузером и веб-сервером.

Как скачать сайт утилитой Telnet

Чтобы закрепить полученные знания о протоколе HTTP, давайте познакомимся с еще одной утилитой, которая позволит трассировать (просматривать содержимое) HTTP-коммуникации. Эта утилита называется Telnet.

Строго говоря, TELNET (TErminaL NETwork) — это сетевой протокол для реализации текстового интерфейса по сети (в современной форме — при помощи транспорта TCP). Однако название "telnet" имеют также некоторые утилиты, реализующие клиентскую часть протокола.

HTTP как раз является текстовым протоколом. То есть HTTP-запросы и HTTP-ответы имеют понятный человеку вид, их можно читать в обычном текстовом ре-

дакторе. С помощью Telnet-клиента мы сможем подключиться к какому-нибудь веб-серверу, самостоятельно сформировать и отправить HTTP-запрос и увидеть содержимое HTTP-ответа.

Для запуска Telnet в ОС Windows выполните:

1. Нажмите кнопку **Пуск**.
2. В строке поиска введите `cmd` и нажмите клавишу <Enter>.
3. В консоли наберите `telnet` и нажмите клавишу <Enter>.

Перед вами появится окно программы Telnet-клиента (рис. 1.5).

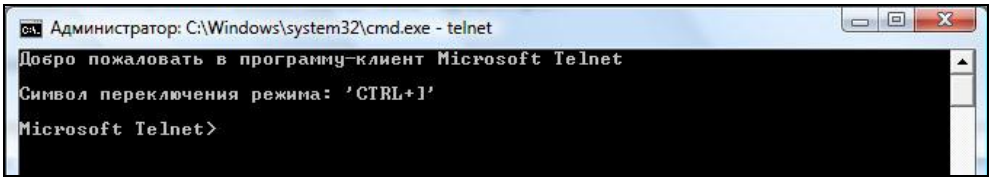


Рис. 1.5. Telnet-клиент Windows

Если же вы увидели окно с сообщением о том, что telnet является незнакомой командой для Windows, выполните следующие действия:

1. Откройте Панель управления.
2. Выберите **Программы и компоненты**.
3. Справа в окне найдите ссылку **Включение или отключение компонентов Windows** и щелкните по ней.
4. В окне **Компоненты Windows** в списке найдите **Клиент Telnet**, поставьте напротив флажок и нажмите кнопку **OK**.

Теперь, когда Telnet успешно включен, можно переходить к основным действиям. В тренировочных целях попробуем подключиться к веб-серверу Яндекса, который использует домен **ya.ru**. Однако знания одного лишь домена не достаточно для взаимодействия с веб-сервером.

Дело в том, что на одной машине в сети могут располагаться сразу несколько сервисов, которые ждут входящих сетевых соединений. Например, помимо веб-сервера на компьютере может быть запущена служба почтового сервера, FTP-сервер и т. п. Каким же образом сервер определяет, какая из программ должна обработать то или иное соединение? Для решения этой задачи используется понятие порта. *Порт* — это абстракция, характерная для проколов сетевого уровня TCP и UDP, которая представляет собой номер, выделяемый сетевому приложению для связи с приложениями на других хостах. Таким образом, за каждым сетевым приложением закреплен уникальный порт, т. е. некий номер, который фигурирует в передаваемых пакетах с данными и позволяет однозначно идентифицировать приложение, которому адресован каждый конкретный пакет.

Веб-серверы обычно используют порт с номером 80. Поэтому для соединения с Яндексом введите следующую строчку и нажмите клавишу <Enter> (рис. 1.6):

```
telnet ya.ru 80
```

Рис. 1.6. Telnet-соединение с веб-сервером

Если вы увидите пустой черный экран и мигающий курсор — все в порядке. Подключение прошло успешно, и сервер ждет вашего запроса.

Теперь давайте попробуем получить от сервера главную страницу сайта **ya.ru**. Для этого нам потребуется передать стартовую строку HTTP-соединения, а также необходимые HTTP-заголовки.

Для начала введите стартовую строку и нажмите клавишу <Enter>:

```
GET / HTTP/1.1
```

Пусть вас не смущает, что ничего не отображается на экране, это особенность Telnet-клиента в Windows. Из данной стартовой строки видно, что мы используем для HTTP-запроса метод GET, хотим получить корневой элемент сайта, о чем говорит URI-путь "/", а также используем HTTP-протокол версии 1.1.

Теперь введите HTTP-заголовок:

```
Host: www.ya.ru
```

Несмотря на то, что самому Telnet-клиенту мы уже указали, что соединение должно быть установлено с хостом **ya.ru**, требования HTTP-протокола обязывают задавать заголовок `Host` в каждом HTTP-запросе. Иначе в ответ мы получим ошибку "Bad request".

Теперь *дважды* нажмите клавишу <Enter>. Двойное нажатие требуется из-за того, что помимо заголовков сообщения мы также можем задать его тело. Но в нашем случае тело сообщения пустое, и мы просто еще один раз нажимаем клавишу <Enter>.

После второго нажатия Telnet-клиент выполнит HTTP-запрос к указанному серверу и получит HTTP-ответ. В нашем случае вы должны увидеть на экране результат, похожий на тот, что представлен на рис. 1.7.

Как видно из рисунка, HTTP-ответ веб-сервера состоит из стартовой строки, семи HTTP-заголовков и HTML-кода страницы.

Если вы все сделали правильно, то стартовая строка HTTP-ответа будет иметь вид:

```
HTTP/1.1 200 OK
```

`200` — это код состояния HTTP-ответа, `OK` — пояснение кода состояния. Данная строчка говорит о том, что веб-сервер успешно выполнил обработку нашего HTTP-запроса.

Таким образом, мы только что научились запрашивать и получать привычные нам веб-страницы не с помощью браузера, а через Telnet-клиент, который отображает данные HTTP-ответа веб-сервера.

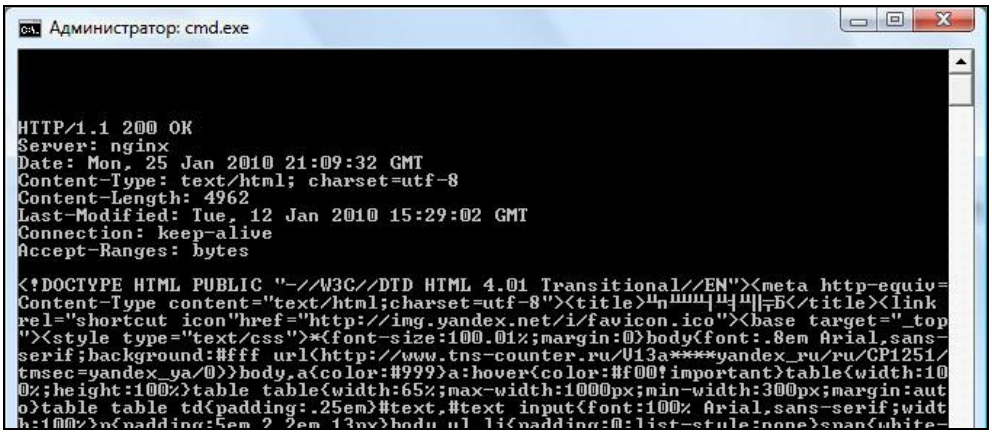


Рис. 1.7. HTTP-ответ от хоста ya.ru

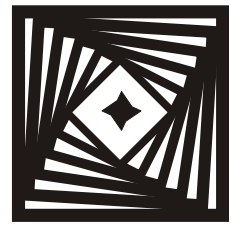
Резюме

После прочтения данного урока и повторения практических примеров вы должны четко понимать, как взаимодействуют браузер и веб-сервер при запросе той или иной интернет-страницы. Если вы еще не выполнили действия, которые мы описали по ходу этого урока, обязательно сделайте их. Только повторенные на собственной практике знания можно считать окончательно усвоенными. А знания этого урока — это важнейший фундамент всего дальнейшего изучения не только PHP, но и других технологий и языков веб-программирования.

Также хотим обратить ваше внимание на огромную роль самообразования для любого программиста. Каким бы не был объем книги, она не сможет включить в себя всех знаний по теме веб-разработки. Поэтому возьмите себе за правило самостоятельно более глубоко изучать интересующие вас моменты из каждого урока. Возможно, в этом уроке вас заинтересовало значение различных HTTP-заголовков, которые вы могли наблюдать в Telnet-клиенте или Firebug. Не ленитесь, найдите информацию о них в Интернете, это вовсе не сложно. Как показывает практика, наибольших успехов в программировании добиваются те, кто с удовольствием занимается самообразованием.

Задания

1. Скачайте и установите для своего браузера плагин, позволяющий просматривать данные HTTP-запросов и ответов, например Firebug. Выполните несколько вызовов веб-страниц с включенным плагином. Изучите отправленные и принятые данные.
2. Скачайте какую-нибудь веб-страницу с помощью Telnet-клиента. Откройте ту же страницу в браузере и просмотрите ее исходный код. Совпадает ли он с тем, что вы увидели в Telnet-клиенте. Почему? Постарайтесь обосновать ответ.



УРОК 2

Подготовка рабочего места

В этом небольшом уроке мы наконец-то закончим подготовку к изучению PHP-программирования и перейдем от слов к делу, написав свой первый PHP-сценарий. Но прежде чем это сделать, нам необходимо установить пакет программ, которые обеспечат удобную разработку на протяжении всего курса обучения.

Как вы уже знаете из первого урока, для функционирования PHP-сценариев необходим веб-сервер и PHP-интерпретатор. Давайте займемся установкой этих программ на локальный компьютер, чтобы мы смогли разрабатывать и тестировать сайты без необходимости их размещения в Интернете. Запомните, что любой разработчик сначала работает с локальной версией сайта или сценария и лишь после успешного тестирования выкладывает новые версии сценариев на реальный сайт.

При серьезной разработке принято организовывать сразу три площадки: для разработки, для тестирования и "боевую", т. е. реально функционирующую систему. Однако для наших целей хватит и одной площадки — локального компьютера, где мы и будем проводить все эксперименты с нашими PHP-сценариями.

Установка пакета Денвер

После изучения *урока 1* вы должны знать, что для полноценной разработки нам потребуется установить как минимум веб-сервер и PHP-интерпретатор. А также, что очень желательно, какую-нибудь СУБД (систему управления базами данных), чтобы мы могли пользоваться функционалом работы с базой данных в своих сценариях.

Мы же предлагаем заменить разрозненную установку вышеперечисленных программ установкой всего одной программы, а точнее пакета программ под названием Денвер. Здесь нет никакого чуда, Денвер — это просто программа-инсталлятор и настройщик всех необходимых инструментов для локальной веб-разработки. Он незаменим на первых этапах обучения веб-программированию, когда ученик еще не обладает глубоким пониманием механизмов функционирования веб-сервера и его компонентов.

Бытует мнение, что Денвер плох тем, что не позволяет провести тонкую настройку компонентов веб-сервера. Это мнение ошибочно, т. к. все что делает Денвер — это

инсталлирует необходимые компоненты и настраивает их взаимодействие. Вы всегда сможете внести необходимые корректировки в настройки любой из составляющих системы, будь то веб-сервер, PHP-интерпретатор или СУБД.

Пакет Денвер позволяет единовременно установить на своем локальном компьютере такие необходимые программы, как веб-сервер Apache, интерпретатор PHP и СУБД MySQL. Стоит заметить, что это наиболее популярная связка программ, которая используется для работы миллионов реально существующих сайтов.

Итак, скачать этот замечательный установщик пакета программ можно по адресу: **http://www.denwer.ru/** (рис. 2.1).

Рис. 2.1. Сайт **denwer.ru**

При нажатии кнопки **Скачать Денвер** вам предложат выбрать версию PHP. Советуем выбирать самую последнюю версию дистрибутива (рис. 2.2).

Рис. 2.2. Выбор версии PHP

После этого нажмите кнопку **Скачать** и в появившейся форме введите свои контактные данные.

По указанному адресу электронной почты вы получите письмо со ссылкой для скачивания дистрибутива. Скачайте предложенный файл, сохраните в любое удобное место на локальном компьютере и запустите его. Сначала вы увидите окно распаковки архива (рис. 2.3), а после окно с предложением начать установку Денвера (рис. 2.4).

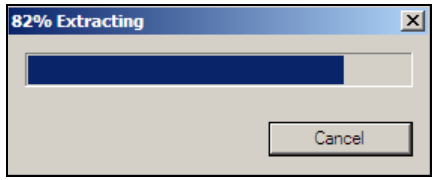


Рис. 2.3. Распаковка архива Денвера

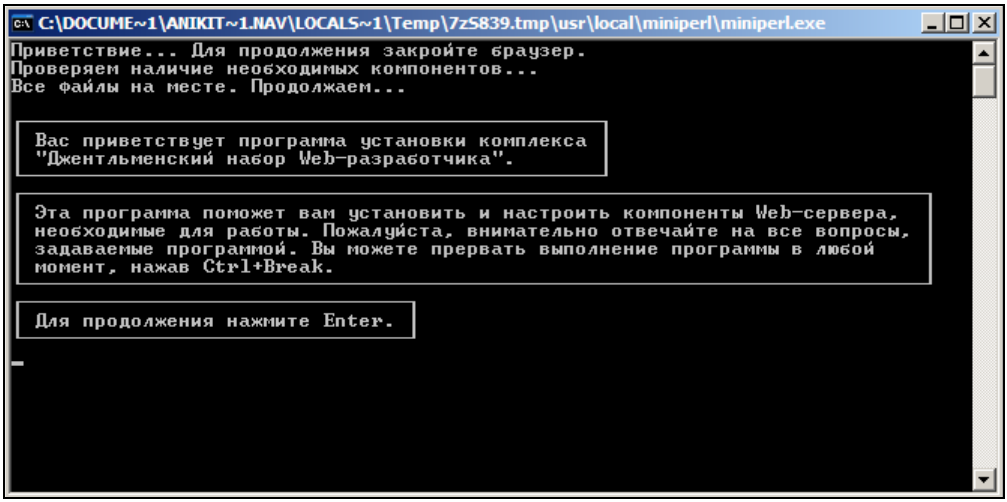


Рис. 2.4. Окно установки Денвера

Последовательно ответьте на вопросы установщика. Рекомендуем при вашей первой установке Денвера по возможности оставлять значения по умолчанию. Если в процессе инсталляции у вас возникли какие-либо сложности, советуем обратиться к видеоуроку с нашего сайта, в котором подробно описан процесс установки:

http://prog-school.ru/2010/02/kak-ustanovit-veb-server-i-napisat-pervyj-php-skript/

После успешного завершения установки вы должны увидеть окно браузера (рис. 2.5).

На своем рабочем столе вы найдете три ярлыка (рис. 2.6).

Это своеобразный пульт управления вашим веб-сервером. Ярлык **Start Denwer** позволяет запустить все программы веб-сервера, **Stop Denwer** — завершить эти программы, **Restart Denwer** — перезапустить их.

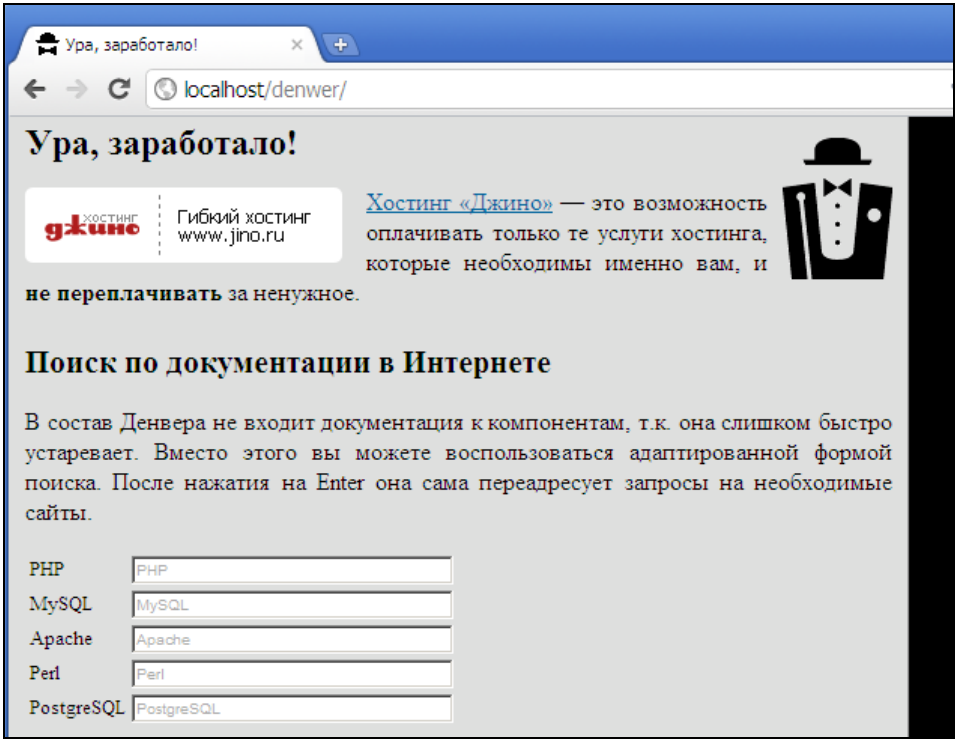


Рис. 2.5. Завершение установки Денвера

Рис. 2.6. Ярлыки Денвера

Структура папок Денвера

Прежде чем приступить к изучению архитектуры каталогов нашего веб-сервера, перейдите на рабочий стол и дважды щелкните по ярлыку **Start Denwer**. Вы должны увидеть следующее окно, подобное представленному на рис. 2.7.

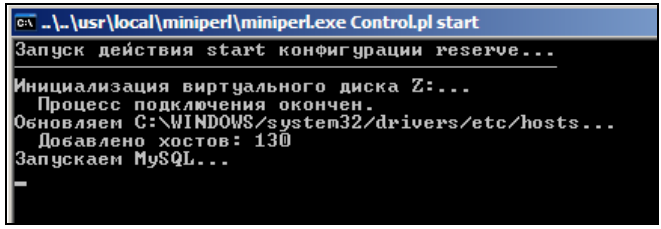


Рис. 2.7. Запуск Денвера

Теперь давайте разберемся, как устроена архитектура каталогов Денвера и куда нам в будущем потребуется выкладывать свои файлы сценариев, чтобы увидеть их работу на локальном компьютере.

Обратите внимание, что после запуска Денвер должен был создать виртуальный диск на вашем компьютере. При установке вы задавали его букву. По умолчанию это была буква "Z". Зайдите в окно **Компьютер** и убедитесь, что такой диск появился (рис. 2.8). Откройте содержимое этого диска. Вы должны увидеть перечень каталогов, как показано на рис. 2.9.

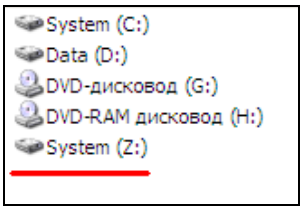


Рис. 2.8. Виртуальный диск веб-сервера

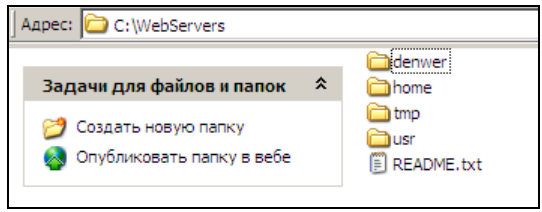


Рис. 2.9. Содержимое диска веб-сервера

Обратите внимание, что созданный диск не является физическим, это лишь виртуальное представление обычной папки на компьютере, необходимое для функционирования веб-сервера.

При установке Денвер просил выбрать вас папку на жестком диске, где будут располагаться все компоненты веб-сервера. По умолчанию был предложен каталог C:\WebServers (рис. 2.10). Давайте зайдем в этот каталог либо в ту папку, которую вы указали в качестве требуемого пути в ходе установки. В ней вы найдете все то же содержимое нашего виртуального диска.

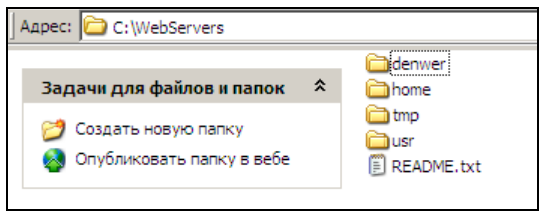


Рис. 2.10. Содержимое папки C:\WebServers

Вам должно быть ясно, что каталог C:\WebServers и виртуальный диск Z: — это фактически одно и то же, а не две копии одной и той же папки.

Мы не будем углубляться в изучение архитектуры папок, в которых находятся файлы веб-сервера Apache, интерпретатора PHP, СУБД MySQL и т. д. На начальном этапе нас интересует принцип создания сайтов на локальном компьютере.

Главный интерес для нас представляет папка home. В ней мы и будем располагать будущие сайты. Для каждого нового сайта будем использовать отдельный каталог внутри папки home. При этом обратиться к новому сайту в браузере мы сможем по адресу **http://<*название_подкаталога_в_папке_home*>.**

Обратите внимание, что на конце адреса нет привычного ".ru" или ".com". Это отличие наших локальных сайтов. Для обращения к ним мы не должны добавлять в конце адреса дополнительные домены (например, ".ru").

Предположим, что мы завели в папке home подкаталог с именем helloworld. Это значит, что к страницам нашего нового локального сайта мы будем обращаться следующим образом: **http://helloworld/*<путь_внутри_сайта>*.**

Все сценарии и статичные HTML-страницы локального сайта требуется располагать внутри подкаталога www нашего локального сайта. Это связано с настройками маршрутизации Денвера. На данном этапе достаточно просто запомнить это правило.

Таким образом, если мы хотим создать станицу index.html для нашего сайта helloworld, то на жестком диске мы должны расположить ее по адресу C:\WebServers\home\helloworld\www (если не менялись пути по умолчанию). А обратиться в браузере к такой странице можно по адресу **http://helloworld/index.html**.

Предлагаем вам прямо сейчас создать предложенную структуру каталогов. В каталог C:\WebServers\home\helloworld\www\ поместите файл index.html, состоящий из одной строчки:

```
Hello, world!
```

После выполнения предложенных действий откройте браузер и перейдите по адресу **http://helloworld/index.html** и... убедитесь, что ничего не работает (рис. 2.11).

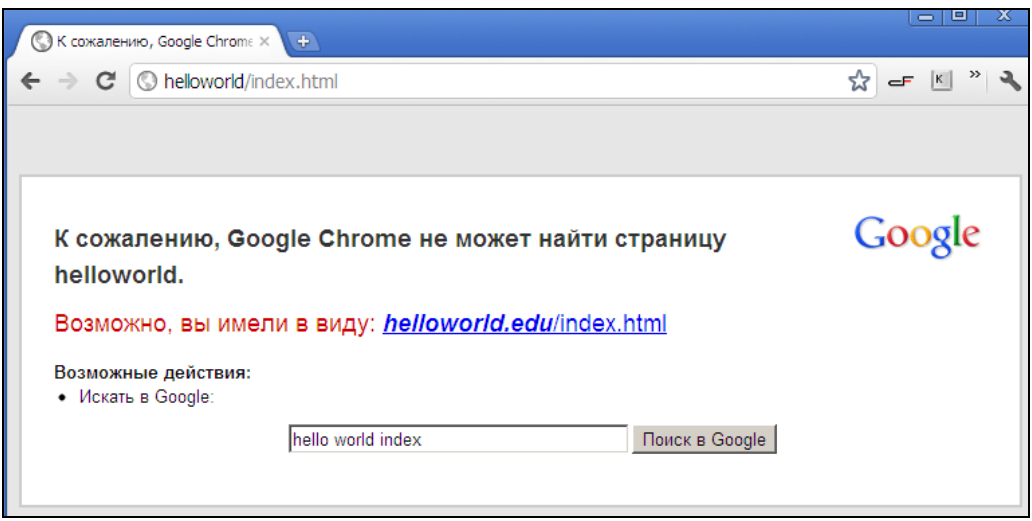


Рис. 2.11. Ошибка при попытке вызова страницы

Дело в том, что веб-сервер еще "не знает" о том, что вы создали папку с новым сайтом. Чтобы Apache "узнал" об этом, перейдите на рабочий стол и дважды щелкните по ярлыку **Restart Denwer**. После перезагрузки веб-сервера попробуйте снова. На этот раз вы должны увидеть правильный результат (рис. 2.12).

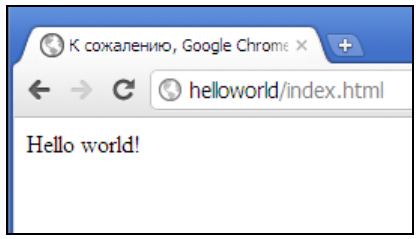


Рис. 2.12. Тестовая HTML-страница

Подводя промежуточный итог, отметим то, что вам следует запомнить:

- все сайты создаются внутри каталога home вашего веб-сервера;
- на каждый сайт отводится отдельная папка, имя которой совпадает с названием сайта в адресной строке браузера;
- для вызова локальных сайтов не нужно приписывать в конце адреса домены первого уровня (".ru", ".com" и т. п.);
- все файлы сайта должны располагаться в каталоге www;
- после создания нового сайта требуется перезапустить веб-сервер.

Выбор текстового редактора

Прежде чем написать свой первый PHP-сценарий, у вас должен возникнуть вопрос: какую программу использовать для написания кода?

Вообще говоря, набирать код вы можете и в стандартном редакторе Блокнот, никто вам этого не запретит. Но разработчики уже давно придумали более удобные инструменты для программирования. В то же время на первом этапе не следует погружаться в изучение громоздких "навороченных" редакторов, т. к. 95% их функциональности вам все равно не пригодится.

Для написания первых PHP-сценариев мы рекомендуем пользоваться бесплатной программой Notepad++. Она имеет целый ряд преимуществ, которые позволяют смело советовать ее в качестве редактора кода:

- она бесплатна;
- ее легко найти и установить. Достаточно воспользоваться ссылкой **http://notepad-plus-plus.org/download/**;
- поддерживается синтаксис PHP. Это значит, что редактор будет подсвечивать различными цветами конструкции языка, что очень удобно и наглядно;
- наконец, она проста. И это является несомненным достоинством программы. Так как хороший редактор должен быть в первую очередь *незаметен* во время написания кода и не должен отвлекать на себя драгоценного внимания программиста.

Установите Notepad++. Для этого перейдите по ссылке **http://notepad-plus-plus.org/download/** и скачайте самую последнюю версию программы.

Теперь, когда редактор установлен, у нас все готово для создания своего первого PHP-сценария.

Первый PHP-сценарий

Перейдите в папку C:\WebServers\home\helloworld\www\. Удалите файл index.html и вместо него создайте файл index.php. Далее откройте его в редакторе Notepad++ и наберите код из листинга 2.1. Не волнуйтесь из-за того, что пока этот код не ясен вам. Его смысл станет вам понятен в ходе изучения этой книги. На данный же момент наша задача состоит в том, чтобы просто убедиться, что PHP-сценарии исполняются на вашем локальном компьютере.

Листинг 2.1. Первый PHP-сценарии

```
<?php
  echo "Hello, world!";
  echo "<br/>";
  echo "Текущее время на сервере: " . date("H:i:s");
  echo "<br/>";
  echo "IP адрес клиента: " . $_SERVER['REMOTE_ADDR'];
?>
```

После этого сохраните файл и перейдите в браузере по ссылке **http://helloworld/index.php**. Если все сделано правильно, то вы должны увидеть страницу, как на рис. 2.13.

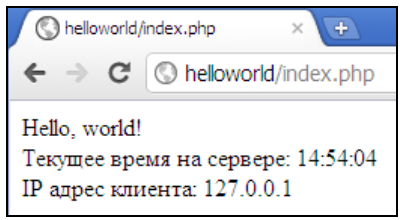


Рис. 2.13. Первый PHP-сценарий

Если вы увидели аналогичную страницу, мы поздравляем вас! Только что вы написали свой первый PHP-сценарий. Кроме того, вы убедились, что веб-сервер исправно работает на вашем локальном компьютере. А это значит, что нет никаких препятствий к дальнейшему успешному изучению PHP!

Резюме

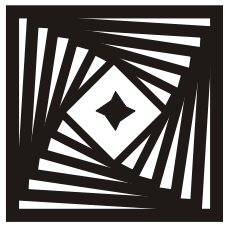
Установка веб-сервера на локальный компьютер — это необходимое требование для дальнейшего обучения. Если у вас возникли проблемы в процессе установки, посмотрите обучающие видео, которое можно скачать по ссылке **ftp://85.249.45.166/9785977506786.zip** либо на сайте **http://prog-school.ru**.

Обязательно установите программы, которые мы предложили в этом уроке. Наши рекомендации выработаны многолетним опытом программирования и обучения программированию. Поэтому предложенные программы — это лучшие инструменты разработки на вашем этапе обучения.

Начиная с этого урока и до самого окончания книги, настоятельно рекомендуем собственноручно переписывать листинги программ на свой локальный компьютер. Профессионализм и опыт программиста характеризуется количеством написанного им кода. Переписывая и разбирая листинги этой книги, вы закладываете первые кирпичики в надежный фундамент своей будущей профессии.

Задания

1. Скачайте и установите пакет Денвер согласно инструкциям из этой главы.
2. Скачайте и установите текстовый редактор Notepad++.
3. Напишите свой первый PHP-сценарий, повторив листинг, который мы привели ранее. Убедитесь, что результат выполнения вашего сценария совпадает с результатом, описанным в данной главе.



УРОК 3

Введение в HTML

В современной практике веб-разработки считается грамотным подход, когда оформление веб-страницы отделяют от логики ее формирования. Иными словами выделяют две важные задачи: определить, *что* выведется в браузере пользователя в ответ на его запрос и то, *как* эта информация будет визуально представлена.

Как мы уже знаем из предыдущего урока, HTML — это язык разметки документов в Интернете. С помощью этого языка и решается задача о том, *как* переданная от сервера информация должна выглядеть в браузере.

Грамотное оформление интернет-документов является отдельной сферой деятельности и называется *версткой веб-страниц*. В рамках же этой книги нас будет больше интересовать непосредственно веб-программирование, т. е. определение и реализация логики выполнения веб-сценариев, задача которых — предоставить некоторую информацию пользователю. Поэтому в данной главе нам будет достаточно изучить лишь основные принципы организации HTML-страниц, чтобы иметь минимально необходимые знания о форматировании интернет-документов в браузерах.

Создание HTML-документа

Для создания HTML-документа подойдет любой текстовый редактор, в том числе Блокнот Windows. Но мы порекомендовали бы использовать специализированный редактор HTML-кода, в котором намного удобнее создавать собственные страницы. Предлагаем использовать редактор ScITE (или Notepad++), минималистичный, но очень практичный и для первых экспериментов — самое оно.

Теперь давайте попробуем создать свою первую веб-страницу. Нет ничего страшного в том, что мы еще ничего не понимаем в HTML-коде. Просто самостоятельно перепишите в редакторе приведенный в листинге 3.1 код. Как он работает, будем разбираться чуть позже.

Совет

Старайтесь всегда самостоятельно переписывать листинги, приведенные в книге. Квалификация программиста определяется не количеством прочитанных им книг, а

объемом написанного кода. Чем больше кода вы будете писать, тем грамотнее и профессиональнее он будет становиться. Изучение теории, не подкрепленной никакой практикой, абсолютно бесполезно.

Листинг 3.1. Первый HTML-документ

```
<html>
  <head>
    <title>Первая веб-страничка</title>
  </head>
  <body>
    Какой-нибудь текст
  </body>
</html>
```

Теперь файл надо сохранить, чтобы просмотреть результат в браузере. Сохраняем файл, указывая расширение html, тогда операционной системе будет понятно, что это именно веб-страница. Давайте назовем ее primer1.html.

Чтобы посмотреть результат, нужно найти сохраненный файл и два раза щелкнуть по нему. Операционная система, поняв по расширению файла, что вы хотите открыть веб-страницу, запустит браузер и передаст ему адрес нашего файла (рис. 3.1).

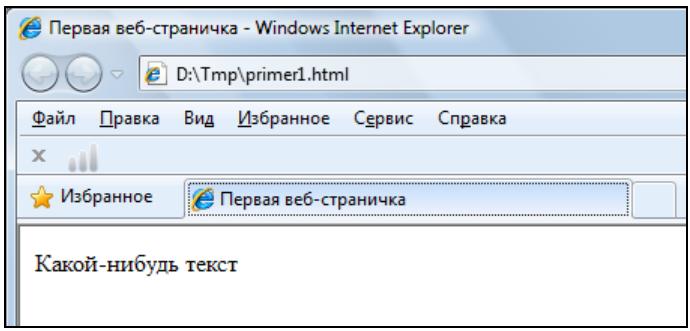


Рис. 3.1. Первая веб-страничка

Поздравляем! Только что вы написали свою первую веб-страницу. Как видите, это оказалось не так уж и сложно. Давайте исследуем, из чего же состоит типичная HTML-страница. Для этого нам в первую очередь необходимо познакомиться с понятием HTML-тегов.

