МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озёрский технологический институт — филиал НИЯУ МИФИ

Вл. Пономарев

Технологии программирования распределенных приложений

Учебно-методическое пособие

Озерск, 2013

УДК 681.3.06
П 56

Пономарев В.В. Технологии программирования распределенных приложений. Учебно-методическое пособие.. Озерск: ОТИ НИЯУ МИФИ, 2013. — 40 с., ил.

В пособии описываются современные технологии, применяемые при создании web-приложений. Дается краткое введение в язык PHP и базы данных MySQL. Рассматриваются структура XML-документа, объектные модели DOM, язык запросов XPath, язык трансформаций XSLT и другие.

Пособие предназначено для студентов-программистов, обучающихся по специальности 230105 — «Программное обеспечение вычислительной техники и автоматизированных систем».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Введение 4
1. Введение в язык PHP 5
1.1. Модель «клиент-сервер» и скрипты PHP 5
1.2. Формы и PHP 6
1.3. Совмещение запроса и ответа 8
1.4. Альтернативный синтаксис 9
1.5. PHP и MySQL 10
1.6. Классы и PHP 13
1.7. Отправление почтового сообщения 15
1.8 Схема Модель–Шаблон–Контроллер (MVC) 16
1.9 Компонентный подход 18
2. Технологии XML 20
2.1. XML-документы 20
Объектная модель XML-документа (DOM) 20
Кодировки 20
Класс domDocument 21
Класс domNode 22
Классы NodeList и NamedNodeMap 23
Создание XML-документа методами domDocument 23
DOM2 24
DOM3 24
Язык XPath 24
Структура запроса 27
Оси 28
Функции 29
Расширение SIMPLEXML 31
Доступ к дочерним узлам 32
Доступ к атрибутам 32
Язык XSLT 33

Введение

1. Введение в язык PHP

1.1. Модель «клиент-сервер» и скрипты PHP

Прежде, чем начать изучение применения языка PHP, нужно уяснить его точное положение в системе «клиент-сервер», действующей в рамках технологий Интернет.

Процесс получения HTML-страницы клиентом (браузером) по протоколу HTTP условно изображен на рисунке 1.

Рисунок 1 — Клиент и сервер

На стороне клиента пользователь в браузере вводит в строку адреса так называемый URL интересующей его страницы HTML. URL — Universal Resource Locator, *универсальный указатель местоположения ресурса*. Через телекоммуникационные сети запрос поступает на некоторый сервер сети Интернет, который содержит требуемый ресурс (вопросы маршрутизации или использования стека протоколов TCP/IP для установления соединения и передачи данных, а также CGI здесь для простоты не рассматриваются).

На рисунке 1 в качестве ресурса выступает скрипт. Будем понимать под скриптом программу, выполняющуюся на сервере и формирующую ответ браузеру в виде HTML-страницы. Заметим, что, в принципе, ресурсом может выступать страница HTML в чистом виде (не требующая предварительной интерпретации). Неправильно будет говорить о таком ресурсе, как о скрипте, потому что в этом случае страница просто отправляется браузеру.

После получения ответа браузер размещает HTML-страницу в своем окне, пользователь ее наблюдает.

Язык PHP (интерпретатор PHP) работает на стороне сервера. Программа на языке PHP и есть тот скрипт, который запрашивается браузером как страница HTML. На самом деле страница HTML формируется в результате интерпретации скрипта (в данном случае PHP).

Представленная модель формирования HTML страниц является простейшей. В действительности схемы формирования web-страниц зачастую несколько сложнее.

1.2. Формы и PHP

Рассмотрим простейшую задачу web-приложения, заключающуюся в посылке на сервер запроса авторизации. Как известно, в чистом HTML для этой цели используются так называемые формы (тег *FORM*).

Рассмотрим, как происходит процесс обмена информацией между браузером и сервером. Страница HTML-запроса имеет вид, показанный на рисунке 2.

```
1: <!-- файл f1q.html -->
2: <HTML><HEAD><TITLE>Форма запроса</TITLE></HEAD><BODY>
3: <P>Введите имя
4: <FORM method='GET' action='f1a.php'>
5:   <INPUT type='text' name='user'><BR><BR>
6:   <INPUT type='submit' value='Отправить'>
7: </FORM>
8: </BODY></HTML>
```

Рисунок 2 - Форма запроса

Как видно из рисунка, код HTML содержит описание формы, состоящей из двух элементов: текстового поля с именем *name* (строка 5) и кнопки отправки формы на сервер с надписью «*Отправить*» (строка 6). Атрибутами тега *FORM* являются (строка 4): метод (указан метод *GET*) и скрипт назначения (указан скрипт *f1a.html*).

Для простоты здесь на сервер отправляется только имя (*логин*).

Ответный скрипт PHP *f1a.html* приведен на рисунке 3.

```
1: <!-- файл f1a.php -->
2: <HTML><HEAD><TITLE>Скрипт ответа</TITLE><HEAD><BODY>
3: <P>Ответ на запрос
4: <P>Ваше имя: <?=$_REQUEST['user']?>
5: </BODY><HTML>
```

Рисунок 3 - Скрипт ответа

Внешне код скрипта ответа выглядит как страница на чистом HTML, за одним небольшим исключением. В строке 4 внутри «скобок» `<?` и `?>` размещен код на языке PHP. Знак «равно» означает, что в данное место HTML-страницы будет вставлено значение последующего выражения. В нашем случае выражением является значение параметра *user*, которое было послано серверу браузером во время отправки формы.

Рассмотрим, что происходит при отправке формы (рисунок 4).

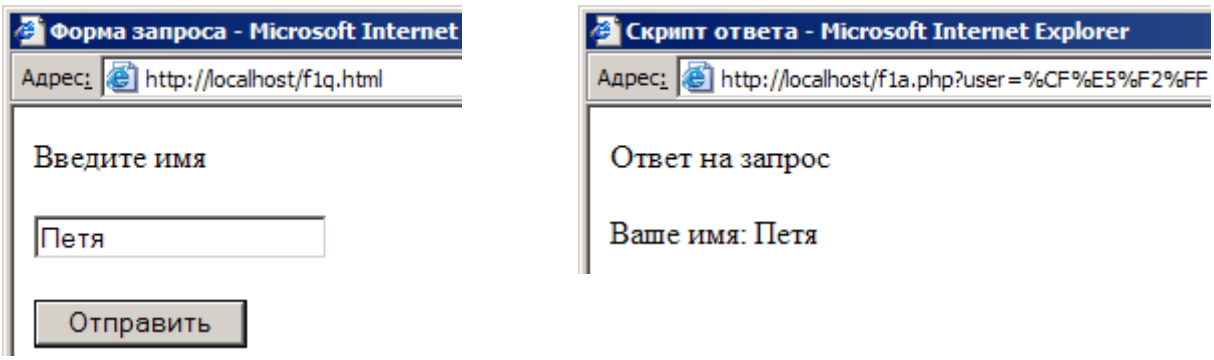

Рисунок 4 — Страница запроса (слева) и страница ответа (справа)

После того, как пользователь ввел имя (*Петя*) и нажал кнопку «*Отправить*», браузер сформировал запрос, который на рисунке справа имеет следующий вид (для простоты слово *Петя* показано как есть, а не в закодированном виде, как на рисунке):

```
http://localhost/f1a.php?user=Петя
```

Здесь после *URL* скрипта стоит знак вопроса, после чего идет строка параметров запроса, в данном случае присутствует только один параметр с именем *user* и значением *Петя*. Все параметры, передаваемые таким образом (методом *GET*), в интерпретаторе PHP (в скрипте PHP) доступны как ассоциативный массив с именем **`$_REQUEST`**. Доступ к элементам этого массива осуществляется по имени параметра, как показано на рисунке 3 в строке 4: **`$_REQUEST['user']`**.

Цель данного простейшего примера — показать, как происходит взаимодействие запроса браузера со скриптом PHP. Выделим главные моменты:

- код на языке PHP должен быть заключен в «скобки» **`<?`** и **`?>`**;
- код на языке PHP может размещаться в тексте скрипта произвольным образом по отношению к другому тексту; этот другой текст тогда воспринимается браузером как код HTML;
- простейший способ вставить в страницу HTML значение выражения, вычисляемого в PHP, является использование конструкции

```
<?=выражение?>
```

- любая переменная PHP начинается со знака «доллар» — "**`$`**";
- переменные не требуют объявления и имеют переменный тип;
- интерпретатору PHP в момент выполнения скрипта доступна любая информация, которая была послана браузером по протоколу *CGI*. Перечень этой информации здесь не приводится;
- дополнительно: параметры, посылаемые методом *POST*, попадают в ассоциативный массив с именем **`$_POST`**.

1.3. Совмещение запроса и ответа

На самом деле нет никакой необходимости создавать два файла для рассматриваемого нами взаимодействия (за исключением, возможно, некоторых особых случаев). Два файла, приведенные на рисунках 2 и 3, можно легко объединить, используя условный оператор языка PHP.

Смысл заключается в том, чтобы каким-то образом выяснить, является ли запрос браузера к скрипту первым или вторым. При помощи функции `isset` можно определить наличие или отсутствие значения. Если, предположим, Вы уверены, что при отправке формы интерпретатор PHP должен обнаружить некоторую переменную *X*, то отсутствие этой переменной укажет на то что запрос является формой, а не ее ответом.

