ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современный Интернет – весьма сложная и высокотехнологичная система, позволяющая пользователю общаться с людьми, находящимися в любой точке земного шара, быстро и комфортно отыскивать любую необходимую информацию, публиковать для всеобщего сведения данные, которые он хотел бы сообщить всему миру. С помощью Интернета можно найти себе подходящую работу и расширить круг знакомств, обсудить интересующие темы и просто приятно провести время.

Интернет-технологии стремительно развиваются, проникая в самые разнообразные сферы профессиональной деятельности, в том числе и экономической. Для компаний присутствие в Интернете – это возможность рассказать о своих товарах и услугах, найти потенциальных партнеров и клиентов, а также снизить издержки за счет Интернет-торговли, использования «облачных» сервисов. Даже такие традиционно замкнутые системы, как промышленные автоматизированные системы управления производством, в том числе и в критических отраслях, также в большинстве случаев прямо или косвенно подключены к Интернету.

Рядовые пользователи активно пользуются Интернет-магазинами, Интернет-банкингом, общаются в социальных сетях, могут получать через Интернет государственные услуги, доверяя Интернет-системам свои персональные данные и другую конфиденциальную информацию.

Вместе с тем, новые возможности порождают и новые угрозы безопасности, которые не всегда в достаточной мере осознаются как рядовыми пользователями, так и владельцами ресурсов. Для компаний наиболее тревожной является тенденция роста атак на корпоративные сайты и web-приложения.

Как показывают различные исследования, 60-75% всех зафиксированных атак за 2010 год было направлено именно на web-приложения, при этом степень распространения уязвимостей существующих web-приложений остается достаточно высокой. Так, по данным компании Positive Technologies, вероятность обнаружения критичной ошибки в web-приложении автоматическим сканером составляет около 35% и достигает 80% при детальном экспертном анализе. Это является следствием того, что большинство компаний недооценивают опасность подобных атак, а разработчики web-приложений недостаточно тщательно следят за безопасностью собственных продуктов.

Однако для рассмотрения вопросов, касающихся угроз безопасности, уязвимостей и методов защиты web-приложений, необходимо обладать базовыми навыками использования основных web-технологий. Данное учебное пособие в сжатой форме знакомит со всеми основными web-технологиями, такими как HTML/XHTML, CSS, JavaScript, PHP и MySQL, современными принципами и инструментальными средствами разработки web-сайтов и серверным программным обеспечением.

РАЗДЕЛ 1. ПРЕДСТАВЛЕНИЕ И ПЕРЕДАЧА ИНФОРМАЦИИ В СЕТИ ИНТЕРНЕТ

1.1. Использование web-технологий

HTML/XHTML и CSS

Web-страницы – это файлы в формате «неформатированный текст» (plain text, текст в ASCII-кодах), распознаваемые любой операционной системой. Поэтому такой файл может быть отравлен практически на любой компьютер, подключенный к Интернет. Кроме того, для создания web-страницы достаточно иметь самый простой текстовый редактор, например, стандартное приложение Windows Блокнот (Notepad).

Файлы web-страниц имеют расширения htm, html или другие, если при их создании были использованы специальные серверные технологии (например, shtml, asp, php). Тип HTML-файлов происходит от названия языка создания web-страниц HTML (*HyperText Markup Language* – язык разметки гипертекста). Поэтому web-страницы часто называют HTML-документами.

HTML-документ содержит текст и команды языка HTML, которые называют дескрипторами разметки или тегами (*tag*).

Теги языка HTML определяют:

- внешний вид документа (формат шрифта, цвет фона и т. д.);
- структуру документа: взаимное расположение текстовой, графической и другой мультимедийной информации;
- ссылки на другие интернет-ресурсы.

HTML позволяет формировать на странице сайта текстовые блоки, включать в них изображения, организовывать таблицы, управлять отображением цвета документа и задавать форматирование текста, добавлять звуковое сопровождение, организовывать гиперссылки для перехода в другие разделы сайта к иным Интернет-ресурсам, а также задавать взаимное расположение текстовых блоков и других компонент страницы.

Встраиваемые в web-страницу с помощью тегов медийные объекты (графика, видео и т. д.) хранятся в отдельных файлах соответствующих типов.

Любой web-браузер содержит интерпретатор языка HTML, что позволяет ему корректно отображать web-страницу вместе со всем ее содержимым на экране. Все сказанное относится и к XHTML – более поздней модификации языка HTML.

С развитием web-технологий разработчики web-страниц стали придерживаться принципа разделения описания логической структуры

web-страницы (которое производится с помощью языков разметки HTML/XHTML) и описания внешнего вида этой web-страницы. Для некоторых вариантов HTML/XHTML соблюдение данного принципа является обязательным. Такой подход позволяет отделить содержимое HTML-документа от дизайна и верстки.

Для описания внешнего вида элементов web-страницы на экране браузера стали использовать технологию стилевого оформления web-страниц – таблицы каскадных стилей CSS (*Cascading Style Sheets*).

Таблицы каскадных стилей CSS – это простая технология определения и присоединения стилей оформления к документам HTML.

Стиль оформления – это все то, что определяет внешний вид документа: цвета, шрифты, рамки и границы, отступы, другое оформление и выравнивание текста, расположения отдельных блоков, обтекание и т.д. Стиль определяется набором правил отображения тегов, задаваемых таблицей стилей. Таблица стилей – это шаблон, который управляет форматированием тегов HTML в web-документе.

Возможности оформления, предоставляемые технологией CSS, гораздо богаче стандартных средств форматирования языка HTML/XHTML. Важно, что стиль можно определить сразу для группы элементов web-страницы; стиль можно сохранить отдельно от страницы и применить сразу к нескольким страницам, задав, таким образом, для них единое оформление. Кроме того, не представляет труда сменить внешний стиль web-страницы или применить к одной и той же странице несколько стилей. Все это значительно повышает гибкость определения и модификации дизайна страницы.

Статические и динамические сайты

Технологии HTML/XHTML и CSS являются базовыми при создании web-страниц. Их использование позволяет создавать красивые несложные информационные сайты, однако страницы таких сайтов будут статичными. Пользователям, возможно, будет не хватать интерактивности, а сами сайты могут показаться скучными. Функциональность таких сайтов также невысока.

Основу статического web-сайта составляют статические web-страницы, разработанные с использованием стандартной HTML-технологии. Страницы сайта хранятся в виде HTML-кода в файловой системе сервера. Естественно, на таком сайте могут присутствовать Flash-заставки, ролики, анимация и т. п.

Основная отличительная особенность *статического сайта* заключается в том, что web-страницы сайта создаются заранее. Для редактирования информации и обновления такого сайта страницы модифицируют

вручную, с применением HTML-редактора, и затем заново загружают их на сайт (рис. 1).

Такая схема приемлема в том случае, если содержимое сайта достаточно постоянно и изменяется сравнительно редко. Если же информация, размещенная на сайте, требует постоянной актуализации и обновления, неизбежны серьёзные трудозатраты на поддержание статического сайта.

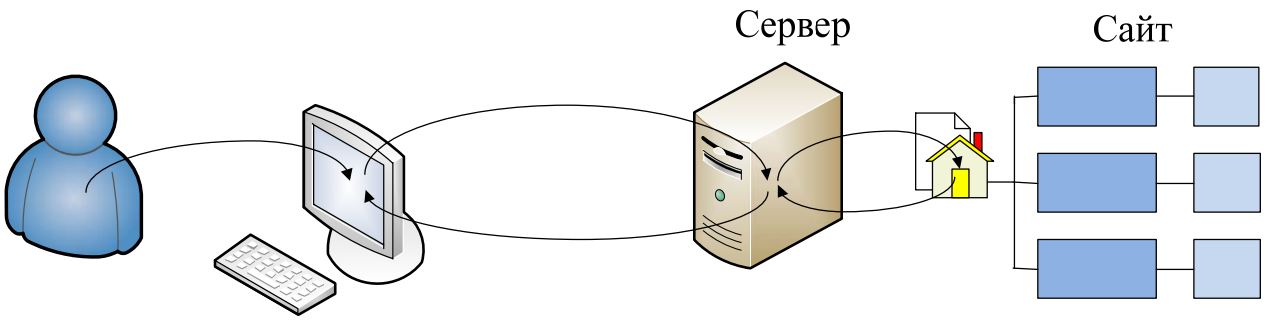


Рис. 1. Статический сайт

Таким образом, статический сайт дешевле в разработке и технической поддержке, но эти достоинства могут нивелироваться серьезными недостатками в оперативности публикации актуальной информации и трудоемкостью модификации.

Динамический сайт – это сайт с динамическим информационным наполнением. Динамические страницы также формируются с помощью HTML, но такие страницы обновляются постоянно, нередко при каждом новом обращении к ним. Динамические сайты основываются на статических данных и HTML-разметке, но дополнительно содержат программную часть – скрипты, благодаря которым страница «собирается» из отдельных фрагментов в режиме реального времени. Это позволяет обеспечить гибкость в подборе и представлении информации, соответствующей конкретным запросам посетителей сайта.

Таким образом, динамический сайт состоит из набора «строительных блоков» динамических страниц – шаблонов дизайна, информационного наполнения (контента), скриптов, хранящихся в виде отдельных файлов. Динамическая web-страница формируется из страницы-шаблона и отдельно расположенного информационного содержимого по запросу (рис. 2). Как правило, для отображения любого количества однотипных страниц используется один шаблон, в который добавляется содержимое, впоследствии отображаемое в обозревателе пользователя.

Рис. 2. Динамический сайт (генерация страниц на стороне сервера)

Динамичность заключается в том, что для изменения страницы достаточно изменить ее информационное наполнение, а сам механизм формирования и вывода страницы остается тем же.

Кроме информационного наполнения, динамически могут создаваться также и элементы навигации по сайту. Таким образом, при обновлении содержимого сайта, необходимо просто добавить текстовое содержимое новой страницы, которое затем вставляется в базу данных с помощью определенного механизма. В результате получается, что сайт как бы сам себя обновляет.

Динамические сайты различаются в зависимости от используемых технологий. Процесс получения динамических страниц также может различаться:

1. Генерация на стороне сервера (осуществляется серверными скриптами на языках PHP, Perl, ASP.NET, Java, Python и др., а информационное наполнение хранится в базах данных).
2. Генерация на стороне клиента (JavaScript).
3. Комбинированная генерация (чаще всего на практике встречается именно комбинация первых двух методов).

Технология клиентского программирования JavaScript

Простейшим средством «оживления» web-страниц, добавления динамических эффектов, задания реакции на пользовательские действия является скриптовый язык программирования JavaScript. Сценарии (скрипты) JavaScript внедряются в web-страницу или связываются с ней, и после загрузки страницы с сервера выполняются браузером на стороне клиента. Все современные браузеры имеют поддержку JavaScript.

JavaScript – компактный объектно-ориентированный язык для создания клиентских web-приложений. JavaScript используется для обра-

ботки событий, связанных с вводом и просмотром информации на web-страницах.

JavaScript обычно используется в клиентской части web-приложений – клиент-серверных программ, в которых клиентом выступает браузер, а сервером – web-сервер, имеющих распределённую между сервером и клиентом логику.

Обмен информацией в web-приложениях происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому web-приложения являются кроссплатформенными сервисами.

Клиентская часть среды проектирования web-приложений содержит следующие основные компоненты:

- браузер (обозреватель) – средство просмотра web-страниц;
- языки разработки web-страниц HTML/XHTML и CSS;
- языки сценариев (в настоящее время в качестве стандарта принят JavaScript);
- клиентские расширения, такие как элементы управления ActiveX, Java-апплеты, подключаемые модули (plugin, плагины), такие как плееры Macromedia Flash или Silverlight и т. д.

Язык сценариев JavaScript позволяет создавать интерактивные web-страницы и содержит средства управления окнами браузера, элементами HTML-документов и стилями CSS.

Технологии серверного программирования

Без использования серверного программирования нельзя обойтись, если необходимо изменять и сохранять какую-нибудь информацию на сервере (например, организовать приём и сохранение отправляемых пользователями сообщений). Без серверных скриптов невозможно представить себе гостевые книги, форумы, чаты, опросы/голосования, счетчики посещений или другие программные компоненты, которые активно взаимодействуют с базами данных.

Серверное программирование позволяет решать такие задачи, как регистрация пользователей, авторизация пользователей и управление аккаунтом (в почтовых web-системах, социальных сетях и др.), поиск информации по базе данных, работа интернет-магазина и т. п.

Серверные языки программирования открывают перед программистом большие функциональные возможности. Современные сайты зачастую представляют собой чрезвычайно сложные программно-информационные системы, решающие значительное количество разнообразных задач, и теоретически могут дублировать функции большинства бизнес-приложений.

Работа серверных скриптов зависит от платформы, то есть от того, какие технологии поддерживаются сервером, на котором расположен сайт. Например, основной технологией, поддерживаемой компанией Microsoft является ASP.NET (ASP), другие серверы могут поддерживать языки PERL, Python, PHP.

Perl – высокоуровневый интерпретируемый динамический язык программирования общего назначения, является одним из наиболее старых языков, используемых для написания серверных скриптов.

По популярности Perl сейчас очень уступает более простому в освоении языку PHP. В настоящее время используются бесплатные технологии EmbPerl и mod_perl, позволяющие обрабатывать html-страницы со вставленными скриптами на языке Perl. Обработчик реализуется как CGI-программа или модуль расширения сервера (Apache, MS IIS).

PHP (*Hypertext Preprocessor*, первоначально Personal Home Page Tools) – язык сценариев общего назначения, в настоящее время интенсивно применяемый для разработки web-приложений.

PHP-код может внедряться непосредственно в HTML. PHP является языком с открытым кодом. PHP крайне прост для освоения, но вместе с тем способен удовлетворить запросы профессиональных web-программистов, так как имеет большой набор специализированных встроенных средств.

РАЗДЕЛ 2. БАЗОВЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ WEB-СТРАНИЦ

2.1. Основы языка HTML/XHTML

Правила отображения браузерами

Web-страницы – это текстовые файлы, содержащие собственно текст содержимого страницы и команды форматирования, называемые *тегами* (*tag*) или *дескрипторами разметки* языка HTML. Язык разметки гипертекста HTML (*HyperText Markup Language*) является базовой технологией разработки web-страниц, которые также называют HTML-документами.

Код HTML интерпретируется браузером (средством просмотра web-страниц), который выводит отформатированное содержимое web-страницу на экран. Просмотр кода загруженной страницы доступен во всех браузерах, например, в MS Internet Explorer это можно сделать с помощью команды *Вид*/*Просмотр HTML-кода*.

Правила синтаксиса и версии языка HTML

При написании кода страницы названия (имена, идентификаторы) тегов заключаются в угловые скобки < >.

Например, для того, чтобы выделить текст в отдельный абзац, его можно заключить в тег p (от paragraph – абзац):

```
<p>Текст абзаца</p>
```

Большинство тегов используются попарно и называются *парными,* или *тегами-контейнерами*. Первый тег называется *открывающим*, а соответствующий парный ему – *закрывающим*. Закрывающий тег имеет то же имя, что и открывающий, но предваряется символом слеша (/).

Действие тегов-контейнеров распространяется на все содержимое, заключенное между открывающей и закрывающей частями тега. Таким образом, открывающая часть тега отмечает начало действия тега, а закрывающая – конец (прекращение) действия тега.

Теги-контейнеры могут содержать не только текст, но и другие теги, которые в этом случае называют *вложенными*. Число вложенных тегов и уровней вложенности не ограничено.

Например, часть текста абзаца можно выделить жирным шрифтом с помощью тега b (от bold – жирный):

```
<p>Текст абзаца <b>с жирным написанием</b></p>
```

Для тега можно задавать *параметры* (атрибуты, attribute), которые уточняют действие тега, то есть определяют дополнительные характеристики и режимы его функционирования. Наборы допустимых параметров индивидуальны для каждого тега. Параметры указываются обязательно

после имени тега и отделяются друг от друга пробелами. Порядок следования параметров в теге произволен.

Большинство параметров требует указания принимаемых значений. Если требуется указать значение, оно записывается после имени параметра через знак равенства = и заключается в двойные кавычки.

Например, текст абзаца можно отцентрировать, задав параметр align со значением center:

```
<p align="center">Текст абзаца</p>
```

Если параметр не указан в теге в явном виде, то принимается его значение по умолчанию. Например, если не указывать параметр align для тега p, то по умолчанию текст абзаца будет выровнен по левому краю страницы.

В HTML некоторые параметры не требуют указания значений. Обычно это параметры, которые выключают некоторое свойство.

Например, отключение переноса текста в ячейке таблицы:

```
<td nowrap>Текст ячейки</td>
```

Принципиальным отличием закрывающей части тега является то, что она никогда не содержит параметров. Таким образом, она прекращает действие тега вместе со всеми его настройками.

Кроме тегов-контейнеров в HTML, определены теги, выполняющее некоторое одиночное действие, например, вставку рисунка img, горизонтальной линии hr, перевод текста на новую строку br и поэтому не требуют закрывающей части. Эти теги называются *непарными*.

Исключение из правил записи тегов составляет тег комментария с ограничителями <!-- и -->. Все, что находится внутри этого тега, считается комментарием и не отображается браузером на экране.