Теги HTML

Тег — это ключевой элемент во всей концепции HTML. Теги используются для оформления текста, который пользователь видит в браузере. Согласитесь, современные веб-сайты не очень-то похожи на текст в MS Word или в книге. Чтобы браузер понимал, что имеет дело не с простым текстом, а с особым элементом, ко-

торый задает его форматирование, и применяются теги. Синтаксис написания тегов следующий.

```
<название_тега параметр1="значение1" параметр2="значение2" ...>
```

либо

```
<название_тега параметр1="значение1" параметр2="значение2" ...>...</
название_тега>
```

Теги бывают одиночными и парными (контейнеры). Одиночный тег используется самостоятельно, а парный может включать внутри себя другие теги или текст. Теги могут иметь различные параметры, которые разделяются между собой пробелом. Встречаются также теги без каких-либо дополнительных параметров. Параметры можно разделить на обязательные и необязательные. Первые должны всегда присутствовать для корректной трактовки тега браузером. Вторые используются при необходимости.

Вот некоторые из особенностей тегов, которые следует запомнить.

- Теги не чувствительны к регистру. Их можно записывать как прописными, так и строчными символами. Однако рекомендуем выработать для себя правила и придерживаться их по мере создания всех страниц. Хорошим тоном считается написание тегов *строчными буквами*.
- Атрибуты тега не обязательно должны находиться на одной строке. Однако лучше все-таки избегать переносов, иначе код будет плохо читаться.
- Значения атрибутов тегов следует брать в *двойные кавычки*, иначе возможна неправильная интерпретация кода браузером.
- Неизвестные теги браузер пропускает, как будто их нет.
- Существует порядок следования тегов, которого следует придерживаться. В частности `<head>` должен идти перед `<body>`, `<meta>` и `<title>` должны находиться между тегами `<head>` и `</head>`. Об этом мы подробнее поговорим в следующем уроке.
- Порядок следования параметров внутри тега не принципиален.

Структура HTML-страницы

Все веб-страницы имеют общие элементы, которые присутствуют вне зависимости от содержимого сайта. Эти элементы описывают базовые составляющие HTML-страницы. Давайте рассмотрим канонический вид HTML-кода (листинг 3.2).

Листинг 3.2. Структура HTML-страницы

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta content="text/html; charset=Windows-1251"
          http-equiv="content-type">
```

```
    <title>Вторая веб-страничка</title>
  </head>
  <body>
    ...
  </body>
</html>
```

Разберем каждую строчку кода.

Тег DOCTYPE описывает тип HTML-документа. На первых порах достаточно запомнить, что первой строчкой ваших HTML-страниц всегда должна быть:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Тег <html> — начало содержимого HTML-файла. Любая страница содержит этот тег. <html> состоит из заголовка (<head>..</head>) и тела документа (<body>..</body>).

Тег <head> может включать текст и теги, но содержимое этого раздела не показывается напрямую на странице.

Тег <meta> используется внутри <head>. Он предоставляет целый класс возможностей, таких как установка кодировки страницы, добавление ключевых слов, описание документа и многое другое. Например, для того чтобы браузер понимал, что мы пишем страничку на русском языке, в примере используется тег

```
<meta content="text/html; charset=Windows-1251"
      http-equiv="content-type">
```

Также в <head> присутствует тег <title>, который определяет заголовок веб-страницы. Обычно заголовок отображается в строке заголовка или на вкладке браузера (см. рис. 3.1).

<head> является парным тегом и требует закрывающего тега — </head>, который определяет окончание заголовка документа.

Далее следует блок <body>..</body>, который несет в себе содержательную часть документа. В нашем примере содержательная часть ограничивается тегом переноса строки
, который не требует соответствующего закрывающего тега.

Документ заканчивается последовательностью </body></html>. На этом браузер прекращает обработку HTML-кода.

Вот и вся структура HTML-файла. Все остальное разнообразие управляющих элементов, таблиц, списков, ссылок, картинок, элементов форматирования и т. д. создается за счет применения различных тегов внутри блока <body>...</body>.

Это справочная информация, которой здесь не имеет смысла уделять много времени. Лучше наберите в поисковике "справочник html" и скачайте понравившийся, чтобы он всегда был под рукой.

Работа с текстом

Одна из основных задач HTML — это наглядное представление текста страницы. Поэтому давайте рассмотрим основные способы форматирования текста в HTML. Еще раз напомним, что приведенные в книге примеры крайне желательно самостоятельно переносить на свой компьютер и проверять результат их работы.

Абзацы

Абзацы позволяют форматировать текст блоками, как мы все привыкли его видеть в книгах. Между абзацами вставляется небольшой вертикальный отступ.

Абзац начинается с тега и заканчивается соответствующим закрывающим тегом .

В листинге 3.3 приведен пример использования абзацев. Поэкспериментируйте с ним на собственной странице.

Листинг 3.3. Использование абзацев в HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta content="text/html; charset=Windows-1251"
          http-equiv="content-type">
    <title>Использование абзацев</title>
  </head>
  <body>
    <p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem
    accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab
    illo inventore veritatis et quasi architecto beatae vitae dicta sunt
    explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit
    aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem
    sequi nesciunt.</p>
    <p> Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet,
    consectetur, adipisci velit, sed quia non numquam eius modi tempora
    incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad
    minima veniam, quis nostrum exercitationem ullam corporis suscipit
    laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum
    iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae
    consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla
    pariatur?</p>
  </body>
</html>
```

Переносы строк

Для простого переноса строки в тексте используется тег
. Он, в отличие от , не требует закрывающего тега, а также не создает вертикального отступа.

Пример использования:

```
Sed ut perspiciatis unde omnis iste natus <br/> error sit voluptatem
accusantium doloremque laudantium...
```

Заголовки

Несмотря на то, что размером текста можно управлять с помощью разных вариантов форматирования, очень желательно использовать заголовки на своей странице. Поисковики считают текст внутри заголовков более значимым, что позволяет быть выше в результатах поиска по ключевым словам.

Чем важнее заголовок, тем больше размер шрифта (листинг 3.4). Самым верхним уровнем является уровень 1 (<h1>), а самым нижним — уровень 6 (<h6>).

Листинг 3.4. Создание заголовков в HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta content="text/html; charset=Windows-1251"
          http-equiv="content-type">
    <title>Заголовки</title>
  </head>
  <body>
    <h1>Введение в HTML</h1>
    <h2>Ключевые понятия</h2>
    <h2>Основные теги</h2>
    <h3>Форматирование текста</h3>
    <h3>Оформление ссылок</h3>
  </body>
</html>
```

Оформление текста: полужирный, курсив, подчеркивание

Три операции, которые идут рука об руку в большинстве текстовых редакторов. В HTML они реализуются с помощью следующих парных тегов:

```
<b>Жирный текст</b>
<i>Курсивный текст</i>
<u>Подчеркнутый текст </u>
```

Ссылки

Ссылки позволяют переходить в браузере с одной страницы на другую с помощью простого щелчка мыши. Переход может осуществляться не только на HTML-страницы, но и на файлы любого типа. Ссылка определяется адресом документа, на который мы ссылаемся.

В HTML ссылки оформляются с помощью тега `<a>`, который имеет обязательный параметр `href`. Значением `href` и будет адрес документа или, другими словами, URL.

Существуют два вида ссылок — абсолютные и локальные. *Абсолютные* начинаются с указания протокола и включают адрес сайта в сети (листинг 3.5).

Листинг 3.5. Пример абсолютной ссылки

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta content="text/html; charset=Windows-1251"
          http-equiv="content-type">
    <title>Ссылки</title>
  </head>
  <body>
    <a href="http://www.prog-school.ru">Школа Программирования</a>
  </body>
</html>
```

В основном абсолютные ссылки применяются, когда нужно обратиться к другому сайту.

Второй тип ссылок — *локальные*, используются для указания на ресурсы того же сайта, на котором находится страница. В этом случае у нас нет необходимости указывать полный путь к странице, будет достаточно лишь относительного пути.

Предположим, у нас в одной папке есть два файла: page1.html и page2.html. И мы хотим в первую страницу встроить ссылку на вторую. Получится код, как в листинге 3.6.

Листинг 3.6. Пример локальной ссылки

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta content="text/html; charset=Windows-1251"
          http-equiv="content-type">
    <title>Ссылки</title>
  </head>
  <body>
    <a href="page2.html">Перейти ко второй странице</a>
  </body>
</html>
```

Если же файлы располагаются в разных папках, то при указании адреса ресурса следует использовать последовательность из двух точек "..", которая обозначает

родительский каталог. Предположим, у нас в одном каталоге есть две папки с именами first и second. В первой находится файл page1.html, во второй — page2.html.

Тогда чтобы со страницы page1.html обратиться к page2.html потребуется следующая ссылка:

```
<a href="../second/page2.html">Перейти ко второй странице</a>
```

".." позволяют сначала перейти в родительский каталог, а уже из него попасть в папку second, где и находится интересующая нас страница.

Отметим, что ".." можно применять несколько раз подряд, если требуется подняться по каталогам на несколько уровней вверх.

Еще один полезный атрибут тега `<a>` — параметр `target`. Он определяет, требуется ли открыть новое окно при переходе по ссылке. Для отображения ссылки в новом окне `target` нужно установить в значение `"_blank"`. Пример:

```
<a href="http://www.prog-school.ru" target="_blank">Школа Программирования</a>
```

По умолчанию новые страницы загружаются в текущее окно браузера.

Изображения

Для показа изображений на странице используется тег `<img>`. Обязательным параметром является `src`, значение которого определяет адрес файла с изображением. Адрес изображения абсолютно аналогичен адресу ссылки и подчиняется тем же правилам записи.

В листинге 3.7 приведен пример, в котором использованы абсолютная и локальная ссылки на изображение. Предполагается, что в папке с HTML-страницей есть каталог img, в котором находится файл img.gif.

Листинг 3.7. Вывод изображения в HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta content="text/html; charset=Windows-1251"
          http-equiv="content-type">
  </head>
  <body>
    <img src="http://www.htmlbook.ru/images/sample.gif"> — абсолютный адрес
размещения изображения
    <img src="img/img.gif"> — адрес размещения изображения относительно
текущего HTML-документа
  </body>
</html>
```

В справочниках можно найти информацию о других параметрах тега `<img>`.

Наиболее полезными могут стать параметры:

- `alt` — альтернативный текст изображения (выводится в случае, если загрузка картинки невозможна);
- `width`, `height` — размеры изображения. В качестве значения задается количество пикселов;
- `border` — толщина рамки вокруг изображения. В качестве значения задается количество пикселов;
- `align` — параметр выравнивания изображения. Может принимать значения:
  - `bottom` — выравнивание нижней границы изображения по окружающему тексту;
  - `left` — выравнивает изображение по левому краю окна;
  - `middle` — выравнивание середины изображения по базовой линии текущей строки;
  - `right` — выравнивает изображение по правому краю окна;
  - `top` — верхняя граница изображения выравнивается по самому высокому элементу текущей строки.

Кстати, изображения можно использовать в качестве ссылки. Для этого нужно расположить тег `<img>` внутри блока `<a>..</a>`. Пример попробуйте написать сами.

Таблицы

Последней важной составляющей HTML-разметки являются таблицы. Использование таблиц — это один из способов верстки сайтов, т. е. формирования контента в соответствии с макетом (рисунком дизайнера). Таблицы состоят из строк и ячеек. Содержимое ячеек может быть абсолютно любым.

Несмотря на то, что в современной верстке для форматирования страниц применяются другие теги, таблицы по-прежнему очень часто используются, т. к. являются удобным и наглядным средством представления информации.

Создание таблиц

Все содержимое таблицы помещается в блок тегов `<table>..</table>`. Элементами таблицы являются строки, которые задаются тегами `<tr>..</tr>`. Содержимым строк являются ячейки, которые задаются тегами `<td>..</td>`. Внутри ячейки может размещаться любой HTML-код, будь то текст, ссылки, картинки или что-то другое (листинг 3.8).

Листинг 3.8. Пример страницы с таблицей

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
```

```
  <body>
    <table>
      <tr>
        <td>1-я ячейка 1-й строки</td>
        <td>2-я ячейка 1-й строки</td>
      </tr>
      <tr>
        <td>1-я ячейка 2-й строки</td>
        <td>2-я ячейка 2-й строки</td>
      </tr>
    </table>
  </body>
</html>
```

Параметры таблиц

Основными параметрами таблиц являются:

- `align` — выравнивание таблицы;
- `bgcolor` — цвет фона таблицы;
- `border` — толщина рамки таблицы;
- `bordercolor` — цвет рамки таблицы;
- `height` — высота таблицы;
- `width` — ширина таблицы;
- `cellpadding` — расстояние между границей ячейки и ее содержимым;
- `cellspacing` — расстояние между ячейками таблицы.

Параметры ячеек

Среди параметров ячеек выделим:

- `align` — выравнивание содержимого ячейки по горизонтали;
- `valign` — выравнивание содержимого ячейки по вертикали;
- `bgcolor` — цвет фона ячейки;
- `bordercolor` — цвет рамки вокруг ячейки;
- `height` — высота ячейки;
- `width` — ширина ячейки;
- `colspan` — число ячеек, которые должны быть объединены в одну по горизонтали. Например, может использоваться для оформления общей шапки таблицы;
- `rowspan` — число ячеек, которые должны быть объединены в одну по вертикали.

Особенности таблиц

В заключение обзора средств HTML-разметки остановимся на некоторых особенностях таблиц:

- таблицы могут образовывать вложенность. То есть одна таблица может быть помещена внутрь ячейки другой. Это очень полезное свойство при верстке сайта с помощью таблиц;
- если размеры таблицы не заданы отдельно, то они вычисляются на основе содержимого ячеек. То есть таблица растягивается в зависимости от наполнения;
- если у таблицы указан атрибут `border` и не указан `bordercolor`, то рамка таблицы будет трехмерной. Указание `bordercolor` убирает трехмерность.

XML и XHTML

Самый популярный язык описания данных — это XML. С его помощью можно описать любую древовидную структуру, например, группы и студенты (листинг 3.9).

Листинг 3.9. Описание данных с помощью XML

```
<university>
  <group name="IU7-11">
    <student name="Lyapin" sex="male" />
    <student name="Kotova" sex="female" />
  </group>
  <group name="IU7-12">
    <student name="Mogelashvili" sex="male" />
    <student name="Ivko" sex="male" />
  </group>
  <group name="IU7-12">
    <student name="Nikitin" sex="male" />
    <student name="Mishanin" sex="male" />
  </group>
</university>
```

Язык очень прост. Описание, как и в HTML, состоит из тегов. Тег может содержать атрибуты и дочерние теги. Каждый тег должен быть закрыт. Так: `<group></group>` или так: `<student />`.

Не правда ли, это очень похоже на HTML? Отличие в том, что классический HTML допускает различные вольности, например, тег переноса строки можно записать как `<br>` и как `<br/>`. Первая запись не соответствует формату XML. Любой тег должен быть закрыт!

HTML допускает значения атрибутов в двойных кавычках, в одинарных и без кавычек:

```
<a href="index.php">На главную</a>
<a href='index.php'>На главную</a>
<a href=index.php>На главную</a>
```

XML допускает запись значений атрибутов только в *двойных* кавычках! XHTML — это HTML, который удовлетворяет формату XML. В этой книге мы не делаем акцент на верстке, но лучше сразу привыкать описывать содержимое страницы на языке XHTML. Такое описание считается "валидным", т. е. правильным, грамотным.

Резюме

На данный момент у вас должно было сложиться понимание принципов формирования HTML-документов. Все остальное изучение HTML сводится к исследованию новых тегов и их параметров. Повторимся, что это справочная информация, ее не нужно учить, достаточно иметь хороший справочник HTML и уметь быстро находить в нем интересующие данные. С опытом все само собой уложится в голове.

Также важно понимать, что этот урок не содержит исчерпывающей информации по верстке. Эта тема настолько обширна, что для ее изучения стоит прочесть отдельную книгу. Форматирование документа с помощью таблиц считается сейчас устаревшим плохим стилем. Хорошая верстка в современном понимании основывается на использовании тегов `<div>` и каскадных таблицах стилей (Cascading Style Sheets, CSS). Так почему же мы рассказали о таблицах и не слова о CSS? Потому что HTML в рамках данной книги — лишь инструмент вывода результатов работы PHP-сценария. Нам важно, чтобы вы имели определенный минимум знаний по HTML, и теперь он у вас есть!

Обратите внимание на форматирование HTML-документов в листингах урока. Заметьте, что каждый новый уровень вложенности элементов сопровождается дополнительными отступами относительно левого края. Это делается для того, чтобы текст хорошо читался, и человек мог с легкостью найти интересующее его место в коде. Большинство начинающих программистов совершенно забывает о необходимости аккуратно оформлять код. Это приводит к тому, что они начинают путаться в своих программах и наживают себе ненужную головную боль. Запомните: аккуратность при оформлении кода программы *очень важна*!

Выработайте дисциплину по оформлению кода, и эти усилия в дальнейшем окупятся сторицей. Мы посвятили немало статей теме оформления сценариев и программ в рубрике "Культура кода" на сайте "Школы программирования": **http://prog-school.ru/category/cultured-code/.**

Задания

1. Выберите и установите у себя на компьютере редактор HTML-кода.
2. Найдите в Интернете HTML-справочник, скачайте его себе. Он станет вашим постоянным помощником на время освоения веб-программирования.

3. Создайте HTML-страницу, в которой используйте ВСЕ теги, встречающиеся в этом уроке. Мы предлагаем сделать страницу по макету на рис. 3.2.

   Для верстки страницы используйте таблицу (все содержимое страницы должно быть помещено в ячейки таблицы). Ячейки таблицы должны быть отделены друг от друга линиями.

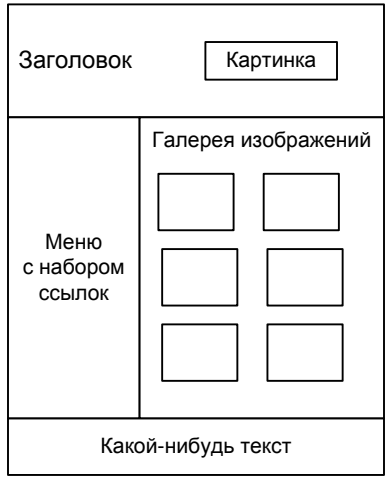


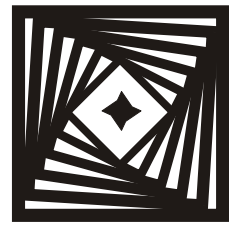
Рис. 3.2. Макет HTML-страницы

   Ссылки меню слева должны быть рабочими и вести на любые другие страницы (пример: **yandex.ru**).

   Дополнительное задание: при щелчке по картинке она должна открываться в отдельном окне браузера.

4. Создайте собственный сайт на бесплатном хостинге в Интернете и разместите там свою страницу. Видеоурок о том, как создать сайт на бесплатном хостинге, вы можете найти у нас на сайте по ссылке: **http://prog-school.ru/2010/02/kak-sozdat-sajt-na-besplatnom-xostinge/**.

   Расскажите о своем сайте друзьям и знакомым. Этот элемент публичности позволит вам более серьезно и осознанно подходить к процессу собственного обучения и не бросить все на полпути. Если уж взялись изучать веб-программирование, так доведите начатое до конца и не стесняйтесь показывать свои результаты окружающим.



УРОК 4

Основы PHP

PHP представляет собой мощный язык написания сценариев для веб и поддерживает невероятно большой диапазон интернет-технологий. Это делает его на сегодняшний день ведущим языком сценариев для веба.

PHP-сценарий (или PHP-скрипт) — это набор инструкций для программы PHP-интерпретатора на сервере. Результатом работы PHP-сценария обычно является HTML-код, который возвращается клиенту в браузер. Тем не менее PHP-интерпретатор может выполнять не только генерацию HTML-кода, но и другие полезные действия, например работать с базой данных или файловой системой сервера.

Прежде чем приступить к изучению PHP...

На данный момент у вас должно быть четко сформировано представление о том, как "работает Интернет", о чем мы рассказали в *уроке 1*. Кроме того, чтобы выполнять задания, приведенные в этом и последующих уроках, вам необходимо установить веб-сервер и PHP-интерпретатор на свой локальный компьютер. Об этом рассказывалось в *уроке 2*.

> ***Примечание***
>
> За более подробной информацией по поводу интерпретируемых и компилируемых языков советуем обратиться на сайт Школы программирования по этой ссылке: **http://prog-school.ru/2010/02/raznica-mezhdu-interpretiruemymi-i-kompiliruemymi-yazykami-programmirovaniya/**.

Базовый синтаксис

Свой первый PHP-сценарий мы уже написали в рамках *урока 2*. Приведем его еще раз здесь, но теперь несколько упростим (листинг 4.1).

Листинг 4.1. Простейший PHP-сценарий

```
<?php
  echo "Hello!";
?>
```

Теперь давайте подробнее рассмотрим, из чего же обычно состоит PHP-сценарий. Все PHP-сценарии пишутся в виде блоков кода. Эти блоки при необходимости могут быть встроены в HTML и обычно определяются с помощью строки `<?php` в начале и конструкции `?>` — в конце.

Весь остальной код, который находится вне этих идентификаторов, интерпретатор PHP игнорирует и передает обратно веб-серверу для отображения в браузере на стороне клиента.

Для иллюстрации видоизмените свой файл index.php так, как представлено в листинге 4.2.

Листинг 4.2. Сценарий PHP внутри HTML-кода

```
<html>
  <head><title>Первый PHP-сценарий</title></head>
  <body>
    Пример работы php:<br/>
    <?php
      echo "Hello!";
    ?>
  </body>
</html>
```

После этого зайдите на свой локальный сайт и посмотрите результат (рис. 4.1).

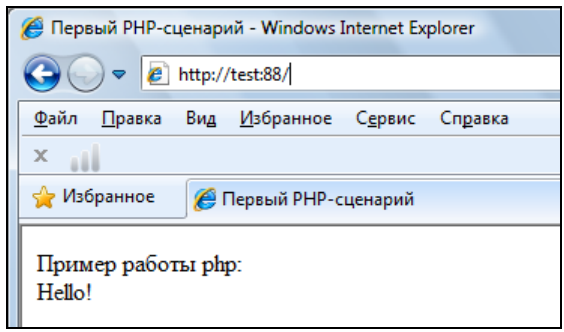


Рис. 4.1. Первый PHP-сценарий

Здесь вы знакомитесь с первым оператором PHP `echo`. Этот оператор представляет собой базовый способ для отправки клиенту некоторого текста из PHP-сценария. Оператор `echo` будет интенсивно использоваться на протяжении всего курса.

Как и во всех Си-подобных языках, в PHP каждое предложение завершается *точкой с запятой*, как в этой сроке:

```
echo "Hello!";
```

Несмотря на то, что в предыдущем листинге мы использовали PHP-код, этот сценарий не выполняет ничего такого, что не могло бы быть сделано средствами стандартного HTML. Чтобы написать что-нибудь более полезное, нам придется научиться пользоваться переменными PHP.

Переменные

Переменная — это атрибут PHP-сценария, который может изменять свое значение по ходу выполнения сценария PHP-интерпретатором. Программист может задавать значение переменных, изменять и считывать эти значения, а также выстраивать логику выполнения сценария в зависимости от них.

В PHP имена переменных всегда начинаются с символа `$` и содержат произвольную комбинацию символов при условии, что первый символ после `$` будет буквой или знаком подчеркивания. К допустимым символам относятся строчные и прописные латинские буквы, цифры, символ подчеркивания ("_"), а также символы с ASCII-кодами в диапазоне от 127 до 255.

Переменную можно назвать по-русски, например, "строка", но мы *категорически не рекомендуем так делать*. Согласно программистской этике переменные должны носить ясные англоязычные названия, максимально четко определяющие их собственный смысл. Иными словами, из названия переменной должно быть сразу понятно ее предназначение: для чего программист ее ввел и как собирается использовать.

Давайте рассмотрим несколько примеров хороших и плохих названий переменных:

- `$kkk` — плохое название. Не понятно предназначение переменной;
- `$cars_count` — хорошее название. По названию можно сделать вывод, что переменная должна хранить количество машин;
- `$carscount` — плохое название. Если название состоит из нескольких слов, нужно каким-либо образом их разделять.

Существует множество различных нотаций и правил именования переменных, функций и прочих функциональных атрибутов сценариев и программ. На данный момент нам не требуется глубоко погружаться в эту область, достаточно выработать простое правило именования переменных: разделение слов подчеркиванием.

Переменные в PHP могут быть определены либо присвоением им значения, либо с помощью оператора `var`. Измените PHP код своей страницы так, как представлено в листинге 4.3.

Листинг 4.3. Работа с переменными

```
<?php
  $string_var = "string variable"; // строковая переменная
```

```
  $int_var = 5;                    /* числовая переменная */
  echo "Hello! " . $string_var . " " . $int_var;
?>
```

Перезапустите страницу, и вы увидите, что в браузере отобразилась строчка "Hello! string variable 5". Здесь в третьей строке мы вывели значения двух переменных, которые объявили в первых двух строчках.

Комментарии

Наверняка в предыдущем примере вы заметили еще незнакомые нам конструкции `/*`, `*/` и `//`. Эти последовательности позволяют вставлять комментарии в текст сценария.

В PHP все, что находится между конструкциями `/*` и `*/`, трактуется как *комментарий*, который используется для пояснений в теле сценария, и игнорируется интерпретатором. Для однострочных комментариев применяются символы `//`, которые комментируют весь текст до конца строки. То есть все, что написано на строке правее символов `//`, будет игнорироваться интерпретатором.

Следует отметить, что комментарии должны стать обязательным атрибутом ваших PHP-сценариев. Хорошо комментированный код понимается в разы быстрее и проще, чем некомментированный. У начинающих программистов часто возникает соблазн отказаться от комментариев в пользу быстроты написания сценария. Это очень опасное заблуждение! Если вы, как программист, работаете над достаточно сложным проектом, а не над тривиальным сайтом-визиткой, то спустя некоторое время после начала разработки вместе с увеличением объема написанного кода значительно сложнее станет в нем ориентироваться. Даже если вы являетесь единоличным его автором. Что уж говорить о совместной разработке или ситуациях, когда программисту приходится разбираться в коде своих коллег. В данном случае только хорошо прокомментированный код позволит в десятки раз снизить трудозатраты на его понимание и дальнейшее использование.

Поэтому никогда не пренебрегайте комментариями. Не стоит впадать в крайности и комментировать каждую строчку, однако старайтесь писать достаточно комментариев для того, чтобы вы могли с легкостью разобраться в коде и через неделю, и через год.

Возвращаясь к предыдущему листингу, отметим, что в третьей строчке вы могли увидеть использование оператора "точка" — `"."`. Он служит для конкатенации (т. е. склеивания) строк. О нем подробнее мы поговорим чуть позже.

Константы

Константы представляют собой контейнеры для данных, как и переменные, но после присваивания константе значения его уже нельзя изменить на всем протяжении выполнения сценария. Константы создаются в PHP с помощью функции `define()`.

Возможно, вам не до конца понятно, что такое функция и как ее использовать, тогда просто запомните приведенный далее листинг, как пример создания константы.

```
define("INDEPENDENCEDAY", "12th June");
```

В данном примере мы создали константу с именем `INDEPENDENCEDAY`, которая имеет значение `"12th June"`.

В теле сценария на константу можно сослаться просто по имени:

```
echo(INDEPENDENCEDAY);
```

Обратите внимание, имя константы не содержит лидирующего символа `$`, в отличие от переменных.

Принято записывать имена констант буквами верхнего регистра, хотя возможно выбирать любые имена, отвечающие правилам именования переменных.

Константы PHP фактически действуют так же, как директива `#define` в препроцессоре Си: можно определить их в некотором месте, а затем выполнять разный код в зависимости от того, определена ли константа и какое значение она имеет. Проверка выполняется с помощью функции `defined()`.

Так как на данный момент мы еще не изучили понятия функций и условных операторов в PHP, то приведенный листинг 4.4 может быть вам не до конца ясен. Не переживайте, по мере освоения материала этой книги понимание правил использования констант появится само собой.

Листинг 4.4. Проверка существования константы

```
if (defined("INDEPENDENCEDAY"))
{  echo ("INDEPENDENCEDAY is defined");  }
else
{  echo("INDEPENDENCEDAY is not defined");  }
```

Константы нужны достаточно редко. Использовать их имеет смысл для того, чтобы не "зашивать" в код абсолютные пути к файлам, IP-адреса, какие-то числовые значения и т. д. Жестко заданные значения в теле сценария часто указывают на низкое качество кода. Вместо этого следует задавать такие значения в переменных или константах, если логика сценария требует, чтобы данное значение не менялось по ходу его выполнения.

Кроме того, константы часто выносят в отдельный файл, который подключается множеством исполняемых сценариев.

Хорошим примером использование константы было бы хранение в ней пути к файлу, в который пишется журнал событий. Или же имя базы данных и пароль для доступа к ней. Обратите внимание, что эта информация может потребоваться совершенно разным сценариям. И было бы в высшей степени неверно "жестко" прописывать такого рода конфигурационные данные в каждом сценарии отдельно. Вместо этого правильным решением будет подключение специального файла (од-

ного на всю систему), в котором в качестве констант заданы все необходимые конфигурационные параметры.

Типы данных

Как вы могли заметить из предыдущих примеров, с помощью переменных в PHP мы можем оперировать совершенно различными данными: целыми числами, дробными числами, текстовыми строками и т. д. Этот факт показывает нам, что у каждой переменной PHP есть свой тип (например, целое число, строка и т. п.).

С точки зрения типизации переменных PHP можно классифицировать, как свободно-типизированный язык. Это значит, что изначально нет необходимости задавать определенный тип переменной. Вместо этого, когда ей присваивается значение, PHP трактует ее соответствующим образом в зависимости от значения и контекста, в котором она применяется. В PHP существуют четыре простых типа переменных (целые, строковые, действительные с плавающей точкой, булевы) и два сложных типа (объекты и массивы). В этом уроке мы будем работать только с простыми типами.

Первый тип данных, который будет представлен, — это целые числа. *Целые числа* — фундаментальный числовой тип PHP, представляющий значения со знаком величиной до чуть более 2 миллиардов. На практике PHP воспринимает целые значения с использованием трех математических представлений: десятичные, восьмеричные и шестнадцатеричные. В большинстве ситуаций PHP-сценарии пишутся в десятичной нотации. То есть так, как привычно вести счет любому человеку. Однако в некоторых случаях восьмеричные и шестнадцатеричные числа могут существенно облегчить жизнь.

В листинге 4.5 показано, как использовать числа в разных нотациях.

Листинг 4.5. Использование чисел в разных нотациях

```
<?php
  // Стандартная десятичная нотация
  $my_int_10 = 50;

  // То же число в восьмеричной нотации (начинается с нуля)
  $my_int_8 = 062;

  // Шестнадцатеричная нотация (начинается с 0x)
  $my_int_16 = 0x32;
?>
```

Когда PHP работает с дробными числами, он представляет значения в виде типа данных с *плавающей точкой*. Числа с плавающей точкой — это любые числа, которые содержат десятичную дробную часть и могут быть выражены в десятичном или экспоненциальном представлении (листинг 4.6).

Листинг 4.6. Представление дробных чисел

```
<?php
  // Стандартная нотация с десятичной точкой
  $my_float = 5.1;

  // То же число в экспоненциальном представлении с плавающей точкой
  $my_float = .051e2;
?>
```

Еще один базовый тип данных, о котором говорится в этой главе, — *строковый*. Для начала отметим, что существуют два типа строк. Первый тип строк определяется с использованием *двойных кавычек*, в то время как второй тип строк представляется в *одинарных кавычках*.

Разница между этими строками в том, что если внутри строки в двойных кавычках написать имя переменной, то интерпретатор PHP подставит значение этой переменной при результирующем выводе. В случае же со вторым типом строк (одинарные кавычки) подстановка не происходит. Для лучшего понимания напишите сценарий из листинга 4.7 и посмотрите результат.

Листинг 4.7. Использование строковых переменных

```
<?php
  $my_int = 50;
  $string_one = "Значение переменной равно $my_int<br/>";
  $string_two = 'Значение переменной равно $my_int<br/>';
  echo $string_one;
  echo $string_two;
?>
```

Если этот сценарий исполнить, его вывод будет таким:

```
Значение переменной равно 50
Значение переменной равно $my_int
```

Также вы можете обратить внимание, что обе строки содержат тег HTML переноса строки `<br/>`. Поскольку вы отправляете выходной текст веб-браузеру, это необходимо для отображения упомянутых двух значений на разных строках экрана.

Последний из базовых типов, о которых говорится в данной главе, это булев (или логический) тип данных. Переменные этого типа могут принимать только два варианта значения: истина и ложь. Для этого используются ключевые слова `true` и `false` соответственно. Пример объявления переменных такого типа приведен в листинге 4.8.

Листинг 4.8. Пример объявления булевых переменных

```
<?php
  $param1 = true;   // истина
  $param2 = false;  // ложь
?>
```

Переменные принимают булев тип в результате выполнения логических операций, о которых мы поговорим позже в этом уроке.

Преобразование типов

Преобразование типов используется для того, чтобы указать интерпретатору на необходимость работать со значением переменной одного типа так, как если бы она имела другой тип. Оператор преобразования типа — это имя типа, заключенное в круглые скобки:

- `(string)` — строка;
- `(integer)` — целое число;
- `(double)` — число с плавающей точкой;
- `(boolean)` — булева переменная (`true`/`false`).

У некоторых есть также сокращенные версии:

- `(int)` — сокращение `(integer)`;
- `(bool)` — сокращение `(boolean)`.

Пример использования приведен в листинге 4.9.

Листинг 4.9. Пример преобразования типа переменной

```
<?php
  $a = "432.123";
  echo((int)$a);
?>
```

Этот код выведет "432", потому что переменная `$a` будет восприниматься как целое число (с типом `int`), а дробная часть будет отброшена.

В PHP преобразование типов используется довольно редко, однако о такой возможности не следует забывать.

Операторы и операции

Арифметические операции

Основные операции, предназначенные в PHP для выполнения математических действий, всем хорошо знакомы. Они аналогичны операциям из обычной арифметики (листинг 4.10).

Листинг 4.10. Арифметические операции PHP

```
<?php
  $answer = 5 + 4;              // сложение
  $answer = $answer – 5;        // вычитание
  $answer = $answer / 2;        // деление
  $answer = $answer * $answer; // умножение
  $answer = $answer % 3;        // остаток от деления
?>
```

Отдельно следует обратить внимание на операцию определения остатка от деления `%`. Точное его определение звучит так: остатком от деления называется неотрицательное число, которое в сумме с произведением неполного частного и делителя дает делимое.

Примеры:

```
$answer = 5 % 3; // $answer будет равен 2
$answer = 4 % 3; // $answer будет равен 1
$answer = 3 % 3; // $answer будет равен 0
```

В операциях сложения и деления нужно помнить одну важную особенность, связанную с типами операндов. Если оба аргумента — целые числа, то результат также будет целым числом. В то же время, если хотя бы один из операндов — число с плавающей точкой, то и результат будет величиной с плавающей точкой, даже если получается целое число. Пример: `0.5 + 1.5 = 2.0`, а не 2. То есть переменная, в которую мы поместим результат операции `0.5 + 1.5`, будет вещественным числом с плавающей точкой.

Для этих пяти операций также существуют сокращенные версии в случае, когда одним из операндов в правой части выражения выступает переменная, которой присваивается значение (переменная из левой части выражения). Например, чтобы не писать `$a = $a + $b`, можно сокращенно записать `$a += $b`. А для того чтобы удвоить значение переменной `$a`, достаточно написать `$a *= 2` (то же самое, что `$a = $a * 2`).