На рисунке 5 приведен код скрипта, который использует данное свойство, используя в качестве переменной *X* значение ассоциативного массива `$_REQUEST['user']`.

```
01: <!-- файл f2.php -->
02: <?php
03: echo "<HTML><HEAD><TITLE>Скрипт 2</TITLE></HEAD><BODY>\r\n";
04: if( isset( $_REQUEST['user'] ) ) {
05:  // это второй запрос
06:  echo "<P>Ответ на запрос";
07:  echo "<P>Ваше имя: ".$_REQUEST['user']."";
08: } else {
09:  # это первый запрос
00:  echo "<P>Введите имя";
11:  echo "<FORM method='GET' action='".$_SERVER['SCRIPT_NAME']."'>";
12:  echo "  <INPUT type='text' name='user'><BR><BR>";
13:  echo "  <INPUT type='submit' value='Отправить'>";
14:  echo "</FORM>\r\n";
15: } /* комментарий */
16: echo "</BODY></HTML>\r\n";
17: ?>
```

Рисунок 5 — Запрос и ответ в одном файле

Заметим, что непосредственно после открывающей скобки `<?` здесь приписано слово `php` (строка 2). Это сделано с целью уточнить, какой язык открывается данной скобкой.

Далее, данный скрипт показывает, как сформировать код HTML при помощи оператора *echo* (строка 7). Заметим, что при необходимости вставить в строку значение выражения, используется конкатенация:

```
echo "строка".выражение-php."строка";
```

Оператором конкатенации является точка ".". Этот оператор допускает форму со знаком «равно», то есть ".=", заменяя, таким образом, конструкцию вида `$a=$a."строка"` на конструкцию `$a.="строка"`.

Кроме того, в строковые литералы значения переменных могут быть вставлены как есть, например:

```
$переменная="строка $переменная строка";
```

1.4. Альтернативный синтаксис

На рисунке 6 приведен код скрипта, выполняющий те же самые действия, что и код на рисунке 5.

```
<!-- файл f3.php -->
<HTML><HEAD><TITLE>Скрипт 3</TITLE></HEAD><BODY>
<? if( isset( $_REQUEST['user'] ) ): ?>
  <P>Ответ на запрос
  <P>Ваше имя: <?=$_REQUEST['user']?>
<? else: ?>
  <P>Введите имя
  <FORM method='GET' action='<?=$_SERVER['SCRIPT_NAME']?>'>
    <INPUT type='text' name='user'><BR><BR>
    <INPUT type='submit' value='Отправить'>
  </FORM>
<? endif ?>
</BODY></HTML>
```

Рисунок 6 — Альтернативный синтаксис

Здесь, очевидно, основная часть текста — код HTML, содержащий несколько «вкраплений» инструкций PHP. Это возможно вследствие использования «альтернативного синтаксиса» операторов PHP. В результате текст скрипта становится более читабельным.

Альтернативный синтаксис основных операторов следующий:

```
<? if (выражение) : ?>
    код HTML
<? elseif (выражение) : ?>
    код HTML
<? else : ?>
    код HTML
<? endif ?>

<? for ([выражение];[выражение];[выражение]) : ?>
    код HTML
<? endfor ?>

<? while (выражение) : ?>
    код HTML
<? endwhile ?>
```

Есть также альтернативный синтаксис для оператора **switch**, но нет для оператора **do** … **while**.

Заметим, что PHP очень лояльно относится к любой промежуточной форме формирования HTML-страницы.

1.5. PHP и MySQL

MySQL — простая реляционная СУБД, часто используемая в сочетании с PHP. Здесь для простоты не рассматриваются вопросы создания баз данных MySQL. Для этой цели есть много различных средств. Предположим, что пустая база данных с именем `b_test` создана. Первое действие, которое должно быть выполнено скриптом — подключение к базе данных. Для этой цели будем использовать специальный дополнительный скрипт *cons.php* (рисунок 7).

```
01: <?php # файл cons.php
02: $dben = 0;
03: $link_id = mysql_connect( 'localhost', 'root', '' );
04: if ( $link_id ) {
05:   if ( mysql_select_db( 'b_test', $link_id ) ) {
06:     $dben = 1;
07:   } else {
08:     $dben = -1;
09:   }
10: }
11: ?>
```

Рисунок 7 — Скрипт для подключения к базе данных

В строке 2 определяется переменная `$dben`, указывающая на наличие подключения. В строке 3 производится подключение к СУБД, расположенной на `localhost`, вход осуществляется под именем `root` с пустым паролем. Результатом выполнения функции `mysql_connect` является переменная `$link_id` (идентификатор подключения). Эта переменная должна быть использована в дальнейшем для выполнения операций.

В строке 5 при помощи функции `mysql_select_db` выбирается база данных для последующей работы. Значение переменной `$dben`, равное нулю, указывает на отсутствие подключения к СУБД, значение –1 указывает на отсутствие подключения к базе данных, значение 1 — успех.

Следующий скрипт предназначен для создания таблицы. Его первоначальный вид приведен на рисунке 8 (скрипт *t_create.php*).

```
01: <?php # файл t_create.php
02:   include_once 'cons.php';
03:   echo $dben;
04: ?>
```

Рисунок 8 — Проверка подключения

Здесь в строке 2 использована инструкция PHP, с помощью которой указанный скрипт включается в данное место, причем включение производится только один раз. В строке 3 в окно браузера выводится значение переменной `$dben`.

Если при запуске скрипта *t_create.php* в окне браузера высвечивается единица, подключение успешно и можно продолжать.

Программно создадим таблицу пользователей с именем **t_user**, в которой есть три поля (атрибута), а именно: **id_user** для уникального идентификатора, **us_logn** для псевдонима и **us_pass** для пароля. Для этого модифицируем скрипт *t_create.php* (рисунок 9).

```
01: <HTML><HEAD><TITLE>Создание таблицы</TITLE></HEAD><BODY>
02: <H2>Создание таблицы пользователей</H2>
03: <?php # файл t_create.php
04: include "cons.php";
05: if ( $dben != 1 ) die( "Нет подключения: $dben " );
06: $table = "t_user";
07: $qs = "drop table if exists ".$table;
08: $qi = mysql_query( $qs, $link_id );
09: if ( !$qi ) die( "Ошибка удаления таблицы $table " );
10: $qs = "CREATE TABLE ".$table." (
11:   id_user INT auto_increment NOT NULL,
12:   us_logn VARCHAR(32) NOT NULL DEFAULT '',
13:   us_pass VARCHAR(32) NOT NULL DEFAULT '',
14:   UNIQUE INDEX ix_$table_logn (us_logn),
15:   PRIMARY KEY (id_user)
16: )";
17: echo "<BR> $qs <BR>";
18: $qi = mysql_query( $qs, $link_id );
19: if ( !$qi ) {
20:   echo "Ошибка создания таблицы $table <BR>";
21:   die( mysql_error() );
22: }
23: echo"Таблица $table создана.<BR>";
24: # добавляем пользователя
25: $qs = "INSERT `t_user` (`us_logn`, `us_pass`)
26:        VALUES ('admin', '".md5("123456")."')";
27: $qi = mysql_query( $qs, $link_id );
28: mysql_close();
29: ?>
30: </BODY></HTML>
```

Рисунок 9 — Скрипт для создания таблицы

В строке 5 проверяется переменная **$dben**, и при отсутствии подключения скрипт завершается функцией **die**, при этом выводится некоторое сообщение. В строке 8 с помощью функции **mysql_query** выполняется SQL-запрос на удаление таблицы, сформированный в переменной **$qs**. В строчках 10-16 формируется SQL-запрос на создание таблицы. В строке 17 SQL-запрос выводится в окно браузера для контроля. В конце скрипта формируется SQL-запрос на добавление новой записи.

Пароль пользователя хранится в базе данных в виде хеша, который формируется в скрипте при помощи функции **md5**. Результатом этой функции является число в шестнадцатеричной форме, длиной 32 байта.

После того, как таблица сформирована, можно приступить к разработке скрипта, осуществляющего авторизацию. Вопросы безопасности здесь для простоты не рассматриваются. Скрипт *auth.php* приведен на рисунке 10.

```
01: <?php # файл auth.php
02: include 'cons.php';
03: if ( $dben != 1 ) die( "Нет подключения: $dben " );
04: $login = 0;
05: if ( @isset( $_POST['user'] ) ) {
06:   $logn = substr($_POST['user'], 0, 32);
07:   $pass = substr($_POST['pass'], 0, 32);
08:   $qs = "SELECT us_pass FROM t_user WHERE us_logn='$logn'";
09:   $qi = mysql_query( $qs, $link_id );
10:   $nn = mysql_num_rows( $qi );
11:   if ( $nn == 1 ) {
12:     $row = mysql_fetch_array( $qi );
13:     if ( md5( $pass ) == $row['us_pass'] ) {
14:       $login = 1;
15:     }
16:   }
17: }
18: ?>
19: <HTML><HEAD><TITLE>Авторизация</TITLE></HEAD><BODY>
20: <H2>Авторизация</H2>
21: <? if ( $login == 1 ): ?>
22:  <P>Ответ на запрос
23:  <P>Ваше имя: <?=$logn?>
24: <? else: ?>
25:  <FORM method='POST' action='<?$_SERVER['SCRIPT_NAME']?>'>
26:    <P>Введите имя<BR>
27:    <INPUT type='text' name='user' size='32'>
28:    <P>Введите пароль<BR>
29:    <INPUT type='password' name='pass' size='32'><BR><BR>
30:    <INPUT type='submit' value='Отправить'>
31:  </FORM>
32: <? endif ?>
33: </BODY></HTML>
```

Рисунок 10 — Скрипт авторизации

В примере функция **mysql_num_rows** возвращает количество записей в запросе (строка 10). Функция **mysql_fetch_array** возвращает очередную запись (строка 12). Эта функция возвращает ложь при отсутствии записей (при достижении конца набора записей). Запись представляет собой ассоциативный массив, ключами которого являются названия полей. Пример выборки поля приведен в строке 13.