Все теги и их атрибуты специфицированы в HTML, их число достаточно ограничено. С течением времени набор тегов и рекомендации по их использованию менялись. В настоящее время часть тегов и атрибутов, входящих в актуальную спецификацию HTML, помечены как устаревшие и не рекомендованные к использованию. Спецификации языка HTML приведены в табл. 1.

Официальной спецификации HTML 1.0 не существует. До 1995 года существовало множество неофициальных вариантов HTML. Чтобы стандартная версия отличалась от них, ей сразу присвоили второй номер.

Версия 3 была предложена Консорциумом всемирной паутины (World Wide Web Consortium, W3C) в марте 1995 года и обеспечивала много новых возможностей, таких как создание таблиц, «обтекание» изображений текстом и отображение сложных математических формул. Даже при том, что этот стандарт был совместим со второй версией, реализация его была сложна для браузеров того времени. Версия 3.1 официально никогда не предлагалась, и следующей версией стандарта HTML стала 3.2, в

которой были опущены многие нововведения версии 3.0, но добавлены нестандартные элементы, поддерживаемые популярными в то время браузерами Netscape и Mosaic.

Таблица 1

Спецификации языка HTML

Название спецификации	**Год выхода**
HTML 2.0 (одобрена как Интернет-стандарт RFC 1866)	1995
HTML 3.2	1996
HTML 4.0	1997
HTML 4.01 – действующая спецификация	1999
Международный стандарт ISO/IEC 15445:2000 (ISO HTML, основан на HTML 4.01 Strict)	2000
HTML 5.0	в разработке с 2004

HTML версии 4.0 содержит много элементов, специфичных для отдельных браузеров, но в то же время произошла некоторая «очистка» стандарта. Многие элементы были отмечены как устаревшие и не рекомендованные (deprecated). В частности, были помечены как устаревшие такие теги, как font, используемый для изменения свойств шрифта, u, задающий подчеркивание символов текста (вместо них рекомендуется использовать таблицы стилей каскадные таблицы стилей CSS). С другой стороны, HTML 4.0 стал поддерживать большее количество опций мультимедиа, языков скриптов, каскадных таблиц стилей.

В версии HTML 4.01 многие, отмеченные как устаревшие, теги (определение шрифта и базового шрифта font, basefont, центрированный абзац center, подчеркнутый u и зачеркнутый текст s, strike и др.), а также ряд атрибутов оформления (таких как выравнивание align, задание фоновых цветов bgcolor, упомянутый ранее nowrap и др.) не поддерживаются строгой версией языка. Версия языка указывается в HTML-документе с помощью определения типа документа (doctype):

- Строгий (*Strict*) – не содержит элементов, помеченных как «устаревшие» или «не рекомендованные»;
- Переходный (*Transitional*) – содержит устаревшие теги в целях совместимости и упрощения перехода со старых версий HTML;
- С фреймами (*Frameset*) – аналогичен переходному, но содержит также теги для создания фреймовых структур.

Начиная с 2004 года W3C ведет разработку HTML версии 5. Черновой вариант спецификации языка появился в Интернете 20 ноября 2007 года.

В HTML5 появляется множество синтаксических особенностей. HTML5 вводит несколько новых элементов и атрибутов, некоторые устаревшие элементы, которые еще можно было использовать в HTML 4.01 (такие как font и center), были исключены, исключена и поддержка фреймов.

Предположительно, спецификация HTML 5 будет рекомендована к применению в 2014 году.

На рабочем семинаре в 1998 г. консорциум W3C решил, что с точки зрения языков разметки будущим Web является развитие HTML в сторону обеспечения совместимости с XML. Поэтому W3C подвел черту под HTML 4.01, и сконцентрировался после этого на спецификации XHTML 1.0 (*Extensible Hypertext Markup Language* – расширяемый язык разметки гипертекста), законченной в начале 2000 г. XHTML по своим возможностям сопоставим с HTML, однако предъявляет более строгие требования к синтаксису.

Вскоре последовал язык XHTML 2.0, который добавил целый пакет новых мощных средств с целью стать следующей основой Web. Проблема с XHTML 2.0 состояла в том, что он не являлся обратно совместимым с уже имеющейся web-разметкой.

HTML5 – это попытка определить единый язык разметки, который мог бы способствовать созданию нового поколения web-приложений, не разрушая – что критически важно – обратной совместимости как с HTML, так и с XHTML.

Пока же web-стандартом де факто остается XHTML 1.0. Рассмотрим подробнее особенности этого языка.

Особенности XHTML

Как и HTML, XHTML является подмножеством языка SGML (*Standard Generalized Markup Language*, стандартный обобщённый язык разметки) – метаязыка, на котором можно определять язык разметки для документов (рис. 3). От SGML произошли HTML и XML (*Extensible Markup Language*, расширяемый язык разметки). HTML – это конкретное приложение SGML, а XML – это подмножество SGML, разработанное для упрощения процесса машинного анализа документов.

HTML не вполне совместим с XML, поскольку менее строг к синтаксису, и не может считаться его подмножеством. Расширяемый язык разметки гипертекста XHTML, в отличие от обычного HTML, является подмножеством языка XML. Фактически, XHTML – это HTML, записанный в соответствии с синтаксическими правилами языка XML.

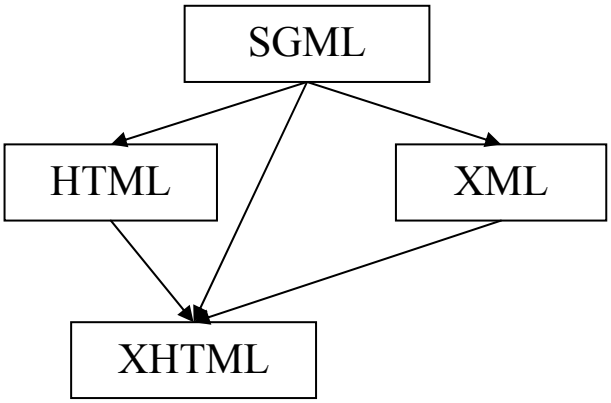


Рис. 3. Эволюция языков разметки

XML – это обобщенный язык разметки, он не имеет набора заранее определенных тегов. В отличие от HTML, XML позволяет создавать собственные теги и таким образом формировать собственные структуры. XML – текстовый формат, предназначенный для хранения произвольных структурированных данных (взамен существующих форматов файлов баз данных), а также для организации универсального обмена данными между приложениями.

Документ XML – это описанная в текстовом формате иерархическая структура, предназначенная для хранения произвольных данных. Визуально структура может быть представлена как дерево элементов. Элементы XML описываются тегами.

Приведем в качестве примера возможное библиографическое описание книги с помощью тегов.

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="1" date="12/04/99">
    <title>Собрание сочинений</title>
    <author>Стругацкий Аркадий</author>
    <author>Стругацкий Борис</author>
    <publisher></publisher>
    <year>1996</year>
</book>
```

Другой пример – описание простого кулинарного рецепта, размеченное с помощью XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe name="хлеб" preptime="5" cooktime="180">
<title>Простой хлеб</title>
    <ingredient amount="3" unit="стакан">Мука</ingredient>
    <ingredient amount="0.25" unit="грамм">Дрожжи </ingredient>
    <ingredient amount="1.5" unit="стакан">Теплая вода </ingredient>
```

```
    <ingredient amount="1" unit="чайная ложка">Соль </ingredient>
    <instructions>
        <step>Смешать все ингредиенты и тщательно замесить. </step>
        <step>Закрыть тканью и оставить на один час в теплом поме-
        щении.</step>
        <step>Замесить еще раз, положить на противень и поставить в
        духовку.</step>
    </instructions>
</recipe>
```

Чтобы утвердить семантические правила нового языка, то есть список допустимых элементов структуры данных, их возможное содержимое и атрибуты, необходимо создать DTD-определения (DTD-схему). Например, можно описать, что внутри структуры book может быть единственный элемент title и year, а элементов author и publisher может быть несколько. Кроме того, можно указать, является ли элемент обязательным, допускает ли он пустое содержимое. Можно также задать порядок следования элементов в структуре.

DTD (*Document Type Definition*, определение типа документа) – описывает схему документа для конкретного языка разметки с точки зрения семантических ограничений этого документа. Также DTD может объявлять конструкции, которые всегда необходимы для определения структуры документа.

Документ XML, написанный в соответствии со всеми общими правилам синтаксиса XML, считается *правильно построенным* (well-formed). Документ, который неправильно построен, не может считаться документом XML; XML-процессор (*парсер*) не должен обрабатывать его обычным образом и обязан классифицировать ситуацию как «фатальная ошибка».

Следующий уровень правильности документа XML – *действительный* (валидный, valid) документ. Действительный документ дополнительно соответствует некоторым семантическим правилам. Это более строгая дополнительная проверка корректности документа на соответствие заранее определённым, но уже внешним правилам, например, структуры и состава данного, конкретного документа или семейства документов. Именно эти правила и задаются DTD-схемой.

В противоположность XML, HTML гораздо более строго определенный язык разметки с ограниченным набором тегов. В любом случае, общий характер XML позволяет рассматривать HTML-документы как XML-документы с набором тегов для отображения в web-браузерах. Однако, старые стандарты HTML не до конца совместимы с XML. Например, в HTML не-

обязательно закрывать тег p, то есть закрывающую часть тега </p> можно опускать. web-браузер сможет правильно интерпретировать эту конструкцию, поскольку он так запрограммирован, однако XML-парсер выдаст ошибку о том, что HTML-документ не является правильно построенным (well-formed).

Чтобы устранить разрыв между этими двумя языками разметки, и был разработан XHTML. По существу это обычный HTML, в который добавили синтаксические правила XML для создания well-formed документов.

Структура XHTML-документа

Корневым элементом документа должен быть html.

Корневой элемент документа должен обозначить пространство имён XHTML путём использования атрибута xmlns (XMLNAMES). Пространство имён XHTML определено в http://www.w3.org/1999/xhtml.

В документе должно присутствовать объявление doctype, предшествующее корневому элементу html. Публичный идентификатор, включённый в объявление doctype, обязан быть ссылкой на одно из определений типа документа.

Каждый документ разделен на две части: *головную* часть (head) и основную часть или *тело* (body) документа (рис. 4).

В документах с фреймами (тип документа Frameset) тело body может отсутствовать (заменено фреймовой структурой).

Головная часть содержит служебную информацию о документе, предназначенную для браузера или поисковых систем, и не отображается на экране.

Все содержимое web-страницы, предназначенное для вывода на экран и просмотра пользователем, помещается в тело документа body. Для того чтобы отделить эти части друг от друга, используются соответствующие теги.

Тег title является обязательным и указывается в разделе head. Элемент title задает название (заголовок) web-страницы. Это название отображается в строке заголовка (в самом верху) окна браузера. Кроме того, содержимое title используется как название web-страницы при ее внесении пользователем в список «Избранное» («Закладки»), а также в результатах поиска, выдаваемых поисковыми системами. Поскольку пользователи часто обращаются к документам вне контекста, следует задавать осмысленные заголовки.

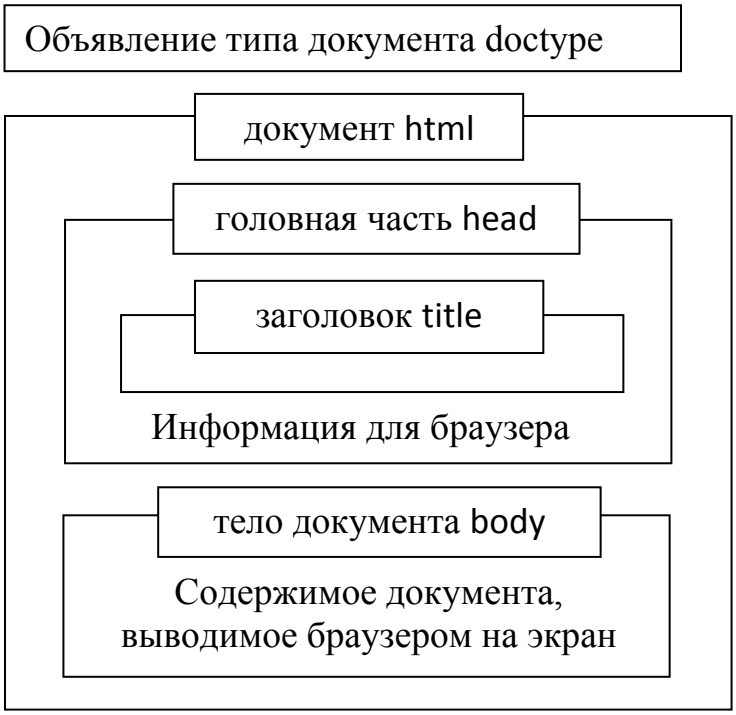


Рис. 4. Структура XHTML-документа

Таким образом, минимальный XHML-документ может выглядеть следующим образом:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="content-type"
    content="text/html; charset=windows-1251">
    <title>Заголовок документа</title>
    </head>
    <body>
    <p>Текст, выводимый на экран</p>
    </body>
</html>
```

В данном примере головная часть документа содержит необязательный тег meta, задающий кодировку web-страницы (для отображения русскоязычных символов). При просмотре web-страницы в окне браузера будет выведена всего одна строка: Текст, выводимый на экран.

Форматирование текста

Теги форматирования могут разбивать текст на блоки (абзацы, списки, таблицы) или определять написание символов текста внутри блока.

Поэтому все теги делятся на теги уровня блока и теги уровня текста. С точки зрения корректности написания XHTML-кода важно следить, чтобы теги уровня текста обязательно указывались внутри какого-либо блока.

Основные теги XHTML, используемые для форматирования абзацев, приведены в табл. 2.

Таблица 2

Основные теги форматирования абзацев в XHTML

Теги	Описание и особенности использования
p	Абзац, отделенный от остального текста пустыми строками. p является тегом уровня блока, однако не может содержать элементы уровня блока (включая сам p). Не рекомендуется использовать пустые элементы p
div	Абзац, не имеющий никакого дополнительного форматирования. Обычно используется для группировки блоков
h1 – h6	Шесть уровней заголовков: от h1 (самый верхний) до h6 (самый нижний). Визуально браузеры обычно отображают более значительный заголовок более крупным шрифтом. Элементы заголовков являются частью глобальной структуры документа и предназначены для краткого описания смысла последующего раздела Теги заголовков могут считываться автоматическими средствами представления документа для создания автоматического оглавления. Важно соблюдать правильный порядок следования уровней заголовков для описания разделов и подразделов с целью их корректной обработки при автоматическом считывании
blockquote	Блок с дополнительным отступом. Выделяется, как и p, пустыми строками
pre	Преформатированный текст – блок, сохраняющий форматирование текста в коде: разбиение на строки, табуляции и пробелы (может отображаться браузером моноширинным шрифтом). Внутри блока pre не действует ряд тегов (img, object, big, small, sub или sup), тег p действует как br.
br	Непарный тег, принудительно обрывает (оканчивает) текущую строку текста.
hr	Непарный тег, вставка горизонтальной линии.

Здесь и далее не приводятся теги и параметры, помеченные как устаревшие.

Тег или параметр, задающий отступ первой строки абзаца (красную строку), не определен. Красная строка может быть задана с помощью таблиц стилей.

Ширина блоков автоматически подстраивается под размер окна браузера (также как абзацы Word подстраиваются под ширину документа за счет автоматического переноса на новую строку). Перенос текста осуществляется по словам. Дополнительную возможность переноса можно задать, используя символ «мягкого дефиса» (или).

В случае если, напротив, необходимо запретить разрыв строки в каком-то месте, можно использовать символ неразрывного пробела   (символы-мнемоники   или).

Теги уровня текста используются для задания специфического написания последовательности символов (указание текста в кавычках, верхние/нижние индексы) или задания параметров шрифта (выделение жирным, курсивом, крупный, мелкий шрифт). Вместе с тем для форматирования шрифтов предпочтительным считается использование таблиц стилей.

Основные теги форматирования символов представлены в табл. 3, рис. 5 показывает их отображение браузером.

Таблица 3

Основные теги форматирования символов

Теги	Описание	Теги	Описание
q	Текст, заключенный в кавычки	big	Крупный шрифт
tt	Моноширинный шрифт	small	Мелкий шрифт
b	Жирный шрифт (*bold*)	sup	Верхний индекс
i	Курсив (*italic*)	sub	Нижний индекс

Обычный текст, «текст в кавычках», моноширинный, **жирный**, *курсив*, ***жирный курсив***, крупный, мелкий, верхний$^{\text{индекс}}$, нижний$_{\text{индекс}}$

Рис. 5. Форматирование символов тегами XHTML

Соответствующий рисунку фрагмент кода:

```
<p>Обычный текст, <q>текст в кавычках</q>,
<tt>моноширинный</tt>, <b>жирный</b>, <i>курсив</i>,
<b><i>жирный курсив</i></b>, <big>крупный</big>,
<small>мелкий</small>, верхний<sup>индекс</sup>,
нижний<sub>индекс</sub></p>
```

2.2. Технология каскадных таблиц стилей CSS

Использование каскадных таблиц стилей

Таблицы каскадных стилей CSS (Cascade Style Sheets) – это простая технология определения и присоединения стилей оформления к документам HTML.

Стиль оформления – это все то, что определяет внешний вид документа: размер, цвет и вид шрифта текста, цвет фона текста, наличие границ, подчеркивания, выравнивание текста и т. д. Стиль определяется набором правил отображения тегов, задаваемых таблицей стилей.

Таблица стилей – это шаблон, который управляет форматированием тегов HTML в web-документе. Таблица стилей состоит из набора *правил* описания стиля.

Любое *правило* каскадных таблиц стилей состоит из двух частей: *селектора* и *определения*. *Селектор* – это любой элемент или группа элементов web-страницы, для которых определяется форматирование. *Определение* описывает конкретный вид форматирования и состоит из двух частей: *свойства* и *значения*, разделенных знаком двоеточия (рис. 6).

Например, цвет текста абзаца может быть задан с помощью стиля как: Текст

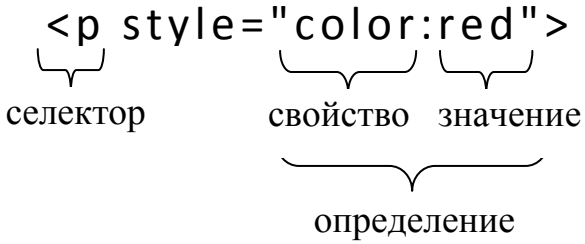


Рис. 6. Правило таблицы стилей

В ряде случаев использование CSS позволяет создать более компактный и эффективный код HTML-документа.

В одном правиле можно задать несколько определений, разделенных знаком точки с запятой (;). Например, задание красного цвета, жирного написания и выравнивания «по ширине» текста абзаца с помощью стиля выглядит как:

 Текст

Некоторые свойства стиля приведены в табл. 4.

Таблица 4

Свойства стиля

Свойство	Описание свойства	Значения
font-family	Тип шрифта	Имя шрифта, список имен
font-size	Размер шрифта	Относительные размеры (small, large, x-large и др.), %, pt, px, mm, cm, in, em и т. п.
font-weight	Толщина шрифта	normal, bold и т. п.
font-style	Наклон шрифта	normal, italic и т. п.
text-transform	Изменение регистра	lowercase, uppercase, capitalize, none
text-decoration	Подчеркивание	underline, overline, line-through, none
color	Цвет текста	цвет
background-color	Цвет фона	цвет
background-image	Фоновый рисунок	url('URL рисунка')
background-repeat	Повторение фонового рисунка (мозаика)	no-repeat, repeat-x, repeat-y, repeat
background-position	Позиционирование фоновой картинки	left, right, center, top, bottom
text-indent	Красная строка	%, pt, px, mm, cm, in, em и т. п.
Свойства стиляborder-style	Тип рамки	double, solid, dashed, dotted, none, outset, inset, ridge
border-width	Толщина рамки	thin, medium, thick, %, pt, px, mm, cm, in, em и т. п.
border-color	Цвет рамки	цвет
word-spacing	Разрядка слов	normal, %, pt, px, mm, cm, in, em и т. п.
letter-spacing	Разрядка символов	normal, %, pt, px, mm, cm, in, em и т. п.
white-space	Управление переносом на новую строку	normal, nowrap, pre, pre-line, pre-wrap
vertical-align	Выравнивание по вертикали	top, bottom, baseline, middle
text-align	Выравнивание текста	left, right, center, justify
padding	Отступы (внутренние)	%, pt, px, mm, cm, in, em и т. п.
margin	Границы блока (внешние отступы)	%, pt, px, mm, cm, in, em и т. п.
visibility	Видимость	visible, hidden

Окончание табл. 4

Свойство	Описание свойства	Значения
display	Отображение на экране	block, inline, none
overflow	Выход контента за границы элемента	hidden, scroll, visible
float	Обтекание текстом	left, right, none
clear	Запрет обтекания	left, right, none, both
position	Тип позиционирования	absolute, fixed, relative, static
z-index	Позиция по «глубине»	неотрицательное целое число
top, right, bottom, left	Позиция элемента	%, pt, px, mm, cm, in, em и т. п.
width, height	Размеры элемента	%, pt, px, mm, cm, in, em и т. п.

Большинство свойств стиля поддерживает также значение inherit – наследование от родительского элемента (контейнера).

Возможности форматирования, предоставляемые CSS, значительно превосходят средства языка HTML как такового; кроме того, использование таблиц каскадных стилей в сочетании со сценариями позволяет добиваться различных динамических эффектов, то есть форматирование страницы может изменяться со временем или в результате каких-то пользовательских действий.

Описание стиля может быть вынесено за рамки конкретного экземпляра тега, то есть задано для всех экземпляров тега или для группы элементов. В этом случае изменение форматирования группы элементов производится однократным внесением изменений в общее описание стиля.

В целом, механизм таблиц каскадных стилей позволяет:

- использовать дополнительные возможности форматирования;
- сократить число используемых в документе тегов;
- задавать более гибкое форматирование, удобное для модификации.

Существует несколько версий технологии CSS: оригинальная версия – CSS1, улучшенная версия – CSS2, исправленная – CSS2.1, версия, разрабатываемая в настоящее время – CSS3.

Не все браузеры поддерживают технологию CSS. Современные версии основных браузеров (IE версии 8 и выше, Mozilla Firefox, Opera, Safari, Google Chrome) обеспечивают ее наиболее полную поддержку. Для проверки поддержки браузером web-стандартов (в том числе и различных частей стандарта CSS) был разработан тест Acid, его версии: Acid2, Acid3 (http://www.acidtests.org/).

Указание правил CSS в HTML-документе может производиться четырьмя разными способами (табл. 5).

Таблица 5

Варианты использования стилей CSS

Область охвата	**Тип таблицы стилей**	**Вариант использования**
Отдельный экземпляр тега /отдельный элемент страницы	Стиль, встроенный в тег	Указание стиля внутри тега
Все или несколько экземпляров тега / группа элементов	Внутренняя таблица стилей	Внедрение стиля (описание стиля – в разделе head страницы)
Несколько страниц сайта	Внешняя таблица стилей	Связывание
Страницы нескольких сайтов		Импорт

Встраивание стиля в теги документа производится с помощью параметра style и позволяет изменить форматирование конкретных экземпляров тегов страницы.

Примеры встраивания стилей в тег p приведены выше.

Внедрение стиля позволяет определять форматирование для всех экземпляров тега или для группы элементов страницы. Таблица стилей внедряется в текст HTML-документа в раздел заголовка head с помощью тега style.

Пример:

```
<html>
<head>
<title>Внедренная таблица стилей</title>
<style type="text/css">
p {     color : red;
        font-weight : bold;
        text-align:justify
        }
</style>
</head>
<body>
<p>Общий стиль всех абзацев</p>
</body>
</html>
```

Связывание с таблицей стилей – таблица стилей сохраняется в виде внешнего файла, а в HTML-документ помещается ссылка на нее. Ссылка на файл с таблицей осуществляется тегом link, который указывается в раз-

деле заголовка head. Связывание позволяет использовать одну таблицу CSS для форматирования элементов сразу нескольких web-страниц.

Пример связывания с файлом стилей.

Содержание файла таблицы стилей (файл с расширением CSS, например, st.css):

```
p {     color : red;
        font-weight : bold;
        text-align:justify
        }
```

Указание в коде web-страницы ссылки на файл таблицы стилей st.css, расположенной в корневой папке сайта:

```
<html>
<head>
        <title>Связанная таблица стилей</title>
        <link rel="stylesheet" type="text/css" href="/st.css">
</head>
<body>
<p>Текст абзаца</p>
</body>
</html>
```

Импорт таблицы стилей – указание месторасположения в сети (URL) файла с таблицей CSS. Вместо тега link в разделе head указывается тег import формата @import url('URL-адрес'); , например:

```
@import url('/st.css');
```

Таблицы стилей называются каскадными из-за порядка применения различных стилей к элементам страницы. Каскад распространяется сверху вниз: сначала внешняя таблица стилей, потом внутренняя, потом стиль, встроенный в тег.

Чем ближе правило к элементу, тем выше его приоритет. Приоритет стиля важен, когда для элемента заданы несколько противоречащих друг другу правил. Приоритет стилей (в порядке убывания) следующий:

1. Стиль, встроенный в тег.
2. Внедренная в страницу таблица стилей.
3. Внешняя таблица стилей (связанная, импортируемая).
4. Стандартные настройки браузера.

На практике разработчики отдают предпочтение какому-либо одному типу таблиц стилей, обычно внешним таблицам, связанным со страницей.

РАЗДЕЛ 3. ТЕХНОЛОГИИ WEB-ПРОГРАММИРОВАНИЯ

3.1. Технология клиентского программирования JavaScript

Использование JavaScript

Язык сценариев JavaScript предназначен для создания интерактивных web-страниц и web-приложений, позволяющих полностью управлять как самими web-страницами, так и браузерами, в которых эти страницы открыты. Это объектно-ориентированный язык программирования, имеющий схожий с языком Си синтаксис.

Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в web-программировании. Первоначально язык предназначался как для программирования на стороне клиента, так и для программирования на стороне сервера. Приложения, написанные на JavaScript, могут исполняться на серверах, использующих Java 6 и более поздних версий. Это используется для построения серверных приложений, позволяющих обрабатывать сценарии JavaScript на стороне сервера. Помимо Java 6, существует ряд платформ, использующих существующие движки (интерпретаторы) JavaScript для исполнения серверных приложений. Однако чаще сценарии JavaScript предназначены для исполнения в среде браузера на стороне клиента.

Поскольку в начале развития скриптового языка для браузеров существовали несколько аналогов от разных компаний-разработчиков, ассоциацией ECMA была проведена стандартизация языка JavaScript. Стандартизированная версия имеет название ECMAScript и описывается стандартом ECMA-262 [28].

ECMAScript не является браузерным языком и в нём не определяются методы ввода и вывода информации. Это скорее основа для построения скриптовых языков. Спецификация ECMAScript описывает типы данных, инструкции, ключевые и зарезервированные слова, операторы, объекты, регулярные выражения, не ограничивая авторов производных языков в расширении их новыми составляющими.

Структурно JavaScript можно представить в виде объединения трёх частей:

- ядро (ECMAScript);
- объектная модель браузера (Browser Object Model или BOM);
- объектная модель документа (Document Object Model или DOM).

Если рассматривать JavaScript в отличных от браузера окружениях, то объектная модель браузера и объектная модель документа могут не поддерживаться. Язык JavaScript применятся не только в Интернете, но и

в таких программах, как, например, Adobe Acrobat Reader или Adobe Photoshop для расширения их возможностей, аналогично использованию языка VBA (Visual Basic for Applications) в Microsoft Office.

Однако чаще всего JavaScript все же используется как язык, встраиваемый в web-страницы для программного доступа к их элементам. Сценарии JavaScript являются основой клиентской части web-приложений. Они загружаются с сервера вместе с web-страницами и выполняются браузером на компьютере пользователя. Сценарии JavaScript обрабатываются встроенным в браузер интерпретатором.

JavaScript широко применяется для решения таких задач, как проверка информации, введенной пользователем в форму, перед ее отправкой на сервер и программирование ответных реакций на действия пользователя. Считается, что язык JavaScript занял лидирующую позицию в разработке клиентских web-приложений в связи с развитием AJAX, поскольку браузер стал превалирующей системой доставки приложений.

Следует, однако, помнить об ограничениях языка. Так, в JavaScript отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода/вывода, нет базовых типов для бинарных данных, отсутствуют стандартные интерфейсы к web-серверам и базам данных. Кроме того, клиентские скрипты могут быть отключены пользователем в настройках браузера.

Сценарии могут встраиваться в HTML-документ тремя стандартными способами:

- с помощью тега script;
- с помощью обработчика события;
- в виде гиперссылки.

Сценарий JavaScript может быть помещен в любом месте web-страницы внутри контейнера script. Если во время интерпретации HTML-документа браузер встретит тег script, он первым делом выполнит код скрипта и лишь затем продолжит интерпретацию страницы дальше. Контейнеров script в одном документе может быть сколько угодно.

В общем виде встраивание скрипта в страницу с помощью тега script имеет вид:

```
<script type="text/javascript">
код сценария
</script>
```

Параметр type можно и не указывать, так как значение text/javascript является значением по умолчанию.

Пример web-страницы со встроенным сценарием, который выводит на экран окошко предупреждения с приветствием «Hello, World!»:

```
<html>
<head>
<title>Приветствие</title>
</head>
<body>
<script type="text/javascript">
alert('Hello, World!');
</script>
<p>Страница с приветствием</p>
</body>
</html>
```

Обычно скрипты JavaScript стараются отделить от собственно HTML/XHTML-документа, с этой целью тег script помещают в головную часть head, а в теле страницы по возможности оставляется чистая верстка.

Сценарии могут содержаться как внутри web-страницы, так и храниться в отдельном файле. Файл скриптов – обычный текстовый файл, имеющий расширение js. Подключение внешнего файла скриптов производится с помощью параметра src тега script, значением которого является URL или относительный адрес файла со сценарием. Например:

```
<script type="text/javascript" src="my_script.js"></script>
```

При этом файл my_script.js содержит код JavaScript, который иначе мог бы находиться внутри тега script.

При указании атрибута src содержимое тега script игнорируется, то есть одним тегом script нельзя одновременно подключить внешний файл скриптов и описать внутренний сценарий. Для решения этой задачи придется создать два разных экземпляра тега script.

Использование отдельных файлов для хранения кода сценариев JavaScript очень удобно, так как позволяет создавать, например, собственные библиотеки функций и объектов, используемые многократно и подключаемые при необходимости к разным страницам.

Подключение внешних файлов со сценариями, объявления функций и всех глобальных переменных обычно помещают внутри тега head, то есть в заголовке страниц.

Обычно выполнение сценария привязано к какому-либо событию и служит для его обработки. В качестве событий выступают любые действия пользователя, например, щелчок мыши (click), перемещение курсора мыши относительно элемента страницы: наведение указателя на элемент (mouseover), перемещение указателя за пределы элемента (mouseout), изменение значения поля формы (change), нажатие пользователем кнопки отправки данных формы (submit) или внутренние события браузера, например, загрузка страницы (load).

Строго говоря, существуют:

- DOM-события, которые инициируются элементами страницы (click, mouseover и т.п.);
- События окна браузера (например, resize – изменение размера окна);
- Другие события (например, load, readystatechange – используется в технологии AJAX).

Любое событие может быть перехвачено и обработано процедурой сценария. Для перехвата события в теги некоторых элементов страницы вводятся параметры обработки события. Имя параметра обработки начинается с приставки on, за которой следует название обрабатываемого события (например, onclick). В качестве значения параметра обработки указывается действие, которое необходимо выполнить.

Пример: при нажатии кнопки будет выдаваться окно предупреждения с текстом «Спасибо!»:

```
<input type="button" value="Нажать" onclick="alert('Спасибо')" />
```

Сценарий JavaScript может быть вызван при активации гиперссылки, для этого в качестве значения адресного параметра href тега гиперссылки a указывается JavaScript-сценарий, предваренный ключевым словом javascript: (со знаком двоеточия).

Пример такой гиперссылки:

```
<a href="javascript:alert('Приветствую вас!')">Нажмите</a>
```

Хорошим стилем программирования является оформление действий, выполняемых при обработке событий, в процедуры.

Определение функции состоит:

- из имени функции, предваренного ключевым словом function. Имена (идентификаторы) JavaScript должны начинаться с латинской буквы и могут содержать буквы латинского алфавита, цифры и знак подчеркивания. Идентификаторы чувствительны к регистру символов;
- из списка формальных параметров, заключённых в скобки и разделяемых запятыми. Если функция не имеет параметров, ее имя указываются с пустыми скобками;
- из последовательности операторов JavaScript, составляющих содержание функции, заключённых в фигурные скобки { }. Операторы функции могут содержать вызовы других функций, определённых в текущем приложении.

Вызов функции производится с помощью обработчика событий или в гиперссылке с указанием ее фактических параметров.

Например, выдача окна с текстом может быть оформлена в виде процедуры.

Описание процедуры (рекомендуется помещать в раздел head):

```
<script type="text/javascript">
```

```
function thanks() {
alert("Спасибо!")
}
</script>
```

Вызов процедуры:

```
<input type="button" value="Нажать" onclick="thanks()" />
```

Функции отличаются от процедур тем, что в результате выполнения функции возвращается некоторое значение. В этом случае значение или содержащее его имя переменной указывается в коде процедуры с ключевым словом return. Функции могут указываться в выражениях и в операции присваивания. Вызов функции в коде web-страницы предваряется словом return. Далее будут приведены примеры использования функций для проверки и управления отправкой данных формы.

3.2. Технологии серверного программирования. Основы языка PHP

Особенности серверного программирования

Серверные программы позволяют решать ряд важных задач, которые невозможно решить с помощью базовых технологий разработки web-страниц и клиентского программирования. К таким задачам относится, например, сохранение полученной от пользователей информации, взаимодействие с внешней базой данных и т. д. Поэтому ни одно сколько-нибудь серьезное web-приложение не обходится без серверной части.

Главной особенностью серверных программ является то, что они могут быть выполнены только на сервере, в отличие от HTML, CSS и сценариев JavaScript, которые интерпретируются пользовательским клиентом – браузером. Обращение браузера к web-серверу может осуществляться и в рамках одного компьютера, в этом случае установка и функционирование web-сервера может использоваться для отладки серверных программ. Для того чтобы гарантировать доступ к данным на компьютере для других пользователей, одного web-сервера недостаточно, потребуются также выделенный IP-адрес и запись в сервере DNS (чтобы пользователи имели возможность обращаться к серверу по символическому имени, а не только по IP-адресу).

Поэтому для разработки и отладки серверной части web-приложения необходимо развернуть и настроить на своем компьютере программное обеспечение web-сервера. Кроме того, необходимо установить транслятор языка серверного программирования, систему управления базой данных (СУБД), с которой будет взаимодействовать web-приложение, и обеспечить взаимодействие всех этих компонентов. Для облегчения этого процесса часто используют устоявшиеся связки – комплексы серверного программного обеспечения:

- web-сервер Apache, язык PHP, СУБД MySQL;
- web-сервер MS IIS, ASP.NET, СУБД MS SQL Server.

Это не значит, например, что web-сервер MS IIS не может работать с PHP, или web-сервер Apache работает только с PHP, или что невозможно организовать взаимодействие скриптов PHP с SQL-сервером от Microsoft, однако внутри связки такое взаимодействие отработано и, как правило, не требует дополнительной настройки. Кроме того, предлагаются готовые пакеты, включающие все основные компоненты связки (и, возможно, некоторые дополнительные), обеспечивающие их интеграцию, а также предоставляющие интерфейс (оболочку) для управления. Наиболее известные пакеты для связки Apache-PHP-MySQL – XAMPP, Denwer, EasyPHP.

Альтернативой является разработка с использованием готовых систем управления контентом (CMS), таких как Drupal, Joomla!, WordPressMS, MS WebMatrix и др., не требующих знания языков программирования, что является их основным преимуществом. В то же время следует помнить, что каждая CMS система имеет свои ограничения, что может затруднить или сделать невозможным ее настройку под решение конкретной (в особенности, не типовой) задачи.

При выборе инструментов разработки web-приложения приходится сравнивать не просто отдельные языки программирования, но и технологии, обеспечивающие взаимодействие с базами данных, работу на стороне клиента и сервера, то есть всю связку, а также возможности технической поддержки, стоимость программного обеспечения и сопровождения, масштабы web-приложения, вопросы безопасности и т. д. Иногда требуется разработать приложение для конкретной платформы, в этом случае выбор технологии серверного программирования ограничен тем набором языков, которые поддерживаются web-сервером. Если стоит задача обеспечения переносимости между платформами, лучше остановить выбор на наиболее популярных серверных технологиях, поддерживаемых большинством платформ. К таким технологиям относится язык PHP, являющийся одним из лидеров среди языков разработки динамических web-сайтов благодаря своей простоте, скорости выполнения, богатой функциональности, кросс-платформенности и открытому исходному коду (Open Source), распространяющемуся под собственной лицензией.

Остановимся на связке Apache-PHP-MySQL и оболочке XAMPP [30, 35], предоставляющей визуальный интерфейс для управления компонентами связки (рис. 7), в том числе и в среде ОС Windows.

Полный пакет XAMPP содержит:

- web-сервер Apache с поддержкой SSL;
- СУБД MySQL;
- язык PHP;

- язык;
- FTP-сервер FileZilla;
- POP3/SMTP сервер;
- утилиту phpMyAdmin для визуального управления базами данных MySQL.

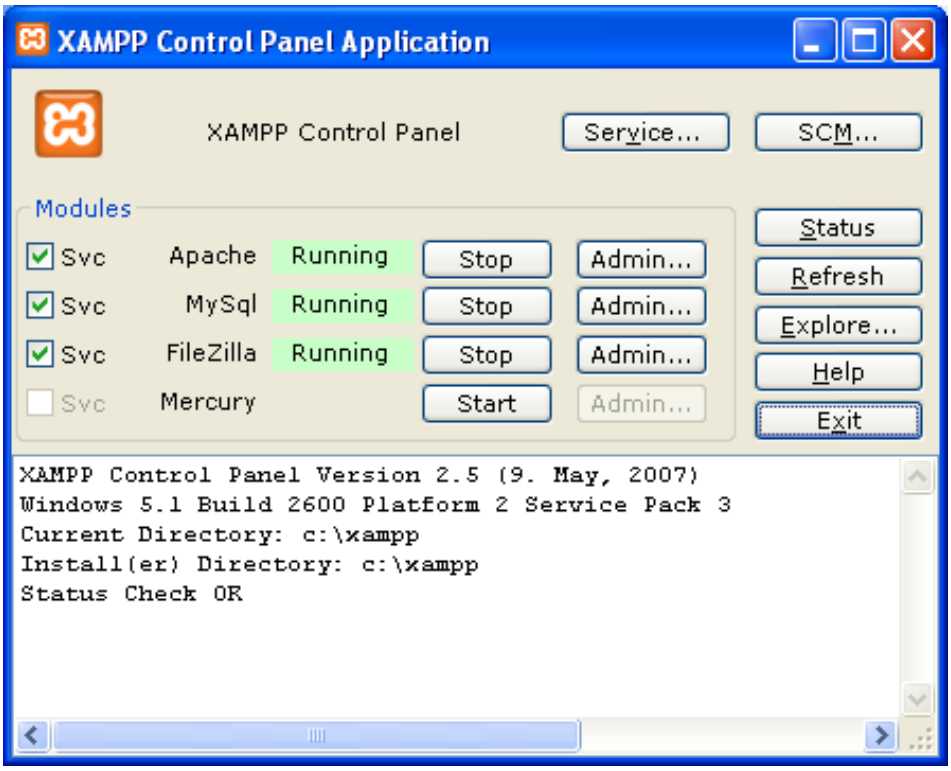


Рис. 7. Панель управления XAMPP

Для простоты работы некоторые возможности и настройки безопасности в XAMPP по умолчанию отключены, и в целом его рекомендуется использовать только в достаточно дружественном окружении, например, для отладки серверных программ в процессе разработки на локальном компьютере.

После запуска XAMPP, служб web-сервера (и MySQL) можно перейти в web-интерфейс XAMPP, набрав в адресной строке браузера: http://localhost или http://127.0.0.1. Web-интерфейс XAMPP содержит ссылки для отображения текущего статуса и настроек компонент XAMPP (*Status* – рис. 8, *phpinfo()*, *perlinfo()*), документацию и примеры, обеспечивает переход в оболочку phpMyAdmin, к почтовому и FTP-серверу, а также позволяет задавать настройки безопасности. Меню *Security* служит для просмотра статуса безопасности компонент XAMPP. Для задания пароля администратора СУБД MySQL и пароля папки XAMPP для ограничения доступа к ней из локальной сети следует перейти по ссылке http://localhost/security/xamppsecurity.php (рис. 9).

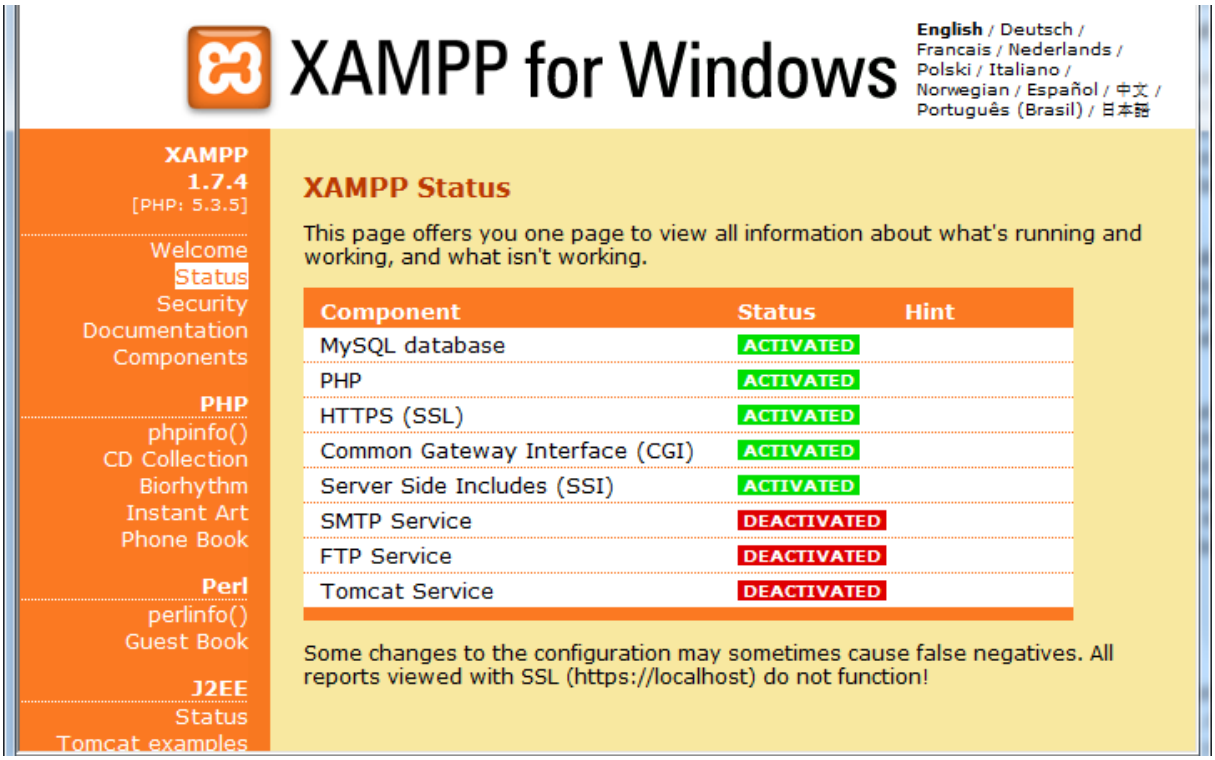


Рис. 8. Просмотр статуса компонент XAMPP

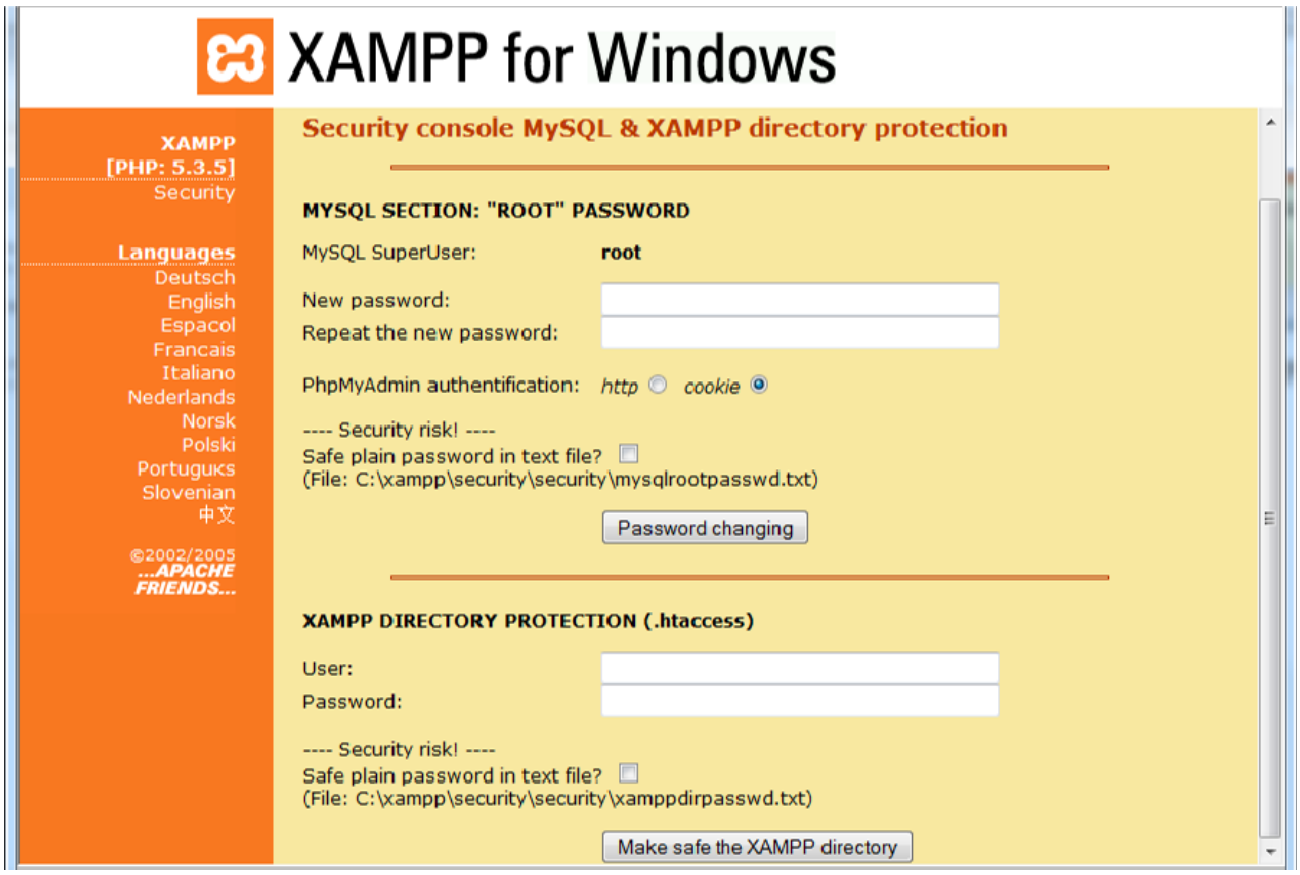


Рис. 9. Настройка параметров безопасности XAMPP

Для того чтобы создать новый сайт на локальном web-сервере, необходимо создать папку с названием сайта (например, site) в папке \xampp\htdocs\ и разместить в ней страницы сайта. Файл домашней страницы должен иметь имя index с расширением htm, html или php. Обращение к локальному web-сайту в этом случае будет иметь вид http://localhost/site.

Если необходимо хранить файлы web-сайта в другой папке, может быть создан виртуальный хост [30]. По умолчанию папка XAMPP создается в корне системного диска C:, назначения подпапок XAMPP и местонахождение некоторых важных файлов его компонент приведено в табл. 6.

Таблица 6

Назначения некоторых подпапок и файлов XAMPP

\xampp\htdocs	Папка для размещения локальных сайтов
\xampp\php	Папка транслятора PHP
\xampp\mysql\data	Папка баз данных MySQL
\xampp\cgi-bin	Папка для CGI-программ
\xampp\anonymous	Папка для анонимного доступа через FTP (пользователь anonymous)
\xampp\apache\logs\error.log	Лог-файл с описанием ошибок сервера Apache
\xampp\mysql\data\mysql.err	Лог-файл с описанием ошибок СУБД MySQL
\xampp\php\php.ini	Файл настроек языка PHP

Основы синтаксиса языка PHP

PHP – скриптовый язык программирования, созданный для генерации динамических страниц на web-сервере и работы с базами данных. PHP поддерживается подавляющим большинством хост-провайдеров. PHP является интерпретируемым языком и устанавливается с web-сервером Apache как расширение сервера. Совместная работа PHP-интерпретатора с web-сервером MS IIS осуществляется через CGI-интерфейс.

Сценарии PHP встраиваются непосредственно в web-страницу и интерпретируются на сервере перед отправкой страницы пользователю. Пользователь получит страницу с результатом выполнения сценария, но не увидит сам PHP-код.

Встраивание сценария в HTML/XHTML-код web-страницы производится с помощью специальных тегов:

```
<?php
...
?>
```

Существует сокращенная форма этого тега, которая доступна, если в файле настроек языка PHP (php.ini) включен параметр short_open_tag (short_open_tag = On):

```
<?
...
?>
```

Некоторые web-сервера пропускают через PHP-интерпретатор не только страницы с расширением php, а все страницы сайта (с расширениями htm и html). Интерпретация обычной web-страницы, не содержащей PHP-инструкций, никак не изменит ее внешний вид. Другие сервера интерпретируют только те страницы, которые имеют расширение php. Поэтому рекомендуется в обязательном порядке менять расширение web-страниц на php при добавлении в них PHP-кода. Следует отметить, что файл с расширением php не обязательно содержит обычный HTML/XHTML-код.

Например, если разместить в папке сайта (site) на сервере файл index.php с кодом:

```
<?php
echo ("Hello, World!");
?>
```

Тогда при обращении к сайту через сервер (например, по адресу http://localhost/site), браузер выведет страницу с приветствием (рис. 10).

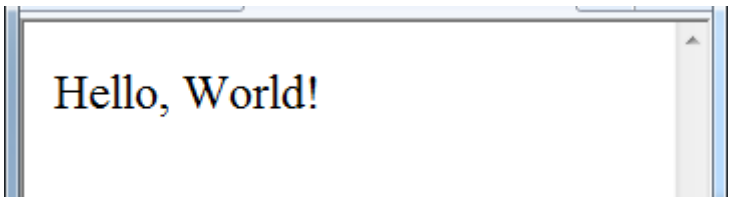


Рис. 10. Результат выполнения простого PHP-сценария

В то же время, если открыть страницу index.php в браузере без посредничества сервера, напрямую с диска, будет выдан текст PHP-сценария (рис. 11).

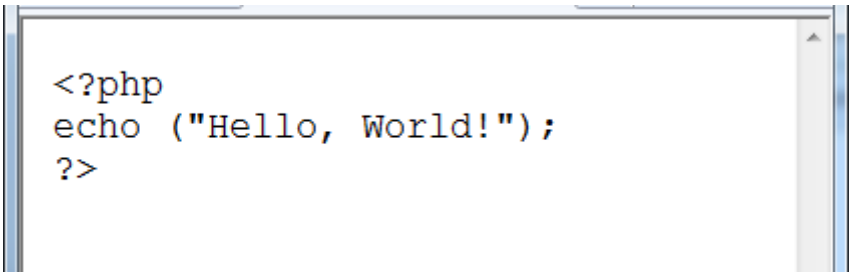


Рис. 11. Отображение PHP-сценария без посредничества сервера

В PHP-выдачу могут быть добавлены теги HTML для форматирования текста, например:

```
<?php
echo ("<i>Hello, World!</i>");
?>
```

После интерпретации сервером текст приветствия будет отображен курсивом.

Сценарий PHP может быть сохранен в отдельном файле с расширением php, а затем вызван из другого сценария функцией include(), например: include("db_info.php");.

Возможность подключения к сценарию внешних файлов обеспечивают также функции require(), include_once() и require_once(). Функции require() и include() отличаются лишь реакцией на невозможность получения запрошенного ресурса. В случае недоступности запрошенного ресурса функция include() выводит предупреждение и пытается продолжить исполнение сценария, а функция require() при недоступности ресурса останавливает обработку. Функцию require() или require_once() рекомендуется использовать, если подключается файл с определениями критически важных функций или переменных (например, описывающих подключение к базе данных), без которых сценарий не сможет функционировать.

При подключении сценариев с многоступенчатой вложенностью сценарий может подключиться повторно, что может привести к возникновению ошибок (например, повторное определение пользовательской функции). Исключить подобные ошибки позволяет использование include_once() или require_once(), которые проверяют, был ли сценарий уже подключен ранее.

Синтаксис языка PHP во многом схож с синтаксисом JavaScript.

Каждый оператор заканчивается знаком точки с запятой (;). Блоки операторов заключаются в фигурные скобки { }.

Функции определяются так же, как и в JavaScript, с помощью ключевого слова function; для возврата значения из функции используется ключевое слово return. Имена функций в PHP не чувствительны к регистру символов.

Комментарий предваряется знаками // или #. Можно также использовать многострочные комментарии в стиле языка Си, заключенные в символы /* ... */.

Константы объявляются с помощью функции define(), например: define("const_name","Зима"); или define("value1",751);. Проверка того, определена ли константа, производится с помощью функции defined(), принимающей логические значения (true, false). В выражениях константы указываются по имени без двойных кавычек.

Переменные предваряются знаком доллара ($), за которым следует идентификатор – последовательность букв, цифр и символов подчеркивания, начинающаяся с буквы. Так же как и в JavaScript, имена переменных в PHP чувствительны к регистру символов.

PHP не требует явного объявления переменных и указания их типа перед использованием, тип переменной определяется ее значением, при этом в процессе выполнения сценария PHP переменная может менять свой тип. Существует возможность явного определения/переопределения типа данных (integer, float, double, real, srting, boolean, array, object) с помощью функции settype; просмотр типа переменной осуществляется функцией gettype.

Пример:

```
<?php
$my_var="3";
echo("Исходный тип переменной ".gettype($my_var));
settype($my_var,"integer");
echo("<br>Новый тип переменной ".gettype($my_var));
?>
```

Результат выполнения данного сценария приведен на рис. 12. В примере = означает оператор присваивания, а знак точки (.) – операцию конкатенации строк.

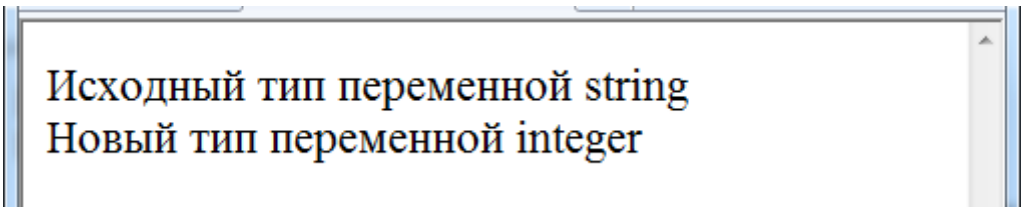


Рис. 12. Изменение типа переменной в PHP-сценарии

Переопределение типа переменной может производиться указанием нового типа в круглых скобках перед именем переменной, например:

```
$my_var=(int)$my_var;
```

Константы, как только они определены, всегда видимы глобально, то есть могут использоваться как внутри, так и вне функций.

Областью видимости *локальных* переменных является функция, внутри которой они определены. Время жизни локальной переменной определяется временем выполнения соответствующей функции или сценария. Локальная переменная при каждом вызове функции инициализируется заново. Объявление статической переменной позволяет сохранить ее значение между вызовами функции.

Глобальные переменные определены в сценарии, видны из любого места сценария, но не внутри функций. Доступ из функции к переменным сценария можно получить через массив $GLOBALS.

В следующем примере в теле функции change_my_var() через массив $GLOBALS идет обращение к переменной, определенной вне функции, то есть внешней по отношению к change_my_var().

```
<?php
function change_my_var()
{
$GLOBALS["my_var"]=5;
echo("Значение из функции: ".$GLOBALS["my_var"]);
}
$my_var=3;
echo("Исходное значение переменной: $my_var<br />");
change_my_var();
?>
```

Вывод этого сценария приведен на рис. 13.

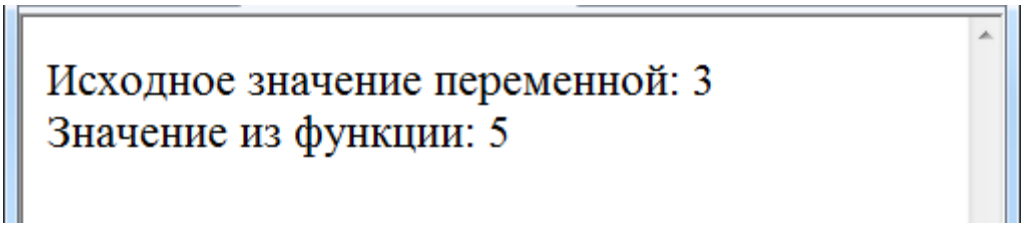


Рис. 13. Изменение значения внешней переменной из функции

Переменную внутри функции можно сделать глобальной, указав перед ней ключевое слово global, например: global $my_var;. Переменные, использованные внутри функции, которые объявлены как глобальные, ссылаются на глобальные переменные с теми же именами.

Когда функции передается аргумент, создается его локальная копия для хранения значения. Изменение этого значения касается только локальной копии переменной в функции, никак не отражаясь на источнике параметра. Однако можно определять *передачу параметров по ссылке*, изменение которых будет отражаться на исходной переменной. Для этого в описании функции перед именем параметра помещают символ амперсанда (&).

Внешние переменные, то есть полученные из окружения PHP-интерпретатора (от браузера или от сервера), являются *суперглобальными*, то есть доступны как любому сценарию, так и внутри любой функции.

В PHP при помощи ссылки можно создать две переменные, которые будут ссылаться на одно и то же содержимое. Для создания ссылки-переменной в правой части оператора присваивания перед именем переменной следует указать символ амперсанда (&), например: $var1=&$var2;

После этого изменение значения одной из переменных автоматически приведет к изменению значения другой. Следует отметить, что использование ссылок-переменных иногда существенно осложняет поиск ошибок в коде.

Оператор присваивания обозначается знаком =; как и в JavaScript, может быть использована сокращенная запись оператора присваивания: +=, -=, *= и т. д.

Арифметические операции: +, -, *, /, % (остаток целочисленного деления), ++ (инкремент), -- (декремент). Существует префиксная и постфиксная формы записи инкремента и декремента. В префиксной форме сначала производится инкремент/декремент, затем вычисляется выражение, в постфиксной – сначала вычисляется выражение, затем инкремент/декремент.

Конкатенация строк обозначается знаком точки (.).

Операторы сравнения: <, >, <=, >=, == (равенство), != (неравенство), === (идентичность), !== (неидентичность).

Логические операторы: and, && (логическое И), or, || (логическое ИЛИ), ! (отрицание НЕ), xor (исключающее ИЛИ). Для логического И, ИЛИ имеется два варианта обозначения, которые различаются приоритетом выполнения.

Побитовые операции: & (побитовое И), | (побитовое ИЛИ), ^ (побитовый XOR), ~ (побитовое НЕ), << (сдвиг влево на указанное число бит), >> (сдвиг вправо).

Приоритет выполнения операторов (в порядке убывания): ++, --, !, ~, - (унарный минус), преобразования типа, *, /, %, +, -, . (конкатенация), <<, >>, <, <=, >, >=, ==, !=, ===, !==, &, ^, |, &&, ||, ? : (условный оператор), = (присваивание), and, or, xor.

В PHP следующие управляющие конструкции: условный оператор, условная операция, оператор выбора, операторы цикла, операторы выхода из цикла.

Условная конструкция имеет вид:

```
if (логическое выражение)
{
операторы1
}
else
{
операторы2
}
```

Допустима сокращенная форма записи условного оператора без части else.

Конструкция if … else … может быть заменена условной операцией, имеющей следующий синтаксис:

выражение1 ? выражение2 : выражение3

Выражение1 – логическое выражение, истинность которого проверяется; результатом будет выражение2, если выражение1 истинно, и выражение3 – в противном случае.

Пример – вычисление абсолютного значения переменной $x:

```
$x<0 ?-$x:$x;
```

Другой пример – задание текстового значения переменной в зависимости от выполнения условия:

```
$ban=($login==true)?"Добро пожаловать!":"Зарегистрируйтесь!";
```

Условная конструкция с elseif позволяет проверять дополнительные условия, пока не будет найдено истинное или достигнут блок else. У каждой инструкции elseif есть собственный блок кода, размещаемый непосредственно после условного выражения инструкции elseif. Инструкция elseif идет после инструкции if и перед инструкцией else, если таковая имеется.

В общем случае синтаксис конструкции elseif имеет вид:

```
if (логическое выражение1) {
операторы1;
}
elseif (логическое выражение2) {
операторы2;
}
elseif (логическое выражение3) {
операторы3;
}
…
else {
операторыN;
}
```

Конструкция выбора switch полезна, если требуется выполнять различные действия в зависимости от различных значений переменной. Синтаксис конструкции switch:

```
switch(выражение) {
case значение1:
операторы1;
break;
case значение2:
```

```
операторы2;
break;
…
case значениеN:
операторыN;
break;
default: операторы по умолчанию;
break;
}
```

Конструкция switch имеет свои особенности. Если найдено соответствие текущего значения выражения (переменной) значению, определенному в каком-либо из блоков case, будет исполнен соответствующий блок операторов, а также все расположенные ниже блоки кода до конца инструкции switch или до ключевого слова break.

Если ни одного соответствия не найдено, исполняется блок default. Блок default не является обязательным, он может отсутствовать. Также не обязательно указание операторов break, которые осуществляют выход из конструкции выбора после нахождения совпадения.

Указание операторов break позволяет выполнять действия, предусмотренные только одной альтернативой. Если операторы break в конструкции выбора отсутствуют, будут выполнены действия, предусмотренные найденной и всеми последующими альтернативами.

Пример конструкции выбора без операторов break:

```
switch ($action) {
case "assemble":
echo ("Укомплектовать заказ<br />");
case "pack":
echo ("Упаковать заказ<br />");
case "send":
echo ("Отправить заказ<br />");
}
```

В данном примере, если переменная $action примет значение «assemble», сценарий выдаст три строки (рис. 14).

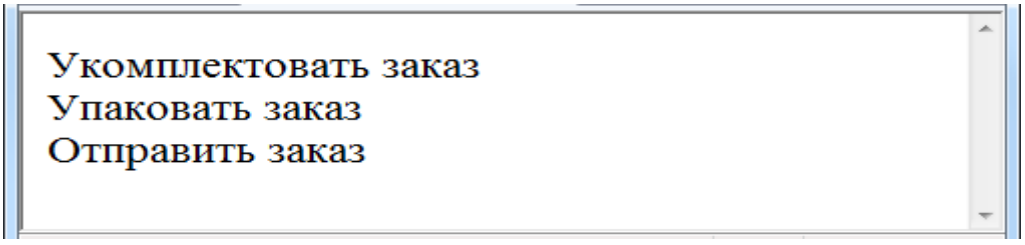


Рис. 14. Вывод конструкции выбора без операторов switch

Если же значением переменной $action оказалось "pack", будут выданы две строки:

Упаковать заказ

Отправить заказ

В то же время конструкция выбора с операторами break всегда выдавала бы одну строку, точно соответствующую значению "assemble" или значению "pack".

Для организации циклов в PHP используются конструкции while (цикл с предусловием), do ... while (цикл с постусловием), for и foreach.

Синтаксис цикла while:

```
while (условие выполнения цикла)
{
операторы
}
```

Синтаксис цикла do...while:

```
do
{
операторы
} while (условие выполнения цикла)
```

Оператор for определяет цикл с заданным числом повторений и имеет синтаксис:

```
for (выражение1;выражение2;выражение3)
{
операторы
}
```

Выражение1 – выражение инициализации счетчика цикла (например, $i=0); выражение2 – условие выполнения цикла (например, $i<5); выражение3 – выражение, задающее изменения счетчика (например, $i++).

Цикл foreach предназначен для работы с массивами.

Для выхода из цикла используется оператор break, для завершения текущей итерации и перехода к следующей – оператор continue.

Оператор exit прекращает выполнение оставшейся части сценария.

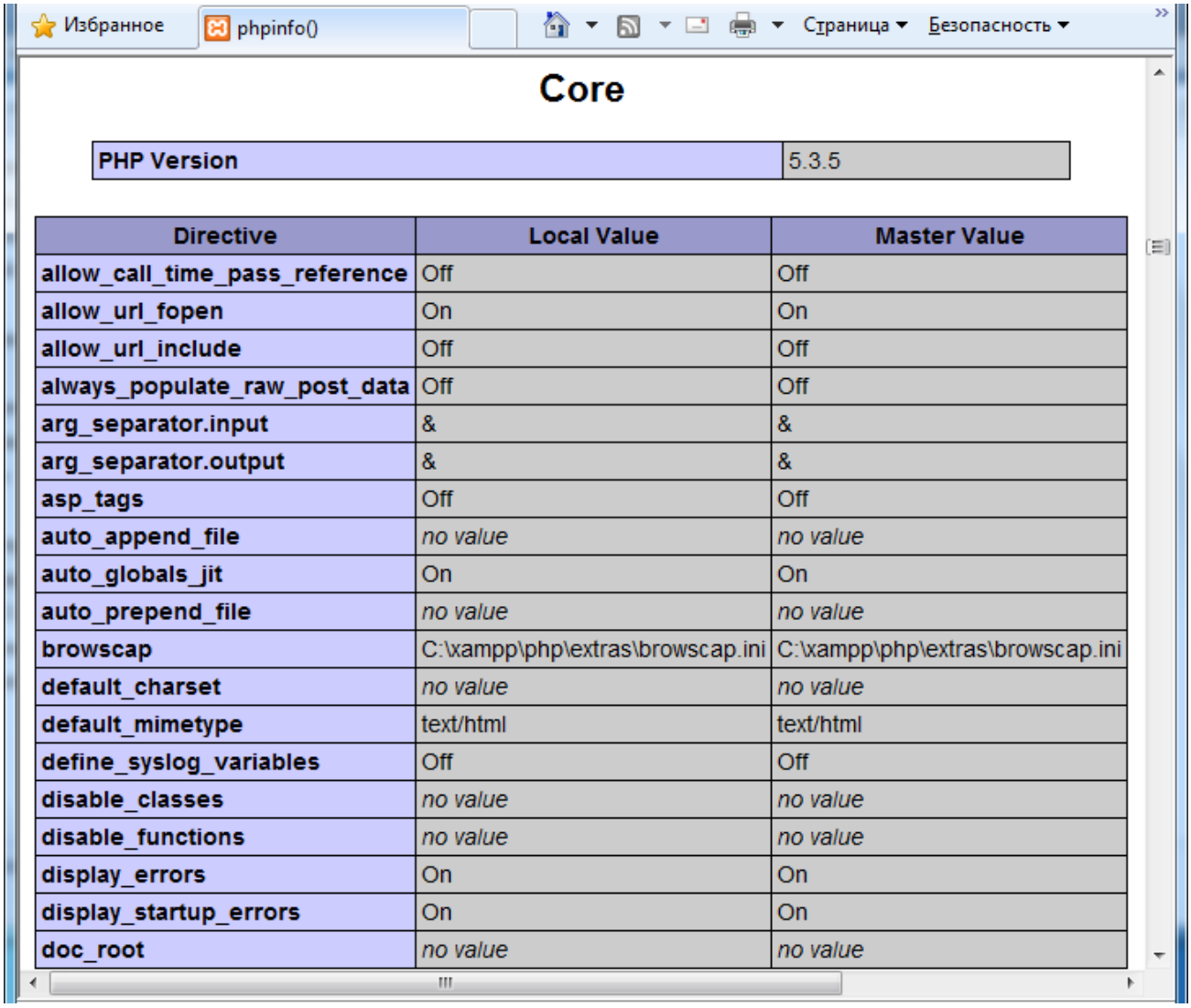
Управляющие конструкции в PHP поддерживают альтернативный синтаксис, в котором открывающая фигурная скобка ({) заменяется знаком двоеточия (:), а закрывающая фигурная скобка (}) – закрывающим ключевым словом (endif, endswitch, endwhile, endfor или endforeach в зависимости от управляющей конструкции).

Обработка данных web-форм

Для того чтобы серверный сценарий на языке PHP мог обрабатывать данные, введенные пользователем в поля web-формы, URL-адрес этого сценария должен быть указан в качестве значения параметра action тега формы form. Способ обработки данных формы зависит от метода отправки данных (get или post) и настроек PHP.

Просмотр текущих настроек PHP можно осуществить из PHP-сценария с помощью функции phpinfo() или с помощью одноименной команды оболочки XAMPP (рис. 15).

Важно, что значения параметров PHP чувствительны к регистру символов.



Core

PHP Version	5.3.5

Directive	Local Value	Master Value
allow_call_time_pass_reference	Off	Off
allow_url_fopen	On	On
allow_url_include	Off	Off
always_populate_raw_post_data	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
asp_tags	Off	Off
auto_append_file	*no value*	*no value*
auto_globals_jit	On	On
auto_prepend_file	*no value*	*no value*
browscap	C:\xampp\php\extras\browscap.ini	C:\xampp\php\extras\browscap.ini
default_charset	*no value*	*no value*
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off
disable_classes	*no value*	*no value*
disable_functions	*no value*	*no value*
display_errors	On	On
display_startup_errors	On	On
doc_root	*no value*	*no value*

Рис. 15. Просмотр настроек PHP командой phpinfo()

Если в файле настроек php.ini включен параметр register_globals (register_globals = On), то при отправке данных с помощью методов get и post автоматически создаются переменные с глобальной областью видимости. При этом имена переменных совпадают с именами передаваемых полей формы.

Пусть в корневом каталоге сайта размещены два файла: файл index.html с простой web-формой и файл сценария 1.php.

В теге формы обязательно должны быть определены метод отправки данных и сценарий, которому будут переданы данные формы:

```
<form name="form1" method="get" action="1.php">
Здравствуйте, как Вас зовут?<br />
<input type="text" name="text1" id="fio" /><br />
<br />
<input type="submit" value="Отправить данные" />
</form>
```

Тогда при включенном параметре register_globals после отправки формы сценарий 1.php имеет доступ к переменной $text1, содержащей введенный пользователем текст.

Пример текста сценария для обработки данных простой формы (файл 1.php):

```
<?php
echo ("<p><b>Добро пожаловать, <br /> $text1 </b></p>");
?>
```

В результате отправки формы пользователю будет выдана страница приветствия. На рис. 16 показан результат выдачи серверного сценария для случая, когда пользователь ввел в поле формы текст «гость».

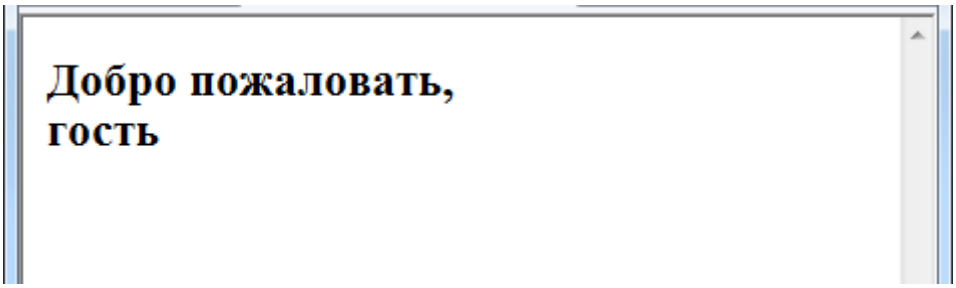


Рис. 16. Пример обработки данных формы серверным сценарием

Рассмотренный прямой доступ к автоматически создаваемым глобальным переменным очень удобен, однако возможны проблемы с безопасностью в связи с невозможностью проверки источника значения переменной перед его использованием. Если параметр register_globals включен, перед выполнением кода инициализируются различные внешние переменные, при этом переменные, определяемые разработчиком внутри скрипта, и передаваемые пользователем внешние данные могут перемешиваться. Опасность заключается в том, что PHP не требует предварительного объявления переменной, и это позволяет злоумышленнику вызвать PHP-сценарий с произвольными get- или post-параметрами, которые не предусмотрены формой.

Например, в описанном выше примере можно напрямую, минуя страницу с формой, вызвать результат выполнения серверного сценария (в предположении, что он расположен в папке site на локальном сервере), набрав в адресной строке браузера:

http://localhost/site/1.php?text1=test

В данном примере никаких проблем с безопасностью не возникнет, однако если имя переданного таким образом параметра совпадет с именем какой-то важной переменной (например, содержащей результат процедуры прохождения аутентификации), то злоумышленник сможет повлиять на функциональность web-приложения, добавив ложный параметр.

Решение данной проблемы заключается в инициализации важных переменных до начала их использования, а также извлечение значений с помощью методов, которые однозначно указывают на их происхождение, чтобы избежать непроверенных значений, которые поступают непосредственно от пользователя. Поэтому был предложен другой способ обработки внешних переменных.

В поздних версиях языка PHP (начиная с версии 4.1.0) параметр register_globals по умолчанию выключен (register_globals = Off), а доступ к значениям внешних переменных осуществляется через суперглобальные массивы:

- $_GET – массив переменных, переданных в сценарий из web-формы методом get;
- $_POST – массив переменных, переданных в сценарий из web-формы методом post;
- $_COOKIES – массив cookie-переменных;
- $_REQUEST – массив пользовательского ввода, включая содержимое массивов $_GET, $_POST и $_COOKIES. Суперглобальный массив $_REQUEST позволяет получать предоставляемые пользователем значения, не заботясь об их происхождении;
- $_SERVER – массив переменных среды сервера;
- $_FILES – массив переменных, связанных с загрузкой файлов;
- $_ENV – массив переменных окружения;
- $_SESSION – массив переменных сеанса;
- $_GLOBALS – массив всех глобальных переменных.

В предположении, что параметр register_globals в настройках PHP выключен, текст сценария 1.php из примера выше примет вид:

```
<?php
echo ("<p><b>Добро пожаловать, <br />". $_GET["text1"]. "</b></p>");
?>
```

Старый же сценарий перестанет работать (рис. 17), поскольку переменной $text1 теперь не существует.

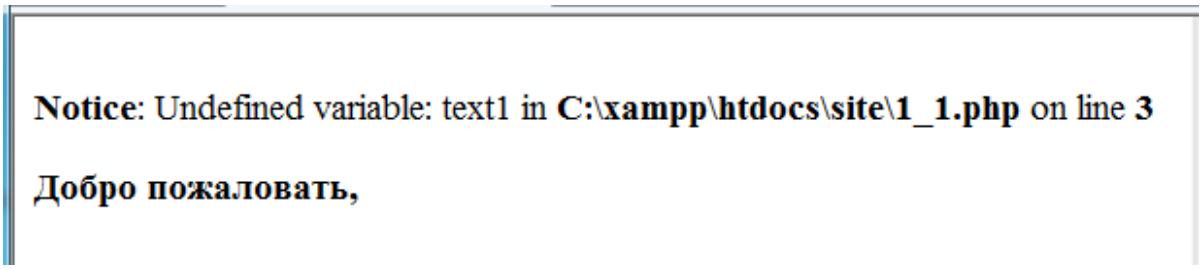


Рис. 17. Сообщение об ошибке после отключения параметра register_globals

Иногда возникает необходимость обеспечить работу старых сценариев, написанных в предположении включенного параметра register_globals, не жертвуя при этом уровнем безопасности. Такая задача может возникнуть, например, при использовании готовых PHP-сценариев, находящихся в открытом доступе в Интернете. Данную проблему можно решить, подключив сценарий с переопределением переменных через суперглобальные массивы. Например, к старому сценарию 1.php можно подключить сценарий old.php следующего содержимого:

```
<?php
$text1 = $_GET["text1"];
?>
```

Текст старого сценария 1.php, написанного в предположении включенного параметра register_globals, тогда может быть модифицирован следующим образом:

```
<?php
include("old.php");
echo ("<p><b>Добро пожаловать, <br /> $text1 </b></p>");
?>
```

Возможна также предварительная проверка существования переменной перед использованием логической функцией isset().

В дальнейшем в примерах серверных сценариев обработка данных форм производится через суперглобальные массивы в предположении выключенного параметра register_globals.

Даже при выключенном в целях повышения безопасности параметре register_globals сохраняется необходимость проверки корректности введенных пользователем данных.

Заполнение поля формы можно проверить с помощью логической функции empty(), которая возвращает значение 1 (true) в том случае, если переменной не существует или она имеет пустое значение.

Пример проверки заполнения поля text1 перед выдачей приветствия сценарием 1.php:

```
<?php
if (empty($_GET["text1"]))
echo ("<p><b>Добро пожаловать, <br /> неизвестный друг</b></p>");
else
echo ("<p><b>Добро пожаловать, <br />". $_GET["text1"]. "</b></p>");
?>
```

PHP поддерживает работу с регулярными выражениями, что позволяет проверять пользовательские данные на соответствие шаблону. Построение регулярных выражений подробно описано в предыдущем параграфе, посвященном работе с JavaScript.

В PHP есть набор функций с именами, начинающимися на preg_, которые осуществляют операции с регулярными выражениями. Эти функции принимают в качестве аргумента регулярное выражение в виде строки и выполняют операции над строками. Чаще всего используется функция preg_match(), которая выполняет поиск в строке всех совпадений по заданному регулярному выражению (шаблону) и возвращает значение true, если совпадение было найдено. Функция preg_match() может также выдавать массив найденных совпадений. Если совпадения не найдено, массив станется пустым.

В общем случае функция preg_match() имеет следующий синтаксис:

```
preg_match (шаблон, строка, массив совпадений)
```

Последний параметр (массив совпадений) является необязательным.

Рассмотрим пример использования шаблонов для проверки данных формы на стороне сервера. В примере используется страница с web-формой для ввода имени, пола и телефона пользователя. Данные формы передаются серверному сценарию с помощью метода post. Процедура проверки данных формы должна проверять заполнение текстовых полей для ввода имени (fio) и телефона (tel), а также соответствие введенного номера телефона заданному шаблону: (XXX)XXX-XX-XX, где X – цифры.

Шаблон для проверки номера телефона с помощью регулярных выражений формируется так же, как и для сценария JavaScript, и имеет вид: /^[(][0-9]{3}[)][0-9]{3}(-)[0-9]{2}(-)[0-9]{2}$/.

Тогда серверная процедура проверки может быть следующей:

```
<?php
$tel_num="/^[(][0-9]{3}[)][0-9]{3}(-)[0-9]{2}(-)[0-9]{2}$/";

if (empty($_POST["fio"]) || empty($_POST["tel"]))
echo ("<p>Заполнены не все поля</p>");
```