```
$a += 2; // аналогично выражению $a = $a + 2
$a -= 2; // аналогично выражению $a = $a – 2
$a *= 2; // аналогично выражению $a = $a * 2
$a /= 2; // аналогично выражению $a = $a / 2
$a %= 2; // аналогично выражению $a = $a % 2
```

Для увеличения или уменьшения значения переменной на единицу есть еще две сокращенных операции: инкремента (`++`) и декремента (`--`). В отличие от рассмотренных ранее, это унарные операции. То есть для выполнения соответствующей операции требуется только один операнд. В применении этих унарных операций есть одна тонкость, связанная с их расположением перед операндом или после него. Напишите и выполните сценарий из листинга 4.11.

Листинг 4.11. Использование унарной операции

```
<?php
  $a = 1;
  echo($a++."<br>");
  $a = 1;
  echo(++$a);
?>
```

В результате выполнения должны быть выведены два числа:

```
1
2
```

Как видите, расположение унарной операции относительно операнда может иметь значение. В первом случае переменная `$a` увеличивается на единицу уже после выполнения команды вывода `echo`. Напротив, во втором случае сначала `$a` становится равной 2, а уже потом происходит вывод с помощью `echo`.

Старайтесь избегать влияния этой особенности на логику выполнения своих сценариев, т. к. такого рода нюансы могут быть неочевидны и малозаметны.

Логические операции

Еще одну группу операций для работы с числами представляют операции сравнения:

- меньше <;
- меньше или равно <=;
- больше >;
- больше или равно >=;
- равно ==;
- не равно !=;
- идентично ===;
- не идентично !==.

Все они сравнивают два заданных значения и возвращают логическое булево значение `true` или `false`. Пример представлен в листинге 4.12.

Листинг 4.12. Пример использования логических операций

```
<?php
  $a = 3 < 4;
  echo($a);
?>
```

В случае, если значение `$a` верно, сценарий выведет единицу, иначе — не выведет ничего. Это связано с особенностью представления логических переменных в PHP.

Отдельно следует рассмотреть операции "идентично" (===) и "не идентично" (!==). Они аналогичны операциям "равно" (==) и "не равно" (!=) с тем лишь дополнением, что при сравнении учитываются также типы операндов. Логику работы операторов продемонстрирует пример из листинга 4.13.

Листинг 4.13. Применение операций "идентично" и "не идентично"

```
<?php
  $a = (2 === 2.0); /* $a будет равно false,
                       т. к. сравниваются значения разных типов */
  $a = (2 === 2);   /* $a будет равно true, т. к. сравниваются
                       значения с одинаковыми типами */
  $a = (2 !== 2.0); /* $a будет равно true */
  $a = (2 !== 2);   /* $a будет равно false */
  $a = (2 !== 3);   /* $a будет равно true, т. к. значения различны */
?>
```

Операции над строками

Наиболее важной операцией над строками является операция их объединения (*конкатенации*, от англ. *concatenate*). В наших примерах она уже не раз встречалась, и мы будем пользоваться ею постоянно. Для конкатенации строк PHP использует символ точки (". "). Пример приведен в листинге 4.14.

Листинг 4.14. Использование конкатенации строк

```
<?php
  $a = "Hello, ";
  $b = "World";
  $c = $a . $b;
  echo($c);
?>
```

По аналогии с арифметическими операциями есть сокращенная операция для конкатенации — .=. Следующая запись:

```
$a .= $b;
```

эквивалентна строке:

```
$a = $a . $b;
```

С помощью оператора "точка" можно конкатенировать несколько строк:

```
$d = $a . $b . "some text" . $c;
```

Язык PHP предоставляет богатый арсенал функций для работы со строками, но т. к. обсуждение функций у нас еще впереди, то дальнейшие средства обработки строк оставим на потом.

Логические операторы

Логические операторы проверяют булевы условия. Операндами в данном случае должны выступать переменные с булевым типом данных. Если же в выражении участвует переменная с другим типом (например, целочисленная), то произойдет автоматическое приведение типа этой переменной к булеву. Обратите внимание: это не значит, что переменная изменит свое значение и тип. Это значит только то, что для вычисления результата выражения будет использоваться не непосредственное значение нашей переменной, а ее приведенное к булеву типу значение.

Существуют четыре главных булева оператора: И (`and` или `&&`), ИЛИ (`or` или `||`), НЕ (`not` или `!`) и исключающее ИЛИ (`xor`).

Давайте разберемся, как работает каждый из них. В табл. 4.1 в колонках `AND`, `OR`, `XOR` представлены результаты выполнения соответствующих операторов для разных вариантов операндов.

Таблица 4.1. *Результаты работы логических операторов*

Операнд 1	Операнд 2	and	or	xor
True	True	True	True	False
True	False	False	True	True
False	False	False	False	False

Если с операторами И и ИЛИ обычно не возникает сложностей в понимании, то работа оператора "исключающее ИЛИ" не так очевидна. Запомните, оператор "исключающее ИЛИ" будет возвращать истину тогда, когда один из операндов равен истине, а другой ложен. В случае, когда оба операнда истинны, оператор `XOR` возвращает `false` (т. е. ложь), в отличие от обычного оператора `OR` (логическое ИЛИ).

Давайте немного попрактикуемся в использовании логических операторов. Напишите самостоятельно следующий код и постарайтесь определить значение каждой переменной, не просматривая правильный ответ, который приведен далее.

```
$a = true;
$b = false;
$c = $a && $b;
$d = $a || $b;
$e = !$a;
$f = $a xor $b;
```

В данном случае получим следующие значения:

- `$c` — `false` (один из операндов ложный);
- `$d` — `true` (один из операндов истинный);
- `$e` — `false` (противоположное значение к `$a`);
- `$f` — `true` (один из операндов истинный, в то время как второй — ложный).

Приоритетность операторов

Когда операторы образуют последовательность, например,

```
$c = $a++ + 2 <= --$b * 4 + $e && $f;
```

то все они выполняются согласно приоритетности. В любом справочнике по PHP можно найти порядок приоритетов операторов. Тем не менее мы настоятельно не рекомендуем допускать таких последовательностей, т. к. они сильно снижают читабельность кода.

Всегда используйте скобки, для облегчения дальнейшего понимания собственного сценария. В нашем случае может получиться что-то вроде:

```
$c = ($a++ + 2) <= ((--$b * 4) + ($e && $f));
```

Согласитесь, здесь, по крайней мере, с первого взгляда видно, что `$c` получает результат операции сравнения.

Далее приведем список операций по убыванию приоритета. Операции из одного пункта имеют равный приоритет и в выражении выполняются слева направо:

1. `++ --`.
2. `! (int) (float) (string)`.
3. `* / %`.
4. `+ —` ..
5. `< <= > >=`.
6. `== != === !==`.
7. `&&`.
8. `||`.
9. `= += -= *= /= .= %=`.
10. `and`.
11. `xor`.
12. `or`.

Не стоит забывать, что это отнюдь не все операторы PHP. Здесь приведены только те, что знакомы нам на данный момент. Полный список приоритетов операторов вы сможете найти по ссылке:

http://www.php.ru/manual/language.operators.html#language.operators.precedence

Резюме

В этом уроке вы познакомились с языком PHP и его ключевыми моментами. Теперь вы умеете объявлять переменные и использовать константы. Вы знаете, что язык PHP свободно типизирован, что, кстати, очень удобно для тех задач, которые он решает. Это вы обязательно оцените в дальнейшем.

Мы не рассмотрели *битовые операции*, поскольку применяются они крайне редко, а тема отнюдь не самая простая. Однако имейте в виду, что они есть и PHP свободно может их осуществлять.

Информации, освещенной в уроке, пока недостаточно для создания полноценного сценария, но запаситесь терпением! Скоро начнется реальная практика!

Задания

Перед выполнением домашнего задания ознакомьтесь со статьей "3 правила расстановки пробелов" по адресу: **http://prog-school.ru/2010/01/3-pravila-rasstanovki-probelov**. Пишите код, руководствуясь этими правилами!

1. С помощью оператора `echo` выведите на страницу:

   а) строковую переменную;

   б) целочисленную переменную;

   в) переменную с дробного типа;

   г) константу;

   д) число в восьмеричной нотации;

   е) число в шестнадцатеричной нотации.

2. Повторите вывод, заключив переменные в двойные кавычки (`"`). Посмотрите, что получится.

3. Повторите вывод, заключив переменные в одинарные кавычки (`'`). Посмотрите, что получится.

4. Выведите в восьмеричной системе числа от 10 до 20 (`echo 010; echo 011; echo 012; ... echo 019; echo 020;`) Объясните результат.

5. Выведите 16 чисел в шестнадцатеричной системе, так чтобы в браузере отобразилось "0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15".

6. Выведите на экран любое четверостишие. Пример:

   Я помню чудное мгновенье:
   Передо мной явилась ты,
   Как мимолетное виденье,
   Как гений чистой красоты.

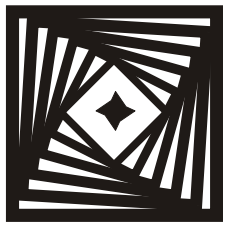
   А. С. Пушкин

   Для каждой новой строки используйте отдельный оператор `echo`. Каждая строчка должна быть отдельной строковой переменной. Также необходимо использовать переводы строки. После четверостишия поставьте инициалы автора и его фамилию и выделите их курсивом.

7. Выполните эти же действия с помощью одного оператора `echo`.

8. Попробуйте в выражении использовать разные типы, например, сложите число 10 и строку "привет". Что получится? Объясните результат.

9. Дайте ответ на вопрос, как работает оператор `xor`? В каких случаях он возвращает значение `true`, в каких — `false`? Для этого напишите сценарий, который выводит значения операций со всеми возможными вариантами операндов (4 варианта). Чему равно `$a xor $a` для любых значений `$a`?



УРОК 5

Ветвления и функции

Ветвление программы

Пожалуй, ни один сколько-нибудь сложный сценарий невозможно представить без ветвления. Они позволяют менять логику выполнения программы в зависимости от входных данных. В PHP существует группа условных операторов, которые и помогают разветвлять код программы. Для лучшего понимания концепции ветвления взгляните на две блок-схемы (рис. 5.1).

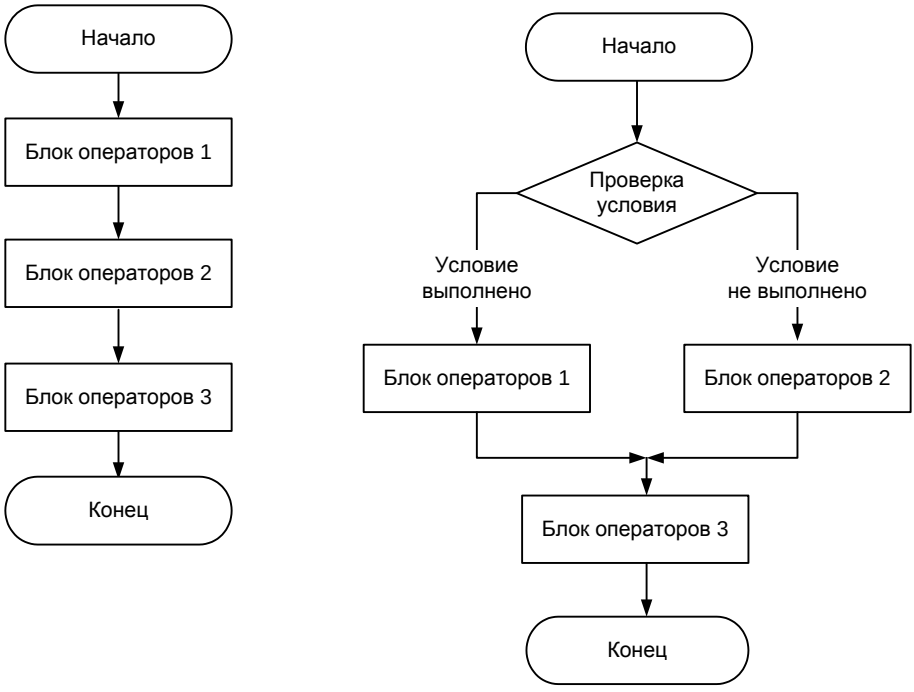


Рис. 5.1. Блок-схемы программ с оператором ветвления (*справа*) и без него (*слева*)

Блок-схема, расположенная слева, не имеет операторов ветвления. Программа, выполняющаяся согласно этой схеме, всегда будет последовательно проходить три блока операторов: *Блок операторов 1*, *Блок операторов 2*, *Блок операторов 3*.

Теперь взгляните на правую блок-схему. В ней присутствует условный оператор, который предлагает две альтернативы выполнения. Если условие выполнено, будет отрабатывать первый блок операторов, если условие не выполнено, отработает второй блок операторов. Таким образом, такая программа имеет два варианта исполнения: *Блок операторов 1, Блок операторов 3* либо *Блок операторов 2, Блок операторов 3*.

Возможность определять альтернативу хода выполнения программы является мощнейшим инструментом программирования.

Оператор *if*

В самом простом виде использование оператора `if` выглядит так:

```
if (Условие) Действие;
```

Условие — это любое выражение с булевым значением (`true` или `false`). Действие выполняется тогда, когда *Условие* истинно (т. е. равно `true`). В результате проверки какого-либо условия может выполняться сразу несколько операторов, тогда они образуют блок, который должен быть заключен в фигурные скобки:

```
if (Условие)
{
  Действие1;
  Действие2;
}
```

Если сравнивать данный сценарий с правой блок-схемой на рис. 5.1, то *Блоком операторов 1* здесь является последовательность

```
Действие1;
Действие2;
```

Блок операторов 2 пуст, так же как и *Блок операторов 3*.

Часто возникает необходимость указать какие-либо действия не только, когда условие верно, но и когда оно ложно. В этом случае используется ключевое слово `else`:

```
if (Условие)
{
  Блок_операторов1
}
else
{
  Блок_операторов2
}
```

Данный сценарий соответствует правой блок-схеме рис. 5.1, если из нее исключить *Блок операторов 3*.

Ключевое слово `elseif` позволяет осуществлять проверку дополнительных условий, если предыдущие условия оказались ложными. Рассмотрим следующий пример:

```
if (Условие1)
{
  Блок_операторов1
}
elseif (Условие2)
{
  Блок_операторов2
}
elseif (Условие3)
{
  Блок_операторов3
}
else
{
  Блок_операторов4
}
```

Данный пример можно изобразить в виде части блок-схемы (рис. 5.2).

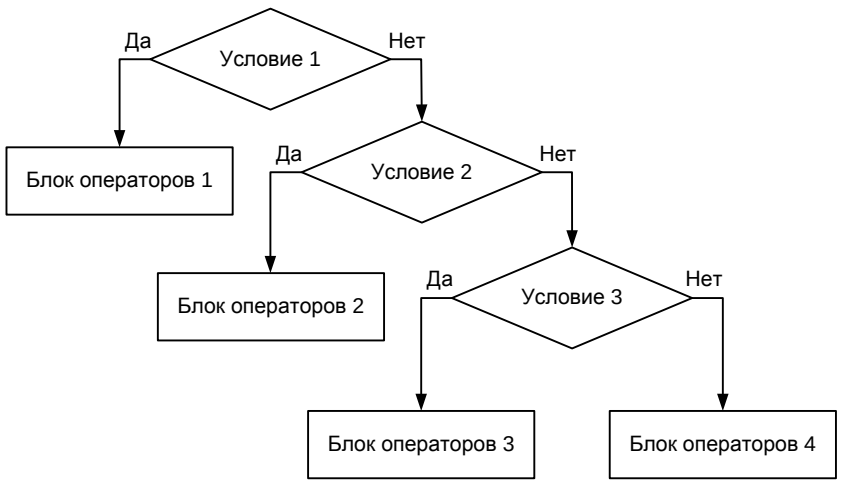


Рис. 5.2. Сложная структура логических операторов

Допустим *Условие1* и *Условие2* ложные, а *Условие3* истинно. В данном случае интерпретатор PHP проверит сначала *Условие1*, убедится, что оно ложное, затем *Условие2*. После этого дойдет до *Условия3*, увидит, что оно истинное, и войдет внутрь третьего блока операторов. После чего выполнение сценария продолжится уже после всей этой конструкции.

А что будет, если истинны сразу несколько условий, например *Условие2* и *Условие3*. В этом случае выполнится только один блок операторов первого истинного условия. В нашем случае, если истинны *Условие2* и *Условие3*, выполнится только *Блок_операторов2*. После чего интерпретатор перейдет к выполнению инструкций, которые идут после закрывающей скобки } последнего блока else.

Чтобы лучше понять принцип работы, напишите небольшой сценарий из листинга 5.1.

Листинг 5.1. Пример работы условных операторов

```
<?php
$a = 1;
if ($a < 2)
{
  echo "a меньше 2";
}
elseif ($a < 3)
{
  echo "a меньше 3";
}
elseif ($a < 4)
{
  echo "a меньше 4";
}
else
{
  echo "a больше либо равно 4";
}
?>
```

Посмотрите результат выполнения, меняя значение $a. Блоки кода операторов elseif и else выполняются только тогда, когда все проверенные ранее условия оказались равными false.

Тернарный оператор условия

Тернарный оператор имеет три операнда (отсюда такое название). Выглядит он следующим образом:

```
Переменная = Условие ? Значение1 : Значение2;
```

Если значение *Условия* истинно, то переменной присваивается *Значение1*, иначе *Значение2*. Пример:

```
$a = (1 < 2) ? "1 меньше 2" : "1 не меньше 2";
```

Напишите этот небольшой сценарий и убедитесь, что переменной $a в данном случае присваивается строка "1 меньше 2", т. к. условие (1 < 2) истинно. Теперь попробуйте поменять знак с "<" на ">" и проверьте результат.

Тернарный оператор появился впервые в языке Си. Он позволяет включать в операцию присваивания проверку условия. Это красивая возможность, делающая код лаконичнее. Пользуйтесь тернарным оператором!

Однако остерегайтесь вложенности таких операторов, т. к. это сильно снижает читабельность кода.

Оператор *switch*

Оператор `switch` вычисляет значение заданного выражения и сравнивает его с предложенными вариантами. В случае совпадения выполняется соответствующий блок кода.

Оператор `switch` имеет следующий синтаксис:

```
switch (Выражение)
{
  case Значение1:
    Блок_операторов1;
    break;
  case Значение2:
    Блок_операторов2;
    break;
  default:
    Блок_операторов3;
}
```

Сначала `switch` вычисляет значение *Выражения*, после чего сравнивает его со значением каждой метки `case` (*Значение1*, *Значение2*). Как только обнаруживается совпадение, выполняется соответствующий блок операторов.

Обратите внимание на ключевое слово `break` в конце каждого блока `case`. `break` говорит о том, что выполнение операторов в блоке `switch` окончено, и нужно переходить к выполнению операторов после всего блока `switch` (после фигурной скобки). В случае, когда в конце блока `case` нет оператора `break`, интерпретатор продолжит выполнять операторы следующих блоков.

Если же после прохода по всем меткам `case` совпадение так и не встретилось, выполняются операторы из блока `default`. Впрочем, это необязательный блок и он может отсутствовать. В таком случае, если интерпретатор не нашел ни одного совпадения, выполнение программы продолжится со строки, следующей за блоком `switch`.

В листинге 5.2 приведен пример.

Листинг 5.2. Применение оператора `switch`

```
<?php
$a = 1;
switch ($a)
```

```
{
  case 1:
    echo "a равно 1";
    break;
  case 2:
    echo "a равно 2";
  case 3:
    echo "a равно либо 2, либо 3. " .
         "Не могу определить точнее, " .
         "потому что забыли поставить " .
         "break в предыдущем case";
    break;
  default:
    echo "a не равно ни 1, ни 2, ни 3";
}
?>
```

Иногда пропуск оператора `break` может использоваться намеренно для обеспечения правильной логики программы.

Особенностью языка PHP является то, что метками `case` могут быть переменные простых типов (не массивы и не объекты). Например, сценарий из листинга 5.3 верен с точки зрения синтаксиса языка.

Листинг 5.3. Применение оператора `switch`

```
<?php
$a = 1;
$b = 2;

switch($a)
{
   case 1:
     echo "a равно 1";
     break;
   case $b:
     echo "a равно 2";
     break;
   default:
     echo "a не равно ни 1, ни 2";
}
?>
```

Оператор `switch` принято использовать, когда алгоритм имеет более двух альтернатив выполнения. Иначе имеет смысл пользоваться связкой операторов `if-else`.

Функции

Практикуясь в программировании, вы быстро заметите, что очень часто появляются ситуации, когда в разных частях программы требуется использовать один и тот же блок кода. На первом этапе обучения вы можете просто копировать этот код в разные части своего сценария. Однако такой подход неверен, и вот почему. Предположим, в какой-то момент вы заметили ошибку в этом блоке кода. Тогда для ее исправления потребуется внести изменения в несколько мест. В крупных проектах это может существенно затормозить разработку. Не говоря уже о том, что зачастую программист исправляет ошибку в одном месте, забывая внести такие же изменения в другое. Решением данной проблемы служит использование функций.

Функция — это блок кода, к которому можно обращаться из разных частей сценария. Функции могут иметь входные и выходные параметры. Входные параметры могут использоваться в операциях, которые содержит функция. Выходные параметры устанавливаются функцией, их значения используются после выполнения функции. Программист сам может определять необходимые ему функции и их логику выполнения.

Определение функции

Функция объявляется с помощью ключевого слова `function`. Если функция имеет параметры, они объявляются как переменные в объявлении функции:

```
function my_function($my_parameter)
{
  echo "Вы передали '$my_parameter' в функцию<br/>";
}
```

Вот и определение нашей первой функции. После ключевого слова `function` идет название функции. Мы назвали ее `my_function`. После этого в круглых скобках перечисляются параметры, которые мы хотим передать в функцию. В нашем случае мы описали параметр `$my_parameter`. Теперь внутри тела функции, которое заключается в фигурные скобки `{` и `}`, мы можем использовать переменную `$my_parameter`. В частности мы выводим ее значение внутри уже известного нам оператора `echo`.

Обратиться к нашей функции теперь можно следующим образом:

```
my_function("это мой параметр");
```

Только что мы вызвали нашу функцию, передав туда строку "это мой параметр". Давайте теперь взглянем на сценарий, в котором используется только что написанная функция (листинг 5.4).

Листинг 5.4. Использование функции

```
<?php
function my_function($my_parameter)
```

```
{
  echo "Вы передали '$my_parameter' в функцию<br>";
}

my_function("это мой параметр");
my_function("а это мой новый параметр");
?>
```

Перепишите и выполните этот сценарий. Как вы можете увидеть, наша функция отработала два раза, при этом на экране появились две строчки:

```
Вы передали 'это мой параметр' в функцию
Вы передали 'а это мой новый параметр' в функцию
```

Если требуется передать несколько параметров, то они разделяются запятой:

```
function my_function($my_parameter1, $my_parameter2)
{
  echo "you send '$my_parameter1' and " .
       "'$my_parameter2' to my function<br>";
}
```

Функция всегда возвращает значение. Если это не указать явно, то будет возвращено `NULL`. Оператор `return` позволяет завершить выполнение функции, вернув конкретное значение (листинг 5.5).

Листинг 5.5. Получение значения из функции

```
<?php
function my_function_sum($a, $b)
{
  return $a+$b;
}

echo "если сложить 2 и 3, то получится ";
echo my_function_sum(2, 3);
?>
```

Можно сохранять полученное из функции значение в переменной:

```
$sum = my_function_sum(2, 3);
```

Оператор `return` может ничего и не возвращать, а просто использоваться для выхода из функции (листинг 5.6).

Листинг 5.6. Использование оператора `return` для выхода из функции

```
<?php
function not_1($a)
```

```
{
  if($a == 1)
    return;
  echo "очевидно, ваш параметр не равен 1";
}
not_1(1);
not_1(2);
not_1(3);
?>
```

Пустой `return` — это то же самое, что

```
return NULL;
```

В данном случае строчка сообщения выведется только два раза, потому что в первом случае (`not_1(1);`) выполнится условие оператора `if`, отработает оператор `return` и произойдет выход из функции.

Значения по умолчанию

PHP позволяет определять значения по умолчанию для параметров функций. Единственное условие — эти параметры должны идти последними в объявлении функции.

Присвоение значения по умолчанию делает параметр необязательным, т. е. при вызове функции его можно опустить (листинг 5.7).

Листинг 5.7. Использование значения по умолчанию при определении функции

```
<?php
function sum($a, $b = 1)
{
  return $a + $b;
}

echo sum(1);
echo sum(1, 2);
?>
```

Сначала функция `echo` выведет значение 2, а потом 3. В первом случае мы вызвали функцию только с одним параметром, поэтому переменная `$b` (второй параметр) получила значение по умолчанию. Во втором случае, как и положено, переменной `$b` присваивается значение 2.

Рекурсия

Внутри блока кода функций можно пользоваться абсолютно любыми средствами языка PHP так же как и вне функций. В том числе существует возможность вызова

функции из самой себя. Эта операция и называется *рекурсией*. Рекурсия часто позволяет найти элегантное решение задачи. Разберем самый распространенный учебный пример использования рекурсии — вычисление факториала.

Факториал `n` — это произведение всех натуральных чисел до `n` включительно. В математике факториал обозначается символом `!`. То есть, к примеру, $5! = 1 \times 2 \times 3 \times 4 \times 5$.

С помощью рекурсии в PHP факториал положительной целочисленной переменной можно вычислить следующим образом:

```
function fact($x)
{
  if ($x == 1)
    return 1;
  else
    return $x * fact($x – 1);
}
```

Область видимости и время жизни переменных

Переменные по области видимости можно разделить на глобальные и локальные. К первым можно обращаться из любого места программы, вторые доступны только в конкретном ее месте, например внутри функции.

Все переменные, которые объявлены в теле сценария вне функций, — глобальные. Переменные, объявленные в функциях, имеют локальную область видимости. То есть за пределами функции нельзя использовать ее переменные.

Рассмотрим пример (листинг 5.8).

Листинг 5.8. Локальные и глобальные переменные

```
<?php
function printA()
{
  $a = 2;
  echo $a;
}

$a = 1;
echo $a;   // выводит 1
printA();  // выводит 2
echo $a;   // выводит 1
?>
```

Переменная `$a`, объявленная внутри функции `printA`, не имеет никакого отношения к глобальной `$a`, объявленной сразу после функции. Когда интерпретатор выполняет инструкции внутри функции `printA`, ему не известно о каких-либо переменных,

которые объявлены за пределами функции. Поэтому сценарий выводит последовательность "121".

Локальные переменные сохраняют свое значение только во время выполнения функции. При новом вызове функции значения локальных переменных инициализируются заново (листинг 5.9).

Листинг 5.9. Инициализация локальных переменных

```
<?php
function printA()
{
  $a = 0;
  $a++;
  echo $a;
}
printA();
printA();
printA();
?>
```

Этот сценарий выведет три единицы, потому что при каждом обращении к функции `printA` значение переменной `$a` сначала устанавливается равным нулю.

Однако есть исключение. Чтобы функция все же "помнила" значения уже инициализированных переменных, используется ключевое слово `static`. Таким образом, можно объявить *статические переменные*, которые инициализируются функцией только один раз (листинг 5.10).

Листинг 5.10. Использование статических переменных внутри функции

```
<?php
function printA()
{
  static $a = 0;
  $a++;
  echo $a;
}
printA();
printA();
printA();
?>
```

Этот сценарий выведет последовательность "123", потому что обнуление `$a` происходит только при первом вызове функции. При последующих вызовах функции PHP будет знать, что переменная `$a` уже существует, и пропустит строчку со словом `static`.

Все глобальные переменные доступны внутри функции через системную переменную `$_GLOBALS`. Она представляет собой ассоциативный массив. О том, что это такое, мы как раз расскажем в следующем уроке.

Также в PHP существует специальная инструкция `global`, позволяющая функции работать с *глобальными переменными*. Рассмотрим данный принцип на примере (листинг 5.11).

Листинг 5.11. Использование инструкции `global` внутри функции

```
<?php
$a = 1;
$b = 2;

function Sum()
{
  global $a, $b;
  $b = $a + $b;
}

Sum();
echo $b;
?>
```

Данный сценарий выведет значение 3. Первой строкой функции идет инструкция

```
global $a, $b;
```

Это значит, что при дальнейшем сложении `$b = $a + $b;` будут использоваться глобальные переменные, объявленные вне функции.

Обратите особое внимание на то, что использование глобальных переменных внутри функций считается плохим тоном. Его следует избегать везде, где это возможно. В большинстве случаев использование глобальных переменных можно заменить передачей дополнительного параметра в функцию. Глобальные переменные повышают связность кода, т. е. взаимозависимость его частей друг от друга. Это плохо, т. к. вместе с увеличением зависимости уменьшается универсальность, переносимость и масштабируемость. Эти моменты кажутся не столь очевидными на данном этапе обучения. Однако сразу возьмите себе за правило избегать использования глобальных переменных там, где это возможно.

Резюме

Использование функций значительно упрощает жизнь программистам. С их помощью можно хорошо структурировать код, избавиться от повторения блоков кода в сценарии, избежать появления ненужных глобальных переменных и значительно снизить трудозатраты на отладку и исправление ошибок. По мере накопления опы-

та у программиста появляются библиотеки собственных функций, которые он может переносить из одного проекта в другой.

Старайтесь минимизировать область видимости переменных. Это сделает вашу программу гибче для внесения изменений и легче в разборе. По возможности избегайте глобальных переменных, передавая каждой функции только ту информацию, которая ей нужна в качестве параметров.

Стоит помнить, что большинство велосипедов уже изобретено. Поэтому перед тем как писать функцию вычисления синуса, нужно поинтересоваться, не решена ли уже ваша задача стандартными средствами PHP, или же какой-нибудь специальной библиотекой, которых в Интернете достаточно.

Задания

1. Объявите в начале сценария две целочисленных переменных `$a` и `$b`, начальные значения определите с помощью констант. Напишите сценарий:

   а) если `$a` и `$b` положительные — выведите их сумму;

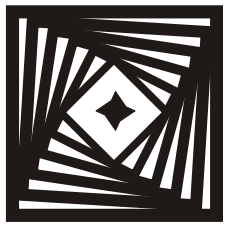
   б) если `$a` и `$b` отрицательные — выведите из разность;

   в) если `$a` и `$b` разных знаков — выведите их произведение.

   Ноль можно считать положительным числом.
2. Выведите большее из чисел, используя тернарный оператор.
3. Присвойте `$a` значение в промежутке `[0..9]`. С помощью оператора `switch` организуйте вывод чисел от `$a` до `9`.
4. Реализуйте все арифметические операции в виде функций с двумя параметрами.
5. Реализуйте функцию с тремя параметрами:

   ```
   function mathOperation($arg1, $arg2, $operation)
   ```

   где `$arg1`, `$arg2` — значения аргументов; `$operation` — строка с названием операции. В зависимости от переданного значения операции выполните одну из арифметических операций (используйте функции из пункта 4) и верните полученное значение (используйте `switch`).
6. С помощью рекурсии организуйте функцию возведения числа в степень. Формат: `function power($val, $pow)`, где `$val` — заданное число; `$pow` — степень.
7. *Реализуйте правильную работу функции возведения в степень при получении нулевых или отрицательных параметров.



УРОК 6

Циклы и массивы

Циклы в PHP

Циклы позволяют многократно выполнять блок кода. Это необходимо для решения множества задач. Например, перебор записей, полученных из базы данных, построчное чтение файла или обход элементов массива. В PHP есть четыре типа циклов: `while`, `do..while`, `for` и `foreach`. Первые три описываются далее, а `foreach` — при обсуждении массивов далее в этом уроке.

Цикл *while*

Конструкция `while` представляет собой простейший оператор цикла. Блок операторов выполняется, пока верно условие

```
while (Условие)
{
  Операторы
}
```

Рассмотрим работу цикла подробнее:

1. Проверка условия.
2. Если оно истинно, выполнение операторов; если ложно — выход из цикла.
3. Переход к шагу 1.

Для управления циклом обычно требуется одна или несколько переменных. Например, целочисленное значение, каждый раз увеличиваемое на единицу. Эти переменные так и называются — *управляющие переменные цикла*.

Рассмотрим простой пример вывода чисел от 1 до *N* (листинг 6.1).

Листинг 6.1. Цикл `while`

```
<?php
$n = 10;
$i = 1;
```

```
while ($i <= $n)
{
  echo "$i<br/>";
  $i++;
}
?>
```

Иногда управляющая переменная цикла бывает булевой. Например, при чтении в цикле строк файла можно использовать переменную булева типа, определяющую конец файла.

Цикл *do..while*

Циклы `do..while` аналогичны циклам `while`, за исключением того, что условие проверяется не в начале, а в конце каждой итерации. Это означает, что цикл будет выполнен, по крайней мере, один раз.