Таким образом, все основные функции PHP для работы с базой данных рассмотрены в приведенных примерах. Это не означает, что нет других функций. При необходимости их легко найти в соответствующих справочниках, или, например, в Интернет.

1.6. Классы и PHP

Язык PHP является объектно-ориентированным, и, соответственно, в нем можно создавать классы. На рисунке 11 приведен пример простого класса для работы с базой данных.

```
<?php # файл db_sql.php
class db_sql {
  public
  $host="", $dbname="", $user="", $pass="",
  $link_id = 0, $query_id = 0,
  $rec=array(), $row,
  $error = "";
  public function connect(){
    $this->link_id = mysql_connect( $this->host,
                                    $this->user,
                                    $this->pass );
    if( !$this->link_id ) {
      $this->error = mysql_error();
      return false;
    }
    if( !mysql_select_db( $this->dbname, $this->link_id ) ){
      $this->error = mysql_error();
      mysql_close();
      return false;
    }
    return true;
  }
  public function query($query_string){
    $this->sql=$query_string;
    $this->connect();
    $this->query_id = mysql_query($query_string, $this->link_id);
    $this->error = mysql_error();
    $this->row = 0;
    return $this->query_id;
  }
  public function next_rec(){
    $this->rec = mysql_fetch_array($this->query_id);
    $this->error = mysql_error();
    $this->row+=1;
    $stat = is_array($this->rec);
    if ( !$stat ) {
      mysql_free_result($this->query_id);
      $this->query_id = 0;
    }
    return $stat;
  }
  function num_rows(){
    n = mysql_num_rows($this->query_id);
    $this->error = mysql_error();
    return n;
  }
}
?>
```

Рисунок 11 — Класс базы данных

Как видно из рисунка, переменные класса объявляются с помощью модификатора **public**, и им можно задавать значения. В старых версиях PHP для этой цели использовалось слово **var**. В новых версиях можно использовать модификаторы **public**, **protected** или **private**.

Методами класса являются любые функции, которым может предшествовать ключевое слово **public**, **protected** или **private**.

Внутри класса переменные класса доступны через указатель **$this**, при этом знак доллара перед переменной опускается.

Пример использования данного класса приведен на рисунке 12.

```
01: <?php # файл db_use.php
02: include_once "db_sql.php";
03: $db = new db_sql;
04: $db->host = "localhost";
05: $db->dbname = "имя_базы_данных";
06: $db->user = "имя_пользователя";
07: $db->pass = "пароль";
08: if ( $db->connect() ) {
09: } else {
10: }
?>
```

Рисунок 12 — Использование класса

При необходимости класс может содержать конструктор и деструктор. Конструктор в новых версиях имеет примерно следующий вид:

```
function __construct() { ... }
```

В старых версиях конструктор имеет имя класса.

Примерный вид деструктора:

```
function __destruct() { ... }
```

Наследование классов объявляется при помощи слова **extends**. Пример наследования приведен на рисунке 13.

```
01: <?php # файл db_admin.php
02: require_once "db_sql.php";
03: class db_admin extends db_sql {
04:   $host = "localhost";
05:   $dbname = "имя_базы_данных";
06:   $user = "имя_пользователя";
07:   $pass = "пароль";
08:   function __construct() {
09:     parent::__construct();
10:   }
11: }
12: ?>
```

Рисунок 13 — Наследование класса

1.7. Отправление почтового сообщения

Для работы с почтовыми сообщениями в PHP используется функция `mail`. С ее помощью можно отправить простое, без вложений, почтовое сообщение.

1.8 Схема Модель–Шаблон–Контроллер (MVC)

Схема MVC (Model–View–Controller) применяется для того, чтобы разделить модули web-приложения на части, с которыми работают специалисты разных областей. Как правило, речь в этом случае идет о дизайнере и программисте. Дизайнер занимается художественным оформлением web-страниц, программист формирует необходимую логику работу web-приложения.

Модель описывает предметную область системы. Она включает в себя базу данных и код, непосредственно с ней работающий.

Шаблон — это заготовка страницы, определяющая ее внешний вид. Страница может иметь несколько шаблонов.

Контроллер — код (бизнес-логика), выступающий посредником между моделью и шаблоном, и принимающий данные от пользователя.

Взаимосвязь элементов MVC показана на рисунке 14.

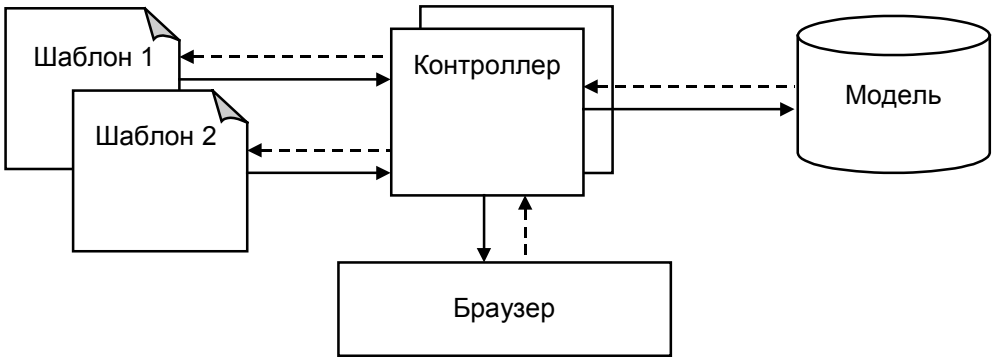

Рисунок 14 — Взаимодействие элементов MVC

В качестве примера рассмотрим простую гостевую книгу. Код шаблона приведен на рисунке 15.

```
01: <!-- файл view.html -->
02: <HTML><HEAD><TITLE>Guest Book</TITLE></HEAD><BODY>
03: <H2>Добавьте свое сообщение</H2>
04: <FORM method='POST' action='controller.php'>
05: Ваше имя: <INPUT type='text' name='new[name]'> <BR>
06: Сообщение:<BR>
07: <TEXTAREA name='new[text]' cols='60' rows='5'> </TEXTAREA> <BR>
08: <INPUT type='submit' name='add' value='Добавить'>
09: </FORM>
10: <H2>Гостевая книга</H2>
11: <? foreach( $book as $id=>$e ) { ?>
12:   Имя: <?=$e['name']?><BR>
13:   Сообщение: <?=$e['text']?><BR>
14: <? } ?>
15: </BODY></HTML>
```

Рисунок 15 — Код шаблона MVC

Шаблон описывает страницу гостевой книги, в которой сначала расположено поле для ввода имени гостя и его сообщения, за которыми следует перечень всех сообщений, обозначенных именами гостей. Первая часть реализуется в виде формы, вторая — циклом **foreach**.

Код контроллера приведен на рисунке 16.

```
01: <?php # файл controller.php
02: define("gbook", "gbook.txt");
03: require_once "model.php";
04: $book=LoadBook( gbook );
05: if ( !empty( $_REQUEST['add'] ) ){
06:   $book = array(time() => $_REQUEST['new']) + $book;
07:   SaveBook(gbook, $book);
08: }
09: include "view.html";
10: ?>
```

Рисунок 16 — Код контроллера MVC

Как видно из кода, контроллер определяет переменную **gbook**, задающую обычный текстовый файл. Далее он загружает записи гостевой книги из этого файла. Если гость отправил новое сообщение, оно приписывается к файлу в его начало и сохраняется. В конечном итоге переменная **gbook** передается шаблону, который отображает ее содержимое.

Код модели показан на рисунке 17.

```
01: <?php # файл model.php
02: function LoadBook( $fname ) {
03:   if ( !file_exists( $fname ) ) return array();
04:   $book = unserialize( file_get_contents( $fname ) );
05:   return $book;
06: }
07: function SaveBook( $fname, $book ) {
08:   file_put_contents( $fname, serialize( $book ) );
09: }
10: ?>
```

Рисунок 17 — Код модели MVC

Код модели определяет две функции, используемые контроллером.

Функция **LoadBook** загружает гостевую книгу из текстового файла, а функция **SaveBook** сохраняет измененную книгу.

Недостатки схемы MVC: а) выбором шаблона управляет контроллер, поэтому дизайнер не может это сделать самостоятельно; б) адресация страниц производится относительно URL контроллера, а не URL шаблона; в) идеологически пользователь хочет видеть страницу (шаблон), а запускает контроллер.