```
elseif (preg_match($tel_num, $_POST["tel"]))
echo ("<p>Спасибо, ваши данные успешно прошли проверку</p>");
//Здесь должна располагаться процедура дальнейшей
//обработки данных
else
echo("<p>Неправильный номер телефона!<br /> Введите
номер телефона в формате<br /> (XXX)XXX-XX-XX");
?>
```

Рассмотренный сценарий фактически не производит с пользовательскими данными никаких действий, кроме проверки. В зависимости от поставленной задачи процедура дальнейшей обработки данных может сохранять их во внешнем файле, либо заносить в базу данных, либо формировать новую web-страницу на основе пользовательских данных, например, с извлеченной из базы данных информацией.

Прежде чем рассмотреть реализацию возможных процедур обработки, следует еще поговорить о проверке полученных от пользователя данных в контексте обеспечения безопасности web-приложения. Речь идет об атаках межсайтового скриптинга (cross-site scripting, XSS) и SQL-инъекции.

Межсайтовый скриптинг предполагает использование злоумышленником web-приложения, для того чтобы передать исполняемый исходный код другому пользователю, использующему данное приложение. Эта атака становится возможной, если сценарий отображает пользовательские данные без их предварительной проверки. Злоумышленник может ввести в поле формы теги, код на языке JavaScript, другой исполняемый код (ActiveX (OLE), VBscript, Flash и т.п.).

Для предотвращения подобного типа атак старые версии интерпретатора PHP имели встроенный механизм «Волшебные кавычки» (Magic Quotes), обеспечивающий автоматическое экранирование входящих данных PHP-скрипта. Начиная с версии PHP 5.4.0 данный механизм не поддерживается. Проверить поддержку «Волшебных кавычек» можно с помощью логической функции get_magic_quotes_gpc(): true – механизм включен, false – механизм отключен или не поддерживается.

Экранирование специальных символов, таких как угловые скобки (<, >), двойные кавычки (") и др., путем их конвертирования в соответствующие мнемоники HTML (< , > , " и др.) осуществляется функцией htmlentities().

Существуют сотни различных вариантов атак, использующих XSS, включая атаки, которые не используют символы угловых скобок (<, >). Поэтому фильтрация не всегда эффективна, для определения легитимности введенных данных рекомендуется всегда сравнивать их с набором

разрешенных значений (например, с помощью шаблона на основе регулярных выражений).

SQL-инъекция – это прямое внедрение вредоносных инструкций в SQL-запросы с целью отображения скрытых данных, их изменения или выполнения вредоносного кода на сервере базы данных. Атака может быть реализована, если приложение строит SQL-запросы из пользовательского ввода и статических параметров.

Поэтому перед отправкой весь пользовательский ввода в базу данных следует пропускать его через функцию mysql_real_escape_string().

Другая возможная атака – подключение файла, определенного пользовательским значением без предварительной проверки (*php-include*). Речь идет о функциях include() или require(). Функция include() будет исполнять только ту часть файла, которая заключена между специальными тегами, обозначающими начало (<?php) и конец (?>) PHP-сценария. Если подключаемый файл имеет содержимое, не заключенное в эти теги, оно будет выдано в текстовом виде.

Когда подключаемый к сценарию файл определяется переменной, значение которой получено от пользователя, злоумышленник может ввести произвольное имя файла, например, файла, предназначенного для хранения паролей, и просмотреть его содержимое.

Помимо доступа к файлам, находящимся на диске сервера, функция include() также предоставляет возможность запускать скрипты, находящиеся на удаленных рабочих станциях.

Для предотвращения подобных атак рекомендуется жестко определять все допустимые альтернативы подключаемых файлов, например, с помощью структуры выбора switch, а также ограничивать доступ к папкам с важными файлами с помощью .htaccess.

Подробнее возможные атаки на web-приложения и способы обеспечения безопасности рассмотрены в [33, 39].

Организация взаимодействия с базой данных

Несмотря на то, что язык PHP имеет ряд функций для манипулирования внешними файлами, непосредственная работа с файлами из PHP-сценария всегда несет в себе угрозы безопасности, поэтому лучшим решением является сохранение информации в базе данных, а не в файлах.

База данных представляет собой структурированную совокупность данных. Система управления базами данных (СУБД) автоматизирует большую часть задач, связанных с хранением и извлечением пользовательской информации на основе заданных критериев. Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, используется язык структурированных запросов SQL (Structured Query Language).

Хотя PHP поддерживает работу с различными СУБД, обычно разработчики используют СУБД MySQL, имеющую взаимоувязанный с PHP программный интерфейс.

MySQL – реляционная СУБД, являющаяся, как и PHP, бесплатно распространяемой кроссплатформенной системой с открытым исходным кодом. MySQL отличается высокой скоростью работы, масштабируемостью, обеспечивает многопользовательский доступ и поддерживает механизм разграничения доступа.

Сервер MySQL входит в комплект установки XAMPP. Каждый сервер MySQL может содержать несколько баз данных, где группируются таблицы.

Создание пользователей базы данных и назначение им полномочий позволяет ограничить круг пользователей, обладающих правом доступа к таблицам на сервере. В MySQL существует три типа полномочий: полномочия обычных пользователей, полномочия администраторов и специальные полномочия. Полномочия обычных пользователей связаны с определенными командами SQL и правами на их выполнение. Административные полномочия включают доступ к базе данных MySQL, поскольку именно в ней хранятся учетные записи пользователей, пароли и т. п.

Если MySQL установлена на том же компьютере, что и web-сервер с PHP (например, в составе XAMPP), работа с СУБД по умолчанию осуществляется от имени пользователя root. Пользователю root можно задать пароль. Как правило, в целях безопасности доступ под именем root разрешается только с того компьютера, на котором стоит сервер MySQL. Однако посетителю сайта совсем не обязательно иметь доступ к MySQL для получения информации – за него это будет делать PHP-сценарий, который выполняется на компьютере с web-сервером. Серверный сценарий предоставляет пользователю не доступ к базе данных, а результат своей работы.

У MySQL есть собственный интерфейс для организации взаимодействия с клиентами. Один из способов взаимодействия основан на использовании командной строки MySQL (MySQL command line client). Для этих целей можно также использовать оболочку phpMyAdmin – инструмент работы с MySQL через web-интерфейс (рис. 18).

Запуск phpMyAdmin производится:

- через адресную строку браузера с помощью следующего URL: http://localhost/phpmyadmin;
- с помощью кнопки Admin в строке MySQL панели управления XAMPP;
- с помощью команды phpMyAdmin в группе Tools web-оболочки XAMPP.

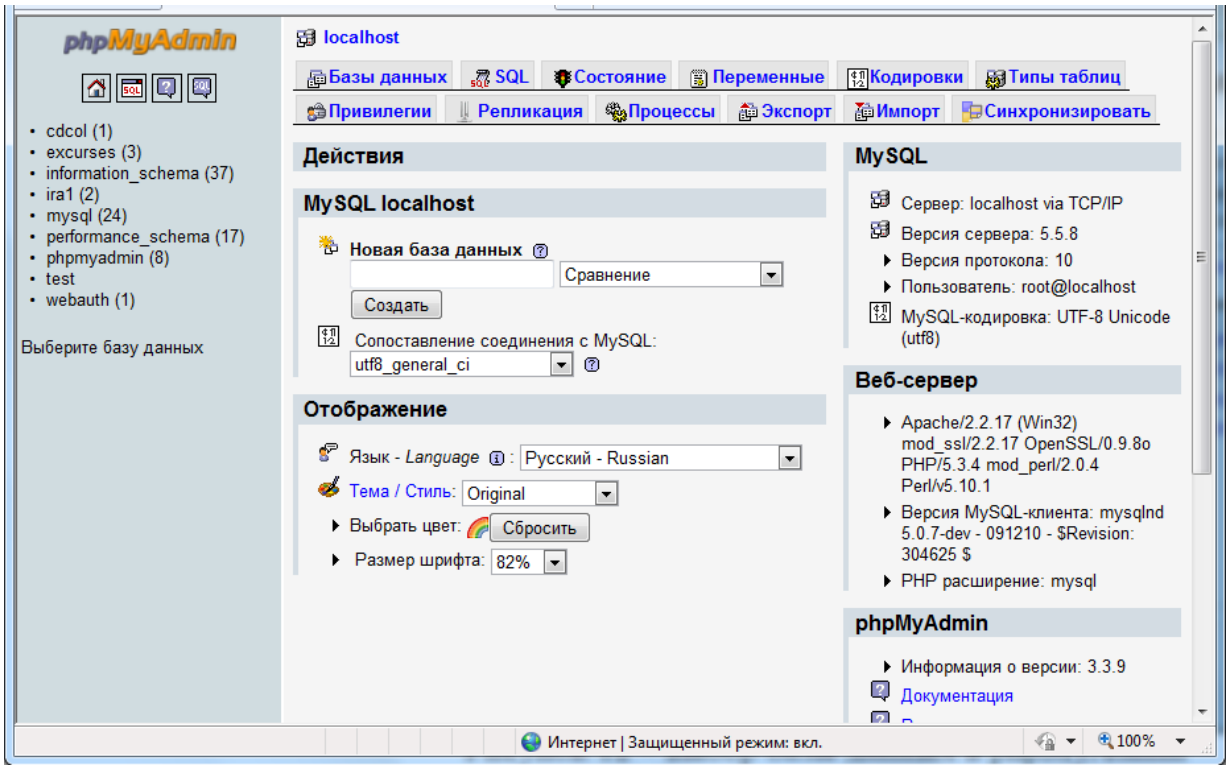


Рис. 18. Стартовое окно оболочки phpMyAdmin

Оболочка phpMyAdmin используется для редактирования и просмотра информации, хранимой в базах данных сервера, она позволяет осуществлять:

- создание новой или выбор существующей базы данных;
- действия с таблицами базы данных (создание, просмотр хранимых данных, удаление данных из таблицы, удаление таблицы и др.);
- формирование и выполнение собственных SQL-запросов (в виде SQL-инструкции или в визуальном режиме с помощью конструктора запросов);
- создание копии базы данных;
- экспорт и импорт структуры и информации базы данных (сохранение во внешних файлах);
- создание пользователей и назначение полномочий.

Оболочка phpMyAdmin имеет интуитивно понятный графический интерфейс и не вызовет затруднения при наличии опыта работы с любой другой визуальной оболочкой СУБД (например, MS Access). Все действия с базой данных в оболочке phpMyAdmin сопровождаются выводом соответствующих SQL-инструкций, что облегчает дальнейшее написание кода PHP-сценария, осуществляющего взаимодействие с базой данных.

MySQL поддерживает транзакции (механизм, который позволяет интерпретировать множественные изменения в базе данных как единую операцию), откат транзакций, а также предоставляет множество функций для работы со строками, датой и временем.

Работа PHP-сценария с информацией из базы данных предполагает выполнение определенной последовательности действий:

1. Подключение к серверу баз данных.
2. Выбор используемой базы данных.
3. Формирование запроса к базе данных (SQL-инструкции).
4. Выполнение запроса.
5. Вывод полученных результатов запроса.
6. Разрыв соединения с базой данных.

Язык PHP имеет встроенные функции для работы с базами данных MySQL (функции специфичны именно для этой СУБД).

Подключение к серверу баз данных (функция возвращает дескриптор соединения):

mysql_connect($адрес_SQL_сервера, $пользователь, $пароль).

Выбор базы данных для использования:

mysql_select_db($имя_базы, $дескриптор_соединения).

Выполнение запроса к базе данных (функция возвращает указатель на данные, полученные с помощью запроса):

mysql_query($строка_SQL_инструкции).

Получение данных запроса (функция формирует массив с данными полученными в результате запроса mysql_query):

mysql_fetch_array($указатель_на_данные, формат),

формат – необязательный параметр, указывает тип массива: MYSQL_NUM – массив с нумерованными индексами (по умолчанию), MYSQL_ASSOC – массив с индексами по имени столбцов таблицы.

Разрыв соединения с базой данных:

mysql_close($дескриптор_соединения).

Поскольку выполнение запросов к базе данных выполняется с помощью SQL-инструкций, приведем значение основных команд языка SQL (табл. 7).

Таблица 7

Основные команды SQL

Команда SQL	Назначение команды
CREATE DATABASE	Создание новой базы данных
SHOW DATABASES	Вывод списка существующих баз данных
USE DATABASE	Выбор базы данных
DROP DATABASE	Удаление базы данных
SHOW TABLES	Вывод списка существующих в базе данных таблиц
CREATE TABLE	Создание таблицы
DESCRIBE	Вывод характеристик полей таблицы

Окончание табл. 7

Команда SQL	Назначение команды
DROP TABLE	Удаление таблицы
INSERT INTO	Ввод данных в таблицу
SELECT	Вывод данных (запросы к базе данных, выборка)
UPDATE	Изменение данных, хранящихся в таблице
DELETE FROM	Удаление строк (записей) таблицы
ALTER TABLE	Операции со столбцами (полями) таблицы

Рассмотрим пример. Пусть в MySQL создана база данных data, хранящаяся на локальном сервере localhost. База данных data содержит одну таблицу users_data, предназначенную для хранения информации о пользователях (имя, пол, номер телефона). Структура таблицы users_data приведена на рис. 19.

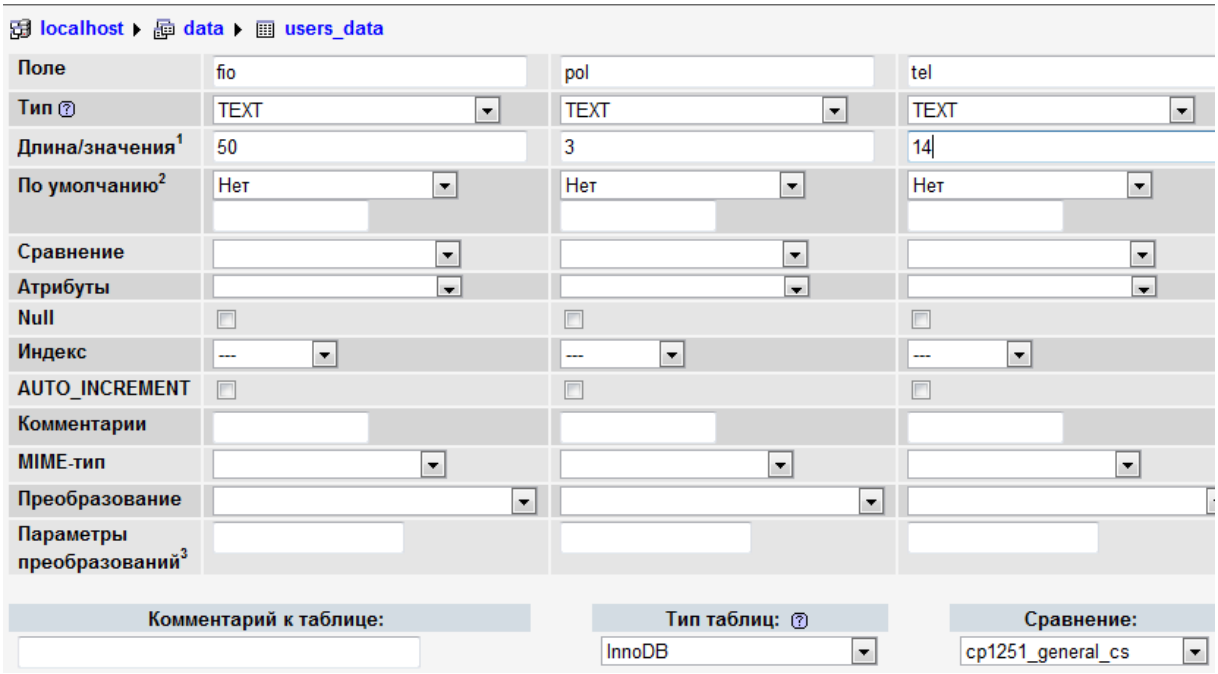
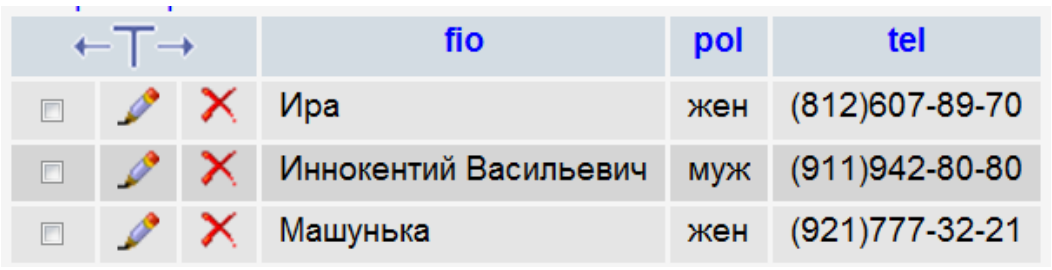


Рис. 19. Структура таблицы users_data

Таблица users_data содержит данные, представленные на рис. 20.



	fio	pol	tel
	Ира	жен	(812)607-89-70
	Иннокентий Васильевич	муж	(911)942-80-80
	Машунька	жен	(921)777-32-21

Рис. 20. Данные таблицы users_data

Сценарий db_connect.php осуществляет подключение к серверу и выбор базы данных:

```
<?php
// Информация о базе данных
$db_host="localhost";
$db_database="data";
$db_username="root";
$db_password="";
// Подключение к базе данных
$connection = mysql_connect($db_host, $db_username, $db_password);
// Указание кодировки соединения
mysql_query("set names cp1251");
// Выбор базы данных
$db_select = mysql_select_db($db_database);
?>
```

В процессе подключения или выбора базы данных могут возникнуть ошибки, связанные с неправильным указанием параметров базы данных, либо отсутствием необходимых объектов. В этом случае дальнейшая работа с базой данных станет невозможной. Поэтому на период отладки следует добавить в сценарий подключения обработку ошибок, например:

```
<?php
// Информация о базе данных
$db_host="localhost";
$db_database="data";
$db_username="root";
$db_password="";
// Подключение к базе данных
$connection = mysql_connect($db_host, $db_username, $db_password);
//Обработка ошибок подключения
if (!$connection) {
echo("<p>Невозможно подключиться к базе данных: <br />".
mysql_error()."</p>");
exit();
}
// Указание кодировки соединения
mysql_query("set names cp1251");
// Выбор базы данных
$db_select = mysql_select_db($db_database);
```

```
// Обработка ошибок выбора базы данных
if (!$db_select) {
echo("<p>Невозможно выбрать базу данных: <br />". mysql_error()."</p>");
exit();
}
?>
```

Выдача информации о причине ошибки работы с базой данных MySQL производится функцией mysql_error().

Вместо пары функций echo() и exit() можно выдать сообщение об ошибке непосредственно из функции exit() или die(), которая остановит исполнение сценария.

После того, как соединение с базой данных установлено, можно приступить к исполнению SQL-запросов. SQL-инструкцию можно сформировать в phpMyAdmin, а затем скопировать. Сценарий show_info.php выводит все данные из таблицы users_data, а затем закрывает соединение с базой данных. Закрытие соединения освобождает все занимаемые им ресурсы и память. В сценарий добавлена проверка выполнения запроса к базе:

```
<?php
// Подключение к базе данных
include("db_connect.php");
// Запись в переменную текста запроса
$query="SELECT * FROM `users_data`";
// Выполнение запроса к базе данных
$result=mysql_query($query);
//Проверка выполнения запроса
if (!$result)
die("<p>Невозможно выполнить запрос: <br />" . mysql_error(). "</p>");
else
// Так как запрос выдает несколько строк,
//для вывода используется цикл
while($showdata=mysql_fetch_row($result, MYSQL_ASSOC))
echo("<p><b>" . $showdata["fio"] . "</b> ". $showdata["pol"] . " ". $show-
data["tel"]. "</p>");
//Разрыв соединения
if(!mysql_close($connection))
echo("<p>Невозможно разорвать соединение с базой
данных</p>");
?>
```

Результат выполнения этого сценария приведен на рис. 21.

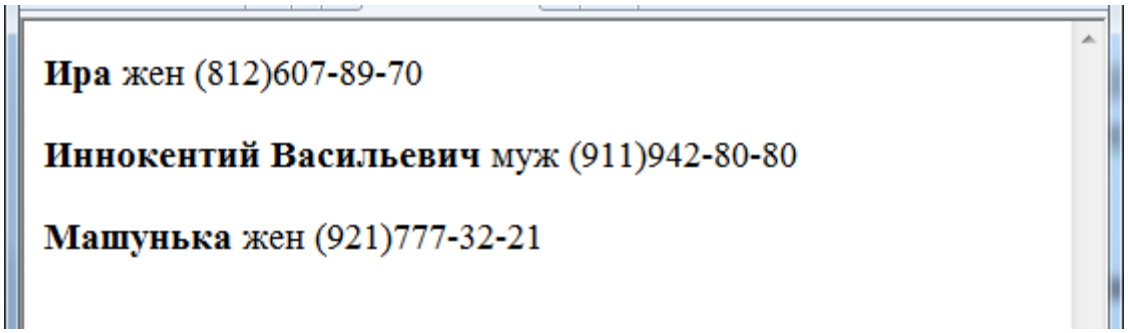


Рис. 21. Вывод PHP-сценарием информации из базы данных

Варианты вывода из базы данных (состав полей, порядок отображения данных, отбор по условию и т. п.) различаются лишь текстом SQL-инструкции.

Другая задача – занесение получаемых из web-формы пользовательских данных в базу. Пусть заполнение данных пользователем производится с помощью web-формы, поля формы имеют имена fio, pol и tel, данные формы передаются серверному сценарию с помощью метода post. Пусть также серверный сценарий, обрабатывающий данные формы (рассмотрен в предыдущем пункте, посвященном проверке данных web-форм) после выполнения всех необходимых проверок вызывает сценарий safe_info.php, который должен сохранить полученную информацию в таблице users_data.

Текст сценария safe_info.php может быть следующим:

```
<?php
// Подключение к базе данных
include("db_connect.php");
//Запись в переменную текста запроса на добавление записи
$sql = "INSERT INTO `data`.`users_data` (`fio`, `pol`, `tel`) VALUES ('".
$_POST["fio"]."', '".$_POST["pol"]."', '".$_POST["tel"]."');";
// Выполнение запроса к базе данных
$result=mysql_query($sql);
//Проверка выполнения запроса
if (!$result)
die("<p>Невозможно выполнить запрос: <br />". mysql_error()."</p>");
else
echo("<p>Данные успешно сохранены</p>");
//Разрыв соединения
if(!mysql_close($connection))
echo("<p>Невозможно разорвать соединение с базой данных</p>");
?>
```

Работа с базами данных может осуществляться с помощью расширения PDO (PHP Data Objects) – библиотеки классов, предоставляющей универсальный интерфейс доступа к различным базам данных [3, 9, 25]. PDO входит в состав PHP, начиная с версии 5.1, в более ранних версиях PDO не работает.

PDO позволяет использовать унифицированные методы для работы с различными базами данных, хотя текст запросов может немного отличаться, так как многие СУБД реализуют свой диалект SQL.

PDO не использует абстрактных слоев для подключения к базам данных, а использует драйверы самих баз данных (рис. 22), что позволяет добиться высокой производительности.

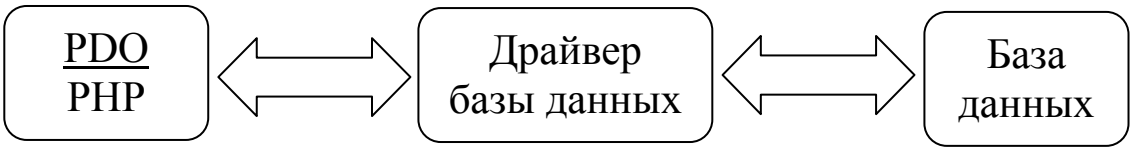


Рис. 22. Взаимодействие PHP-сценария с базой данных
с помощью расширения PDO

Расширение поддерживает любую базу данных, для которой есть PDO драйвер. В настоящее время для PDO существуют драйверы практически ко всем общеизвестным СУБД и интерфейсам (MySQL, MS SQL Server, PostgreSQL, ODBC, Oracle Call Interface и др.).

Просмотр списка доступных в системе драйверов осуществляется командой print_r(PDO::getAvailableDrivers());.

Универсальные функции PDO используют ту же основную информацию, что и PHP-функции работы с MySQL, но при вызове дополнительно указывается тип базы данных, с которой осуществляется взаимодействие. Далее примеры команд приведены для СУБД MySQL.

Подключение к базе данных с помощью PDO имеет вид:

```
$connection = new
PDO("mysql:host=$db_host;dbname=$db_database", $db_username,
$db_password);
```

Рекомендуется использовать механизм исключений из библиотеки PDO (PDOException), чтобы в случае неудачного подключения к базе данных получить сообщение об ошибке. Пример обработки исключений с помощью конструкции try…catch:

```
try {
$connection = new
PDO("mysql:host=$db_host;dbname=$db_database", $db_username,
$db_password);
```

```
//Установка режима обработки ошибок
$connection->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
}
catch(PDOException $e) {
echo $e->getMessage();
}
```

В режиме PDO::ERRMODE_EXCEPTION необработанные ошибки будут вызывать исключения и остановку выполнения сценария. В этом примере используются операции работы с объектами: new – создание нового экземпляра объекта, -> – вызов свойства или метода объекта.

Сценарий db_connect.php для подключения базы данных data с использованием PDO примет вид:

```
<?php
// Информация о базе данных
$db_host="localhost";
$db_database="data";
$db_username="root";
$db_password="";
// Подключение к базе данных
try {
$connection = new
PDO("mysql:host=$db_host;dbname=$db_database",
$db_username,$db_password);
//Установка режима обработки ошибок
$connection->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
// Задание кодировки
$connection->query("SET NAMES cp1251");
}
//Обработка исключений
catch(PDOException $e) {
die($e->getMessage());
}
?>
```

Для исполнения SQL-инструкции используется метод query() дескриптора соединения. Пример:

```
$query="SELECT * FROM `users_data`";
$result=$connection->query($query);
```

Для вывода информации, полученной из базы данных, используется метод fetch() дескриптора выполнения запроса (указателя на данные). Перед использованием метода можно указать, в каком виде будут возвращены данные, например, PDO::FETCH_ASSOC – массив, индексированный по именам столбцов, PDO::FETCH_BOTH – массив, индексированный по именам столбцов и по номерам (по умолчанию), PDO::FETCH_NUM – массив, индексированный по номерам столбцов, и т. д.

Пример установки метода извлечения данных:

```
$result->setFetchMode(PDO::FETCH_ASSOC);
```

Пример вывода данных (используется формат вывода, установленный по умолчанию):

```
while($showdata=$result->fetch())
echo("<p><b>".$showdata["fio"]."</b>  ".$showdata["pol"]."  ". $showdata["tel"]."</p>");
```

Разрыв соединения производится назначением дескриптору соединения значения NULL, например: $connection = null;.

Сценарий show_info.php для вывода информации из базы данных с использованием PDO примет вид:

```
<?php
// Подключение к базе данных
include("db_connect.php");
// Запись текста запроса в переменную
$query="SELECT * FROM `users_data`";
try {
// Выполнение запроса к базе данных
$result=$connection->query($query);
// Так как запрос выдает несколько строк, используется цикл
while($showdata=$result->fetch())
echo("<p><b>".$showdata["fio"]."</b>  ".$showdata["pol"]. "  ". $showdata["tel"]."</p>");
// Разрыв соединения
$connection=null;
}
//Обработка исключений
catch(PDOException $e) {
die($e->getMessage());
}
?>
```

Вставка новых данных (INSERT INTO) или обновление существующих (UPDATE) – наиболее часто используемые общие операции баз данных.

При использовании PDO для реализации этих операций можно использовать метод exec(), например:

```
$sql = "INSERT INTO `data`.`users_data` (`fio`, `pol`, `tel`) VALUES
('".$_POST["fio"]."', '".$_POST["pol"]."', '" . $_POST["tel"]."');";
$result=$connection->exec($sql);
```

Однако в целях безопасности рекомендуется использовать подготовленные выражения, что разбивает каждую из операций (изменение, вставка данных) на два этапа (рис. 23) с использованием методов prepare() и execute().

Рис. 23. Схема выполнения обновления и добавления данных
с использованием подготовленных выражений

Подготовленные выражения – это предварительно скомпилированные инструкции SQL, которые могут быть выполнены много раз с пересылкой на сервер только данных. Подготовленные выражения имеют большую скорость выполнения, а также позволяют четко разделить структуру и входные данные запроса. Кроме того, их использование обеспечивает защиту от атак типа SQL-инъекция, так как все передаваемые в них параметры автоматически экранируются.

Если подготовленные выражения не используются для экранирования специальных символов в пользовательских данных, можно использовать метод дескриптора соединения quote(), например, $fio=$connection->quote($_POST["fio"]);.

Работа с подготовленными выражениями выполняется следующим образом. Сначала следует сформировать шаблон инструкции SQL, который будет затем скомпилирован.

Пример *неименованного* шаблона:

```
$sql = "INSERT INTO `data`.`users_data` (`fio`, `pol`, `tel`) VALUES (?, ?, ?);";
```

Пример *именованного* шаблона:

```
$sql = "INSERT INTO `data`.`users_data` (`fio`, `pol`, `tel`) VALUES (:fio, :pol, :tel);";
```

Подготовка выражения осуществляется с помощью метода prepare() дескриптора соединения, например:

```
$result = $connection->prepare($sql);
```

Затем следует определить данные, которые будут подставлены в шаблон. Легче всего это сделать через объявление массива. Выполнение шаблона запроса, заполненного данными, осуществляется методом execute().

Пример объединения с данными и выполнения запроса с неименованным шаблоном:

```
$data = array($_POST["fio"], $_POST["pol"], $_POST["tel"]);
$result->execute($data);
```

Пример объединения с данными и выполнения запроса с именованным шаблоном:

```
$data = array("fio" => $_POST["fio"], "pol" => $_POST["pol"], "tel" =>
$_POST["tel"]);
$result->execute($data);
```

Остановимся на варианте использования неименованного шаблона. Тогда сценарий safe_info.php, выполняющий сохранение пользовательского ввода в базе данных, с использованием PDO и подготовленных выражений примет вид:

```
<?php
// Подключение к базе данных
include("db_connect.php");
// Запись в переменную шаблона запроса
$sql = "INSERT INTO `data`.`users_data` (`fio`, `pol`, `tel`) VALUES (?, ?,
?);";
try {
// Подготовка выражения
$result = $connection->prepare($sql);
// Подготовка данных для вставки в шаблон
$data = array($_POST["fio"], $_POST["pol"], $_POST["tel"]);
// Выполнение запроса к базе данных
$result->execute($data);
echo ("Данные успешно сохранены");
// Разрыв соединения
$connection=null;
}
//Обработка исключений
catch(PDOException $e) {
die($e->getMessage());
}
?>
```

Итак, были подробно рассмотрены методы работы с базами данных, как специфичные для СУБД MySQL, так и универсальные, позволяющие организовать взаимодействие с различными базами данных.

Вместе с тем, за пределами данного учебного пособия остались такие возможности языка PHP, как управление внешними файлами, загрузка файлов на сервер, работа с маркерами cookie, авторизация пользователей и управление сеансами (сессиями), а также вопросы обеспечения безопасности, связанные с решением этих задач.

Однако основы языка PHP в рассмотренном объеме позволяют понять наиболее важные концепции и принципы серверного программирования.

ЗАКЛЮЧЕНИЕ

Современные web-технологии дают в руки разработчика мощный инструмент, позволяющий создавать полнофункциональные приложения, доступные как в глобальной сети Интернет, так и в корпоративной Интранет-среде. Вместе с тем острой проблемой остается обеспечение безопасности web-приложений. При выборе тех или иных инструментов web-разработки необходимо четко представлять функциональные возможности, предоставляемые используемой технологией, ее особенности, возможные проблемы и способы обеспечения информационной безопасности. Поэтому крайне важным является освоение основ разработки web-страниц и web-программирования.

Данное пособие знакомит читателя с принципами размещения и передачи информации в Интернете, концепциями и возможностями наиболее популярных технологий современной web-разработки, подходами к оптимизации web-сайтов, а также базовыми элементами обеспечения информационной безопасности web-приложений.

Задачей пособия является формирование системного представления об использовании современных web-технологий и подготовка к восприятию информации о web-уязвимостях и методах защиты информации в Интернете. Более глубокое изучение рассмотренных тем потребует привлечения дополнительных литературных источников, в том числе учебной литературы и специальных справочных изданий, в том числе стандартов, рекомендаций и спецификаций W3C.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Акимов С. В*. Введение в Интернет-технологии. [электронный ресурс]. URL: http://www.structuralist.narod.ru/it/internet/ internet.htm (дата обращения: 21.05.2012).
2. Алгоритм работы поисковой системы. Что учитывает и анализирует поисковая система. [электронный ресурс]. URL: http://www.antula.ru/ algoritm.htm (дата обращения: 21.03.2012).
3. *Андреев Д*. PHP PDO. Краткое руководство. 27.04.2011. [электронный ресурс]. URL: http://dandreev.com/blog/php-pdo-kratkoe-rukovodstvo/ (дата обращения: 12.06.2012).
4. Архитектура сайтов. // SEO бесплатный дистанционный курс. [электронный ресурс]. URL: http://seo.abronova.com/ 62.php (дата обращения: 21.03.2012).
5. *Веллинг Л*., *Томсон Л*. Разработка Web-приложений с помощью PHP и MySQL. – М.: Изд. дом «Вильямс», 2008. – 880 с.
6. Виновникам утечки «секретных» документов в «Яндекс» и Google ничего не угрожает // CNews, 27.07.11. [электронный ресурс]. URL: http://safe.cnews.ru/news/top/index.shtml?2011/07/ 27/448914 (дата обращения: 12.09.2011).
7. *Выскорко М. С*. Регулярные выражения в Javascript. 24.11.2006. [электронный ресурс]. URL: http://www.opennet.ru/ base/dev/ pcre_javascript.txt.html (дата обращения: 29.05.2012).
8. *Гарретт Д. Д*. Ajax: Новый подход к веб-приложениям /пер. А. Наконечного [электронный ресурс]. URL: http://www.codenet.ru/webmast/ js/ajax/AJAX-New.php (дата обращения: 20.02.2012).
9. Доступ к базе данных в PHP /пер. С. Фастунова. 04.06.2012. // ruseller.com, Уроки. [электронный ресурс]. URL: http://ruseller.com/ lessons.php?rub=37&id=1463 (дата обращения: 12.06.2012).
10. *Дэвис Е. М*., *Филлипс Дж. А*. Изучаем PHP и MySQL / пер. с англ. – СПб.: Символ-Плюс, 2008. – 448 с.
11. *Евдокимов Н. В*., *Лебединский И. В*. Раскрутка веб-сайта: практическое руководство по SEO 3.0. – М.: Изд-во «Вильямс», 2011. – 288 с.
12. *Ерижоков А. А*. SSI. DH's Linux Site. 2000. [электронный ресурс]. URL: http://dh.opennet.ru/main.html (дата обращения: 20.02.2012).

13. Интернет. История. [электронный ресурс]. URL: http://www.sunhome.ru/journal/12971 (дата обращения: 14.02.2012).

14. *Кан М*. Основы программирования на JavaScript. 2006 [электронный ресурс]. URL: http://www.intuit.ru/department/ internet/jsbasics/ (дата обращения: 29.04.2012).

15. *Кантор И*. DOM: работа с HTML-страницей. [электронный ресурс]. URL: http://javascript.ru/tutorial/dom (дата обращения: 17.05.2012).

16. *Кантор И*. Введение. DOM в примерах. // DOM: работа с HTML-страницей. [электронный ресурс]. URL: http://javascript.ru/tutorial/dom/ intro (дата обращения: 17.05.2012).

17. *Кантор И*. Введение в Ajax. [электронный ресурс]. URL: http://javascript.ru/ajax/intro (дата обращения: 29.02.2012).

18. *Кент П*. Поисковая оптимизация для чайников, 4-е изд. = Search Engine Optimization For Dummies, 4th Edition. – М.: Изд-во «Вильямс», 2011. – 432 с.

19. *Кузнецов М*., *Симдянов И*. PHP 5/6. – СПб.: БХВ-Петербург, 2010. – 1024 с.

20. Метатег keywords – правила составления и частые ошибки. 10.05.2009. [электронный ресурс]. URL: http://seointelligent.ru/ seo-secrets/teg-keywords-oshibki-i-pravila-sostavleniya/ (дата обращения: 25.03.2012).

21. Настройка веб-сервера Apache через Htaccess. [электронный ресурс]. URL: http://www.htaccess.net.ru/ (дата обращения: 28.03.2012).

22. *Олифер В*., *Олифер Н*. Введение в IP-сети: [электронный ресурс]. URL: http://citforum.ru/nets/ip/contents.shtml (дата обращения: 14.02.2012).

23. Оптимизация сайта. Поисковая оптимизация сайта. SEO. [электронный ресурс]. URL: http://webstudio2u.net/ru/site-optimization.html (дата обращения: 21.03.2012).

24. Поисковые системы и алгоритмы // SEO бесплатный дистанционный курс. [электронный ресурс]. URL: http://seo.abronova.com/22.php (дата обращения: 21.03.2012).

25. Почему следует использовать PDO для доступа к базам данных? /пер. С. Фастунова. 23.06.2010. // ruseller.com, Уроки. [электронный ре-

сурс]. URL: http://ruseller.com/ lessons.php?rub=28&id=610 (дата обращения: 12.06.2012).

26. *Савельев А*. Технология, которая перевернет веб. // Компьютерра. 16.06.2005. [электронный ресурс]. URL: http://www.computerra.ru/ hitech/39239/ (дата обращения: 29.02.2012).

27. *Севостьянов И. О*. Поисковая оптимизация: практическое руководство по продвижению сайта в Интернете. – СПб.: Питер, 2010. – 240 с.

28. Стандарт ECMA-262, 3-я редакция. [электронный ресурс]. URL: http://javascript.ru/ecma (дата обращения: 03.05.2012).

29. *Ташков П.А*. Веб-мастеринг на 100%: HTML, CSS, JavaScript, PHP, CMS, AJAX, раскрутка. – СПб.: Питер, 2010. – 512 с.

30. Установка и настройка сервера XAMPP на Windows. [электронный ресурс]. URL: http://makegood.ru/tools/8/ (дата обращения: 29.05.2012).

31. *Храмцов П. Б*., *Брик С. А*., *Русак А. М*., *Сурин А. И*. Введение в JavaScript. 2003. [электронный ресурс]. URL: http://www.intuit.ru/ department/internet/js/ (дата обращения: 29.04.2012).

32. Цифровой потоп: почему утекают персональные данные в Интернете // РБК. – 16.08.2011. [электронный ресурс]. URL: http://top.rbc.ru/ economics/16/08/2011/610694.shtml (дата обращения: 23.08.2011).

33. *Шейда В. В*. Защита информации в компьютерных сетях. Web уязвимости: учебно-методическое пособие. – Томск: Томский гос. ун-т систем управления и радиоэлектроники, 2007. – 68 с.

34. Эффекты перехода между страницами в HTML. [электронный ресурс]. URL: http://perkoka.ru/article/10/664.html (дата обращения: 21.03.2012).

35. Apache friends – XAMPP. [электронный ресурс]. URL: http://www.apachefriends.org/ru/xampp.html (дата обращения: 29.05.2012).

36. Favicon // Википедия. Свободная энциклопедия. [электронный ресурс]. http://ru.wikipedia.org/wiki/Favicon (дата обращения: 21.03.2012).

37. *Harish Kamath*. Регулярные выражения в JavaScript. 07.10.2007 / пер. В. Черкасова [электронный ресурс]. URL: http://netsago.org/ru/docs/1/6/ (дата обращения: 25.05.2012).

38. HTML DOM: учебник. [электронный ресурс]. URL: http://www.wisdomweb.ru/HDOM/hdom-first.php (дата обращения: 15.05.2012).

39. PHP: Безопасность – Manual. [электронный ресурс]. URL: http://www.php.net/manual/ru/security.php (дата обращения: 25.05.2012).

40. TIOBE Software: TIOBE index. [электронный ресурс]. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (дата обращения: 29.02.2012).

41. XML-формат файла Sitemap. [электронный ресурс]. URL: http://www.sitemaps.org/ru/protocol.html#location (дата обращения: 25.03.2012).

Учебное издание

Васильева Ирина Николаевна
Федоров Дмитрий Юрьевич

WEB-ТЕХНОЛОГИИ

Учебное пособие

Редактор М. В. Манерова

Подписано в печать 07.04.14. Формат 60×84 1/16.
Печ. л. 4,25. Тираж 90 экз. Заказ 82.

Издательство СПбГЭУ. 191023, Санкт-Петербург, Садовая ул., д. 21.

Отпечатано на полиграфической базе СПбГЭУ