```
do
{
  Операторы
} while (Условие)
```

Рассмотрим работу цикла:

1. Выполнение блока операторов.
2. Проверка условия.
3. Если оно истинно, переход к шагу 1; если ложно — выход из цикла.

В примере из листинга 6.2 единица будет выведена, даже если $N = 0$.

Листинг 6.2. Цикл `do..while`

```
<?php
$n = 10;
$i = 1;

do
{
  echo "$i<br/>";
  i++;
} while ($i <= $n)
?>
```

Цикл `do..while` используют достаточно редко ввиду его громоздкости и плохой читаемости. Лучше слегка изменить алгоритм (от чего он, скорее всего, выиграет) и применить `for` или `while`.

Цикл *for*

Цикл `for` — шедевр лаконичной организации кода, пришедший из языка Си. Конструкция позволяет одной строкой полностью определить поведение цикла.

```
for (Выражение1; Выражение2; Выражение3)
{
  Операторы
}
```

Выражение1 вычисляется перед началом цикла. Обычно в нем инициализируется управляющая переменная. *Выражение2* вычисляется в начале каждой итерации цикла. Это выражение ведет так же, как условие цикла `while`, если значением *Выражения2* оказывается `true`, цикл продолжается, иначе — останавливается. *Выражение3* вычисляется в конце каждой итерации и, как правило, используется для изменения значения управляющей переменной цикла.

Ход выполнения цикла:

1. Выполнение *Выражения1*.
2. Проверка *Выражения2*.
3. Если оно истинно, переход к шагу 4; если ложно — выход из цикла.
4. Выполнение блока операторов.
5. Выполнение *Выражения3*.
6. Переход к шагу 2.

Рассмотрим тот же пример вывода чисел от 1 до *N* (листинг 6.3).

Листинг 6.3. Цикл `for`

```
<?php
$n = 10;

for ($i = 1; $i <= $n; $i++)
{
  echo "$i<br/>";
}
?>
```

Пример демонстрирует наиболее частое применение конструкции `for`: инициализировать управляющую переменную, сравнить переменную со значением, инкрементировать или декрементировать значение. Однако `for` может использоваться и другими способами.

Так же, как в Си и других языках, допускается не указывать одно или более выражений (если не указано *Выражение2*, считается, что оно `true`). Цикл `while` всегда можно заменить циклом `for`.

Следующие две конструкции эквивалентны:

- вариант 1:

```
while (Условие)
{
  Операторы
}
```

- вариант 2:

```
for ( ; Условие; )
{
  Операторы
}
```

Бесконечный цикл, операторы выхода из цикла и прерывания итерации цикла

Бесконечным называется цикл такого вида:

```
while (true)
{
  ...
}
```

или такого (что одно и то же):

```
for (;;)
{
  ...
}
```

Для выхода из цикла можно использовать оператор `break` (листинг 6.4).

Листинг 6.4. Использование оператора `break`

```
<?php
$n = 10;
$i = 1;

while (true)
{
   echo "$i<br/>";
   $i++;

   if ($i > $n)
     break;
}
?>
```

Оператор `break` мгновенно завершает выполнение цикла. Если же нужно закончить текущую итерацию цикла и вернуться к проверке его условия, можно воспользоваться оператором `continue`. Следующий пример демонстрирует вывод нечетных чисел от 1 до *N* (листинг 6.5).

Листинг 6.5. Использование оператора `continue`

```
<?php
$n = 10;

for ($i = 1; $i <= $n; $i++)
{
  if ($i % 2 == 0)
    continue;

  echo "$i<br/>";
}
?>
```

Остаток от деления на два четного числа равен нулю, такие числа мы пропускаем в приведенном примере. Правильнее считается не использовать операторы прерывания цикла, а возлагать логику управления на условие. Старайтесь организовывать циклы именно так.

Массивы

Что такое массив

Массив — это именованный набор однотипных переменных. Он состоит из элементов. Обращение к каждому элементу происходит по его номеру. Нумерация начинается с нуля, т. е. у первого элемента индекс 0, у второго элемента индекс 1 и т. д.

Массивы в PHP — чрезвычайно мощный инструмент. По сути, это даже не массив, а словарь. Словарь (он же хэш-таблица) — такая структура данных, которая хранит множество пар "ключ—значение". Ключи не могут повторяться. В качестве ключа может использоваться как целое число, так и строка.

Есть несколько методов инициализации массива. Один из них состоит просто в том, чтобы начать присваивать значения элементам массива. Приводимый далее код создает массив с именем `$aLanguages` из трех элементов. Поскольку индексы не указаны, PHP по умолчанию присваивает числовые индексы 0, 1 и 2:

```
$aLanguages[] = "Arabic";
$aLanguages[] = "German";
$aLanguages[] = "Korean";
echo($aLanguages[2]); // Выводит "Korean"
```

Чтобы явно указать индекс, заключите его в квадратные скобки:

```
$aLanguages[0] = "Arabic";
$aLanguages[1] = "German";
$aLanguages[2] = "Korean";
echo($aLanguages[2]); // Выводит "Korean"
```

Элементы массива не обязательно должны объявляться последовательно. Следующий код создает массив элементов с индексами 100, 400, 300 и 401:

```
$aLanguages[100] = "Arabic";
$aLanguages[400] = "German";
$aLanguages[300] = "Korean";
$aLanguages[] = "Tagalog";
echo($aLanguages[300]); // Выводит "Korean"
echo($aLanguages[401]); // Prints "Tagalog"
```

Поскольку индекс последнего элемента не был задан, PHP присвоил ему первый доступный индекс после самого большого использованного до сих пор индекса: 401. Функция `array()` дает альтернативный способ определения массивов. `array()` принимает разделенный запятыми список значений, подлежащих помещению в массив:

```
$aLanguages = array("Arabic", "German", "Korean", "Tagalog");
echo($aLanguages[2]); // Выводит "Korean"
```

И снова, поскольку индексы не были заданы, они присваиваются элементам массива по умолчанию. Для явного указания индексов в конструкции `array()` применяется оператор =>:

```
$aLanguages = array("Arabic", 3 => "German", "Korean", "Tagalog");
echo($aLanguages[0]);  // Выводит "Arabic"
echo($aLanguages[3]);  // Выводит "German"
echo($aLanguages[4]);  // Выводит "Korean"
echo($aLanguages[5]);  //Выводит "Tagalog"
```

Как упоминалось ранее, индексы массива могут быть строками:

```
$aLanguages = array("ar" => "Arabic",
                    "de" => "German",
                    "tl" => "Tagalog"
                   );
echo($aLanguages["tl"]); // Выводит "Tagalog"
$aLanguages["ko"] = "Korean";
echo($aLanguages["ko"]); // Выводит "Korean"
```

Обход массивов в цикле

Самый простой способ обойти все элементы массива:

```
for ($i = 0; $i < count($arr); $i++)
{
  ...
}
```

Однако способ годится только тогда, когда в качестве ключа используется порядковый номер элемента (начиная с нуля). Для обхода любых массивов (в общем случае — словарей) существует специальный оператор `foreach`.

Его синтаксис прост:

```
foreach (Массив as [$key =>] $value)
{
  Операторы
}
```

Оператор `foreach` проходит каждый элемент массива по одному разу. В каждом проходе в переменную `$key` помещается индекс этого элемента, а в переменную `$value` — значение. Имена этих двух переменных произвольны.

```
foreach ($aLanguages as $sIndex => $sVal)
{
  echo("$sIndex is $sVal <br/>");
}
```

Переменная для ключа необязательна, поскольку не всегда нужна внутри цикла. В данном примере переменная `$key` опущена, а вместо `$value` используется `$sLang`:

```
echo("Available Languages: <br/><ul>");
foreach ($aLanguages as $sLang)
{
  echo( "<li>$sLang</H>" ) ;
}
echo("</ul>");
```

Функции для работы с массивами

PHP предлагает массу функций, облегчающих работу с массивами. Ряд полезных функций мы приводим ниже. Полный список можно найти в электронной документации на английском языке: **http://www.php.net/manual/en/ref.array.php**.

count()

```
int count(mixed var)
```

Функция `count()` принимает в качестве аргумента массив и возвращает количество элементов в нем. Если переменная не установлена или не содержит элементов, возвращается ноль.

in_array()

```
boolean in_array(mixed needle, array haystack [, bool strict])
```

Эта функция ищет в массиве *haystack* значение *needle* и возвращает `true`, если оно найдено, и `false` в противном случае.

sort()

```
void sort(array array [, int sort_flags])
```

Эта функция применяется для сортировки значений в массиве. Необязательный второй параметр *sort_flags* указывает, как должны сортироваться данные. Допустимыми значениями являются SORT_REGULAR, SORT_NUMERIC, устанавливающие сравнение значений как чисел, и SORT_STRING, устанавливающее сравнение значений как строк.

PHP содержит много функций сортировки, синтаксис которых очень близок к sort(). Эти функции по-разному ведут себя, предоставляя разные варианты процедуры сортировки, включая направление сортировки, обработку ключей и алгоритмы сравнения. Подробнее смотрите в документации такие функции, как arsort(), asort(), ksort(), natsort(), natcasesort(), rsort(), usort(), array_multisort() и uksort().

explode() и implode()

Эти две функции официально считаются строковыми, но они касаются массивов. explode() расщепляет строку на отдельные элементы, помещаемые в массив, используя разделитель, переданный в качестве аргумента. implode() осуществляет противоположную операцию. Она сжимает элементы массива в одну строку, соединяя их с помощью переданного аргумента:

```
// Превратить массив в строку, с разделителем — точкой с запятой:
$sLangString = implode(';', $aLanguages);
echo($sLangString);
$sSentence = 'Per aspera ad astra';
// Превратить предложение в массив отдельных слов:
$aWords = explode(' ', $sSentence);
```

Многомерные массивы

Многомерный массив возникает, когда элементы некоторого массива сами содержат массивы (которые, в свою очередь, могут содержать массивы и т. д.). Для инициализации многомерных массивов используются те же средства, включая вложенные конструкции array():

```
$aLanguages = array("Slavic" => array("Russian", "Polish", "Slovenian"),
"Germanic" => array("Swedish", "Dutch", "English"),
                    "Romance" => array("Italian", "Spanish", "Romanian")
);
```

Для доступа к элементам многомерных массивов, вложенным глубоко внутрь, применяются дополнительные скобки. Таким образом, $aLanguages["Germanic"] указывает на массив, содержащий германские языки, а $aLanguages["Germanic"][2] указывает на третий элемент ("English") вложенного массива.

Обход многомерных массивов может осуществляться с помощью вложенных циклов (листинг 6.6).

Листинг 6.6. Использование вложенных циклов для обхода многомерных массивов

```
<?php
foreach ($aLanguages as $sKey => $aFamily)
{
  // Вывести название семейства языков:
  echo(
    "<h2>$sKey</h2>" .
    "<ul>"
  );

  // Теперь перечислить языки в каждом семействе:
  foreach ($aFamily as $sLanguage)
  {
    echo("<li>$sLanguage</li>");
  }

  // Завершить список:
  echo("</ul>");
}
?>
```

При каждом проходе внешнего цикла переменной `$sKey` присваивается в качестве значения название семейства языков, а переменной `$aFamily` — соответствующий внутренний массив. Внутренний цикл обходит массив `$aFamily`, помещая значение каждого элемента в переменную `$sLanguage`.

Предопределенные массивы

- `$GLOBALS`

  Содержит ссылку на каждую переменную, доступную в данный момент в глобальной области видимости сценария. Ключами этого массива являются имена глобальных переменных.

- `$_SERVER`

  Переменные, установленные веб-сервером либо напрямую связанные с окружением выполнения текущего сценария.

- `$_GET`

  Переменные, передаваемые сценарию через HTTP GET.

- `$_POST`

  Переменные, передаваемые сценарию через HTTP POST.

- `$_COOKIE`

  Переменные, передаваемые сценарию через HTTP cookies.

- `$_FILES`

  Переменные, передаваемые сценарию через HTTP POST-загрузку файлов.

- `$_ENV`

  Переменные, передаваемые сценарию через окружение.

- `$_SESSION`

  Переменные, зарегистрированные на данный момент в сессии сценария.

- `$_REQUEST`

  Переменные, передаваемые сценарию через механизмы ввода GET, POST и cookie. Не рекомендуется для использования. Следует обращаться к конкретному массиву.

Резюме

Массивы PHP — мощный инструмент, который можно использовать и как обычные списки, и как хэш-таблицы (словари). При их обходе удобно использовать конструкцию `foreach`.

На этом уроке изучение самого языка PHP можно считать оконченным. Далее будем рассматривать непосредственно веб-программирование с помощью языка.

Задания

1. С помощью цикла `while` выведите все числа в промежутке от 0 до 100, которые делятся на 3 без остатка.
2. С помощью цикла `do..while` напишите функцию для вывода чисел от 0 до 10, чтобы результат выглядел так:

```
0 — это ноль
1 — нечетное число
2 — четное число
3 — нечетное число
...
10 — четное число
```

3. *Выведите с помощью цикла `for` числа от 0 до 9, *не* используя тело цикла. То есть выглядеть должно вот так: `for(...){// здесь пусто}`.
4. Объявите массив, где в качестве ключей будут использоваться названия областей, а в качестве значений — массивы с названиями городов из соответствующей области.

   Выведите в цикле значения массива, чтобы результат был таким:

```
Московская область:
 Москва, Зеленоград, Клин
Ленинградская область:
 Санкт-Петербург, Всеволожск, Павловск, Кронштадт
Рязанская область
 ...
```

   (Названия городов можно найти на **maps.yandex.ru**.)

5. *Повторите предыдущее задание, но выводите на экран только города, начинающиеся с буквы "К".
6. Объявите массив, индексами которого являются буквы русского языка, а значениями — соответствующие латинские буквосочетания ('а' ⇒ 'a', 'б' ⇒ 'b', 'в' ⇒ 'v', 'г' ⇒ 'g', ..., 'э' ⇒ 'e', 'ю' ⇒ 'yu', 'я' ⇒ 'ya'). Напишите функцию транслитерации строк.
7. Напишите функцию, которая заменяет в строке пробелы на подчеркивания и возвращает видоизмененную строчку.
8. Объедините две ранее написанные функции в одну функцию, которая получает строку на русском языке, производит транслитерацию и замену пробелов на подчеркивания (аналогичная задача решается при конструировании URL-адресов на основе названия статьи в блогах).

УРОК 7

Запросы HTTP, параметры URL и формы HTML

Типы запросов HTTP

HTTP (Hyper Text Transport Protocol) — тот самый язык, на котором "разговаривают" браузеры с веб-серверами, важнейший протокол Интернета.

Запросы можно разделить на два вида:

- GET;
- POST.

GET используется при наборе адреса сайта в строке браузера или перехода по ссылке. POST служит для отправки форм, например при регистрации на сайте, публикации комментария к статье. Для отправки формы обычно нужно нажать кнопку **Отправить** или подобную ей (рис. 7.1).

Для простоты понимания различие можно представлять так:

- GET используется для чтения сайтов (читаем Интернет);
- POST служит для публикации информации на сайтах (пишем Интернет).

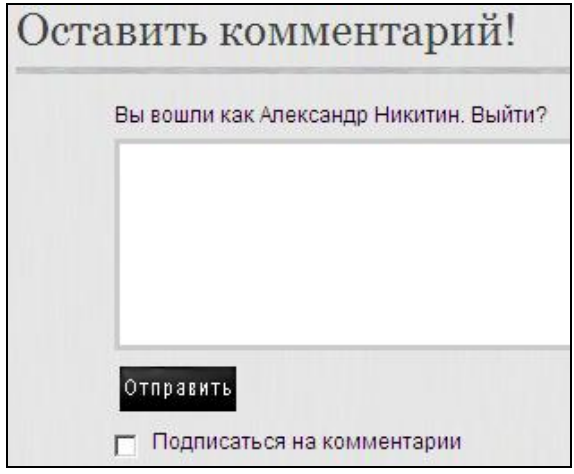


Рис. 7.1. Пример HTML-формы

URL и параметры запроса

В обоих случаях требуется URL (Uniform Resource Locator) запрашиваемого документа.

URL — это адрес страницы в Интернете. Как правило, он имеет такой вид:

http://*<хост>*/*<путь>*

Например:

http://www.example.ru/about.php

Или же такой, если необходимо передать параметры сценарию:

http://*<хост>*/*<путь>*?*<параметры>*

где ***<параметры>*** — это набор пар вида:

<имя>*=*<значение>

разделенных символом **&**.

Пример:

http://www.example.ru/news.php?id=100&show_comments=yes

У вас может возникнуть вопрос: для чего сценарию передавать параметры? Динамическая страница (она же сценарий), в отличие от статической, может выдавать различную информацию. Например, сценарий новостной ленты отображает либо список анонсов последних новостей, либо целиком текст конкретной статьи. Что именно хочет увидеть пользователь, сценарий понимает, исходя из переданных ему параметров.

Это могло бы работать следующим образом. Получение списка последних новостей: **http://www.example.ru/news.php** (URL без параметров). Получение полного текста новостной статьи: **http://www.example.ru/news.php?id=1** (URL включает в качестве параметра номер новости).

Обработка параметров URL

А сейчас мы напишем сценарий этой самой новостной ленты. У нее будут два режима:

- ❒ демонстрация списка всех новостей (если нет параметров) — рис. 7.2;
- ❒ отображение текста конкретной новости (если ее номер передан в качестве параметра) — рис. 7.3.

Всего новостей у нас будет три:

- ❒ "За качество ответят. Контролировать продукты питания начали по-новому";
- ❒ "Варшава не раскрывает перечень возможных мер против Минска";
- ❒ "Павел Астахов намерен добиваться отставки ряда чиновников Удмуртии".

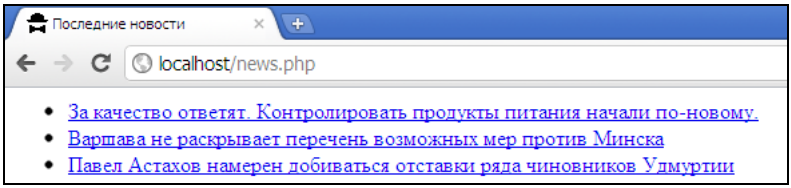


Рис. 7.2. Новостной сценарий: список новостей

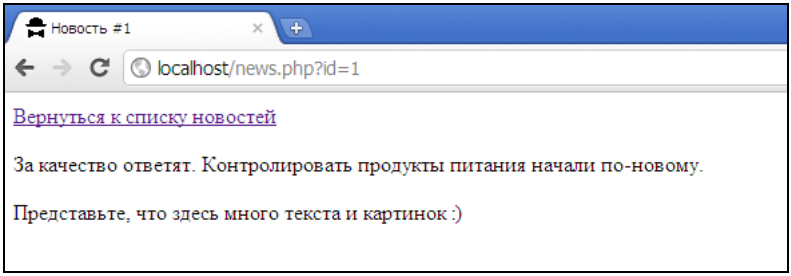


Рис. 7.3. Новостной сценарий: просмотр новости

Внимание!

Пример упрощен. Никто никогда не хранит новости в коде сценария. Хранить подобную информацию следует в базе данных. Но это предмет совсем другого урока!

Сейчас же нам важно научиться обрабатывать параметры, переданные через URL. Итак, создайте файл news.php (листинг 7.1).

Листинг 7.1. Пример сценария новостей

```
<?php
// Функция вывода всего списка новостей.
function show_list($news)
{
    echo '<html>';
    echo '<head>';
    echo '<title>Последние новости</title>';
    echo '</head>';
    echo '<body>';
    echo '<ul>';

    for ($i = 0; $i < count($news); $i++)
    {
        echo '<li>';
        echo '<a href="news.php?id=' . ($i + 1) . '">';
        echo $news[$i];
        echo '</a>';
        echo '</li>';
    }
```

```
    echo '</ul>';
    echo '</body>';
    echo '</html>';
}

// Функция вывода конкретной новости.
function show_item($news, $id)
{
    echo '<html>';
    echo '<head>';
    echo "<title>Новость #$id</title>";
    echo '</head>';
    echo '<body>';
    echo '<a href="news.php">Вернуться к списку новостей</a>';
    echo '<p>';
    echo $news[$id — 1];
    echo '</p>';
    echo '<p>';
    echo 'Представьте, что здесь много текста и картинок :)';
    echo '</p>';
    echo '</body>';
    echo '</html>';
}

// Точка входа.

// Создаем массив новостей.
$news = array();
$news[0] = 'За качество ответят. Контролировать продукты питания начали по-
новому.';
$news[1] = 'Варшава не раскрывает перечень возможных мер против Минска';
$news[2] = 'Павел Астахов намерен добиваться отставки ряда чиновников
Удмуртии';

// Был ли передан id новости в качестве параметра?
if (isset($_GET['id']))
{
    show_item($news, $_GET['id']);
}
else
{
    show_list($news);
}
?>
```

Теперь подробно разберем, что же мы написали.

Вначале объявляем две функции, которые будут генерировать HTML. Первая отображает список новостей, вторая — текст конкретной новости. Управление будет передано в эти функции только тогда, когда мы их вызовем. Вернемся к ним позже.

Выполнение сценария начинается с того места, где комментарием помечена точка входа. Мы создаем массив, состоящий из трех новостей. Напомним, нумерация элементов в массиве начинается с нуля!

Далее проверяем, был ли передан `id` новости в качестве параметра. Параметры, переданные через URL, хранятся в системной переменной `$_GET`. Она представляет собой ассоциативный массив (или, по-другому, словарь).

Напомним, ассоциативный массив (или словарь) — это такая структура данных, которая содержит пары "ключ—значение".

Ключи словаря `$_GET` — это имена параметров. Функция `isset()` возвращает `true`, если переменная определена. Таким образом,

```
if (isset($_GET['id']))
```

следует читать: "если URL запроса содержит параметр id".

Теперь возвращаемся к функциям. Здесь все просто, но мы хотели бы обратить внимание на два момента. Во-первых, может быть не совсем понятно, для чего в одном месте прибавляем к `$i` единицу, а в другом — вычитаем. Сделано это для того, чтобы пользователь видел URL первой новости так: "news.php?id=1", а не "news.php?id=0". Это хороший тон и не более того.

Во-вторых, обратите внимание на строчку:

```
echo "<title>Новость #$id</title>";
```

Двойные кавычки отличаются от одинарных тем, что если внутри них встречаются имена переменных (со знаком `$`), то они заменяются значениями этих самых переменных. Строка в одинарных кавычках остается как есть без подстановки значений переменных.

Обработка отправки HTML-формы

В предыдущем примере мы отправляли запросы методом GET. Теперь познакомимся ближе с методом POST и формами HTML. Для этого напишем очень полезный сценарий — сумматор. Он будет складывать числа.

Создайте файл sum.php (листинг 7.2).

Листинг 7.2. Сценарий сумматора

```
<?php
// Функция отображения формы ввода.
function show_form()
{
    echo '<html>';
```

```
    echo '<head>';
    echo '<title>Сумматор</title>';
    echo '</head>';
    echo '<body>';
    echo '<form action="sum.php" method="post">';
    echo '<input type="text" name="a" />';
    echo '+';
    echo '<input type="text" name="b" />';
    echo '<input type="submit" value="=" />';
    echo '</form>';
    echo '</body>';
    echo '</html>';
}

// Функция вывода результата.
function show_result($a, $b)
{
    $result = $a + $b;

    echo '<html>';
    echo '<head>';
    echo '<title>Сумматор</title>';
    echo '</head>';
    echo '<body>';
    echo '<p>';
    echo "$a + $b = <b>$result</b>";
    echo '</p>';
    echo '<p>';
    echo '<a href="sum.php">Хочу суммировать еще</a>';
    echo '</p>';
    echo '</body>';
    echo '</html>';
}

// Точка входа.

// Показываем результат операции или форму ввода.
if (isset($_POST['a']) && isset($_POST['b'])) {
    show_result($_POST['a'], $_POST['b']);
}
else
{
    show_form();
}
?>
```

Принцип работы сценария похож на новостную ленту.

Тут так же два режима:

- режим ввода параметров операции сложения (рис. 7.4);
- режим показа результата (рис. 7.5).

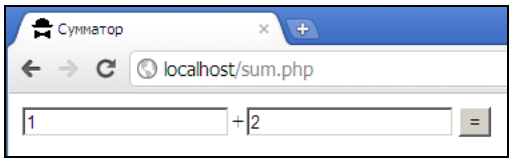


Рис. 7.4. Сумматор. Форма ввода данных

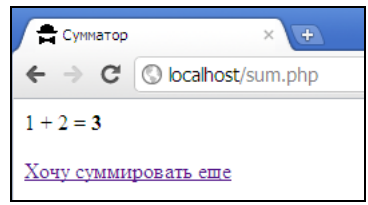


Рис. 7.5. Сумматор. Результат выполнения операции

Форму ввода генерирует функция `show_form()`, HTML-код которой приведен в листинге 7.3.

Листинг 7.3. Форма ввода сумматора

```
<form action="sum.php" method="post">
  <input type="text" name="a" />
  +
  <input type="text" name="b" />
  <input type="submit" value="=" />
</form>
```

Тег `<input>` бывает различных типов (что определяется атрибутом `type`). Нам нужны два текстовых поля (`type="text"`) и одна кнопка (`type="submit"`). Текстовым полям мы дали имена (`"a"` и `"b"`), чтобы из сценария получить введенные туда пользователем значения.

Тег `<input>` должен обязательно находиться внутри тега `<form>`. У тега `<form>` мы указали адрес сценария обработки отправки формы и метод передачи данных. Вместо `"post"` тут можно было бы указать `"get"`, тогда параметры и значения передались бы через URL (как правило, используют все же `"post"`).

Параметры, переданные методом POST, содержатся в системной переменной `$_POST`. Это так же, как и `$_GET`, ассоциативный массив (словарь). Наш сценарий проверяет, были ли переданы методом POST аргументы операции сложения. Если да, пользователю возвращается результат, иначе — форма ввода.

Резюме

В этом уроке мы, наконец, перешли от скучной теории к долгожданной практике! Теперь вы можете писать сценарии, обрабатывающие настоящие HTTP-запросы.

Теперь вы понимаете разницу между GET и POST, знаете, что такое URL и как передать через него параметры запросу. И самое главное — вы знаете, как обработать эти параметры и отправку формы.

Задания

1. Обязательно создайте сценарии, приведенные в качестве примеров в этом уроке.
2. Измените сумматор таким образом, чтобы два режима его отображения объединить в один. Как это должно выглядеть, показано на рис. 7.6.

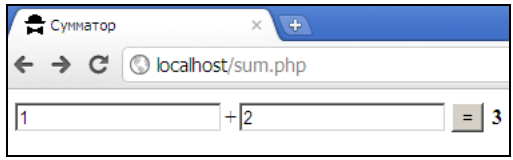
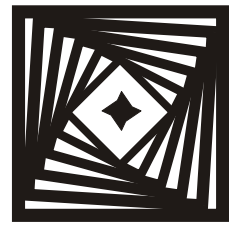


Рис. 7.6. Новый режим работ сумматора

3. Превратите получившийся сумматор в калькулятор с четырьмя операциями: сложение, вычитание, умножение, деление. Не забудьте обработать деление на ноль!



УРОК 8

Cookies и сессии

Что такое cookies и с чем их едят

Cookies (или, как принято называть их в рунете, "куки") переводятся с английского языка как "печенки". В контексте веб — это небольшой кусочек текстовой информации. Cookies нужны для того, чтобы сайт мог сохранить некоторые данные на компьютере пользователя. Например, когда вы авторизуетесь на сайте и устанавливаете флажок **Запомнить меня**, ваши данные сохраняются в cookies браузера. Если вы оставляли комментарии на сайте Школы программирования (**http://prog-school.ru**) более одного раза, то могли заметить, что при отправке второго комментария ваше имя и адрес e-mail уже введены. Это также реализовано с помощью cookies.

Cookies различных сайтов изолированы друг от друга. Представляют они собой пары "имя = значение". Для простоты можно представить их как текстовый файл такого вида:

```
yandex.ru:
  login = dima
  password = a123
  some_name = some_value

prog-school.ru:
  comment_name = dr.zlo
  comment_email = dmitriy@yandex.ru

proglive.ru:
  comment_name = dr.zlo
  comment_email = dmitriy@yandex.ru
```

На самом деле, каждый браузер хранит cookies по-своему, что сейчас не имеет для нас с вами никакого значения.

Управляет cookies удаленный сервер, т. е. он может их читать, добавлять и изменять. А для хранения информации используется компьютер пользователя.

Как это работает?

Давайте вспомним, как происходит обмен информацией в Интернете. Запрашивая страницу, браузер отправляет веб-серверу короткий текст с HTTP-запросом. Например, для доступа к странице **http://www.example.org/index.html** браузер отправляет на сервер **www.example.org** следующий запрос:

```
GET /index.html HTTP/1.1
Host: www.example.org
```

Сервер отвечает, отправляя запрашиваемую страницу вместе с текстом, содержащим HTTP-ответ. Там может содержаться указание браузеру сохранить cookies:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value

(содержимое страницы)
```

Строка `Set-Cookie` отправляется лишь тогда, когда сервер желает, чтобы браузер сохранил cookies. В этом случае, если cookies поддерживаются браузером и их прием включен, веб-обозреватель запоминает строку `name=value` (имя = значение) и отправляет ее обратно серверу с каждым последующим запросом. Например, при запросе страницы **http://www.example.org/spec.html** браузер пошлет серверу **www.example.org** следующий запрос:

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: name=value
Accept: */*
```

Этот запрос отличается от первого запроса тем, что содержит строку, которую сервер отправил браузеру ранее. Таким образом, сервер узнает, что этот запрос связан с предыдущим. Сервер отвечает, отправляя запрашиваемую страницу и, возможно, добавив новые cookies. Количество cookies для одного сайта никак не ограничивается.

Кроме пары "имя = значение", cookie может содержать срок действия, путь и доменное имя. Домен и путь говорят браузеру, что cookie должно быть отправлено обратно на сервер при запросах URL для указанного домена и пути. Если они не указаны, используются домен и путь запрошенной страницы.

Фактически cookies определяются тройкой параметров "имя—домен—путь". Иными словами, cookies с разными путями или доменами являются разными, даже если имеют одинаковые имена.

Дата истечения указывает браузеру, когда удалить cookies. Если срок истечения не указан, cookies удаляется по окончании пользовательского сеанса, т. е. с закрытием браузера.

Дата истечения указывается в формате "Нед, ДД-Мес-ГГГГ ЧЧ:ММ:СС GMT". Например:

```
Set-Cookie: RMID=732423sdfs73242; expires=Fri, 31-Dec-2011 23:59:59 GMT;
path=/; domain=.example.net
```

Cookie из примера имеет имя `RMID` и значение `732423sdfs73242`. Срок его хранения истечет 31 декабря 2011 года в 23:59:59. Путь "/" и домен `example.net` показывают браузеру, что нужно отправить cookies при просмотре любой страницы в домене **example.net**.

Манипулируем cookies средствами PHP

PHP имеет удобный набор функций для работы с cookies. Чтобы установить cookies на период сессии (до закрытия браузера), воспользуйтесь функцией `setcookie()`. Например, вот так:

```
setcookie("name", "value");
```

Если вы хотите запомнить cookies на некоторое время, предположим, на неделю, воспользуйтесь той же функцией, но с тремя параметрами:

```
setcookie("name", "value", time() + 3600 * 24 * 7);
```

Здесь последний параметр — время истечения cookies. Он указывается в формате timestamp. Timespamp — это число секунд, прошедших с 00:00:00 1 января 1970 года.

Функция `time()` возвращает текущее время, к нему мы прибавляем неделю ($3600 \times 24 \times 7$ секунд). Начинающему программисту может показаться странным такой формат времени, однако, поверьте, очень часто намного удобнее работать с целочисленным значением, чем со строковым представлением даты.

Функцию `setcookie()` можно вызывать и с большим числом параметров, если есть необходимость установить прочие атрибуты cookies. Однако в большинстве случаев достаточно двух приведенных ранее примеров.

Чтобы прочитать cookies, следует воспользоваться системной переменной `$_COOKIES`:

```
echo $_COOKIE['name'];
```

Проверить, установлены ли cookies, можно так:

```
if (isset($_COOKIE['name']))
 echo $_COOKIE['name'];
```

Кстати, используя словарь `$_COOKIES`, можно и сохранять значения cookies, но только на период сессии (без указания времени истечения). Операция

```
$_COOKIE['name'] = 'value';
```

аналогична

```
setcookie("name", "value");
```

Что такое сессии PHP и как они работают?

Протокол HTTP является протоколом "без сохранения состояния" и не имеет встроенной возможности поддерживать сеанс работы с сайтом. Иными словами, если не прибегать к различным ухищрениям, сайт не "помнит" ваших предыдущих взаимодействий.

Представьте себе интернет-магазин. Пользователь выбирает товар, кладет его в корзину, а затем оплачивает. Для оформления покупки нужно посетить несколько страниц. Причем сайт должен понимать, что это один и тот же пользователь, а также запоминать товары, которые он отложил в корзину.

Для решения таких задач в PHP реализован механизм поддержки *сессий*. Если есть необходимость запоминать состояние сеанса, все, что нужно сделать, — это вызвать в начале сценария функцию `session_start()`.