1.9 Компонентный подход

Более удачным решением распределения модулей между дизайнером и программистом является компонентный подход. В нем страница представляется как последовательность блоков, которые дизайнер размещает в необходимой для его идеи последовательности.

Блоки могут иметь разную природу и содержать различные данные, например, баннер, новости сайта, меню раздела, основной текст и т.п.

Компоненты — это модули, которые генерируют данные блоков.

Взаимодействие элементов в этой схеме приведено на рисунке 18.

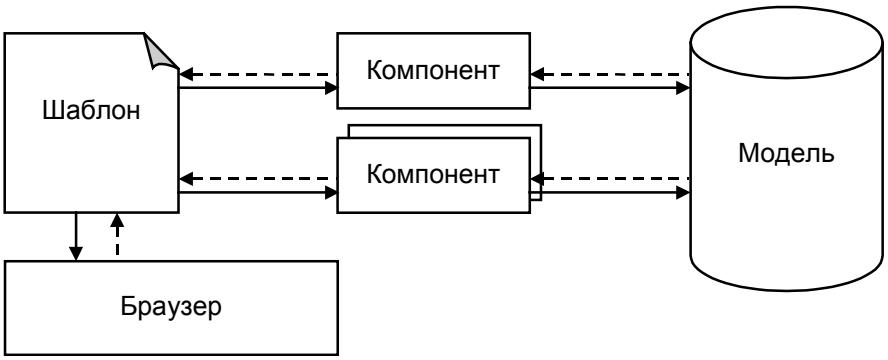

Рисунок 18 — Взаимосвязь элементов в компонентной модели

На рисунке 19 приведен примерный код шаблона гостевой книги для компонентного подхода.

```
01: <!-- файл view.html -->
02: <HTML><HEAD><TITLE>Guest Book</TITLE></HEAD><BODY>
03: <!-- Блок ввода нового сообщения -->
04: <? include 'gb_add.php'; ?>
05: <H2>Добавьте свое сообщение</H2>
06: <FORM method='POST'>
07: Ваше имя: <INPUT type='text' name='new[name]'> <BR>
08: Сообщение:<BR>
09: <TEXTAREA name='new[text]' cols='60' rows='5'> </TEXTAREA> <BR>
10: <INPUT type='submit' name='add' value='Добавить'>
11: </FORM>
12: <!-- Блок сообщений гостевой книги -->
13: <? include 'gb_show.php'; ?>
14: <H2>Гостевая книга</H2>
15: <? foreach ( $data as $id=>$e ) { ?>
16:   Имя: <?=$e['name']?><BR>
17:   Сообщение: <?=$e['text']?><BR>
18: <? } ?>
19: </BODY></HTML>
```

Рисунок 19 — Код шаблона компонентной модели

Как видим, страница содержит два блока: ввод нового сообщения (строки 03-11) и блок сообщений (строки 12-18).

Контроллер, управляющий блоком ввода нового сообщения, показан на рисунке 20.

```
01: <?php # файл gb_add.php
02: define( "gbook", "gbook.txt" );
03: require_once 'model.php';
04: if ( !empty( $_REQUEST['add'] ) ) {
05:   $t = LoadBook( gbook );
06:   $t = array(time() => $_REQUEST['new']) + $t;
07:   SaveBook( gbook, $t );
08: }
09: // Не генерирует никаких данных
10: $data = null;
11: ?>
```

Рисунок 20 — Код контроллера ввода сообщения

На рисунке 21 показан код контроллера, формирующего гостевую книгу.

```
01: <?php # файл gb_show.php
02: $data = LoadBook( gbook );
03: ?>
```

Рисунок 21 — Код контроллера формирования гостевой книги

Код модуля **model.php** остается прежним, как и в модели MVC, т.к. функции модели не изменились.

Кроме того, потребуется еще один специальный файл, код которого приведен на рисунке 22. Это файл с именем **.htaccess**, расположенный в текущем каталоге. Он позволяет включить интерпретатор PHP для обработки HTML-файлов.

```
01: # файл .htaccess включает PHP для файлов HTML
02: AddHandler application/x-httpd-php html
```

Рисунок 22 — Файл .htaccess

Достоинства компонентного подхода:

а) страницы строятся из независимых блоков (представленных компонентами);

б) дерево сайта с точки зрения дизайнера выглядит так же, как и для пользователя;

в) шаблон можно использовать для создания других страниц.

2. Технологии XML

2.1. XML-документы

XML (*eXtensible Markup Language*, расширяемый язык разметки) — язык описания документов, использующий представление в виде иерархии. Предназначен для хранения структурированных данных (взамен существующих методов хранения в базах данных), для обмена информацией между программами и создания более специализированных языков. Является упрощенным подмножеством языка SGML (*Standard Generalized Markup Language*).

Основным понятием XML-документа является узел (*node*). Центральный узел имеет тип Document. Базовый узел имеет тип Element, он может иметь любое количество подузлов.

Конечными узлами (не имеющими подузлов) являются узлы типов:

- Text (текст);
- Comment (комментарий);
- ProcessingInstruction (команды);
- CDATASection (набор данных символьного типа).

Существуют также узлы, входящие в поддерево узла DOCTYPE. Этот узел описывает структуру документа, список его компонентов, формат элементов и т.п. Они служат для формирования окончательного вида XML-документа, проверки его корректности.

Далее будем рассматривать некоторый XML-документ по частям и одновременно рисовать его частичную структуру. Документ описывает программу передач.

Объектная модель XML-документа (DOM)

При обработке XML-документов в программе выстраивается дерево объектов, отражающих структуру документа, которое затем используется для построения выходного документа. Метод отображения XML-документа в дерево объектов определяет стандарт DOM (Document Object Model).

Существует три версии этого стандарта — DOM1, DOM2, DOM3.

Стандарт DOM1 определяет общую структуру XML-документа, типы узлов. Узлы описываются в виде классов. Последняя версия этого стандарта вышла в 2000 году, она имеет название *Document Object Model* (*DOM*) *Level 1 Specification* (*Second Edition*). Стандарты DOM2 и DOM3 являются расширением и развитием стандарта DOM1.

Кодировки

Стандарт DOM определяет три типа кодировки.

1) *Входная кодировка* задает кодировку исходного XML-документа, которая определяется по заголовку исходного документа:

```
<?xml version="1.0" encoding="кодировка" ?>
```

2) *Внутренняя кодировка* определяет кодировку строк в объектах. В качестве внутренней кодировки стандарт определяет Unicode UTF-16. Реализация DOM в языке PHP задает кодировку Unicode UTF-8.

3) *Выходная кодировка* определяет, в какой кодировке будет формироваться выходной XML-документ. Она отображается в заголовке выходного документа.

Установку кодировок выполняет следующая функция PHP5:

```
int iconv_set_encoding(string type, string charset)
```

Параметр type может принимать значения:

`input_encoding` — установить входную кодировку;
`internal_encoding` — установить внутреннюю кодировку;
`output_encoding` — установить выходную кодировку.

Для перекодировки строк в PHP5 используется функция:

```
string iconv(string in_charset, string out_charset, string string)
```

Пример перекодировки из кодовой страницы 1251 в UTF-8:

```
$s = iconv("WINDOWS-1251", "UTF-8", $str);
```

Следующая функция вычисляет длину строки в указанной кодировке:

```
int iconv_strlen(string string [, string charset])
```

Существуют и другие функции для работы с кодированными строками.

Класс domDocument

Экземпляр класса `domDocument` представляет XML-документ в целом. Этот класс является расширением более общего класса `domNode`.

Экземпляр класса `domDocument` создается следующим образом:

```
$dd = new domDocument([string version[, string encoding]]);
```

Первый параметр задает версию XML-документа, второй — выходную кодировку. Следующий пример показывает, как создать XML-документ, построить дерево объектов (метод `loadXML`) и вывести документ на страницу (метод `saveXML`):

```
<?php ## Загрузка XML-документа методом loadXML
$dd = new domDocument("1.0", "WINDOWS-1251");
$xml = "<?xml version='1.0' encoding='WINDOWS-1251'?>
<HTML><HEAD><TITLE>Пример XML-документа</TITLE></HEAD><BODY>
<H1>Пример XML-документа</H1>
```

```
<P>Строка 1</P>
<P>Строка 2</P>
</BODY></HTML>";
// Построить дерево объектов
$dd->loadXML($xml);
// Получить XML-документ
$out = $dd->saveXML();
echo $out;
?>
```

Следующие свойства класса **domDocument** управляют вводом и выводом:

$documentURI — задает адрес XML-документа;
$formatOutput — формирует XML-документ с отступами;
$preserveWhiteSpace — задает режим обработки пустых текстовых полей;
$resolveExternals — разрешает подключать внешние XML-документы;
$substituteEntities — разрешает подстановку ссылок на компоненты;
$validateOnParse — позволяет проверять корректность XML-документа.

Класс domNode

Класс **domNode** является корневым для всех типов узлов дерева объектов XML-документа. Расширениями этого класса являются классы: **domDocument**, **domDocumentType**, **domEntity**, **domNotation**, **domElement**, **domAttr**, **domEntityReference**, **domDocumentFragment**, **domProcessingInstruction**, **domCharacterData**.

Рассмотрим свойства данного класса. Первые два свойства должны быть непустыми.

unsigned int nodeType — тип узла;
domString nodeName — имя узла;
domString nodeValue — значение узла;
Document ownerDocument — узел документа;
Node parentNode — родительский узел;
NamedNodeMap attributes — атрибуты узла;
NodeList childNodes — дочерние элементы узла;
Node firstChild — первый дочерний узел;
Node lastChild — последний дочерний узел;
Node previousSibling — предыдущий узел данного уровня;
Node nextSibling — следующий узел данного уровня.

Следующая таблица содержит возможные значения свойства **domType**.

Таблица 1 — Константы свойства **domType**

Узел	*Константа DOM*	*Константа PHP*
Элемент	**ELEMENT_NODE**	**XML_ELEMENT_NODE**
Атрибут	**ATTRIBUTE_NODE**	**XML_ATTRIBUTE_NODE**
Текст	**TEXT_NODE**	**XML_TEXT_NODE**

CDATA	CDATA_SECTION_NODE	XML_CDATA_SECTION_NODE
Ссылка	ENTITY_REFERENCE_NODE	XML_ENTITY_REF_NODE
Компонент	ENTITY_NODE	XML_ENTITY_NODE
Команды	PROCESSING_INSTRUCTION_NODE	XML_PI_NODE
Комментарии	COMMENT_NODE	XML_COMMENT_NODE
Документ	DOCUMENT_NODE	XML_DOCUMENT_NODE
DOCTYPE	DOCUMENT_TYPE_NODE	XML_DOCUMENT_TYPE_NODE

Классы NodeList и NamedNodeMap

Класс `NodeList` — это упорядоченный список узлов. Доступ к узлу списка возможен только по его порядковому номеру. Класс имеет одно свойство `length` и один метод `item` с параметром — номером узла. Первый узел списка имеет номер 1.

Класс `NamedNodeMap` — это неупорядоченный список узлов с уникальными именами (список атрибутов). Этот класс также имеет свойство `length` и метод `item`. Доступ к узлу по имени осуществляется при помощи метода `getNamedItem(string Name)`. Метод `setNamedItem(Node node)` добавляет заданный узел в список. Метод `removeNamedItem(string Name)` удаляет из списка узел с заданным именем.

В следующем примере на страницу выводятся узлы документа (в продолжение предыдущего примера):

```
// Корневой узел
$root = $dd->documentElement;
// Список дочерних узлов
$nodeList = $root->childNodes;
$count = $nodeList->length;
echo "Дочерних узлов: $count <BR>";
for ($i = 0; $i < $count; $i++) {
  $child = $nodeList->item($i);
  echo "Узел $i:<BR>";
  echo iconv("UTF-8","WINDOWS-1251",
    "$child->nodeName='$child->nodeValue'."<BR>");
}
```

Создание XML-документа методами domDocument

Документ XML можно создавать методами вида `createXXXX` класса `domDocument`. Методы `createXXXX` не добавляют созданный узел в дерево объектов. Для одновременного построения дерева объектов требуется применение метода `appendChild`.

В следующем примере создается XML-документ.

```
<?
$dd = new domDocument("1.0", "WINDOWS-1251");
$doc = $dd->createElement("Document");
$dd->appendChild($doc);
$para = $dd->createElement("Paragraph-1");
$doc->appendChild($para);
```

```
$text = $dd->createTextNode("Line 1");
$para->appendChild($text);
$para = $dd->createElement("Paragraph-2");
$doc->appendChild($para);
$text = $dd->createTextNode("Line 2");
$para->appendChild($text);
$dd->preserveWhiteSpace = true;
$dd->formatOutput = true;
$out = $dd->saveXML($doc);
echo $out;
?>
```

Данный скрипт генерирует следующий XML-документ:

```
<Document>
  <Paragraph>Line</Paragraph>
  <Paragraph>Line</Paragraph>
</Document>
```

Заметим, что для того, чтобы сформировать узлы с русскими наименованиями или текстом, потребуется применить перекодирование строк с кириллическими символами в кодировку UTF-8.

DOM2

Стандарт DOM2 позволяет указывать в имени элемента область имен (*namespace*), к которой принадлежит данное имя. Для этого имя элемента разбивается на две части — префикс и локальное имя — которые разделяются двоеточием. Префикс указывает на область имен, а локальное имя определяет имя элемента в этой области. Примером может служить использование в XML-документе тега `<I>`:

```
<HTML:I>Text</HTML:I>
```

Кроме того, стандарт DOM2 требует, чтобы область имен задана некоторым стандартом. Так, область имен HTML может быть задана в виде:

```
<HTML:I xmlns="http://www.w3.org/1999/xhtml">Text</HTML:I>
```

Таким образом, стандарт DOM2 решает проблему совмещения нескольких XML-документов в одном.

DOM3

Стандарт DOM3 описывает модули, отображающие определенные аспекты обработки XML-документа.

Язык XPath

Назначение — выборка из дерева объектов узлов, удовлетворяющих условиям, заданным в запросе на языке XPath. Название стандарта «Document Object Model Level 3 XPath», описание можно найти по адресу:

http://www.w3.org/TR/2003/CR-DOM-Level-3-XPath-20030331.

В PHP5 существует класс `domXpath`, создать экземпляр которого можно следующим образом:

```
$d = new domXpath($domdocument);
```

Класс `domXpath` имеет свойство `$document`, которое хранит ссылку на анализируемый документ, и два метода:

```
nodeList query($string xpath[, domNode $context])
```

— производит поиск в документе узлов, удовлетворяющих условию запроса и формирует список;

```
void registerNamespace(string prefix, string namespace)
```

— связывает префикс с указанной областью имен.

В качестве примера будем рассматривать следующий XML-документ:

```
<?xml version="1.0" encoding="WINDOWS-1251" ?>
<Часть имя="Имя-1">
  <Глава имя="Имя-2">
    <P>Текст-1</P>
    <Раздел имя="Имя-3">
      <P>Текст-2</P>
      <Таблица имя="Имя-4">...</Таблица>
      <P>Текст-3</P>
    </Раздел>
    <Раздел имя="имя-5">
      <P>Текст-4</P>
      <Рисунок имя="Имя-6" src="URL-1" />
    </Раздел>
  </Глава>
  <Глава имя="Имя-7">
    <P>Текст-5</P>
    <Раздел имя="Имя-8">
      <P>Текст-6</P>
        <UL>
          <LI>Текст-7</LI>
          <LI>Текст-8</LI>
        </UL>
      <P>Текст-9</P>
      <Листинг имя="Имя-9" />
    </Раздел>
  </Глава>
</Часть>
```

Подчеркивание выделяет часть документа, соответствующая запросу

```
../../* | /*/Глава[2]//text() .
```

Запрос состоит из частей, разделенных символом «вертикальная черта». В примере две части задают *адрес* (*Location Path*) до искомых узлов. Язык XPath имеет два способа адресации:

- относительная (*RelativeLocationPath*);

- абсолютная (*AbsoluteLocationPath*).

Относительная адресация задает путь до искомых узлов относительно некоторого базового узла, который носит название *контекстного* (*context node*). В примере контекстный узел выделен двойным подчеркиванием — это узел с атрибутом «имя» таблицы 1 раздела 1 главы 1. На языке XPath адрес этого узла имеет вид:

```
/Часть/Глава[1]/Раздел[1]/Таблица[1]/@имя
```

Первая часть запроса описывает путь относительно к контекстного узла. Запрос указывает на все элементы (`*`) узла `Раздел`, который является родителем родителя узла `имя`. Эти элементы выделены в разделе с именем `Имя-3`. При этом узел `Таблица` является родительским по отношению к атрибуту `имя`, однако узел атрибута не является дочерним узлом узла `Таблица`.

Вторая часть запроса задает абсолютный адрес. Абсолютные адреса начинаются с символа «обратный слеш» (узел документа).

Для выполнения запроса используется следующий скрипт.

```
<?
$cpathw = "//Часть/Глава[1]/Раздел[1]/Таблица[1]/@имя";
$xpathw = "../../* | /*/Глава[2]//text()";
$cpath = iconv( "WINDOWS-1251", "UTF-8", $cpathw );
$xpath = iconv( "WINDOWS-1251", "UTF-8", $xpathw );
$dd = new domDocument();
$dd->preserveWhiteSpace = false;
$dd->formatOutput = true;
$dd->load( "t.xml" );
$dx = new domXpath( $dd );
// Найти контекстный узел
$context = @$dx->query( $cpath );
if ( !$context || $context->length == 0 ) {
  echo "Неверный контекстный узел, используется корневой.<BR><BR>";
  $context = $dx->query( "/" );
}
$context = $context->item(0);
// Узлы, удовлетворяющие запросу
$list = $dx->query( $xpath, $context );
$count = $list->length; // Количество узлов
echo "Найдено узлов: $count <BR>";
for ($i = 0; $i < $count; $i++) {
  $child = $list->item($i);
  echo "Узел $i:<BR>".iconv("UTF-8", "WINDOWS-1251",
    "$child->nodeName='$child->nodeValue'")."<BR>";
}
?>
```

Результат работы скрипта следующий:

```
Найдено узлов: 8
Узел 0:
P='Текст-2'
Узел 1:
Таблица='...'
```

```
Узел 2:
P='Текст-3'
Узел 3:
#text='Текст-5'
Узел 4:
#text='Текст-6'
Узел 5:
#text='Текст-7'
Узел 6:
#text='Текст-8'
Узел 7:
#text='Текст-9'
```

Таким образом, в документе выделены три узла по первой части запроса и 5 узлов по второй части. В первой части содержанием узлов являются элементы, а во второй части — текст элементов.