```
session_start();
```

Эта функция проверит, существует ли идентификатор сессии. Если нет, то он будет выделен и создастся файл, в котором можно будет сохранять информацию, актуальную в рамках сессии (например, список товаров в корзине).

Но где искать этот самый идентификатор сессии? PHP предлагает два механизма:

- ❒ хранить идентификатор сессии в cookies;
- ❒ добавлять идентификатор сессии к внутренним ссылкам в качестве параметра URL.

Последний вариант имеет смысл только тогда, когда в браузере отключены cookies. Он плох тем, что внутренние ссылки станут выглядеть не очень красиво. Например, ссылка **index.php?page=main** превратится в

index.php?page=main&PHPSESSID=1dc9fcb731a123ec16fb2e49ece325ed

Мы рекомендуем пользоваться только первым вариантом. Сейчас все современные браузеры поддерживают работу с cookies, и можно смело полагаться на то, что сессии на вашем сайте будут работать с их помощью. Если пользователь отключил cookies в браузере, можно считать, что ему не нужна поддержка сеансов работы с веб-сайтами.

Информация, актуальная в рамках сессии, хранится в системной переменной (словаре) `$_SESSION`. Вот так можно сохранить данные:

```
$_SESSION['username'] = 'Vasya';
```

А вот так прочитать:

```
echo 'Привет, ' . $_SESSION['username'] . '!';
```

Когда сессия больше не нужна, например пользователь нажал кнопку **Выход**, следует уничтожить ее. Для этого существует функция `session_destroy()`. Однако перед ее вызовом нужно удалить все сохраненные в сессии данные. Например, вот так:

```
unset($_SESSION['username']);
session_destroy();
```

Практическое использование cookies и сессий. Авторизация на сайте

Давайте перейдем к практике. Сейчас мы создадим сайт с авторизацией. Он будет состоять из трех страниц:

- страница авторизации;
- страница с буквой "А";
- страница с буквой "Б".

Страницы "А" и "Б" могут посещать только авторизованные пользователи. Выглядеть это будет так, как показано на рис. 8.1—8.3.

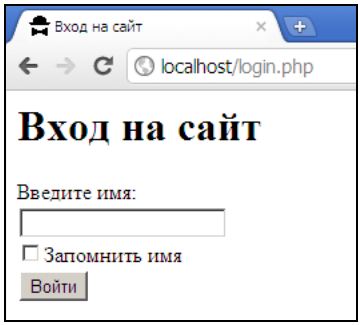


Рис. 8.1. Страница входа на сайт

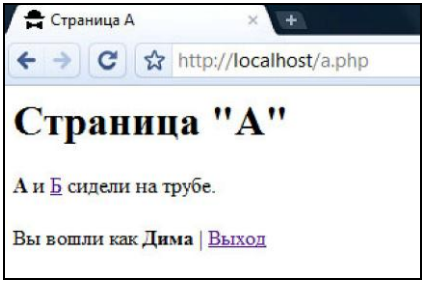


Рис. 8.2. Первая страница закрытой части сайта

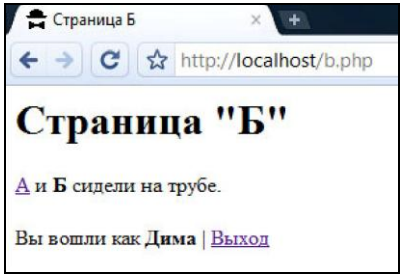


Рис. 8.3. Вторая страница закрытой части сайта

Начнем со страницы login.php (листинг 8.1).

Листинг 8.1. Код страницы авторизации

```
<?php
// АВТОРИЗАЦИЯ.
// Параметр $username — имя пользователя.
// Параметр $remember — булева переменная, указывающая на то,
// желает ли пользователь быть автоматически авторизованным
// при следующих заходах на сайт
```

```
function Login($username, $remember)
{
  // Имя не должно быть пустой строкой.
  if ($username == '')
    return false;

  // Запоминаем имя в сессии...
  $_SESSION['username'] = $username;

  // и в cookies, если пользователь пожелал запомнить его (на неделю).
  if ($remember)
    setcookie('username', $username, time() + 3600 * 24 * 7);

  // Успешная авторизация.
  return true;
}

// СБРОС АВТОРИЗАЦИИ.
//
function Logout()
{ // Делаем cookies устаревшими (единственный способ их удаления).
  setcookie('username' , '', time() — 1);

  // Сброс сессии.
  unset($_SESSION['username']);
}

// ТОЧКА ВХОДА.
//
session_start();
$enter_site = false;

// Попадая на страницу login.php, авторизация сбрасывается.
Logout();

// Если массив POST не пуст, значит, обрабатываем отправку формы.
if (count($_POST) > 0)
  $enter_site = Login($_POST['username'], $_POST['remember'] == 'on');

// Если авторизация пройдена, переадресуем пользователя
// на одну из страниц сайта.
if ($enter_site)
{
  header("Location: a.php");
  exit();
}
?>
```

```
<html>
<head>
  <title>Вход на сайт</title>
</head>
<body>
  <h1>Вход на сайт</h1>
  <form action="" method="post">
    Введите имя:
    <br/>
    <input type="text" name="username" />
    <br/>
    <input type="checkbox" name="remember" /> Запомнить меня
    <br/>
    <input type="submit" value="Войти" />
  </form>
</body>
</html>
```

Комментарии в коде вполне позволяют понять смысл происходящего. При вводе имени пользователь перенаправляется на страницу a.php. Вот так происходит перенаправление:

```
header("Location: a.php");
exit();
```

Вызовом функции `header()` мы отправляем заголовок, говорящий браузеру о том, что необходимо обратиться к другой странице. Функция `exit()` прекращает выполнение сценария. Ее мы вызываем для того, чтобы HTML-код формы авторизации не был передан в ответе.

Переходим к странице a.php (листинг 8.2).

Листинг 8.2. Страница a.php

```
<?php
// ТОЧКА ВХОДА.
//
session_start();

// Если в контексте сессии не установлено имя пользователя,
// пытаемся взять его из cookies.
if (!isset($_SESSION['username']) && isset($_COOKIE['username']))
  $_SESSION['username'] = $_COOKIE['username'];

// Еще раз ищем имя пользователя в контексте сессии.
$username = $_SESSION['username'];

// Неавторизованных пользователей отправляем на страницу регистрации.
if ($username == null)
```

```
{
  header("Location: login.php");
  exit();
}
?>

<html>
<head>
  <title>Страница A</title>
</head>
<body>
  <h1>Страница "A"</h1>
  <b>A</b> и <a href="b.php">Б</a> сидели на трубе.
  <br/>
  <br/>
  Вы вошли как <b><?php echo $username; ?></b> | <a
href="login.php">Выход</a>
</body>
</html>
```

Вначале мы проверяем факт авторизации. Неавторизованные пользователи отправляются на страницу входа, доступ к содержимому a.php им закрыт.

Рассмотрим подробнее проверку авторизации. Если имени пользователя нет в контексте сессии, то оно может быть запомнено в cookies. В этом случае копируем его из cookies в файл сессии.

```
if (!isset($_SESSION['username']) && isset($_COOKIE['username']))
  $_SESSION['username'] = $_COOKIES['username'];
```

Затем снова проверяем контекст сессии.

```
$username = $_SESSION['username'];
```

Если и на этот раз имя пользователя найти не удалось, значит, отправляем его на страницу входа.

```
if ($username == null)
{
  header("Location: login.php");
  exit();
}
```

Ссылка **Выход** ведет на страницу входа, которая сбрасывает данные авторизации.

Страница b.php почти такая же (листинг 8.3).

Листинг 8.3. Страница b.php

```
<?php
// ТОЧКА ВХОДА.
//
session_start();
```

```
// Если в контексте сессии не установлено имя пользователя,
// пытаемся взять его из cookies.
if (!isset($_SESSION['username']) && isset($_COOKIE['username']))
  $_SESSION['username'] = $_COOKIE['username'];

// Еще раз ищем имя пользователя в контексте сессии.
$username = $_SESSION['username'];

// Неавторизованных пользователей отправляем на страницу регистрации.
if ($username == null)
{
  header("Location: login.php");
  exit();
}
?>

<html>
<head>
  <title>Страница Б</title>
</head>
<body>
  <h1>Страница "Б"</h1>
  <a href="a.php">А</a> и <b>Б</b> сидели на трубе.
  <br/>
  <br/>
  Вы вошли как <b><?php echo $username; ?></b> | <a
href="login.php">Выход</a>
</body>
</html>
```

Резюме

На сегодняшний день огромное количество интернет-сайтов имеет возможность авторизации. Теперь вы знаете, как это реализуется в PHP. Концепция сессий и cookies может показаться сложной на первый взгляд, но с опытом вы будете уверенно пользоваться этими инструментами. Сейчас мы настоятельно рекомендуем вам собственноручно реализовать пример, приведенный в этой главе.

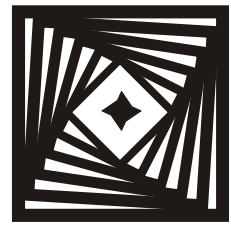
Вы можете также сделать какой-либо другой сайт с сохранением состояния между запросами, например интернет-магазин. Для этого служит механизм сессий, который в свою очередь реализован на основе cookies (как правило).

Cookies можно использовать и на период более длительный, чем сессия для хранения различной информации. Например, для блога с возможностью оставлять комментарии без регистрации удобно хранить псевдоним и e-mail пользователя.

PHP предоставляет удобные механизмы работы с сессиями и cookies: это системные переменные `$_SESSION` и `$_COOKIE`, а также функции `session_start()`, `session_destroy()`, `setcookie()`.

Задания

1. Создайте главную страницу сайта index.php, которая будет отправлять неавторизованных пользователей на страницу авторизации, а авторизованных — на ту страницу, которую они посещали последний раз ("А" или "Б"). Для пользователя главной страницы (index.php) видно не будет, она нужна только для перенаправления.
2. Придумайте, чем еще дополнить сайт, и реализуйте это. Дополнение должно быть связано с сессиями и/или cookies. Творческий подход очень важен в обучении.



УРОК 9

Работа с файлами

Особенности работы с файлами в PHP

При работе с файлами в PHP можно забыть о различии прямых и обратных слэшей для UNIX-систем и Windows. Вы можете писать их как угодно, интерпретатор самостоятельно преобразует путь к нужному виду. Мы рекомендуем всегда использовать прямые слэши, вот такие:

```
media/images/logo.png
```

Вы можете работать с файлами на удаленных серверах в точности так же, как и со своими собственными. Если вы предваряете имя файла строкой http:// или ftp://, то PHP "понимает", что нужно на самом деле установить сетевое соединение и работать именно с файлом с удаленного сервера. При этом при работе такой файл не будет ничем отличаться от обычного.

Два режима работы с файлом

Можно выделить два режима работы с файлом:

- текстовый;
- бинарный.

Первый режим предполагает, что файл состоит из печатных символов, т. е. букв, цифр и тех знаков, которые есть на клавиатуре. С текстовыми файлами обычно работают построчно, т. е. записывают и читают по одной строке. Для разделения строк используют специальные символы переноса строки.

Бинарный режим служит для работы с любыми файлами. В этом случае читают и записывают не строки текста, а байты, т. е. двоичную информацию.

В PHP в обоих случаях при чтении и записи используется строковый тип данных, т. к. типа "массив байтов" здесь просто нет. Но отличие все же есть, и заключается оно в трансляции символа переноса строки.

В системах семейства UNIX для переноса строки используется символ "\n", в то время как в Windows — последовательность из двух символов "\r\n". Трансляция

символа переноса строки нужна для того, чтобы программист не зависел от операционной системы. Таким образом, можно считать, что для новой строки следует всегда использовать "\n", интерпретатор сам откорректирует чтение и запись.

Функции для работы с файлами

Прежде чем приступить к обзору наиболее важных функций для работы с файлами, мы должны познакомиться с понятием "дескриптор файла". *Дескриптор* — это уникальный номер, который используется для идентификации объектов во время выполнения программы. Например, если *одновременно* открывается несколько файлов или соединений, их требуется как-то различать друг от друга, для этого используются дескрипторы. Чтобы произвести какие-либо манипуляции с файлом, вам необходимо сообщить соответствующей функции его дескриптор. Тогда интерпретатор будет знать, над каким файлом требуется произвести то или иное действие. Теперь давайте рассмотрим наиболее популярные функции для работы с файлами.

`int fopen(string $filename, string $mode, bool $use_include_path=false)`

Открывает файл с именем `$filename` в режиме `$mode` и возвращает дескриптор открытого файла. Если операция "провалилась", то, как это принято, `fopen()` возвращает `false`.

Необязательный параметр `$use_include_path` говорит PHP о том, что, если задано относительное имя файла, его следует искать также и в списке путей, используемом инструкциями `include` и `require`. Обычно этот параметр не используют.

Параметр `$mode` может принимать следующие значения:

- `r` — файл открывается только для чтения. Если файл не существует, вызов регистрирует ошибку. После удачного открытия указатель файла устанавливается на его первый байт, т. е. на начало;
- `r+` — файл открывается одновременно на чтение и запись. Указатель текущей позиции устанавливается на его первый байт. Как и для режима `r`, если файл не существует, возвращается `false`;
- `w` — создает новый пустой файл. Если на момент вызова уже был файл с таким именем, то он предварительно уничтожается;
- `w+` — аналогичен `r+`, но если файл изначально не существовал, создает его. После этого с файлом можно работать как в режиме чтения, так и записи. Если файл существовал до момента вызова, его содержимое удаляется;
- `a` — открывает существующий файл в режиме записи и при этом сдвигает указатель текущей позиции за последний байт файла. Этот режим полезен, если требуется что-то дописать в конец уже имеющегося файла. Вызов неудачен в случае отсутствия файла;
- `a+` — открывает файл в режиме чтения и записи, указатель файла устанавливается на конец файла, при этом содержимое файла не уничтожается. Отличается от

`a` тем, что если файл изначально не существовал, то он создается. Этот режим полезен, если вам нужно что-то дописать в файл (например, в журнал), но вы не знаете, создан ли уже такой файл.

В конце любой из строк `r`, `w`, `a`, `r+`, `w+` и `a+` может находиться еще один необязательный символ — `b` или `t`. Если указан `b` (или не указан вообще никакой), то файл открывается в режиме бинарного чтения/записи. Если же это `t`, то для файла устанавливается режим трансляции символа перевода строки, т. е. он воспринимается как текстовый.

`int tmpfile()`

Создает новый файл с уникальным именем и открывает его на чтение и запись. В дальнейшем вся работа должна вестись с возвращенным файловым дескриптором, потому что имя файла недоступно. Функцию следует использовать, если нужен временный файл, который при завершении программы хотелось бы удалить. Пространство, занимаемое временным файлом, автоматически освобождается при его закрытии и при завершении работы программы.

`int fclose(int $fp)`

Закрывает файл. Возвращает истину при успехе, `false` при "провале". PHP автоматически закрывает открытые файлы при завершении выполнения сценария. Однако хорошим тоном считается самостоятельно закрывать дескриптор сразу после завершения работы с файлом.

`string fread(int $f, int $numbytes)`

Из файла `$f` читает `$numbytes` символов и возвращает строку этих символов. После чтения указатель файла продвигается к следующим после прочитанного блока позициям (это происходит и для всех остальных функций чтения и записи). Если `$numbytes` больше, чем можно прочитать из файла (например, раньше достигается конец файла), возвращается то, что удалось считать.

`int fwrite(int $f, string $st)`

Записывает в файл `$f` все содержимое строки `$st`. Эта функция составляет пару для `fread()`, действуя "в обратном направлении". При работе с текстовыми файлами (т. е. когда указан символ `t` в режиме открытия файла) все "\n" автоматически преобразуются в тот разделитель строк, который принят в вашей операционной системе.

`string fgets(int $f, int $length)`

Читает из файла одну строку, заканчивающуюся символом новой строки "\n". Этот символ также считывается и включается в результат. Если строка в файле занимает больше `$length-1` байтов, то возвращаются только ее `$length-1` символов. Функция полезна, если вы открыли файл и хотите "пройтись" по всем его строкам. Однако

даже в этом случае лучше (и быстрее) будет воспользоваться функцией `file()`, которая рассматривается далее.

`int fputs(int $f, string $st)`

Эта функция — синоним `fwrite()`. Отличие в том, что пару `fread`/`fwrite` принято использовать при работе с бинарными файлами, а пару `fputs`/`fgets` — с текстовыми (когда запись и чтение осуществляются построчно).

`int feof(int $f)`

Возвращает `true`, если достигнут конец файла.

`int fseek(int $f, in $offset, int $whence = SEEK_SET)`

Устанавливает указатель файла на байт со смещением `$offset` (от начала файла, от его конца или от текущей позиции, в зависимости от параметра `$whence`).

Параметр `$whence`, как уже упоминалось, задает, с какого места отсчитывается смещение `$offset`. В PHP для этого существуют три константы, равные соответственно 0, 1 и 2:

- `SEEK_SET` — отсчитывает позицию относительно начала файла;
- `SEEK_CUR` — отсчитывает позицию относительно текущей позиции;
- `SEEK_END` — отсчитывает позицию относительно конца файла.

В случае использования последних двух констант параметр `$offset` вполне может быть отрицательным (а при применении `SEEK_END` он будет отрицательным наверняка).

Как это ни странно, но в случае успешного завершения эта функция возвращает 0, а в случае неудачи — –1. Должно быть, так сделано по аналогии с Си-эквивалентом функции.

`int ftell(int $f)`

Возвращает положение указателя файла. Вот так можно определить длину файла:

```
$f = fopen('x.txt', 'r');
fseek($f, 0, SEEK_END);
$size = ftell($f);
echo "Размер файла: $size байт";
fclose($f);
```

`int filesize(string $filename)`

Более простой способ определения размера файла.

`bool file_exists(string $filename)`

Возвращает `true`, если файл с именем `$filename` существует.

`bool is_file(string $filename)`

Возвращает `true`, если `$filename` — обычный файл.

`bool is_dir(string $filename)`

Возвращает `true`, если `$filename` — каталог.

`string basename(string $path)`

Выделяет основное имя файла из пути `$path`.

`string dirname(string $path)`

Возвращает имя каталога, выделенное из пути `$path`.

`bool copy(string $src, string $dst)`

Копирует файл с именем `$src` в файл с именем `$dst`. При этом, если файл `$dst` на момент вызова существовал, осуществляется его перезапись. Функция возвращает `true`, если копирование прошло успешно, а в случае провала — `false`.

`bool rename(string $oldname, string $newname)`

Переименовывает (или перемещает, что одно и то же) файл с именем `$oldname` в файл с именем `$newname`. Если файл `$newname` уже существует, регистрируется ошибка, и функция возвращает `false`.

`bool unlink(string $filename)`

Удаляет файл с именем `$filename`. В случае неудачи возвращает `false`, иначе — `true`.

`array file(string $filename)`

Считывает файл с именем `$filename` целиком и возвращает массив, каждый элемент которого соответствует строке в прочитанном файле. Функция работает очень быстро — гораздо быстрее, чем если бы мы использовали `fopen()` и читали файл по одной строке.

Неудобство этой функции состоит в том, что символы конца строки не вырезаются из строк файла, а также не транслируются, как это делается для текстовых файлов. Так что, каждый элемент массива иногда имеет смысл преобразовать с помощью функции `rtrim()`.

Это не полный перечень функций для работы с файлами, однако на первое время их более чем достаточно. Если вам потребовалось что-то сделать с файлом и вы не нашли в приведенном выше списке нужной функции, обратитесь к справочнику! Наверняка такая функция есть в PHP.

Журнал посещений сайта

Пришло время попрактиковаться в работе с файлами. Первый учебный пример — журнал посещений сайта.

Когда посетитель попадает на главную страницу, будем фиксировать время захода, IP-адрес и страницу, откуда он к нам пришел. Просматривать эту информацию мы будем с помощью страницы журнала посещений.

Как должна выглядеть главная страница, показано на рис. 9.1, а страница журнала посещений представлена на рис. 9.2.

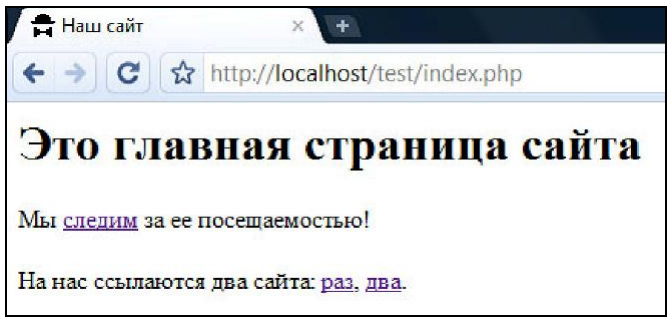
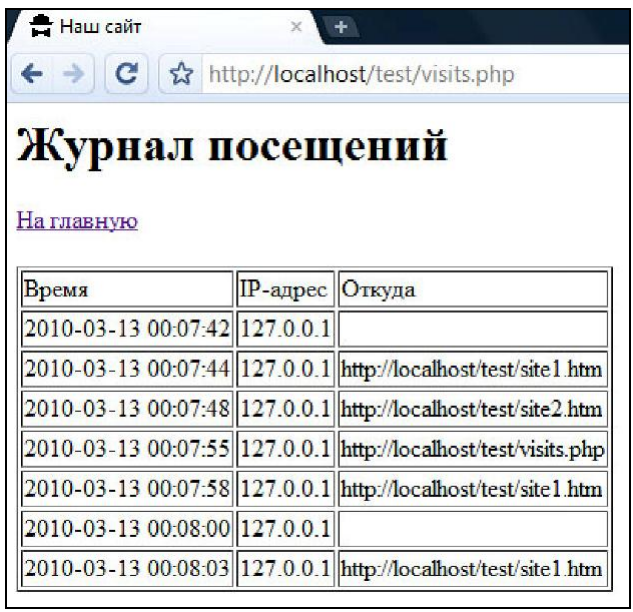


Рис. 9.1. Главная страница сайта с журналом посещений



Время	IP-адрес	Откуда
2010-03-13 00:07:42	127.0.0.1	
2010-03-13 00:07:44	127.0.0.1	http://localhost/test/site1.htm
2010-03-13 00:07:48	127.0.0.1	http://localhost/test/site2.htm
2010-03-13 00:07:55	127.0.0.1	http://localhost/test/visits.php
2010-03-13 00:07:58	127.0.0.1	http://localhost/test/site1.htm
2010-03-13 00:08:00	127.0.0.1	
2010-03-13 00:08:03	127.0.0.1	http://localhost/test/site1.htm

Рис 9.2. Журнал посещений сайта

Помимо этого добавим две статичные странички, которые просто ссылаются на главную страницу нашего сайта (рис. 9.3 и 9.4). Они нужны для того, чтобы продемонстрировать возможность определять, откуда пользователь попал на сайт.

Рис. 9.3. Первый сайт со ссылкой на главный сайт

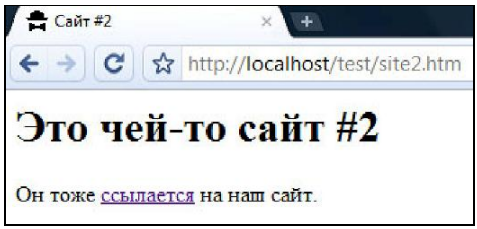


Рис. 9.4. Второй сайт со ссылкой на главный сайт

Информацию о посещениях мы будем записывать в файл visits.txt:

```
2010-03-13 00:07:42
127.0.0.1

2010-03-13 00:07:44
127.0.0.1
http://localhost/test/site1.htm
2010-03-13 00:07:48
127.0.0.1
http://localhost/test/site2.htm
2010-03-13 00:07:55
127.0.0.1
http://localhost/test/visits.htm
2010-03-13 00:07:58
127.0.0.1
http://localhost/test/site1.htm
2010-03-13 00:08:00
127.0.0.1

2010-03-13 00:08:03
127.0.0.1
http://localhost/test/site1.htm
```

Файл состоит из троек:

- дата и время;
- IP-адрес;
- откуда пришел.

Поле "откуда пришел" может быть пустым, если это прямой заход на сайт (когда адрес набирается вручную).

Теперь рассмотрим исходные тексты сценариев. Файл index.php приведен в листинге 9.1.

Листинг 9.1. Главная страница сайта с журналом посещений

```
<?php
// Запись в файл информации о посещении страницы.
  $f = fopen('visits.txt', 'a+');
```

```
  fwrite($f, date('Y-m-d H:i:s') . "\n");
  fwrite($f, $_SERVER['REMOTE_ADDR'] . "\n");
  fwrite($f, $_SERVER['HTTP_REFERER'] . "\n");
  fclose($f);
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
        content="text/html; charset=windows-1251">
  <title>Наш сайт</title>
</head>
<body>
  <h1>Это главная страница сайта</h1>
  Мы <a href="visits.php">следим</a> за ее посещаемостью!
  <br/><br/>
  На нас ссылаются два сайта: <a href="site1.htm">раз</a>, <a
href="site2.htm">два</a>.
</body>
</html>
```

Мы открываем файл режимом `a+`, что означает следующее:

- ❒ файл будет открыт для работы в текстовом режиме;
- ❒ данные будут записываться в конец файла;
- ❒ если файл не существует, он будет создан.

(См. список режимов открытия файла ранее в этой главе.)

При открытии файла мы получаем его целочисленный дескриптор, который будем использовать при дальнейшей работе. Далее мы записываем в файл три строки. Обратите внимание, что для перехода на новую строку используется последовательность "\n". Ее следует писать в двойных кавычках, тогда она заменится на спецсимвол переноса строки. Если же '\n' написать в апострофах, то это будет просто символ "\", за которым следует символ "n".

Наконец, когда работа с файлом завершена, его следует закрыть. Для этого служит функция `fclose()`.

Теперь рассмотрим код журнала посещений (листинг 9.2).

Листинг 9.2. Страница журнала посещений

```
<?php
// Чтение из файла всей информации о посещениях.
$lines = file('visits.txt');
?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
        content="text/html; charset=windows-1251">
  <title>Наш сайт</title>
</head>
<body>
  <h1>Журнал посещений</h1>
  <a href="index.php">На главную</a>
  <br/>
  <br/>
  <table border="1">
  <tr>
    <td>Время</td>
    <td>IP-адрес</td>
    <td>Откуда</td>
  </tr>
  <?php
    $n = count($lines);
    for ($i = 0; $i < $n; $i += 3)
    {
      echo '<tr>';
      echo '<td>' . $lines[$i + 0] . '</td>';
      echo '<td>' . $lines[$i + 1] . '</td>';
      echo '<td>' . $lines[$i + 2] . '</td>';
      echo '</tr>';
    }
  ?>
  </table>
</body>
</html>
```

Мы легко читаем все строки файла в массив с помощью функции `file()`. При использовании этой функции не нужно открывать и закрывать файл — она все делает сама.

Исходный код статичных страничек site1.htm и site2.htm ничем не примечателен (листинг 9.3). Они просто содержат ссылку на главную страницу сайта.

Листинг 9.3. Страницы со ссылкой на главный сайт

site1.htm:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
        content="text/html; charset=windows-1251">
```

```
  <title>Сайт #1</title>
</head>
<body>
  <h1>Это чей-то сайт #1</h1>
  Он <a href="index.php">ссылается</a> на наш сайт.
</body>
</html>
```

site2.htm:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
        content="text/html; charset=windows-1251">
  <title>Сайт #2</title>
</head>
<body>
  <h1>Это чей-то сайт #2</h1>
  Он тоже <a href="index.php">ссылается</a> на наш сайт.
</body>
</html>
```

Загрузка файлов на сервер

Существует определенный механизм загрузки файлов на сервер, отработаем его на следующем примере.

Предположим, что пользователь может выбрать файл и отправить его на сервер. Далее на сайте отобразится текст этого файла, причем строки будут пронумерованы. Этот сценарий не будет иметь большого практического смысла, однако как пример загрузки файлов на сервер подойдет.

Форма загрузки будет выглядеть так, как показано на рис. 9.5, а вывод файла представлен на рис. 9.6.

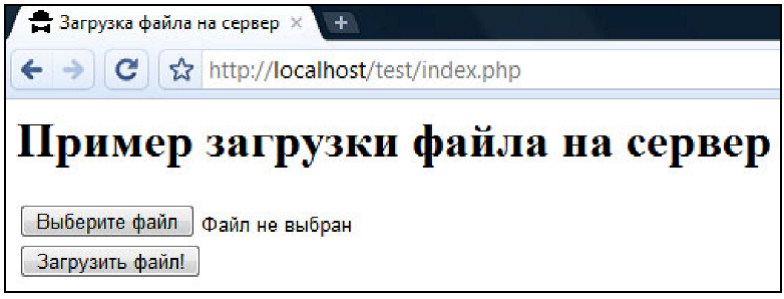


Рис. 9.5. Форма загрузки файлов на сервер

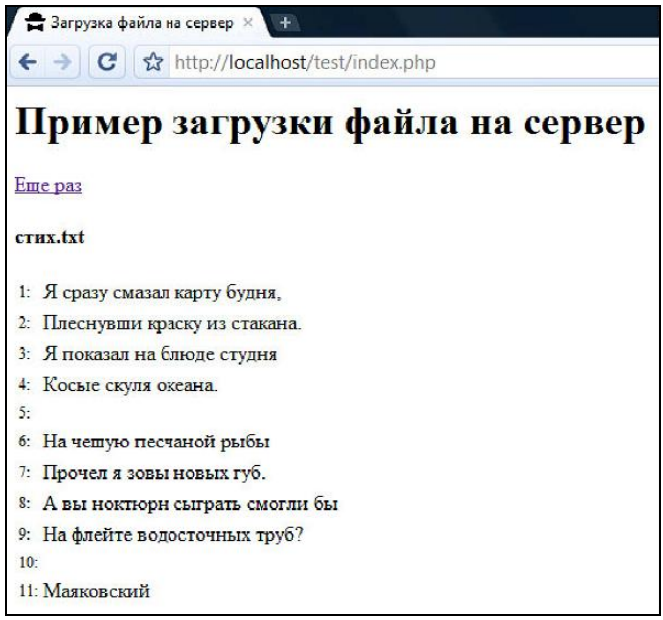


Рис 9.6. Вывод содержимого файла на экран

Загружать следует только текстовые файлы, т. к. проверки на тип файла в сценарии нет.

Теперь рассмотрим код сценария (листинг 9.4).

Листинг 9.4. Сценарий загрузки файла на сервер