Структура запроса

Относительный адрес задается в виде:

```
Шаг/Шаг/...
```

Абсолютный адрес задается в виде:

```
/Шаг/Шаг/...
```

Каждый из шагов может состоять из трех частей.

- *Ось* (*Axis*) задает направление, в котором выбираются узлы. XPath поддерживает 13 направлений, из которых основными являются семь: `self`, `parent`, `following-sibling`, `preceding-sibling`, `child`, `attributes`, `namespace`. В частности, в используемом запросе использовались оси `parent` (обозначаемая с помощью двух точек) и `child` (обозначаемая знаком `@`).

- *Шаблон_Узла* (*NodeTest*) задает шаблон выбора узла по его типу (*Text*, *Comment*, *ProcessingInstruction* и т.п.), или по имени тега элемента.

- *Предикат* (*Predicate*) задает дополнительные условия по выбору узла, например, его номеру, числу потомков, длине имени т.п.

Название оси отделяется от шаблона узла двумя двоеточиями `::`. Предикат заключается в квадратные скобки `[]`. Шаблон узла и предикат могут отсутствовать.

Шаблон узла может принимать следующие значения:

`node()` — указывает все узлы оси;

`text()` — указывает на текстовые узлы;

`comment()` — указывает на узлы комментариев;

`processing-instuction()` — указывает на узлы приложений;

`*` — в зависимости от типа оси указывает либо на элементы, либо на узлы атрибутов с заданным именем.

Оси

В следующей таблице приведены описания осей XPath.

Таблица 2 — Оси XPath

Ось	*Описание*	*Сокращение*
`self::`	Текущий (контекстный) узел	"."
`attribute::`	Узлы атрибутов	"@"
`namespace::`	Узлы описания области имен	
`child::`	Дочерние узлы	По умолчанию
`parent::`	Родительский узел	".."
`following-sibling::`	Последующий узлы родителя	
`preceding-sibling::`	Предыдущие узлы родителя	
`descendant::`	Потомки узла	
`descendant-or-self::`	Потомки, включая текущий	"//"
`ancestor::`	Предки узла	
`ancestor-or-self::`	Предки, включая текущий	
`following::`	Узлы после текущего	
`preceding::`	Узлы перед текущим	

Полная форма запроса `child::` имеет вид `child::node()`. Эта ось принимается по умолчанию, поэтому сокращенная форма: `node()`.

Примеры (контекстный узел "/"):

`/node()` — выбирает узел «Часть»;
`//node()` — выбирает все узлы документа;
`//node()[1]` — выбирает все первые узлы всех узлов;
`/*/node()[2]` — выбирает узел «Глава имя="Имя-7"»;
`/*/*/*/*` — выбирает все узлы уровня 4.

Предикат позиции `[1]` имеет полный вид `[position()=1]`. Обычно функцию опускают, но иногда требуется ее указание: `[position()!=1]`.

В ось `descendant::` входят все потомки текущего узла, а в ось `descendant-or-self::` входят потомки и сам узел.

Примеры (контекстный узел "/"):

`descendant::*/P/descendant::node()` — выбирает все тексты узлов P;
`descendant::node()[position() > 27]` — выбирает узел «Листинг»;
`//Часть/Глава/Раздел/descendant-or-self::node()` — выбирает узлы, являющиеся потомками узлов `Раздел`, включая сами узлы `Раздел`.

Ось `follow-sibling::` охватывает все узлы, находящиеся после контекстного и имеющие общего с ним родителя. Например, запрос

```
//descendant-or-self::node()/child::P/following-sibling::P
```

/descendant-or-self::node()/self::node() — выбирает все поддеревья элементов **P**, которые находятся в родительском элементе после другого элемента **P**.

В ось **following::** входят все узлы, расположенные после контекстного, исключая дочерние. Например, при использовании контекстного узла

```
//Часть/Глава[1]/P[1]
```

запрос

```
following::*
```

выбирает узлы всех элементов после первого элемента **P**.

Ось **attribute::** выбирает узлы атрибутов, которые не входят в другие запросы. Шаблоном является либо имя атрибута, либо звездочка (все атрибуты). Следующий запрос выбирает все узлы атрибутов **имя**:

```
//*/@имя .
```

Контекстный узел — "/".

Ось **parent::** выбирает родительский узел контекстного узла. Например, запрос **//*/@src/..** выбирает узел элемента **Рисунок**.

Ось **ancestor::** выбирает всех предков контекстного узла. Например, запрос **ancestor::*** при контекстном узле **//LI[1]** выбирает узлы **Часть**, **Глава**, **Раздел**, **UL**.

Функции

В предикатах можно использовать множество функций, которые определяет язык XPath. Приведем некоторые из них.

Функции для работы с узлами

position() — возвращает позицию узла в списке от единицы.
last() — возвращает количество узлов в списке.
count(node-set) — возвращает количество узлов в выражении.
id(object) — выбирает элементы по уникальному идентификатору. Параметр должен быть описан в поддереве **DOCTYPE** описателем **<ATTLIST ...>**.
name(node-set) — возвращает полное имя первого элемента.

Строковые функции

string(object) — преобразует параметр к строковому типу. Если параметр не задан, используется значение контекстного узла.
normalize-space(string) — удаляет начальные и конечные символы разделителей (пробелы, табуляторы, символы перевода строки) и заменяет последовательность разделителей внутри строки одним пробелом.
concat(string, string, ...) — конкатенирует строки.

starts-with(string-1, string-2) — если строка, заданная первым параметром, начинается со строки, заданной вторым параметром, возвращает **true**.

contains(string-1, string-2) — если строка, заданная первым параметром, содержит строку, заданную вторым параметром, возвращает **true**.

substring-before(string-1, string-2) — возвращает часть строки 1 до позиции строки 2.

substring-after(string-1, string-2) — возвращает часть строки 1 после строки 2.

substring(string, number-1, number-2) — возвращает часть строки, начиная с позиции **number-1** длиной **number-2** символов. Нумерация символов с 1.

string-length(string) — возвращает длину строки.

translate(string-1, string-2, string-3) — в строке 1 заменяет символы из строки 2, символами из строки 3.

Математические функции

number(object) — преобразует объект в число. Если сделать это невозможно, возвращается значение **NaN**.

sum(node-set) — возвращает сумму числовых значений узлов.

floor(number) — возвращает ближайшее целое, меньшее или равное **number**.

ceiling(number) — возвращает ближайшее целое число, большее или равное **number**.

round(number) — возвращает ближайшее целое к **number**.

Поддержка областей имен

namespace-uri(node-set) — возвращает область имен, связанную с первым узлом в множестве **node-set**. Если параметр отсутствует, возвращает область имен контекстного элемента.

local-name(node-set) — возвращает локальную часть имени первого элемента множества **node-set**.

С помощью данных функций можно сформировать запрос для выборки элементов с определенной областью имен:

```
//*[namespace-uri()='http://...' and local-name()='...']
```

Привязка префиксов запроса

boolean registerNamespace(prefix, uri) — после вызова данного метода можно использовать префикс **prefix** в шаблоне узла запроса.

Дополнительно

Операторы, используемые в шаблонах:

`or`, `and`, `div`, `mod`, `<`, `>`, `<=`, `>=`, `=`, `+`, `-`, `*`.

Язык XPath поддерживается библиотекой PHP `libxml2`.

Расширение SIMPLEXML

SIMPLEXML — это расширение PHP, реализующее упрощенную версию языка XML. В нем используется единственный класс `simplxml_element`, представляющий собой корневой узел XML-документа (`Document`). Доступ к узлам элементов XML-документа осуществляется по имени тегов, а к атрибутам — как к элементам ассоциативного массива.

В качестве примера рассмотрим следующий XML-документ.

Листинг sxml 1 — Файл sxml.xml

```
<?xml version='1.0' encoding='WINDOWS-1251' ?>
<Document name='name'>
  <P name='P1'>Text-1</P>
  <P name='P2'>Text-2</P>
</Document>
```

Следующий PHP-скрипт заменяет атрибут `name` элемента `Document` и атрибут `name` второго элемента `P`:

Листинг sxml 2 — Файл sxml-2.php

```
<?php
$sxml = simplexml_load_file( "sxml.xml" ); // строка 1
$name = $doc['name'];                      // строка 2
echo "Name=".$name."<BR>";                 // строка 3
$sxml['name']="game";                      // строка 4
$sxml->P[1]['name'] = "para-2";            // строка 5
echo $sxml->asXML();                       // строка 6
?>
```

В строке 1 создается объект `simplxml_element` из файл XML-документа. В строке 2 считывается значение атрибута `name` элемента `Document`. В строке 3 полученное значение выводится в HTML-страницу. В строке 4 значение данного атрибута заменяется на новое. В строке 5 заменяется значение атрибута `name` второго элемента `P`. В строке 6 измененный XML-документ выводится в HTML-страницу. Заметим, что выполнить аналогичные преобразования с помощью расширения DOM значительно сложнее.