```
<?php
// Функция вывода формы отправки файла.
function print_form()
{
  echo '<form method="post" enctype="multipart/form-data">';
  echo '<input type="file" name="text" />';
  echo '<br/>';
  echo '<input type="submit" value="Загрузить файл!" />'
  echo '</form>';
}

// Функция вывода содержимого файла.
function print_file($file)
{
  echo '<a href="index.php">Еще раз</a>';
  echo '<br/><br/>';

  if ($file['name']  == '')
  {
    echo 'Файл не выбран!'
```

```
    return;
  }

  $lines = file($file['tmp_name']);
  $i = 1;

  echo "<b>" . $file['name'] . "</b>";
  echo "<br/><br/>";
  echo "<table>";

  foreach ($lines as $s)
  {
    echo "<tr><td><small>$i:</small></td><td>$s</td></tr>";
    $i++;
  }

  echo "</table>";
}
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
        content="text/html; charset=windows-1251">
  <title>Загрузка файла на сервер</title>
</head>
<body>
  <h1>Пример загрузки файла на сервер</h1>
  <?php
  if (isset($_FILES['text']))
  {
    print_file($_FILES['text']);
  }
  else
  {
    print_form();
  }
  ?>
</body>
</html>
```

Для загрузки файла используется HTML-элемент `input` с типом "файл":

```
<input type="file" name="text" />
```

При этом важно указать у формы атрибут `enctype`:

```
<form method="post" enctype="multipart/form-data">
```

Если это не сделать, файл не будет загружен на сервер.

При обработке отправки формы информацию о загруженных файлах можно найти в системной переменной (словаре) `$_FILES`. Информация о файле включает:

- `name` — имя файла (как он называется у пользователя);
- `tmp_name` — путь к временному файлу (на сервере);
- `size` — размер файла (в байтах);
- `type` — тип выбранного файла (например, "`image/jpg`");
- `error` — код ошибки (возникает в том случае, если попытка загрузки была неудачной).

Когда браузер отправляет файл на сервер, PHP сохраняет его во временном каталоге. Добраться до файла помогает свойство `tmp_name`, там указан его путь. Вот так мы читаем содержимое файла в массив строк:

```
$lines = file($file['tmp_name']);
```

Функции для работы с каталогами

Очевидно, что организация любого серьезного сайта требует хорошо продуманной структуры хранения информации. А здесь нам не обойтись без работы с каталогами на стороне сервера. Итак, давайте рассмотрим основные функции для работы с каталогами сервера.

bool mkdir(string *$name*, int *$perms*)

Создает каталог с именем *$name* и правами доступа *$perms*. Права доступа для каталогов указываются точно так же, как и для файлов. Чаще всего значение *$perms* устанавливают равным 0770 (предваряющий ноль обязателен — он указывает PHP на то, что это — восьмеричная константа, а не десятичное число).

Атрибут доступа 0770 означает "доступен для чтения, записи и исполнения для владельца и его группы". Атрибут исполнения, установленный для каталога, показывает, что пользователь сможет просмотреть содержимое каталога. Но это специфично для операционных систем семейства UNIX.

В случае успеха функция возвращает `true`, иначе — `false`.

bool rmdir(string *$name*)

Удаляет каталог с именем *$name*. В случае успеха возвращает `true`, иначе — `false`.

bool chdir(string *$path*)

Сменяет текущий каталог на указанный. Если такого каталога не существует, возвращает `false`. Параметр *$path* может определять и относительный путь, задающийся от текущего каталога.

Вот несколько примеров:

```
chdir("/tmp/data");   // переходим по абсолютному пути
chdir("./somathing"); // переходим в подкаталог текущего каталога
chdir("something");   // то же самое
chdir("..");          // переходим в родительский каталог
```

string getcwd()

Возвращает полный путь к текущему каталогу, начиная от "корня" (/). Если такой путь не может быть отслежен (это иногда бывает в UNIX из-за того, что права на чтение для родительских каталогов могут быть сняты), вызов "проваливается" и возвращает `false`.

int opendir(string $*path*)

Открывает каталог `$path` для дальнейшего считывания из него информации о файлах и подкаталогах и возвращает его идентификатор. Дальнейшие вызовы `readdir()` с идентификатором в параметрах будут обращены именно к этому каталогу. Функция возвращает `false`, если произошла ошибка.

string readdir(int $*handle*)

Считывает очередное имя файла или подкаталога из открытого ранее каталога с идентификатором `$handle` и возвращает его в виде строки. Вместе с именами подкаталогов и файлов будут также получены два специальных элемента: это "." (ссылка на текущий каталог) и ".." (ссылка на родительский каталог). В подавляющем большинстве случаев нам нужно их игнорировать. Когда считывать больше нечего функция возвращает `false`.

void closedir(int $*handle*)

Закрывает ранее открытый каталог с идентификатором `$handle`. Не возвращает ничего.

void rewinddir(int $*handle*)

"Перематывает" внутренний указатель открытого каталога на начало. После этого можно воспользоваться функцией `readdir()`, чтобы заново начать считывать содержимое каталога.

Получение списка файлов и подпапок в каталоге

Рассмотрим простой пример: выведем содержимое некоторого каталога (листинг 9.5).

Листинг 9.5. Вывод содержимого каталога

```
<?php
$handle = opendir('/path/to/files');

if ($handle != false)
{
  echo "Дескриптор каталога: $handle<br/>";
  echo "Файлы:<br/>";

  while (false !== ($file = readdir($handle)))
    echo "$file<br/>";

  closedir($handle);
}
?>
```

Обратите внимание на строгую проверку (включающую проверку типов):

```
false !== ($file = readdir($handle))
```

Иначе файл с именем "0" обработался бы неверно.

Резюме

Теперь вы знаете все необходимые функции для работы с файловой системой. Это позволит сохранять информацию на сайте, делать ее доступной многим пользователям.

Однако для хранения текстовых и числовых данных, как правило, используют базу данных. Она позволяет осуществлять поиск записей, связывать данные разных типов, решает проблемы синхронизации. Часто информацию хранят совместно: частично в базе данных (текстовые и числовые данные), частично в виде файлов (картинки, документы, архивы). Работа с базой данных будет рассмотрена в следующем уроке.

Работа с файлами — это еще один шаг на пути к полнофункциональному современному сайту. Обязательно потренируйтесь и выполните предложенное домашнее задание для закрепления навыков работы с файлами и каталогами.

Также не забывайте, что профессиональный программист — это человек, которого не пугает временное незнание той или иной функции, т. к. он может с легкостью получить о ней информацию в многочисленных открытых источниках в Интернете. Поэтому во время выполнения практических занятий потренируйтесь также и в поиске необходимой вам информации, если это потребуется.

Задания

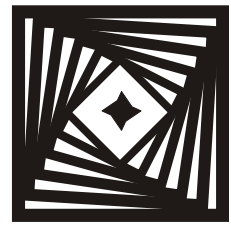
Создайте галерею фотографий.

Требования:

- возможность загрузки фотографии на сервер (при загрузке должен проверяться тип файла) при загрузке должна автоматически создаваться ее уменьшенная копия (не более 200 пикселов по наибольшей стороне);
- возможность просмотра уменьшенных фотографий (все на одной странице);
- возможность увеличения фотографии при щелчке по ней (переход на отдельную страницу).

Комментарии:

- для хранения файлов создайте два каталога — один для маленьких изображений, другой — для больших;
- ответ, как получить уменьшенную копию картинки, самостоятельно найдите в Интернете;
- для получения списка фотографий галереи воспользуйтесь механизмом обхода каталога.



УРОК 10

Работа с базой данных

Перед тем как приступить к уроку, необходимо просмотреть видеоурок "Как создать базу данных MySQL с помощью PhpMyAdmin" по этой ссылке:

http://prog-school.ru/2010/03/kak-sozdat-bazu-dannyx-v-subd-mysql-s-pomoshhyu-phpmyadmin-videourok/.

Вы должны знать, что такое phpMyAdmin, и уметь создавать с его помощью базу данных и таблицы в ней.

Для чего нужна база данных?

База данных — неотъемлемая часть современного веб-приложения. В ней хранится вся информация, необходимая для работы сайта. Исключение составляют файлы: изображения, документы. Они обычно и хранятся в виде файлов в предназначенном для них каталоге на сервере. Однако существует возможность и такую информацию хранить в базе данных, но ее используют редко. Мы также будем придерживаться принципа, по которому текстовую информацию следует хранить в БД, а документы и медиаинформацию — в файлах.

Работа с базами данных происходит с помощью специального языка запросов. Получение информации с помощью операций выборки осуществляется очень быстро благодаря специфике внутреннего устройства хранилища данных. Использование базы данных позволяет не беспокоиться о совместном доступе к данным (что обязательно произойдет, если ваш сайт посещают хотя бы 100 человек в день), все функции синхронизации хранилище берет на себя.

Представьте себе ситуацию, при которой несколько пользователей пытаются одновременно записать и считать информацию из одной и той же строки в файле. Это приведет как минимум к ошибкам в считываемой строке. Поэтому программистам так удобно пользоваться базами данных, которые берут работу по разделению доступа на себя.

Отличие базы данных от СУБД

Следует различать термины "база данных" (БД) и "система управления базами данных" (СУБД). Под первым термином понимается как информация, которую вы храните, так и сама структура этой информации. СУБД — это программа, которая предоставляет доступ внешним приложениям к базам данных, обеспечивает их работу.

База данных проектируется и создается для каждого конкретного проекта, СУБД же выбирается из небольшого списка стандартных средств. На сегодняшний день наиболее популярны СУБД Oracle, Microsoft SQL Server, MySQL, Sybase, PostgreSQL. Сайты PHP чаще всего работают в связке с MySQL. Именно эту СУБД мы будем рассматривать в текущем уроке.

Фраза "я использую на сайте базу данных MySQL" неверная, правильно сказать "я использую на сайте СУБД MySQL", однако в разговорной речи понятие СУБД очень часто подменяется словосочетанием "база данных". Обратите на это внимание!

Реляционная база данных

Базы данных имеют различные типы и принципы организации. Наиболее популярными являются *реляционные базы данных*.

Реляционная база данных состоит из таблиц. Таблица имеет ограниченное количество столбцов (обычно небольшое) и сколь угодно много строк. Другие типы баз данных (не реляционные) используются крайне редко. Их обсуждение для нас на данный момент не актуально.

Такую популярность реляционная модель получила ввиду своей простоты. Ее можно использовать практически для любых задач. Слово "реляционная" происходит от английского *relation* — отношение. Таблицы, из которых состоит реляционная база данных, как правило, связаны друг с другом, т. е. находятся "в отношениях".

Предположим, нам нужно хранить информацию о сотрудниках и отделах (это самый распространенный пример). Приведем пример реляционной базы данных, решающих эту задачу.

Нам потребуются две таблицы: сотрудники и отделы, назовем их `emps` и `depts` соответственно.

Столбцы таблицы отделов (`depts`):

- `id_dept` (первичный ключ);
- `name` (название отдела).

Столбцы таблицы сотрудников (`emps`):

- `id_emp` (первичный ключ);
- `id_dept` (внешний ключ, ссылающийся на таблицу `depts`);
- `first_name` (имя);

- `middle_name` (отчество);
- `last_name` (фамилия).

Каждая строка таблицы описывает одну конкретную сущность. Каждый столбец — это некоторая характеристика сущности, информацию о которой мы храним.

Таблицы реляционной базы данных характерны тем, что каждая должна содержать первичный ключ. Первичный ключ — это столбец с уникальными значениями, т. е. по значению этого столбца можно однозначно идентифицировать строку таблицы. В качестве первичных ключей обычно используются числовые значения.

Каждый сотрудник работает в каком-то определенном отделе. Чтобы зафиксировать такую связь, в таблице `emps` предусмотрен столбец `id_dept`. Он будет хранить идентификатор отдела, в котором работает сотрудник. Столбец `id_dept` в данном случае называется *внешним ключом*.

Пусть в нашей фирме два отдела: "Бухгалтерия" и "Маркетинг". В бухгалтерии работают Иванов Иван Иванович, Петров Петр Петрович и Сидорова Елена Николаевна.

Маркетингом занимаются Ушаков Павел Павлович и Ефремов Илья Викторович.

На рис. 10.1 и 10.2 приведены таблицы, описывающие данную ситуацию.

id_dept	name
1	Бухгалтерия
2	Маркетинг

Рис. 10.1. Таблица `depts`

id_emp	id_dept	first_name	middle_name	last_name
1	1	Иван	Иванович	Иванов
2	1	Петр	Петрович	Петров
3	2	Павел	Павлович	Ушаков
4	2	Илья	Викторович	Ефремов
5	1	Елена	Николаевна	Сидорова

Рис. 10.2. Таблица `emps`

Язык SQL

SQL (Structured Query Language, язык структурированных запросов) — универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. Структурированными запросы называются потому, что в общем случае могут содержать вложенные подзапросы.

Проще говоря, SQL — это язык, на котором можно общаться с базой данных. Любую операцию, от создания таблицы до выборки данных, возможно осуществить

только посредством запроса на языке SQL. Запрос, как правило, отправляется внешней программой к СУБД. Та в свою очередь выполняет его, производя необходимые операции над базой данных, и возвращает результат.

Запросы делятся на два вида:

- DDL;
- DML.

К DDL (Data Definition Language, язык описания данных) относятся запросы, меняющие структуру базы данных. Например, создание таблицы, удаление таблицы, добавление столбца к существующей таблице.

К DML (Data Modification Language, язык модификации данных) относятся запросы, меняющие содержимое базы данных, т. е. операции над строками таблиц. Сюда относятся вставка, удаление, изменение и выборка строк.

Вернемся к базе данных отделов и сотрудников.

С помощью такого запроса можно создать таблицу `depts`:

```
CREATE TABLE depts
(
 id_dept INT NOT NULL,
 name VARCHAR(32) NOT NULL,
 PRIMARY KEY (id_dept)
)
```

А таким запросом — таблицу `emps`:

```
CREATE TABLE emps
(
 id_emp INT NOT NULL,
 id_dept INT NOT NULL,
 first_name VARCHAR(32) NOT NULL,
 middle_name VARCHAR(32),
 last_name VARCHAR(32) NOT NULL,
 PRIMARY KEY (id_emp),
 FOREIGN KEY (id_dept) REFERENCES depts (id_dept)
)
```

Обычно DML-операции приходится производить один раз — при создании базы данных. Их чаще выполняют не вручную, а с помощью специальных утилит. Для MySQL такой утилитой является phpMyAdmin. С помощью ее графического интерфейса можно создать таблицы, не задумываясь о синтаксисе оператора `CREATE TABLE`.

Сценарий, как правило, работает со строками таблиц, не меняя структуру базы данных, т. е. производит только DML-операции: вставку, удаление, изменение и выборку строк.

Вставка строк

Для вставки строк в языке SQL служит оператор `INSERT`. Вот так можно наполнить базу данных сотрудников и отделов:

```
INSERT INTO depts (id_dept, name) VALUES ('1', 'Бухгалтерия');
INSERT INTO depts (id_dept, name) VALUES ('2', 'Маркетинг');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
  VALUES ('1', '1', 'Иванов', 'Иван', 'Иванович');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
  VALUES ('2', '1', 'Петров', 'Петр', 'Петрович');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
  VALUES ('3', '2', 'Ушаков', 'Павел', 'Павлович');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
  VALUES ('4', '2', 'Ефремов', 'Илья', 'Викторович');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
  VALUES ('5', '1', 'Сидорова', 'Елена', 'Николаевна');
```

Удаление строк

Предположим, руководство решило уволить всех сотрудников отдела маркетинга. В этом случае поможет оператор `DELETE`, удаляющий строки из таблицы:

```
DELETE FROM emps WHERE id_dept = '2'
```

Изменение строк

Елена Николаевна вышла замуж за Ивана Ивановича и поменяла фамилию. Для изменения строк таблицы служит оператор `UPDATE`. Воспользуемся им:

```
UPDATE emps
  SET last_name = 'Иванова'
  WHERE id_emp = '5'
```

Выборка строк

Для выборки строк служит оператор `SELECT`. С его помощью можно составлять сложнейшие запросы, выбирающие данные сразу из множества таблиц. В рамках текущего урока рассмотрим лишь самые простые примеры.

Все сотрудники:

```
SELECT *
  FROM emps
```

Сотрудники отдела бухгалтерии:

```
SELECT *
  FROM emps
  WHERE id_dept = '1'
```

Сотрудники отдела бухгалтерии, отсортированные по фамилии, имени, отчеству:

```
SELECT *
  FROM emps
  WHERE id_dept = '1'
  ORDER BY last_name, first_name, middle_name
```

Как зовут сотрудника № 1?

```
SELECT last_name, first_name, middle_name
  FROM emps
  WHERE id_emp = '1'
```

Сколько всего в фирме работает человек?

```
SELECT count(*)
  FROM emps;
```

Средства PHP для работы с MySQL

Подключение к БД

Перед тем как работать с базой данных, необходимо установить с ней соединение. Для этого служит функция `mysql_connect()`. Результат выполнения функции — дескриптор соединения, который пригодится, только если вы собираетесь работать сразу с несколькими подключениями. В большинстве случаев это не требуется, и результат выполнения функции проверяют лишь на неравенство `false` (это означает, что подключение прошло успешно).

```
int mysql_connect (
  [ string $server = ini_get("mysql.default_host") [,
  string $username = ini_get("mysql.default_user") [,
  string$password = ini_get("mysql.default_password") [,
  bool $new_link = false [,
    int $client_flags = 0]]]]])
```

Функция принимает множество параметров, и все они необязательны. Подробнее об их назначении можно прочитать в документации. Чаще всего функция вызывается с тремя параметрами:

```
mysql_connect($server, $username, $password);
```

Если вы тестируете сайт на локальном компьютере и у вас установлен пакет Денвер, то параметры должны быть следующими (если вы, конечно, не меняли конфигурацию):

```
$server = 'localhost';
$username = 'root';
$password = '';
```

Когда сайт работает на удаленном сервере, параметр `$server` остается `"localhost"`, `$username` и `$password` определяются при создании пользователя базы данных. Параметр `localhost` говорит сценарию о том, что база данных находится одном и том же сервере, что и исполняемый сценарий, что является истиной в большинстве случаев и при разработке сайта на локальном компьютере и при работе сценария в Интернете.

Выбор БД

`int mysql_select_db(string $dbname [, int $link_identifier])`

До того как послать первый запрос серверу MySQL, необходимо указать, с какой базой данных мы собираемся работать. Для этого и предназначена функция `mysql_select_db()`.

Она уведомляет PHP, что в дальнейших операциях с соединением `$link_identifier` (или с последним открытым соединением, если указанный параметр не задан) будет использоваться база данных `$dbname`. Функция возвращает `true` в случае успеха, иначе — `false`.

Выполнение SQL-запросов

`int mysql_query(string $query [, int $link_identifier])`

Запрос к базе данных. Текст запроса формулируется на языке SQL. Для запросов на выборку данных функция возвращает идентификатор результата в случае успеха и `false` в случае ошибки. Для запросов, не подразумевающих результат (`INSERT`, `UPDATE`, `DELETE`), функция в случае успеха возвращает `true`.

Получение результата выборки в виде массива

`array mysql_fetch_array(int $result)`

Функция извлекает очередную строку результата выборки данных. В качестве параметра принимает идентификатор, полученный вызовом функции `mysql_query()`. Возвращает массив, состоящий из значений каждого столбца текущей строки. В качестве ключа — порядковый номер столбца (начиная с нуля). Когда извлекать больше нечего, функция возвращает `false`. Пример приведен в листинге 10.1

Листинг 10.1. Получение информации из БД

```
$server = 'localhost';
$username = 'root';
$password = '';
$dbname = 'lesson8';
```

```
mysql_connect($server, $username, $password);
mysql_select_db($dbname);
$result = mysql_query('SELECT id_dept, name FROM depts');

while ($row = mysql_fetch_array($result))
{
  echo '<li>';
  echo '<a href="dept.php?id_dept=' . $row[0] . '">';
  echo $row[1];
  echo '</a>';
  echo '</li>';
}
```

Получение результата выборки в виде ассоциативного массива

array mysql_fetch_assoc(int $*result*)

Функция аналогична `mysql_fetch_array()`, но возвращает ассоциативный массив, в котором в качестве ключа выступает имя столбца (листинг 10.2).

Листинг 10.2. Получение информации из БД

```
$result = mysql_query('SELECT id_dept, name FROM depts');

while ($row = mysql_fetch_assoc($result))
{
  echo '<li>';
  echo '<a href="dept.php?id_dept=' . $row['id_dept'] . '">';
  echo $row['name'];
  echo '</a>';
  echo '</li>';
}
```

Получение числа строк в выборке

int mysql_num_rows(int $*result*)

Функция возвращает число строк, содержащееся в результате выборки данных (листинг 10.3).

Листинг 10.3. Получение информации из БД с вычислением количества строк

```
$result = mysql_query('SELECT id_dept, name FROM depts');
$count = mysql_num_rows($result);

for ($i = 0; $i < $count; $i++)
```

```
{
  $row = mysql_fetch_array($result);
  echo '<li>';

  echo '<a href="dept.php?id_dept=' . $row['id_dept'] . '">';
  echo $row['name'];
  echo '</a>';
  echo '</li>';
}
```

Получение числа измененных строк

int mysql_affected_rows([resource $*link_identifier*])

Функция возвращает число строк, затронутых последним запросом `INSERT`, `UPDATE` или `DELETE` (листинг 10.4).

Листинг 10.4. Удаление всех сотрудников из определенного отдела

```
mysql_query("DELETE FROM emps WHERE id_dept='2'");
$count = mysql_affected_rows();
echo 'Уволены все сотрудники из отдела маркетинга. Их было $count чел.';
```

Получение информации об ошибках

int mysql_errno([int $*link_identifier*])
string mysql_error([int $*link_identifier*])

Если в процессе работы с MySQL возникают ошибки, то сообщение об ошибке и ее номер можно получить с помощью этих двух функций. Первая возвращает номер последней зарегистрированной ошибки. Вторая — строку, содержащую текст сообщения об ошибке. Ее удобно применять в отладочных целях.

Пример приведен в листинге 10.5.

Листинг 10.5. Получение информации об ошибках при работе с БД

```
$result = mysql_query($query_text);

if ($result == false)
{
  $err_code = mysql_errno();
  $err_text = mysql_error();

  die("Ошибка MySQL #$err_code: $err_text" . "<br/>" .
      "при выполнении SQL запроса: query_text");
}
```

Резюме

Тема баз данных слишком обширна, чтобы уместить ее в одном уроке. Однако собранные в документе знания помогут начать изучение этой области и использовать базы данных в ваших проектах уже сейчас!

Ни один серьезный проект не обходится без использования базы данных. Советуем поглубже изучить эту область. Чтобы составлять быстрые грамотные запросы к БД, вы должны хорошо владеть языком SQL. Умение работать с базами данных будет требоваться вам на всем протяжении работы в ИТ.

Задание

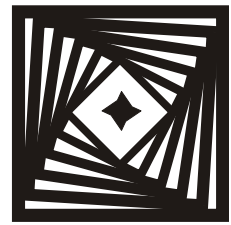
В прошлом уроке домашним заданием было реализовать галерею фотографий с помощью файлов и папок. Теперь необходимо доработать сайт с использованием базы данных.

Требования:

- на странице просмотра фотографии полного размера под картинкой должно быть указано число ее просмотров;
- на странице просмотра галереи список фотографий должен быть отсортирован по популярности. В начале списка должны находиться наиболее популярные фотографии. Популярность определяется числом щелчков по фотографии (просмотров увеличенного изображения).

Комментарии:

- при загрузке фотографии ей следует выделять уникальный идентификатор;
- помимо идентификатора в базе следует хранить тип картинки ("gif", "jpg" и т. д.) и число щелчков по ней;
- уменьшенные изображения надо хранить в отдельной папке с именем вида "идентификатор.тип";
- изображения полного размера также нужно хранить в отдельной папке с именем вида "идентификатор.тип".



УРОК 11

Архитектура сайта

Что такое архитектура программы

Несмотря на то, что значение словосочетания "архитектура сайта" может быть вам интуитивно понятным, давайте попробуем изучить ряд определений, принадлежащих авторитетным источникам.

Архитектура — это базовая *организация системы*, воплощенная в ее *компонентах*, их *отношениях* между собой и с *окружением*, а также принципы, определяющие проектирование и развитие системы [IEEE 1471].

Архитектура программы или компьютерной системы — это *структура* или структуры системы, которые включают элементы программы, *видимые* извне свойства этих *элементов* и *связи* между ними [Басс (Bass)].

Архитектура — это *структура* организации и связанное с ней *поведение* системы. Архитектуру можно рекурсивно разобрать на части, взаимодействующие посредством интерфейсов, связи, которые соединяют части, и условия сборки частей. Части, которые взаимодействуют через интерфейсы, включают классы, компоненты и подсистемы [UML 1.5].

Архитектура программного обеспечения системы или набора систем состоит из всех важных *проектных решений* по поводу *структур* программы и *взаимодействий* между этими структурами, которые составляют системы. Проектные решения обеспечивают желаемый набор свойств, которые должна поддерживать система, чтобы быть успешной. Проектные решения предоставляют *концептуальную* основу для разработки системы, ее поддержки и обслуживания [McGovern (Мак-Говерн)].

Так что же такое архитектура программы?

Приведенные в предыдущем разделе определения слишком сухие для их восприятия неподготовленным читателем. Постараемся объяснить суть как можно проще.

Когда программа становится достаточно большой, программист разбивает ее на несколько файлов. Если не задумываться над выделением групп похожих функций

и вынесением их в отдельные модули, такое разбиение принесет мало пользы. Код нельзя будет использовать повторно, в нем будет трудно ориентироваться. Программу будет сложно расширять и изменять.

Таким образом, назревает первый вопрос: как разбить программу на файлы. Файловая архитектура программы — это один из аспектов ее структуры.

Выделяя модули системы, необходимо понимать, в каком отношении друг к другу они будут находиться. Например, модули доступа к базе данных и обработки графических изображений вряд ли должны знать что-либо о существовании друг друга (рис. 11.1). Но модуль пользовательского интерфейса будет знать о них обоих (а они о нем не будут).

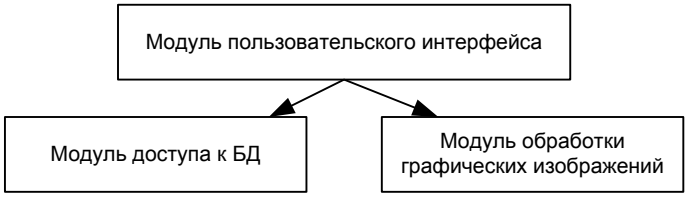


Рис. 11.1. Отношения модулей программы

Отношения между компонентами системы также определяются ее архитектурой.

В целом, все *значимые решения*, направленные на *организацию* программы, а не на проблемы, решаемые с ее помощью, можно отнести к архитектуре.

Уровни абстракции

При проектировании модулей (выбора групп функций и распределения их по файлам в случае процедурного подхода) важно выделять абстракции и пытаться их разложить по нескольким уровням.

Модули низкого уровня максимально автономны, они не зависят от других частей программы. Хорошо спроектированные модули изолируют "внешний мир" от тонкостей решения поставленной перед ними задачи. Вызывающая сторона знает лишь интерфейс модуля (внешние функции), внутренняя часть для нее закрыта.

Рассмотрим в качестве примера галерею фотографий. Информация об изображениях и пользователях хранится в БД, пользовательский интерфейс разделен на клиентскую часть и панель администратора.

Структура программы могла бы быть такой, как на рис. 11.2.

В этом примере прослеживаются три уровня абстракции.

В любой программе, предназначенной для использования человеком, можно выделить, по крайней мере, два уровня: пользовательский интерфейс и модель. Модуль пользовательского интерфейса предназначен для представления и визуализации данных, обработки реакции пользователя. Модель заключает в себе логику решаемой задачи и не должна ни в коей мере быть зависимой от способа отображения

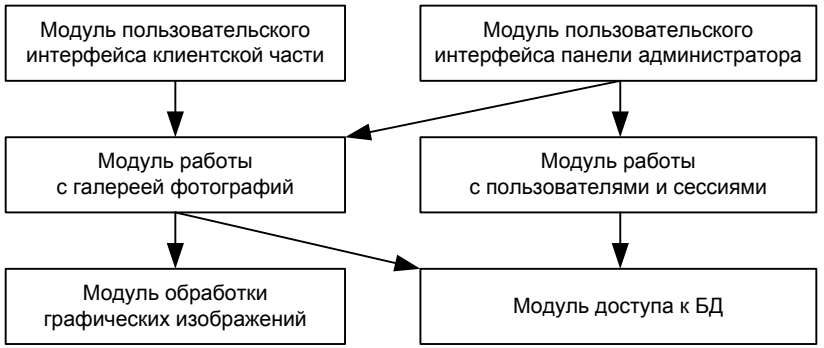


Рис. 11.2. Структура сайта с галереей фотографий

данных. Модель должна быть легко переносима между разными типами пользовательского интерфейса.

Архитектура MVC

Сейчас популярен шаблон проектирования MVC. Он служит для отделения логики приложения от пользовательского интерфейса. Но сначала проясним, что такое шаблон проектирования.

Это набор типовых решений проектирования, каркас архитектуры или ее фрагмента. Если библиотека — это пакет повторно используемого кода, то шаблон проектирования — это пакет повторно используемых решений.

Что же предлагает нам MVC для отделения логики приложения от пользовательского интерфейса?

Шаблон MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

- Модель (Model). Модель предоставляет данные (обычно для Представления), а также реагирует на запросы (обычно от Контроллера), изменяя свое состояние;
- Представление (View). Отвечает за отображение информации (пользовательский интерфейс);
- Контроллер (Controller). Интерпретирует данные, введенные пользователем, и информирует модель и представление о необходимости соответствующей реакции.

На рис. 11.3 показаны отношения между компонентами каркаса. Проиллюстрируем рисунок небольшим примером.

Представьте форму, где можно ввести текст, нажать кнопку **Edit** и получить его транслитерацию (рис. 11.4).

Повторим шаги, изображенные на схеме:

1. Пользователь нажимает кнопку **Edit**, при этом Представление (View) посылает сообщение Контроллеру (Controller): "Команда: edit"

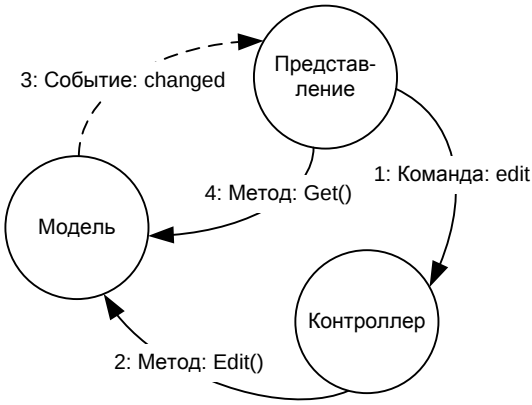


Рис. 11.3. Взаимодействие компонентов системы при архитектуре MVC

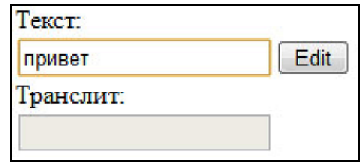


Рис. 11.4. Форма ввода текста

2. Контроллер принимает сообщение и обращается к Модели (Model), вызывая метод `Edit()`.
3. В результате модель меняет свое состояние (запомненный в ней транслитерированный текст) и оповещает об этом представление: "Событие: changed".
4. Представление принимает сигнал и обращается к модели за новым значением результата, вызывая ее метод `Get()`.

Реализация MVC

Реализация MVC предполагает объектно-ориентированный подход (ООП). Однако шаблон проектирования — это всего лишь набор решений. Адаптируем их для PHP без применения ООП. Упрощение делается для того, чтобы сконцентрироваться на сути разделения логики, а также для того, чтобы материал смог применить читатель, не знакомый с ООП.

Рассмотрим снова пример с галереей фотографий.

У нее есть два режима просмотра:

- ❒ режим просмотра уменьшенных изображений (всех сразу);
- ❒ режим просмотра фотографии полного размера (одной).

Также есть возможность загружать фотографии на сервер. Дополнительно реализуем поддержку типов визуализации, чтобы оценить гибкость каркаса.

Вид нашего сайта приведен на рис. 11.5—11.7.

На сайте будут две точки входа:

- ❒ index.php (просмотр галереи);
- ❒ photo.php (просмотр полноразмерной фотографии).

Эти два файла будем считать *Контроллерами*.

В качестве Модели будет выступать модуль, обеспечивающий работу с хранилищем изображений. Назовем его gallery.php и поместим в папку model.

Рис. 11.5. Внешний вид сайта. Табличный вид отображения фотографий

Рис. 11.6. Внешний вид сайта. Отображение фотографий списком

Рис. 11.7. Внешний вид сайта. Страница просмотра фотографии

В роли Представления будут выступать HTML-шаблоны, они будут находиться в папке templates. Что такое шаблоны и для чего они нужны — будет видно дальше.

Страницы просмотра галереи и просмотра фотографии будут иметь общую шапку и подвал страницы, отличаться будет только центральная часть.

Просмотр галереи будет иметь два типа визуализации:

- в виде таблицы (по умолчанию);
- в виде списка.

Нам потребуются четыре шаблона:

- main.php (каркас страницы);
- content_index_table.php (табличный вид содержимого галереи);
- content_index_list.php (списочный вид содержимого галереи);
- content_photo.php (содержимое страницы просмотра фотографии).

Получается следующая структура сайта:

```
model/
  gallery.php
templates/
  main.php
  content_index_table.php
  content_index_list.php
  content_photo.php
index.php
photo.php
```

Файловая структура разделена двумя горизонтальными чертами, образующими три секции. Файлы верхней секции относятся к Модели, файлы средней секции — к Представлению, файлы нижней секции — к Контроллеру.

Модель

Начнем с реализации Модели. В листинге 11.1 код приведен не полностью для минимизации и лучшей наглядности примера.

Листинг 11.1. Код model/gallery.php

```
<?php
// Функция возвращает объект фотографии (ассоциативный массив).
function gallery_item($id)
{
  // Реализация пропущена,
  // тип результата — array
}

// Функция возвращает список фотографий.
function gallery_list()
{
  // Реализация пропущена,
  // тип результата — array
}

// Функция возвращает путь к уменьшенному изображению.
function gallery_icon($photo)
{
  // Реализация пропущена,
  // тип результата — string
}

// Функция возвращает путь к полноразмерному изображению.
function gallery_image($photo)
{
  // Реализация пропущена,
  // тип результата — string
}

// Функция добавляет в галерею новую фотографию.
function gallery_add($file_path, $file_name)
{
  // Реализация пропущена
}
?>
```

Мы определили лишь интерфейс Модели, оставив реализацию пропущенной. Однако для примера реализации каркаса MVC она вовсе и не нужна.

Представление

Теперь рассмотрим шаблоны. Начнем с общего каркаса страницы (листинг 11.2).

Листинг 11.2. Код templates/main.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta content="text/html;charset=windows-1251"
        http-equiv="Content-Type">
  <title><?=$title?></title>
</head>
<body>
  <h1><?=$title?></h1>
  <?php include $content; ?>
</body>
</html>
```

Вас не должно смущать, что в шаблоне используются непонятно откуда взявшиеся переменные `$title` и `$content`. Их подставит Контроллер. Но об этом позже.

`<?=$title?>` — это сокращенный вариант записи `<?php echo $title; ?>`.

Его удобно использовать в шаблонах. Также в шаблонах удобнее использовать альтернативные варианты записи конструкций `if-else`, `foreach`, `for`, `while`. Выглядят они так:

```
if (<условие>):
 <тело>
endif;

foreach (<инициализация цикла>):
 <тело>
endforeach;
```

Остальные шаблоны будут подставляться в main.php таким образом:

```
<?php include $content; ?>
```

В листингах 11.3 и 11.4 приведен их код.

Листинг 11.3. Код templates/content_index_table.php

```
<b>Таблица</b> | <a href="index.php?view=list">Список</a>

<br/><br/>
```

```
<table>
  <tr>
  <?php $i = 0; ?>
  <?php foreach ($photos as $photo): ?>

    <?php if ($i % 3 == 2): ?>
      </tr><tr>
    <?php endif; ?>
    <td>
      <a href="photo.php?id=<?=$photo['id']?>">
        <img src="<?=gallery_icon($photo)?>" />
      </a>
    </td>
    <?php $i++; ?>

  <?php endforeach ?>
  </tr>
</table>

<br/><br/>

<form method="post" enctype="multipart/form-data">
  <input type="file" name="photo" />
  <br/>
  <input type="submit" value="Загрузить файл!" />
</form>
```

Листинг 11.4. Код templates/content_index_list.php

```
<a href="index.php">Таблица</a> | <b>Список</b>

<br/><br/>

<?php foreach ($photos as $photo): ?>

  <a href="photo.php?id=<?=$photo['id']?>">
    <img src="<?=gallery_icon($photo)?>" />
  </a>
  <br/>

<?php endforeach ?>

<br/><br/>

<form method="post" enctype="multipart/form-data">
  <input type="file" name="photo" />
  <br/>
```

```
  <input type="submit" value="Загрузить файл!" />
</form>
templates/content_photo.php:
<a href="index.php">Назад</a>

<br/><br/>

<img src="<?=gallery_image($photo)?>" />
```

Контроллер

И, наконец, соберем все вместе, описав наши два Контроллера. Их задача заключается в обработке запроса, выборе шаблона и подстановке нужных шаблону данных. Данные берутся, как правило, из модели.

Контроллер галереи загружает фотографию, если пользователь отправил файл. Иначе он извлекает из модели список фотографий, выбирает нужный шаблон (в зависимости от желания пользователя) и выводит этот шаблон, передав ему список фотографий (листинг 11.5).

Листинг 11.5. Код index.php

```
<?php
// Подключение библиотек.
require_once('model/gallery.php');

// Загружаем фотографию, если пользователь отправил файл.
if (isset($_FILES['photo']))
{
  gallery_add($_FILES['photo']['tmp_name'], $_FILES['photo']['name']);
  header('Locaton: index.php');
  exit();
}

// Подготовка данных.
$photos = gallery_list();

// Заголовок страницы.
$title = 'Галерея фотографий';

// Выбор шаблона содержимого.
$content = ($_GET['view'] == 'list')
  ? 'templates/content_index_list.php'
  : 'templates/content_index_table.php';

// Вывод HTML.
include 'templates/main.php';
?>
```

Контроллер просмотра фотографии еще проще (листинг 11.6).

Листинг 11.6. Код photo.php

```
<?php
// Подключение библиотек.
require_once('model/gallery.php');

// Подготовка данных.
$photo = gallery_item($_GET['id']);

// Заголовок страницы.
$title = 'Просмотр фотографии';

// Выбор шаблона содержимого.
$content = 'templates/content_photo.php';

// Вывод HTML.
include 'templates/main.php';
?>
```

Резюме

Старайтесь уделять достаточно внимания разработке архитектуры вашего приложения. Если изначально создать прочный и расширяемый каркас, усилия окупятся сторицей.

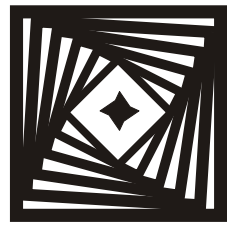
Для реализации модели MVC лучше выбрать объектно-ориентированный подход.

Существует множество готовых решений каркаса, например в Zend Framework. Однако информации, изложенной в текущем уроке, достаточно для того, чтобы понять архитектурные решения MVC и начать их использовать уже сейчас.

Задание

Адаптируйте предыдущую разработку галереи фотографий под архитектуру MVC так, как это было рассказано в текущем уроке.

Используйте листинги этого урока в качестве начального шаблона и реализуйте недостающие части кода. Например, работу с изображениями в БД в модели.



УРОК 12

Введение в объектно-ориентированное программирование

Вот мы и добрались до чрезвычайно важной темы, без которой современное профессиональное программирование просто немыслимо. В этой главе речь пойдет об объектно-ориентированном подходе при написании PHP-сценариев.

Сначала тема объектно-ориентированного программирования (ООП) может показаться вам довольно сложной. Однако не следует пугаться и опускать руки. При тщательном изучении темы и, главное, регулярной практике приемы ООП станут для вас незаменимыми помощниками при разработке собственных PHP-сценариев.

Объектно-ориентированный подход стал негласным стандартом при разработке программ и сценариев на высокоуровневых языках. И мы, как пропагандисты профессионального отношения к программированию, настоятельно рекомендуем освоить эту концепцию, что в итоге позволит не только качественно писать код, но и сэкономит ваше время при разработке.

ООП неспроста получило широкое распространение. Дело в том, что оно обладает рядом неоспоримых преимуществ.

- *Простота.* Действительно объектно-ориентированный код легко воспринимается при чтении. Кроме того, объектно-ориентированный подход позволяет достаточно легко проектировать большие сложные системы.
- *Масштабируемость и расширяемость.* Объектно-ориентированный подход дает возможность легко дописывать и модернизировать программный код.
- *Ускорение цикла разработки.* Хорошо выделенная модульность программ, свойственная объектно-ориентированному подходу, а также легкость в изменении, исправлении написанного кода позволяют значительно сократить общее время на разработку программ.

Что такое объектно-ориентированное программирование

В ООП нам предстоит познакомиться с рядом новых синтаксических конструкций, которые дают новые программные возможности. Но для начала, чтобы уяснить,

в чем заключается суть ООП, нужно познакомиться с двумя базовыми понятиями этого подхода — понятием *класса* и понятием *объекта*. Потратьте достаточно времени на то, чтобы определения класса и объекта были вам *полностью ясны*.

Понятие класса

Класс в ООП — это контейнер для логически связанных данных и функций.

Фактически класс — это тип данных. Как вы помните, в начале книги мы обсуждали типы данных в PHP и делили их на простые и сложные. Одним из сложных типов данных были массивы. Массивы можно представить как некоторую совокупность элементов различных (или одинаковых) типов.

Класс также является совокупностью элементов. Однако у него есть ряд отличий от массива. Во-первых, объявление класса, с которым мы познакомимся чуть позже, четко декларирует его состав. Иными словами, объявление класса определяет, какие элементы он включает в себя. Элементы класса обычно называют *полями* или *свойствами*. Во-вторых, класс может определять функции для работы с его полями.

Еще раз повторим, что класс — это тип данных, такой же, как строка или целое число. Он не имеет конкретной реализации, а лишь описывает структуру хранимой информации и функции для ее обработки.

Читателям, знакомым с такими языками как Си или Паскаль, должно быть известно понятие *структуры* как набора элементов простых типов данных. Класс, по сути, является структурой, в которой дополнительно можно объявлять функции для работы с хранимыми ею данными.

Можно придумать огромное количество примеров различных классов из обычной жизни. Допустим, если мы пишем сценарий, который должен работать с информацией о геометрических фигурах, то в качестве класса можно выделить фигуру *"круг"*. Круг в программе может иметь ряд параметров: координаты центра, длину радиуса, цвет. Эти свойства в программе будут представлены элементами класса. Для определения координат центра нам потребуются две целочисленные переменные, длина радиуса может быть описана также целочисленной переменной, цвет может быть задан с помощью строки с его названием. Таким образом, для определения круга в программе нам понадобятся три целочисленные переменные и одна строковая. Они и будут свойствами класса, описывающего круг.

Простейшее объявление класса выглядит следующим образом:

```
class Article
{
  // тело класса
}
```

Объявление класса начинается с ключевого слова `class`. Далее следует название класса, которое принято писать с большой буквы. За ним идет блок в фигурных

скобках, который может включать объявление полей и функций класса. Давайте рассмотрим класс с тремя полями.

```
class Article
{
  // ниже представлены три свойства класса: id, title, content
  $id;
  $title;
  $content;
}
```

Как видите, объявление полей класса аналогично объявлению простых переменных в PHP-сценарии.

Понятие объекта

Объект — совокупность конкретных данных и функций для их обработки. Фактически объект — это переменная, тип данных которой задается соответствующим классом. Переменная и простой тип данных находятся в том же отношении, что и объект с классом. Объект — это конкретная реализация какого-либо класса.

В нашем примере с кругом объектом будет являться какой-то конкретный круг с определенными координатами центра, радиусом и цветом.

Для объявления объекта используется ключевое слово `new`.

```
$a = new Article();
```

Приведенная строчка является командой PHP-интерпретатору для создания объекта класса `Article`. Это значит, что PHP-интерпретатор подготовит структуру для хранения всех полей, описанных в классе `Article`.

Обращение к полям объекта происходит посредством оператора `->`. Пример:

```
$a = new Article();
echo $a->id;
```

Этот код выведет на экран содержимое поля `id` объекта `$a`. Еще раз обратим ваше внимание: когда мы выводим, записываем, изменяем данные, то говорим об объектах, т. е. о конкретных переменных сложного типа. Класс не может что-либо хранить, он лишь *описывает структуру данных и функции* для работы с ними.

Обратите внимание на рис. 12.1. Пример класса представлен на нем верхним прямоугольником. Два нижних прямоугольника являются объектами, конкретными реализациями класса.

Приводя аналогии из реальной жизни, можно сказать, что класс — это некоторое понятие. Например, понятие "стол". Не какой-то конкретный стол, а само понятие стола как деревянного предмета с четырьмя ножками.

Объекты же — это конкретные воплощения понятия. Стол, который находится сейчас перед вами — это объект. Объект класса "стол".

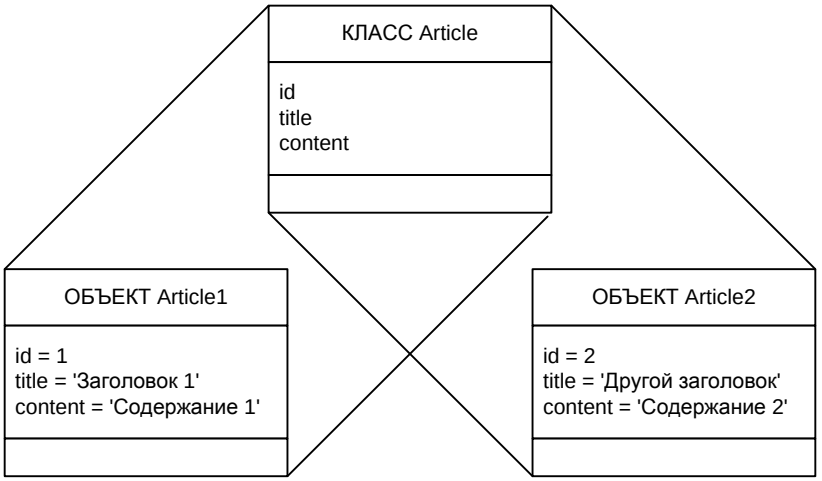


Рис. 12.1. Классы и объекты

Методы

Как уже было сказано, помимо свойств классы могут описывать функции для работы с ними. Такие функции принято называть *методами* класса.

Метод — функция, объявленная внутри класса. Метод имеет доступ ко всем полям класса, в котором объявлен. Для обращения к собственным полям внутри методов класса используется ключевое слово `this`. Пример приведен в листинге 12.1.

Листинг 12.1. Пример реализации класса

```
class Article
{
  // Объявление полей класса
  $id;
  $title;
  $content;

  // Функция для вывода статьи
  function view()
  {
    echo "<h1>$this->title</h1><p>$this->content</p>";
  }
}
```

Поздравляем! Только что вы познакомились с двумя ключевыми понятиями ООП — понятиями класса и объекта. Если принципы использования объектов и классов вам ясны, то можно сказать, что вы на 90% владеете ООП. Действительно, ООП больше не содержит каких-либо еще принципиальных понятий. Все осталь-

ные принципы объектно-ориентированного подхода основаны на использовании этих двух ключевых понятий.

Тем не менее за кажущейся простотой и ограниченностью инструментов ООП скрывается невероятная мощь этой методологии, которую вам еще предстоит ощутить. Давайте ознакомимся с тремя очень важными понятиями, которые несет в себе объектно-ориентированный подход.

Инкапсуляция

Инкапсуляция — свойство языка программирования, позволяющее объединить и защитить данные и код в объект и скрыть реализацию объекта от пользователя (программиста). При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Иными словами при работе с грамотно спроектированным классом мы можем пользоваться только его методами, не вникая в то, как они устроены и как они работают с полями класса. Речь идет о ситуации, когда мы работаем с классом, разработанным другими программистами. Мы же просто пользуемся уже реализованным функционалом.

Приведем другой пример. Принципы инкапсуляции как скрытия внутренней реализации заложены в любой функции PHP. Возможно, вам уже известна функция `strpos()` для работы со строками из стандартной библиотеки PHP. Эта функция ищет в строке заданную последовательность символов и возвращает ее позицию в виде числа. Если задуматься над реализацией этой функции, то можно предположить, что нам потребуется в цикле просматривать каждый символ от начала строки на совпадение с начальным символом искомой последовательности и в случае такового сравнивать следующие два символа и т. д. Но нам как программистам нет необходимости задумываться над этим и вникать в тонкости реализации данной функции. Нам достаточно знать параметры, которые она принимает, и формат возвращаемого значения. Функция `strpos()` *инкапсулирует* в себе решение задачи поиска подстроки, предлагая нам лишь внешний интерфейс для ее использования.

Аналогичным образом правильно спроектированные классы скрывают свою внутреннюю реализацию, предоставляя внешним пользователям интерфейс в виде набора методов.

В языке PHP концепция инкапсуляции реализована в виде специальных *модификаторов доступа* к полям и методам классов. Об этом мы поговорим далее.

Наследование

Наследование — механизм объектно-ориентированного программирования, позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Давайте приведем пример наследования из реальной жизни. В качестве класса можно взять геометрическую фигуру. При этом мы не уточняем, какой конкретно фигура должна быть. Какие свойства фигуры можно выделить? Предположим, она обладает цветом. Тогда в классе, описывающем фигуру, должно быть соответствующее поле строкового типа, задающее цвет фигуры. Также любая геометрическая фигура обладает площадью. Пусть площадь будет вторым свойством нашего класса.

Теперь предположим, что нашей программе требуется работать с конкретными геометрическими фигурами: квадратами и треугольниками, в том числе с их геометрическим положением на плоскости. Очевидно, что описание треугольников и квадратов с помощью класса `Фигура` будет недостаточным, потому что она не хранит информацию о геометрическом положении. Поэтому нам потребуется ввести еще два класса: `Квадрат` и `Треугольник`. При этом допустим, что в нашей программе нам также потребуются цвета и площади фигур. Эта ситуация как раз и требует использования наследования. Потому что любой квадрат и треугольник в программе заведомо является фигурой, т. е. имеет цвет и площадь. В то же время каждая фигура требует дополнительных данных (помимо цвета и площади) для своего описания, что решается вводом двух дополнительных классов для квадратов и треугольников, которые *наследуются* от класса `Фигура`.

Это значит, что в классах `Квадрат` и `Треугольник` нам не придется повторно задавать поля цвета и площади. Достаточно указать, что упомянутые классы наследуются от класса `Фигура`.

Теперь давайте рассмотрим еще один пример, более приближенный к реалиям веб-программистов. Сейчас в Интернете огромную популярность завоевали различные блоги. Фактически блог — это просто набор статей. И ключевой сущностью при разработке блога является именно статья. В этой главе мы уже приводили пример простейшего класса статьи, состоящего из трех полей и функции вывода текста статьи (см. листинг 12.1).

А теперь давайте представим, что в нашем будущем блоге статьи могут иметь различные типы. Для начала остановимся на двух типах: обычная статья-заметка и новостная статья. Для новостной статьи важна дата ее публикации, ведь она несет в себе некоторую конкретную новость.

Чтобы реализовать эту ситуацию в PHP, нам потребуется определить два класса: класс для обычной статьи-заметки и класс для новостной статьи. При этом пусть новостная статья расширяет возможности обычной статьи, т. е. *наследуется* от нее. Для наследования классов в PHP используется ключевое слово `extends`.

```
class Article
{
  ... // содержимое класса
}
class NewsArticle extends Article
{
  ... // содержимое класса
}
```

Приведенный код определяет класс `NewsArticle` как *наследника* `Article`. Класс `Article` в свою очередь является *родительским* для класса `NewsArticle`. Это значит, что поля и методы класса `Article` будут также присутствовать в классе `NewsArticle`, и заново их определять не нужно.

С помощью наследования можно выстраивать целую иерархию классов, наследуя один от другого. В то же время у любого класса может быть только один родитель (рис. 12.2).

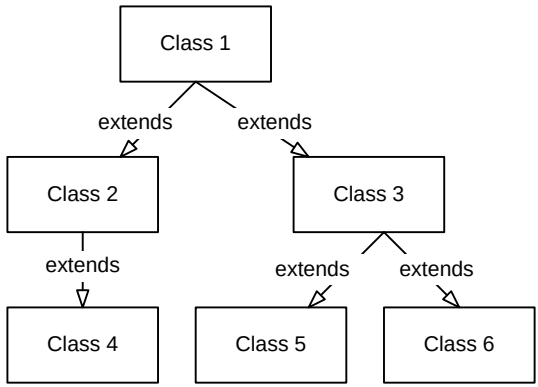


Рис. 12.2. Иерархия классов

Иногда у нас может появиться необходимость *переопределить* один из методов родительского класса. Давайте еще раз приведем реализацию класса `Article`:

```
class Article
{
  ... // поля класса

  // Функция для вывода статьи
  function view()
  {
    echo "<h1>$this->title</h1><p>$this->content</p>";
  }
}
```

Предположим, что вывод новостной статьи должен отличаться от представления обычной статьи, и мы должны дополнительно выводить время публикации новости. При этом в классе `Article` уже существует метод `view()`, отвечающий за вывод статьи. Можно поступить двумя способами. В первом случае можно придумать новый метод в классе `NewsArticle`, например, с именем `viewNews()` специально для вывода новости. Однако правильнее использовать одинаковые методы для выполнения схожих действий в наследуемых классах. Поэтому будет лучше, если метод для вывода новости в классе `NewsArticle` будет называться так же, как и в родительском классе — `view()`. Для реализации такой логики в PHP существует возможность *переопределять* родительские методы, т. е. задавать в дочерних классах методы

с названиями, совпадающими в родительских классах. Реализация этих методов в родительских классах в таком случае становится неактуальной для класса-потомка. Давайте приведем пример класса `NewsArticle` с переопределенным методом `view()`:

```
class NewsArticle extends Article
{
  $datetime; // дата публикации новости

  // Функция для вывода статьи
  function view()
  {
    echo "<h1>$this->title</h1><span style='color: red'>".
         strftime('%d.%m.%y', $this->datetime).
         " <b>Новость</b></span><p>$this->content</p>";
  }
}
```

В приведенном коде используется функция `strftime()`, которая позволяет выводить даты в удобном виде. Для лучшего понимания кода ознакомьтесь со спецификацией этой функции в справочнике. Для нас же сейчас важно, чтобы вы обратили внимание на то, что класс `NewsArticle`, как и `Article`, определяет метод `view()`. Соответственно, все объекты этого класса будут использовать метод `view()`, объявленный в классе `NewsArticle`, а не в `Article`.

У вас может возникнуть вопрос: почему же все-таки важно переопределять некоторые методы вместо того, чтобы вводить новые методы у классов-потомков? Понимание этого придет вместе с пониманием следующей важнейшей концепции ООП.

Полиморфизм

Полиморфизм — взаимозаменяемость объектов с одинаковым интерфейсом.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования. Это именно то, что мы видели в предыдущем примере со статьями.

Давайте рассмотрим следующий пример, который дает представление о сути полиморфизма (листинг 12.2).

Листинг 12.2. Пример полиморфизма

```
class A
{
  function Test() { echo 'Это класс A<br />'; }
  function Call() { $this->Test(); }
}
```

```
class B extends A
{
  function Test() { echo 'Это класс B<br />'; }
}

$a = new A();
$b = new B();
$a->Call(); // выводит: "Это класс A"
$b->Test(); // выводит: "Это класс B"
$b->Call(); // выводит: "Это класс B"
```

Обратите внимание на комментарии к трем последним строчкам. Попытайтесь самостоятельно объяснить такой результат. Желательно собственноручно реализовать и протестировать этот пример. Потратьте время на то, чтобы работа сценария стала вам полностью ясна, т. к. в этом небольшом примере заложен глубокий смысл ООП. Теперь давайте попробуем вместе разобрать предложенный код.

```
$a->Call(); // выводит: "Это класс A"
```

В этой строке происходит вызов метода `Call()` у объекта класса `A`. Как и определено в функции `Call()` класса `A`, происходит вызов метода `Test()`. Отрабатывает метод `Test()` у объекта класса `A`, и на экран выводится текст "Это класс `A`".

```
$b->Test(); // выводит: "Это класс B"
```

В данной строке происходит вызов метода `Test()` у объекта класса `B`. Метод `Test()` класса `B` выводит на экран текст "Это класс `B`".

```
$b->Call(); // выводит: "Это класс B"
```

Наконец, в последней строке происходит вызов класса `Call()` у объекта класса `B`. Но в реализации класса `B` мы не увидим такого метода, а это значит, что он наследуется от класса `A`, т. к. класс `B` — это потомок класса `A`. Что же мы видим в реализации метода `Call()` класса `A`? Следующий код:

```
$this->Test();
```

Метод `Call()` вызывает метод `Test` *того объекта, в котором находится*. Это значит, что отработает метод `Test()` объекта класса `B`. Именно этим объясняется результат, выведенный на экране.

В этом и заключается принцип полиморфизма. Классы могут иметь одинаковые методы, но разную их реализацию. Разрабатывая код сценария, мы можем знать лишь общую для группы классов спецификацию их методов, но не иметь представления, экземпляр какого именно класса будет использоваться в конкретный момент времени.

Понятия инкапсуляции, наследования и полиморфизма можно назвать тремя китами ООП. Понимание и грамотное применение принципов этих концепций — залог успеха разработки с применением ООП.

Спецификаторы (модификаторы) доступа

Ранее мы уже говорили о том, что классы *инкапсулируют* логику своей работы, предоставляя вовне свой интерфейс в виде методов и, возможно, отдельных свойств. Эта возможность реализована в PHP с помощью так называемых спецификаторов доступа к полям и методам класса. *Спецификаторы* — это ключевые слова, которые записываются перед полем и методом класса и определяют его доступность. В PHP существуют три спецификатора, или по-другому — *модификатора*, доступа.

Модификатор `public` позволяет обращаться к свойствам и методам отовсюду.

Модификатор `private` позволяет обращаться к свойствам и методам только внутри текущего класса.

Модификатор `protected` позволяет обращаться к свойствам и методам только текущего класса и класса, который наследует свойства и методы текущего класса.

Продемонстрируем работу модификаторов на примере (листинг 12.3).

Листинг 12.3. Пример работы модификаторов доступа

```
class Article
{
  private $id;
  protected $content;
  public $title;

  // Функция вывода статьи имеет доступ ко всем полям класса
  function view()
  {
    echo "<h1>$this->id – $this->title</h1>".
         "<p>$this->content</p>";
  }
}

echo $a->id;       // Ошибка: модификатор private
                   // не позволяет получить доступ

echo $a->content;  // Ошибка: модификатор protected
                   // не позволяет получить доступ

echo $a->title;   // Ошибки нет: модификатор public
                  // позволяет получить доступ

class NewsArticle extends Article
{
  ...
```

```
  // Функция для вывода статьи имеет доступ
  // к полям content и title, но не id!
  function view()
  {
    echo "<h1>$this->title</h1><p>$this->content</p>";
    echo "$this->id";  // Ошибка: нет доступа к полю id
                       // с модификатором private
  }
}
```

Модификаторы доступа организуют некое искусственное ограничение возможностей классов, которое служит для того, чтобы согласовать их функционал с изначально продуманной логикой их работы.

Поля и методы классов, объявленные без использования модификаторов, считаются публичными, т. е. как если бы они были объявлены с модификатором `public`.

Конструкторы

Практикуясь в применении ООП, вы быстро заметите, что в большинстве случаев вам будет необходима функция инициализации объекта некоторыми начальными значениями. Это вполне очевидно. Когда вы создаете объект некоторого класса, вам, скорее всего, требуется заполнить его поля определенными значениями.

Конечно, можно каждый раз предусматривать в классах некоторую функцию инициализации. Однако в PHP для этой цели предусмотрена специальная функция *конструктора*, которая вызывается всякий раз при создании объекта. Программист может определить эту функцию, ее входные параметры и реализацию, и тогда она будет вызываться всякий раз, когда будет создаваться новый объект. То есть выполняться оператор `new`.

Для объявления конструктора используется ключевое слово `__construct`. Обратите внимание на два подчеркивания, с которых начинается название функции. Двойное подчеркивание — это признак специально зарезервированного названия. В PHP у классов существует еще ряд функций, начинающих с двойного подчеркивания и служащих для решения определенных системных задач. Однако на данном этапе нас интересует только функция `__construct()`. Давайте посмотрим пример ее определения.

```
class Article
{
  // поля класса
  private $id;
  private $title;
  private $content;

  function __construct($id, $title, $content)
  {
    $this->id = $id;
```

```
    $this->title = $title;
    $this->content = $content;
  }
}
```

В данном примере конструктор используется для того, чтобы задать значения для полей класса. Обратите внимание, что все поля класса объявлены как `private`. То есть непосредственно изменить их значение извне невозможно. Их можно задать только через конструктор.

Теперь функция конструктора будет вызываться всякий раз, как мы будем создавать экземпляр класса `Article`. Например, вот так:

```
$first_article = new Article(1, 'Первая!', 'Это моя первая статья');
```

Статические члены классов

Определения классов могут включать статические члены (как свойства, так и методы), доступ к которым осуществляется через класс.

Когда мы имеем дело с объектами, то говорим об *отдельных экземплярах* какого-либо класса. У каждого объекта свои значения для полей класса. Статические же члены классов всегда существуют в единственном экземпляре. Они как бы общие для всех объектов данного класса. Иными словами, нельзя сказать, что статический член принадлежит какому-либо объекту. Статические члены принадлежат именно классу.

Если вернуться к примеру с геометрическими фигурами, то статическим членом класса `Фигура` может стать какая-либо геометрическая константа, например число π. Оно может использоваться в различных вычислениях внутри методов классов, описывающих геометрические фигуры, но его значение едино для всех объектов.

Для объявления статических членов класса используется ключевое слово `static`.

```
static public $instance = NULL;
```

Так можно объявить статическое поле класса и инициировать его нулевым значением.

Обращение к статическим свойствам внутри методов класса происходит с помощью ключевого слова `self`, после которого следует название статического поля с лидирующим символом `$`. Пример:

```
class Singleton
{
  static private $instance = NULL;

  static public function getInstance()
  {
    if (self::$instance == NULL)
    {
      self::$instance = new Singleton();
    }
```

```
    return self::$instance;
  }
}
```

Обратите внимание, что в приведенном примере функция `getInstance()` также является статической. Это значит, что она может работать только со статическими элементами класса, т. к. она не принадлежит какой-либо конкретной реализации объекта.

Для вызова статических методов класса используется следующий синтаксис:

```
<имя_класса>::<имя_метода>(<передаваемые_параметры>)
```

Пример:

```
$instance = Singleton::getInstance();
```

Статические члены используются, когда требуется выделить данные, которые должны присутствовать в программе в единственном экземпляре. Например, это может быть дескриптор соединения с базой данных. В большинстве своих сценариев мы работаем с одной базой данной, с которой устанавливаем соединение в начале работы сценария и закрываем соединение в момент завершения сценария. Несмотря на то, что в ходе работы сценария мы можем выполнять большое количество запросов к базе данных, дескриптор соединения для этих целей требуется всего один. Таким образом, его можно хранить в статическом члене класса, описывающего работу с базой данных.