Для загрузки и сохранения XML-документов в SIMPLEXML используются функции, а не методы. Кроме уже приведенной функции для загрузки документа из файла есть функция загрузки документа из строки:

```
simplexml_element simplexml_load_string(string)
```

Для сохранения XML-документа в виде текста используется метод

```
string asXML([string filename]) .
```

Метод записывает документ в файл или возвращает документ в виде строки, если параметр не задан.

Следующие две функции выполняют экспортировать загруженный в SIMPLEXML документ в объект DOM и наоборот, импортировать объект DOM в объект расширения SIMPLEXML:

```
domNode dom_Import_SimpleXML(simplexml_element)

simplexml_element simplexml_Import_DOM(domDocument)
```

Корневым элементом в SIMPLEXML является не узел документа, как в DOM, а корневой элемент документа. Поэтому в SIMPLEXML недоступны узлы, расположенные на уровне узла документа (например, `DOCTYPE`), однако в выходном документе эти узлы восстанавливаются.

В дереве корневого объекта `simplexml_element` недоступны также узлы CDATA и текстовые узлы, состоящие из пробельных символов, являющихся разделителями тегов (хотя эти узлы формируются).

Доступ к дочерним узлам

Дочерние узлы являются свойствами родительских узлов, имеющими названия соответствующих тегов. Например, для документа, приведенного в листинге 1, теги `P` образуют свойство `P` корневого элемента. Указанные свойства являются индексируемыми массивами, что позволяет получать доступ к элементу по номеру (от нуля).

Доступ к дочерним узлам в порядке следования выполняется при помощи цикла `for-each` (листинг 3).

Листинг sxml 3 — Файл sxml-3.php

```
<?php
$sxml = simplexml_load_file( "sxml.xml" );
foreach ( $sxml as $n => $v ) echo "$n: $v<BR>";
?>
```

Метод `xpath` позволяет использовать запросы XPath (листинг 4).

Листинг sxml 4 — Файл sxml-4.php

```
<?php
include_once "unit.php";
$sxml = simplexml_load_file( "sxml.xml" );
$list = $sxml->xpath("//P/@*");
foreach ($list as $element) echo toWIN("$element")."<BR>";
?>
```

Доступ к атрибутам

Доступ к атрибутам возможен по имени, являющимся индексом ассоциативного массива (листинг 2). Для доступа к атрибутам в порядке следования используется метод `attributes` (листинг 5).

Листинг sxml 5 — Файл sxml-5.php

```
<?php
include_once "unit.php";
$sxml = simplexml_load_file( "sxml.xml" );
$element = $sxml->P;
foreach ($element->attributes() as $n => $v) echo toWIN("$n:
$v")."<BR>";
?>
```

Язык XSLT

XSLT (*Extensible Stylesheet Language Transformations*) — язык преобразования XML-документов. Спецификация XSLT входит в состав XSL.

XSL (*Extensible Stylesheet Language*) — семейство рекомендаций консорциума W3C, описывающее языки преобразования и визуализации XML-документов. Состоит из трех частей:

1. XSLT — язык преобразований XML-документов.

2. XSL Formatting Objects (XSL-FO) — язык разметки типографских макетов и других предпечатных материалов.

3. Язык запросов XPath.

При применении таблицы стилей XSLT, состоящей из набора шаблонов, к XML-документу образуется дерево, которое может быть преобразовано в документ XML, XHTML, HTML или простой текстовый файл. Правила выбора (и, частично, преобразования) данных из исходного дерева формируются на языке запросов XPath. Одной из задач XSLT является отделение данных от их представления, как часть парадигмы MVC.

В процессе выполнения XSLT-преобразования используются:

- один или несколько входных XML-документов;
- одна или несколько таблиц стилей XSLT;
- XSLT-процессор;
- один или несколько выходных документов.

В простейшем случае XSLT-процессор получает на входе два документа — входной XML-документ и таблицу стилей XSLT — и создает на их основе выходной документ (рисунок 1).

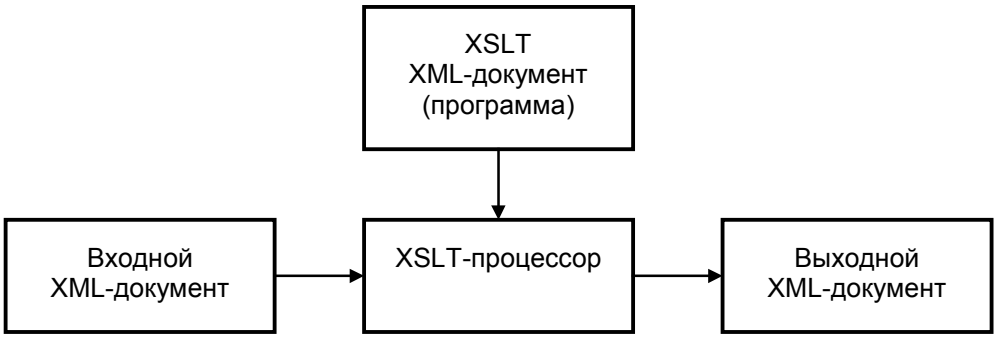

Рисунок 1 — Схема XSL преобразования

Язык XSLT состоит из инструкций, общий вид которых следующий:

```
<префикс:имя-инструкции [атрибут="значение" [атрибут="значение"] ... ]>
  <!-- Тело, если есть -->
</префикс:имя-инструкции>
```

Префиксом обычно является `<xsl>`, но возможны и другие префиксы.

XSLT-документ всегда является состоятельным XML-документом и имеет следующую структуру:

```
<?xml version='1.0' encoding='...' ?>
<xsl:stylesheet version='1.0'
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ... >
  <!-- инструкции (XSLT программа) -->
</xsl:stylesheet>
```

Общий вид программы на языке XSLT приведен в листинге 1.

Листинг xslt 1 — Общий вид программы XSLT

```
<?xml version='1.0' encoding='...'?>
<xsl:stylesheet version='1.0'
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ... >
<!-- Определение формата вывода -->
  <xsl:output method="xml|html|text" encoding="..." indent="yes|no"/>
  <xsl:strip-space elements="element element ..."/>
  <xsl:preserve-space elements="element element ..."/>
  <xsl:decimal-format name="..."/>
<!-- Загрузка внешних стилей -->
  <xsl:include href="..."/>
  <xsl:import href="..."/>
<!-- Определение переменных и параметров -->
  <xsl:param name='...'>выражение</xsl:param>
  <xsl:variable name="...">выражение</xsl:variable>
<!-- Описания шаблонов -->
  <xsl:template match='/'>    <!-- Шаблон 1 -->
    <!-- Описание действий -->
    <xsl:apply-templates select="запрос xpath"/>
    <!-- Описание действий -->
    <xsl:apply-templates select="запрос xpath"/>
    . . .
  </xsl:template>
  <xsl:template match='запрос xpath'>  <!-- Шаблон 2 -->
    <!-- Описание действий -->
  </xsl:template>
```

```
  <xsl:template name="имя">   <!-- Шаблон 3 -->
    <!-- Описание действий -->
  </xsl:template>
  <!-- Другие шаблоны -->
</xsl:stylesheet>
```

Элементы первого уровня можно поделить на следующие группы:

- элементы, определяющие форматирование вывода;
- элементы, включения внешних стилей;
- элементы определения параметров и переменных;
- элементы описания шаблонов.

Группы могут располагаться в произвольном порядке, при этом первая или вторая группы могут отсутствовать.

Формат выходного документа задают следующие элементы:

`<output ...>` — формат выходного документа, кодировку, отступы;

`<strip-space ...>` — список элементов выходного документа, при выводе которых следует удалить незначащие пробелы;

`<preserve-space ...>` — список элементов выходного документа, при выводе которых следует сохранить незначащие пробелы;

`<decimal-format ...>` — определяет формат вывода десятичных чисел.

Внешние стили подключаются элементами `<include...>` и `<import...>`, которые задают адрес включаемого XSLT-файла. Стили, включаемые вторым оператором, имеют более низкий приоритет, чем стили, определенные в исходном документе.

Элементы `<param...>` и `<variable...>` задают параметры и переменные. Переменные могут быть глобальными и локальными (объявленными в инструкции `template`). В отличие от переменной, параметр можно указывать при вызове XSLT-программы или шаблона (если параметр объявлен в инструкции `template`).

Алгоритм обработки исходного документа задается инструкциями `template` (шаблон). Шаблон с атрибутом `match` задает часть входного документа, к которой будут применяться действия шаблона. Шаблон с атрибутом `name` описывает именованный шаблон (вызывается по имени).

Вызов шаблона с атрибутом `match` производится оператором

```
<xsl:apply-templates select="путь">
```

Вызов именованного шаблона производится оператором

```
<xsl:call-template name="имя" />
```

Список параметров шаблона задается вложенными элементами

```
<xsl:with-param name="имя">значение</xsl:with-param>
```

Общий вид шаблона `template` приведен в листинге 2.