Абстрактные классы и методы

В PHP 5 существует понятие *абстрактных классов* и *методов*. Вернемся к примеру с геометрическими фигурами. Как вы помните, мы ввели класс `Фигура`, который хранит данные о цвете и площади.

Полей класса `Фигура` недостаточно для описания какой-либо реальной геометрической фигуры. Тем не менее его объявление оправдано, т. к. он позволяет выделить общие для любой фигуры характеристики (цвет, площадь) и методы для работы с ними. В данном случае `Фигура` является типичным примером абстрактного класса. Его реализация в программе не предусмотрена, он служит лишь для того, чтобы быть родителем классам, которые будут наследоваться от него. PHP позволяет объявлять классы абстрактными с помощью ключевого слова `abstract`.

```
abstract class Article
{
  ...
}
```

Создавать экземпляр класса, который был объявлен абстрактным, нельзя.

Помимо классов абстрактными могут быть и методы, которые также определяются с помощью слова `abstract`. Абстрактные методы не имеют реализации. Они нужны для того, чтобы дочерние классы их переопределили. PHP-интерпретатор выдаст

ошибку, если в неабстрактном классе-наследнике не будет переопределен хотя бы один абстрактный метод его родителя. Обычно абстрактные классы создаются для того, чтобы наследующие их дочерние классы переопределили недостающие методы. Класс, в котором объявлен хотя бы один абстрактный метод, должен также быть объявлен абстрактным. Проиллюстрируем теорию примером.

```
abstract class Article
{
  ...
  abstract function intro();

  function view()
  {
    $this->intro();
    echo "<h1>$this->id — $this->title</h1>".
         "<p>$this->content</p>";
  }
}

class NewsArticle extends Article
{
  ...
  function intro()
  {
    echo '<span style="color: red">'.
         strftime('%d.%m.%y', $this->datetime).
         ' <b>Новость</b></span>';
  }
}
```

В приведенном примере класс `Article` является абстрактным, а значит, в программе не может быть объекта этого класса. `Article` содержит абстрактный метод `intro()`, который должен быть переопределен в потомках. Класс `NewsArticle` является наследником класса `Article` и переопределяет метод `intro()`. Поэтому в программе мы можем создавать объекты *неабстрактного* класса `NewsArticle`.

Резюме

На этом мы предлагаем на время приостановить свое знакомство с ООП и попрактиковаться в отработке пройденной теории. Не волнуйтесь, если на данный момент вы чувствуете некоторый сумбур в голове. Действительно, парадигма ООП вовсе не элементарна, и чтобы научиться применять в своих сценариях новые принципы, требуется время и много практики. Не устаем повторять вам, что только через регулярные практические занятия можно научиться хорошо программировать.

В этой главе мы сознательно сократили объем излагаемой информации об ООП и объяснили только наиболее важные концепции. На самом деле, реализация ООП

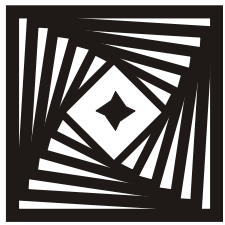
в PHP несколько шире. Она имеет ряд тонкостей, которые мы в данной книге решили опустить, чтобы не перегружать информацией читателей. Предложенного материала и так хватит на несколько недель, а то и месяцев, освоения и отработки.

Остается повторить лишь одно: ООП — это стандарт современного профессионального программирования в первую очередь потому, что предоставляет новый уровень эффективности разработки. Без освоения этой парадигмы вы не можете считать себя полноценными программистами.

Задание

Переработайте галерею фотографий, созданную при выполнении задания предыдущего урока, с учетом концепций ООП. Выделите базовые сущности, которыми вы оперируете на своем сайте. Оформите их в виде классов. Продумайте иерархию классов и их взаимодействие.

Не беспокойтесь, что такая переработка приведет к изменению и переписыванию практически всего кода, такое порой случается в профессии программиста. Помните, что правильное применение концепции ООП — это как раз то, что позволит строить хорошо масштабируемые и расширяемые системы без изменения их базовой функциональности.



УРОК 13

Совместное использование принципов MVC и ООП

В двух предыдущих главах мы познакомились с парой ключевых понятий в современном веб-программировании. Первое из них относится к архитектуре сайта, это шаблон проектирования MVC. Второй важнейшей концепцией является объектно-ориентированное программирование. Совместное использование этих парадигм является ключом к профессиональной веб-разработке.

В последней главе нашей книги не будет принципиально новых сведений. Мы посвящаем ее отработке навыков использования изученных инструментов программирования, т. к. для того чтобы быть полноценным специалистом в области программирования, недостаточно знать теорию и уметь писать код с применением современных средств разработки, таких как ООП. Нужно делать это уместно и грамотно. Иначе никакой пользы от применения этих инструментов не будет.

Мы уже проделали большой путь с начала книги и теперь готовы практиковаться на примере, максимально приближенном к реальности. Сегодня мы разработаем систему для редактирования и публикации контента веб-страницы.

Подчеркнем: если вы добросовестно изучили все предыдущие главы книги, то технически для вас не составит труда реализовать подобный пример. Однако важно сделать это грамотно, так, чтобы написанные PHP-сценарии были образцом хорошего стиля программирования. Это значит, что код должен быть понятным, т. е. легко читаться, функционал сайта при необходимости должен легко расширяться, не нарушая общую архитектуру системы, а разработанные модули системы должны быть спроектированы так, чтобы их можно было использовать в других проектах.

В книге мы дадим общую канву системы. Реализация конкретных модулей остается за вами. Считайте это собственной дипломной работой. Отнеситесь к ней со всей ответственностью. Ну а те, кто успешно справятся с этой задачей, могут рассчитывать на специальный подарок от нас, авторов этой книги. Подробнее об этом поговорим в конце данной главы. Итак, приступим к описанию будущей системы.

Описание задачи

Представьте сайт, состоящий из двух страниц. На главной старице отображается некоторый текст (рис. 13.1). Вторая страница представляет собой его редактор

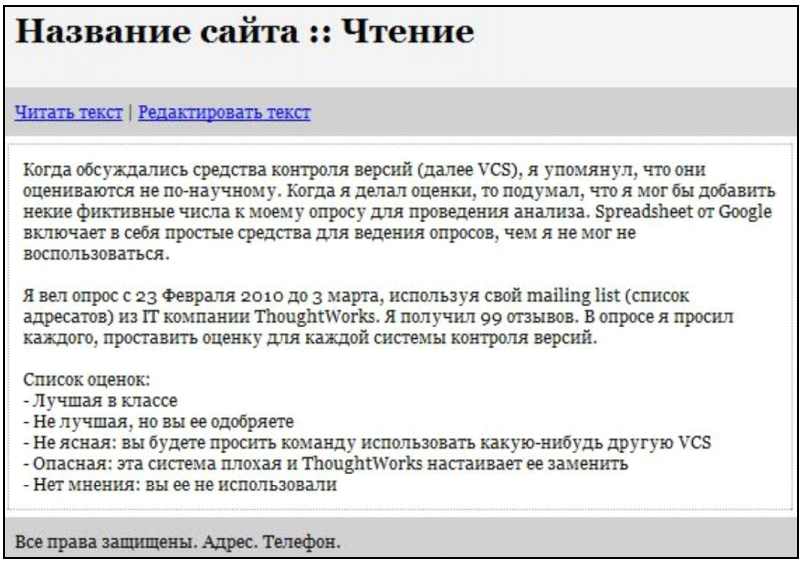


Рис 13.1. Главная страница сайта

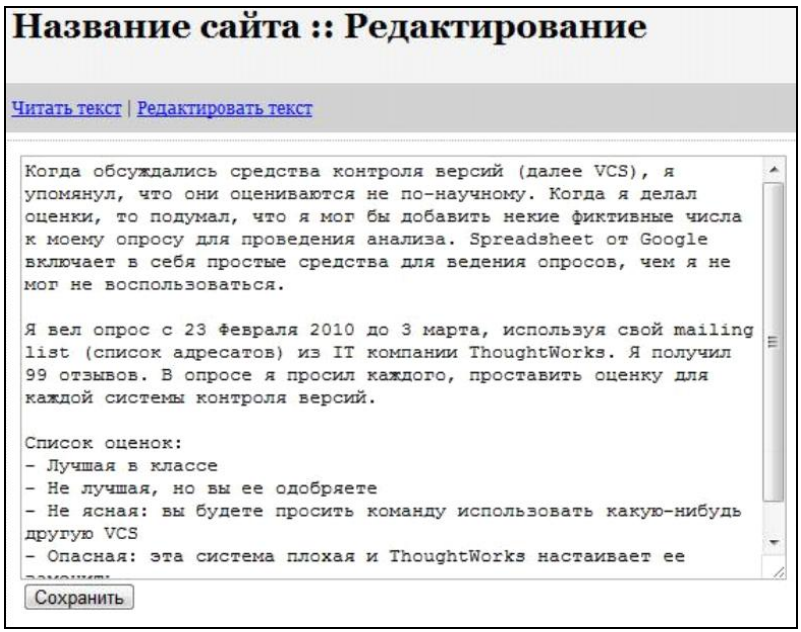


Рис. 13.2. Страница редактирования контента

(рис. 13.2). На ней мы можем изменять содержание главной страницы и сохранять изменения.

В верхней части страниц расположена "шапка" сайта, которая состоит из двух элементов. Первый из них неизменен и отражает название сайта. Второй содержит название текущей страницы: либо "Чтение", либо "Редактирование".

Ниже шапки расположено меню, которое одинаково для всех страниц сайта и содержит ссылки на эти страницы.

Также неизменным остается "подвал" страницы, в котором отображается дополнительная информация о сайте, например, адрес, телефон, информация о создателе страницы.

Центральную часть станицы "Чтение" занимает текст, который мы можем задавать на странице "Редактирование", используя обычную HTML-форму с элементом `textarea`, о котором шла речь в *главе 3*.

Исходные положения

Нам предстоит объединить два подхода — MVC и ООП. Для этого при разработке архитектуры системы будем руководствоваться следующими правилами.

- Согласно шаблону проектирования MVC файлы контроллеров, моделей и представлений будут разнесены по трем отдельным каталогам.
- Каждый контроллер будет представлен отдельным файлом, в котором будет объявлен класс контроллера, инкапсулирующий логику работы контроллера и предоставляющий методы для обработки HTTP-запроса пользователя.
- Каждой странице сайта будет соответствовать отдельный контроллер, отвечающий за ее вывод.
- Модели системы также будут реализованы в виде классов.
- Представления будут реализованы в виде обычных PHP-файлов с HTML-разметкой с вкраплениями PHP-кода для отображения значения переменных сценария.
- HTTP-запросы к системе будут осуществляться через специальную точку входа — файл index.php.

Хоть эта тема уже и обсуждалась в *главе 11*, повторим, что следует четко представлять различия между назначением классов контроллеров и классов модели. Классы контроллеров служат для того, чтобы обеспечить обработку HTTP-запроса пользователя, т. е. получить параметры входного запроса, и подготовить результирующую HTML-страницу. При выполнении этой работы контроллеры задействуют классы модели, которые представляют отдельные функциональные блоки: работу с базой данных, работу с файлами, работу с аккаунтами пользователей и т. п. Каждый такой логически связный блок представляется отдельным классом. Например, в вашей системе может быть класс для работы с базой данных, который будет хранить дескриптор текущего соединения с БД и предоставлять набор методов для выполнения запросов к БД и получения различных наборов данных.

Точка входа

Точкой входа называют сценарии, которые пользователь может вызывать непосредственно из адресной строки браузера. Практически все примеры, написанные вами ранее, содержали несколько точек входа. Однако на данный момент хорошим то-

ном веб-программирования считается использование одной точки входа на сайт. Вся логика перехода между страницами реализуется с помощью дополнительных параметров, передаваемых сценарию, а также за счет сессий и cookies. В единственной точке входа программист может определять действия, характерные для каждой страницы сайта. Это может быть аутентификация пользователя, установление соединения с базой данных, инициализация глобальных переменных и прочие задачи.

В нашем примере мы также будем пользоваться единой точкой входа — файлом index.php. Давайте рассмотрим его содержание (листинг 13.1).

Листинг 13.1. Единая точка входа на сайт. Файл index.php

```
<?php
  include_once('inc/C_View.php');
  include_once('inc/C_Edit.php');

  // Выбор контроллера.
  switch ($_GET['c'])
  {
    case 'edit':
      $controller = new C_Edit();
      break;
    default:
      $controller = new C_View();
  }

  // Передача управления нужному контроллеру.
  $controller->Request();
?>
```

Давайте разберем работу данного сценария построчно.

```
include_once('inc/C_View.php');
include_once('inc/C_Edit.php');
```

Первые две строчки подключают файлы контроллеров для страниц "Чтение" и "Редактирование".

```
switch ($_GET['c'])
```

Здесь происходит анализ GET-параметра с именем `c`.

```
case 'edit':
  $controller = new C_Edit();
  break;
```

Если данный параметр хранит строку `"edit"`, тогда переменной `$controller` присваивается объект класса `C_Edit`, описание которого находится в файле inc/C_Edit.php, подключенном в самом начале сценария.

В случае если параметр c не задан или имеет значение, отличное от "edit", выполняется другая ветка оператора switch:

```
default:
  $controller = new C_View();
```

Здесь переменной $controller присваивается экземпляр класса C_View.

Наконец, последняя строка вызывает метод Request() у созданного объекта:

```
$controller->Request();
```

Обратите внимание, здесь мы можем увидеть типичный пример полиморфизма. Переменная $controller может хранить как экземпляр класса C_Edit, так и экземпляр класса C_View. Поэтому мы не можем заранее знать, метод Request() какого именно класса будет вызван в сценарии. Важно лишь то, что для корректной работы сценария оба класса должны иметь данный метод.

В целом же задача точки входа проста — передать управление одному из контроллеров. Какому именно, определяется с помощью параметра GET-запроса с именем c.

Иерархия контроллеров

Давайте приступим к проектированию системы и для начала определим, какие контроллеры будут в ней присутствовать. Из файла index.php видно, что в системе есть как минимум два контроллера, представленные в виде классов C_Edit и C_View. Задача класса C_Edit — организовать вывод страницы "Редактирование", класса C_View — страницы "Чтение". Обработка входных параметров и вывод страницы инкапсулируется внутри метода Request, который должен присутствовать как в классе C_View, так и в C_Edit. Это может натолкнуть на мысль о создании общего родительского класса для этих двух контроллеров. И действительно, реализация C_View будет во многом совпадать с C_Edit, поэтому выделение родительского класса базового контроллера — это полезное действие.

Таким образом, мы пришли к мысли о необходимости создания базового класса контроллера. Назовем его C_Base. Он будет абстрактным и не будет иметь конкретных реализаций. Мы вводим его для того, чтобы другие классы контроллеров наследовались от него. Класс C_Base будет хранить базовые свойства и методы, характерные для каждого контроллера данного сайта.

Давайте поразмышляем, какие параметры являются общими для каждой страницы сайта? Во-первых, это название сайта. Во-вторых, текст, расположенный внизу каждой страницы. Сюда же можно отнести пункты меню, которые одинаковы для каждой страницы сайта. Обобщая, можно сказать, что контроллер C_Base отвечает за базовый шаблон сайта.

Очевидно, каждый контроллер будет содержать метод Request, отвечающий за обработку запроса. Следовательно, его объявление также можно вынести в класс C_Base и сделать его абстрактным для того, чтобы классы-наследники были обязаны реализовать данный метод.

Возможно, имеет смысл выделить еще какие-то общие свойства или методы для всех контроллеров сайта. Подумайте над этим самостоятельно. Подчеркнем, `C_Base` — это базовый класс, который объединяет в себе все общие элементы каждого контроллера *конкретного* сайта.

Но давайте попробуем пойти еще дальше и подумаем, можно ли выделить класс контроллера, который инкапсулировал бы логику, неизменную для любого контроллера *любого* сайта? В такой класс мы уже не сможем вынести свойство названия сайта, но, к примеру, метод `Request()`, отвечающий за обработку запроса и формирование результирующей страницы, неизменно должен присутствовать в любом контроллере.

Не будем томить читателя и продемонстрируем иерархию классов контроллеров, которую мы вам предлагаем принять за образец (рис. 13.3).

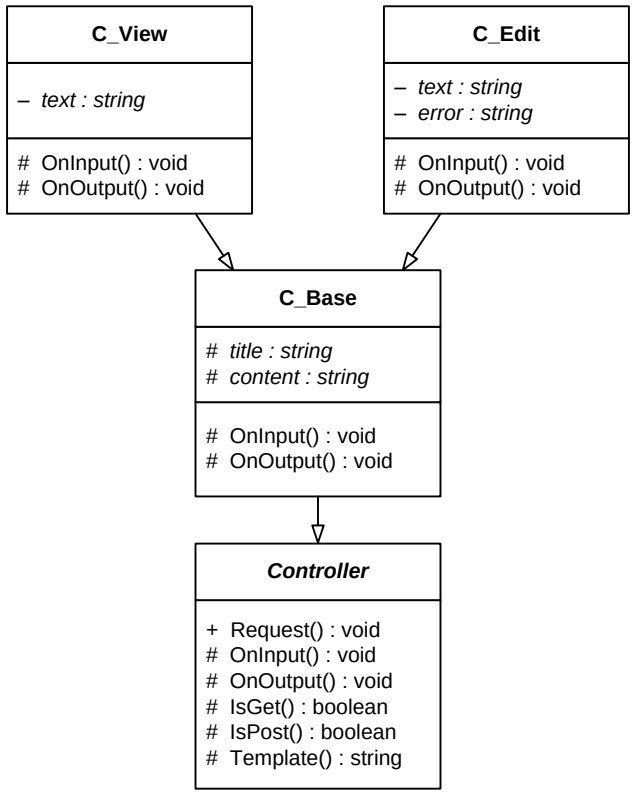


Рис. 13.3. Иерархия классов контроллеров

Данная диаграмма используется в описательном языке UML и называется *диаграммой классов*. Каждый прямоугольник на ней соответствует отдельному классу. В верхней части прямоугольника жирным выделено название класса. Стрелками показаны отношения наследования. Курсивом выделены свойства классов, прямым шрифтом — его методы. Знак – перед свойствами или методами говорит о том, что

свойство или метод объявлены с модификатором `private`, знак `#` соответствует модификатору `protected`, `+` — модификатору `public`.

Класс *Controller*

`Controller` является базовым классом контроллера, который призван инкапсулировать самые общие элементы, характерные для контроллеров любых сайтов. Сюда мы вынесли ряд вспомогательных функций:

- `Template()` — для подстановки в шаблон набора переменных и вывода его на экран;
- `IsGet()` — для проверки, был ли выполнен GET-запрос;
- `IsPost()` — для проверки, был ли выполнен POST-запрос.

В классе `Controller` присутствует объявление уже известного нам метода `Request()`. Давайте посмотрим на его реализацию.

```
public function Request()
{
  $this->OnInput();
  $this->OnOutput();
}
```

В методе `Request()` мы делим обработку запроса на две части: обработку входных данных (метод `OnInput()`) и формирование результирующей страницы (метод `OnOutput()`). Такое разделение довольно удобно, и мы предлагаем его использовать и в будущих проектах. Методы `OnInput()` и `OnOutput()` должны быть переопределены в дочерних классах.

Класс *C_Base*

`C_Base` является базовым контроллером для нашего конкретного сайта. В большинстве случаев на весь сайт нужен один базовый контроллер. Класс `C_Base` отвечает за базовый шаблон HTML, в нем могут быть определены базовые для всего сайта переменные, например, название сайта.

Кроме того, в классе `C_Base` нужно переопределить методы `OnInput()` и `OnOutput()` и вынести в них действия, общие для всех контроллеров сайта. Контроллеры, которые будут наследоваться от класса `C_Base`, будут переопределять эти методы, но они смогут вызвать родительские методы с помощью ключевого слова `parent`.

Мы предлагаем следующую реализацию методов `OnInput()` и `OnOutput()` в классе `C_Base`.

```
protected function OnInput()
{
  $this->title = 'Главная страница';
  $this->content = '';
}
```

```
protected function OnOutput()
{
  $vars = array('title' => $this->title,
                'content' => $this->content);

  $page = $this->Template('main', $vars);
  echo $page;
}
```

В методе `OnInput()` мы можем инициализировать переменные шаблона значениями по умолчанию. В методе `OnOutput()` мы сначала формируем массив переменных шаблона, а затем используем метод `Template()`, который определен в классе `Controller` и отвечает за подстановку переменных в шаблон. Его реализация остается за вами. По нашей задумке метод `Template()` принимает два параметра: первый из них — название подключаемого шаблона, второй параметр — это массив переменных, которые должны быть подставлены в шаблон. В завершение сформированная страница выводится на экран. Как вы понимаете, это последняя операция, которую должен выполнить сценарий. Поэтому функция `OnOutput()` класса `C_Base` должна вызываться в самом конце нашего PHP-сценария.

Контроллеры *C_View* и *C_Edit*

`C_View` и `C_Edit` — контроллеры конкретных страниц сайта, они переопределяют методы `OnInput()` и `OnOutput()`, но также передают управление контроллеру `C_Base`.

Давайте рассмотрим пример реализации этих методов для класса `C_View`.

```
protected function OnInput()
{
  parent::OnInput();
  $this->title = $this->title . ' :: Чтение';
  $this->text = $this->getText();
}
protected function OnOutput()
{
  $vars = array('text' => $this->text);
  $this->content = $this->Template('theme/v_view.php', $vars);
  parent::OnOutput();
}
```

В методе `OnInput()` сначала идет вызов родительского метода, затем определяется заголовок страницы и извлекается ее текст с помощью метода `getText()`. Конкретная реализация метода `getText()` для нас сейчас не важна. Он может получать текст из базы данных или из файла, или же вовсе возвращать некоторую предопределенную константу. Важно то, что в результате мы сохраняем текст в некотором поле класса.

Метод `OnOutput()` начинается с формирования массива, который передается в функцию формирования шаблона `Template()`. Результат ее выполнения сохраняется в поле `content`. После чего вызывается родительский метод `OnOutput()`. Вернитесь

к тексту метода `OnOutput()` класса `C_Base` и обратите внимание на то, что в нем как раз-таки используется значение поля `content`, который мы формируем в текущем методе `OnOutput()` класса `C_View`.

Если вы взглянете на код еще раз, то заметите, что вызов родительских методов `OnInput()` и `OnOuput()` как бы *обрамляет* цикл обработки запроса в классе `C_View`.

Цикл обработки запроса

Теперь давайте посмотрим на общий вид обработки запроса, который мы получили. Выполнение сценария начинается с файла index.php. Здесь выбирается нужный контроллер, и ему передается управление вызовом метода `Request()`.

Этот метод запускает две фазы обработки запроса:

- обращение к модели (`OnInput()`);
- генерация HTML (`OnOutput()`).

Первая фаза должна отработать в такой последовательности:

- базовый контроллер (`C_Base`);
- конкретный контроллер (`C_View` или `C_Edit`).

Далее начинается фаза генерации HTML, она должна пройти в обратной последовательности:

- конкретный контроллер (`C_View` или `C_Edit`);
- базовый контроллер (`C_Base`).

На диаграмме данный цикл можно представить так, как показано на рис. 13.4.

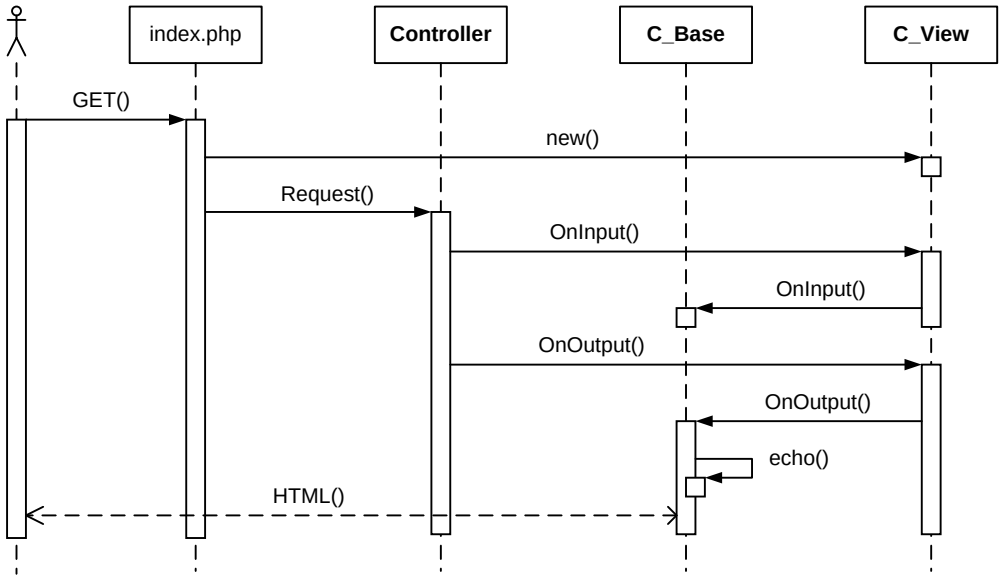


Рис. 13.4. Цикл обработки запроса к сайту

Задание

Реализовать пример из данной главы с использованием концепций MVC и ООП. За основу возьмите предложенную иерархию классов контроллеров.

Резюме

На этом мы завершаем рассказ о применении ООП совместно с концепцией MVC, а вместе с ним и о языке PHP. Оставшиеся функциональные блоки сайта вам предстоит реализовать самостоятельно. Не пугайтесь высокой сложности задания. Постепенно новые концепции станут для вас знакомыми, и вы сможете намного более уверенно в них ориентироваться.

Программирование еще никому не давалось без усилий. С каждым днем практики, с каждым новым PHP-сценарием вы набираете собственный бесценный опыт, который в будущем позволит вам быстро, грамотно и эффективно разрабатывать интернет-системы любой сложности. Залог такой возможности — это профессиональный подход к веб-разработке.

Поэтому именно профессиональному подходу мы уделяем особое внимание в нашем центре компьютерного обучения "Школа программирования" (**www.proglive.ru**), т. к. считаем это наиболее важным и приоритетным. Также мы считаем, что любое обучение не должно быть односторонним, т. е. без возможности задать вопрос преподавателю. Поэтому мы предлагаем вам выполнить последнее задание книги из этой главы и отправить его нам по адресу **support@prog-school.ru**.

Всем читателям, хорошо выполнившим задание, мы подарим видеокурс "Секреты профессии программиста: карьера, фриланс, бизнес", который, без сомнения, будет интересен любому программисту.

Надеемся, что изучение данной книги было для вас увлекательным, и после ее прочтения у вас появилось еще большее желание заниматься веб-программированием. Это очень хороший знак. Успехов вам на этом пути!

Заключение

Мы хотим поздравить вас с завершением обучения! Если вы честно выполняли задания каждого урока, то на текущий момент обладаете существенным багажом знаний и опыта веб-разработки. Позвольте на прощание дать несколько полезных советов.

В первую очередь мы настоятельно рекомендуем вам не прекращать свое профессиональное развитие. В наш век от успешного программиста требуется постоянная активность. Новые технологии появляются чуть ли не каждый день. Обращайте внимание на то, как выглядят современные сайты, изучайте их функционал. Находите глобальные тенденции развития Интернета, такие как социальные сети, интеграция веб-сервисов и т. п. Следите также за миром веб-разработки. Узнайте, что такое фреймворки и как их использовать. Использование фреймворков — это один из будущих шагов в вашем профессиональном развитии.

На первый взгляд вам может показаться, что за прогрессом буквально невозможно угнаться, но это не так. С накоплением опыта вы все чаще будете замечать, что многие концепции в программировании похожи друг на друга. Главное — не останавливаться, постоянно совершенствоваться и не терять интереса к разработке.

Ну а чтобы интерес не пропадал, вы должны заниматься не только обучением, но и применять полученные знания для достижения своих профессиональных целей, будь то работа или реализация собственных проектов. Мы советуем как можно раньше заняться поиском заказов на создание веб-сайтов и накапливать опыт реальной разработки. Поверьте, между программированием обучающих примеров и созданием реальных веб-проектов существует большая разница. Не столько в написании кода, сколько в общем подходе к разработке. Поэтому желательно как можно раньше начать практиковаться на реальных проектах.

Еще одним важнейшим секретом успеха программиста является его "группа поддержки". Как и в любом деле, в программировании сложно существовать в одиночку. Ищите вокруг себя единомышленников, людей со схожими интересами, сообщества, которые были бы вам интересны. Развиваться на порядок легче, если рядом с вами будут люди, разделяющие ваши стремления.

Такое сообщество стараемся создать и мы в своем проекте "Школа программирования". Заходите на наши сайты **http://prog-school.ru**. Там вы найдете большой архив полезных статей по программированию, форум, где уже общаются ваши единомышленники, и множество других полезных аудио-, видео- и текстовых материалов. Мы всегда рады общению. Присоединяйтесь и вы, давайте совершенствоваться в нашей профессии вместе!

ПРИЛОЖЕНИЕ

Описание компакт-диска

Компакт-диск содержит видеоуроки, которые соответствуют урокам книги, хотя и более расширены по тематике, и представляют собой файлы в формате MP4. Просмотреть их можно с помощью любого видеопроигрывателя, поддерживающего этот формат.

- Урок 1. Основы протокола HTTP.
  - Как работать с книгой.
  - Что происходит между тем, как вы набрали адрес страницы в браузере и увидели ее на экране.
  - Введение в протокол HTTP.
  - Как посмотреть запрос и ответ в браузере.
  - Утилита Telnet.
  - Как скачать страницу с помощью Telnet.
- Урок 2. Подготовка рабочего места.
  - Что такое локальный веб-сервер.
  - Пакет Денвер.
  - Установка Денвера.
  - Первый PHP-сценарий.
  - Текстовый редактор Notepad++.
- Урок 3. Введение в HTML и CSS.
  - Как создать CSS-файл.
  - Структура HTML документа.
  - Текст.
  - Картинки.

- Заголовки.
- Абзацы.
- CSS.

❒ Урок 4. Основы PHP.

- PHP-сценарий.
- Вызов функций.
- Переменные.
- Комментарии.
- Генерация HTML.
- Работа с датой и временем.
- Практика использования CSS.

❒ Урок 5. Ветвления и функции.

- Условный оператор.
- Операции сравнения.
- Арифметические операции.
- Работа с шестнадцатеричными числами.
- Работа с фоном страницы.

❒ Урок 6. Циклы и массивы.

- Работа с массивами.
- Цикл `while`.
- Цикл `for`.
- Бесконечный цикл.
- Сортировка пузырьком.

❒ Урок 7. Запросы HTTP, параметры URL и формы HTML.

- GET-запросы.
- Параметры URL.
- POST-запросы.
- Формы-HTML.
- Динамический контент.
- MD5.

❒ Урок 8. Cookies и сессии.

- Простейшая авторизация на сайте.
- Защита страниц.

- Сессии PHP.
- Cookies.

❒ Урок 9. Работа с файлами.
- Создание файла.
- Добавление строк в файл.
- Чтение строк из файла.
- Подсчет статистики посещений сайта.

❒ Урок 10. Работа с базой данных.
- Работа с БД с помощью phpMyAdmin.
- Подключение PHP сценария к БД.
- Чтение строк из БД.
- Добавление строк в БД.
- Изменение строк БД.

❒ Урок 11. Архитектура сайта.
- Как правильно разделять программный код.
- Концепция MVC.
- Контроллер.
- Модель.
- Представление.
- Темы оформления сайта.

❒ Урок 12. Основы ООП.
- Практические примеры использования ООП.
- Именование классов.
- Методы класса.
- Конструктор.
- Область видимости.
- Инкапсуляция.
- Уровни абстракции.
- Класс модели.
- Драйвер работы с БД.

❒ Урок 13. Реализация MVC с помощью ООП.
- Часть 1.
  - Класс контроллера.
  - Класс протоколирования.

- Выделение базового класса.
- Один интерфейс — несколько реализаций.
- Полиморфизм.

- Часть 2.
  - Каркас сайта MVC+ООП.
  - Структура файлов и папок.
  - Единая точка входа.
  - Несколько контроллеров.
  - Рендеринг шаблонов.
  - Вложенные шаблоны.

Литература

1. [IEEE 1471]

   http://www.iso-architecture.org/ieee-1471/.

2. [McGovern (Мак-Говерн)]

   James McGovern, et al. A Practical Guide to Enterprise Architecture. — Prentice Hall, 2004. (Мак-Говерн Дж. и др. Практическое руководство по архитектуре корпораций.)

3. [UML 1.5]

   http://www.omg.org/spec/UML/1.5/PDF/.

4. [Басс (Bass)]

   Басс Л., Клементс П., Кацман Р. Практическая архитектура программного обеспечения на практике. 2-е изд. — СПб.: Питер, 2006. (Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Second Edition. — Addison Wesley, 2003.)

Предметный указатель

Ф

Ц