Листинг xslt 2 — общий вид шаблона template

```
<xsl:template match='путь'>
  <!-- Действия -->
  <элемент-1 атрибут="значение" ...> текст
    <элемент-2 атрибут="значение" ...> текст
      ...
      <xsl:apply-templates select="путь" />
      ...
    </элемент-2>
    ...
    <xsl:call-tamplate name="имя">
      <xsl:with-param name="...">...</xsl:with-param>
      <xsl:with-param name="...">...</xsl:with-param>
    </xsl:call-tamplate>
    ...
    <xsl:apply-templates select="путь">
      <xsl:with-param name="...">...</xsl:with-param>
      <xsl:with-param name="...">...</xsl:with-param>
    </xsl:apply-templates>
    ...
  </элемент-1>
  ...
</xsl:template>
```

Для управления последовательностью выполнения инструкций XSLT используются следующие управляющие инструкции (*операторы*):

```
<xsl:for-each select="путь">действия</xsl:for-each>
```

Оператор `for-each` выполняет указанные действия для всех дочерних узлов заданного пути.

```
<xsl:if test="условие">действия</xsl:if>
```

Оператор `if` выполняет указанные действия при истинном условии.

```
<xsl:choose>
  <xsl:when test="условие">действия</xsl:when>
  <xsl:when test="условие">действия</xsl:when>
  ...
  <xsl:otherwise>действия</xsl:otherwise>
</xsl:choose>
```

Оператор `choose` просматривает условия, указанные в атрибутах `test` элементов `when`. Для первого истинного условия выполняются указанные действия. Если все условия ложны, выполняются действия, указанные в элементе `otherwise`.

Как правило, XSLT-программа содержит шаблон для обработки корневого узла:

```
  <xsl:template match='/'>
    <!-- Описание действий -->
  <xsl:template />
```

Этот шаблон вызывается в первую очередь и содержит алгоритм вызова последующих шаблонов. Если указанный шаблон отсутствует,

XSLT-процессор рекурсивно обходит узлы документа, подыскивая подходящий шаблон `<xsl:template match='путь'>` , и в случае успеха выполняет его.

В качестве примера рассмотрим обработку XSLT-процессором XML-документа, приведенного в листинге 3.

Листинг xslt 3 — Файл xslt.xml

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<Root>
  <Part name="Part-1" id='1'>
    <Item name="Item-1-1">Item-1-1 value</Item>
    <Item name="Item-1-2">Item-1-2 value</Item>
  </Part>
  <Part name="Part-3" id='3'>
    <Item name="Item-3-1">Item-3-1 value</Item>
    <Item name="Item-3-2">Item-3-2 value</Item>
  </Part>
  <Part name="Part-2" id='2'>
    <Item name="Item-2-1">Item-2-1 value</Item>
    <Item name="Item-2-2">Item-2-2 value</Item>
    <Item name="Item-2-3">Item-2-3 value</Item>
  </Part>
</Root>
```

Исходный XML-документ состоит из трех уровней — `Root`, `Part` и `Item`. В соответствии с этой структурой, обработка уровней производится отдельными XSLT-программами, хотя это не является обязательным условием. XSLT-Программа второго уровня (`Item`) приведена в листинге 4.

Листинг xslt 4 — Файл item.xsl

```
<?xml version="1.0" encoding="WINDOWS-1251 ?>
<xsl:stylesheet version='1.0'
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Шаблон уровня 2 -->
  <xsl:template name='item' match='Item'>
    <!-- Значение параметра по умолчанию -->
    <xsl:param name='m'>0</xsl:param>
    <P>Элемент <xsl:value-of select='$m'/>: <xsl:value-of
select='.'/></P>
  </xsl:template>
</xsl:stylesheet>
```

Здесь описывается именованный шаблон `item`, обрабатывающий соответствующий узел XML-документа. Шаблон использует параметр `m`, указывающий номер элемента. Шаблон выводит порядковый номер элемента и его значение (используя выражение XPath `"."`).

XSLT-Программа первого уровня приведена в листинге 5.

Листинг xslt 5 — Файл part.xsl

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<xsl:stylesheet version='1.0'
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="item.xsl"/>
  <!-- Шаблон уровня 1 -->
  <xsl:template name='part' match='Part'>
    <!-- Значение параметра по умолчанию -->
    <xsl:param name='n'>0</xsl:param>
    <H2>Раздел <xsl:value-of select='$n'/>:
        <xsl:value-of select='@name'/></H2>
    <TABLE>
      <xsl:for-each select="Item">
        <xsl:sort select="@name" order='ascending'/>
        <xsl:variable name="m" select="position()"/>
        <TR><TD valign='middle' align='left'>
          <DIV class='item'>
            <xsl:call-template name="item">
              <xsl:with-param name='m' select="$m"/>
            </xsl:call-template>
          </DIV>
        </TD></TR>
      </xsl:for-each>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

Здесь описывается именованный шаблон `part`, обрабатывающий соответствующие узлы. Этот шаблон также использует параметр `n`, указывающий номер раздела. Кроме того, он определяет переменную `m`, используемую в качестве параметра вызова шаблона `item`.

XSLT-Программа нулевого уровня приведена в листинге 6.

Листинг xslt 6 — Файл root.xsl

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<xsl:stylesheet version='1.0'
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="part.xsl"/>
  <xsl:output methos="xml" encoding="WINDOWS-1251" indent="yes"/>
  <xsl:template match='/'>  <!-- Главный шаблон -->
    <xsl:param name="count">0</xsl:param>
    <HTML>
      <HEAD>
        <LINK rel='stylesheet' type='text/css' href='style.css'/>
        <TITLE>Contents</TITLE>
      </HEAD>
      <BODY>
        <TABLE align='center' width='200px'>
          <xsl:for-each select="Root/Part">
            <xsl:sort select="@id" order='ascending'/>
            <xsl:variable name="n" select="$count+position()"/>
            <TR><TD valign='middle' align='left'>
              <DIV class='part'>
                <xsl:call-template name="part">
                  <xsl:with-param name='n' select="$n"/>
                </xsl:call-template>
              </DIV>
            </TD></TR>
          </xsl:for-each>
```

```
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Здесь описывается главный шаблон, обрабатывающий корневой узел XML-документа. Программа задает также содержание документа в виде разметки на языке HTML.

В листинге 7 приведен PHP-скрипт, запускающий XSLT-процессор.

Листинг xslt 7 — Файл xslt.php

```
<?php
$domxml = new domDocument();
$domxml->load("xslt.xml");
$domxsl = new domDocument();
$domxsl->load("root.xsl");
$xsl = new xsltProcessor();
$xsl->importStylesheet($domxsl);
$xsl->setParameter('', 'count', '1');
echo $xsl->transformtoXML($domxml);
?>
```

Метод `importStylesheet` транслирует XSLT-документ. Если документ не содержит ошибок или они незначительны (например, отсутствие подключаемых оператором `import` файлов), метод возвращает значение `true`.

Метод `setParameter` задает значение параметра корневого шаблона. Первый параметр метода — пространство имен.

Метод `transformtoXML` импортирует исходный XML-документ в документ XML в соответствии с XSLT-программой. Результат трансформации приведен в листинге 8.

Листинг xslt 8 — Результат XSLT-трансформации

```
<HTML><HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=WINDOWS-
1251">
<LINK rel="stylesheet" type="text/css" href="style.css">
<TITLE>Contents</TITLE>
</HEAD>
<BODY><TABLE align="center" width="200px">
<TR><TD valign="middle" align="left"><DIV class="part">
<H2>Раздел 1: Part-1</H2>
<TABLE>
  <TR><TD valign="middle" align="left">
    <DIV class="item"><P>Элемент 1: Item-1-1 value</P></DIV>
  </TD></TR>
  <TR><TD valign="middle" align="left">
    <DIV class="item"><P>Элемент 2: Item-1-2 value</P></DIV>
  </TD></TR>
</TABLE></DIV></TD></TR>
<TR><TD valign="middle" align="left"><DIV class="part">
<H2>Раздел 2: Part-2</H2>
```

```
<TABLE>
  <TR><TD valign="middle" align="left">
    <DIV class="item"><P>Элемент 1: Item-2-1 value</P></DIV>
  </TD></TR>
  <TR><TD valign="middle" align="left">
    <DIV class="item"><P>Элемент 2: Item-2-2 value</P></DIV>
  </TD></TR>
</TABLE></DIV></TD></TR>
<TR><TD valign="middle" align="left"><DIV class="part">
<H2>Раздел 3: Part-3</H2>
<TABLE>
  <TR><TD valign="middle" align="left">
    <DIV class="item"><P>Элемент 1: Item-3-1 value</P></DIV>
  </TD></TR>
  <TR><TD valign="middle" align="left">
    <DIV class="item"><P>Элемент 2: Item-3-2 value</P></DIV>
  </TD></TR>
  <TR><TD valign="middle" align="left">
    <DIV class="item"><P>Элемент 3: Item-3-3 value</P></DIV>
  </TD></TR>
</TABLE></DIV></TD></TR>
</TABLE></BODY></HTML>
```

Другие методы процессора XSLT:

`bool transformtoURI(class domDocument, string filename)` — записывает документ в указанный файл.

`domDocument transformtoDOM(class domDocument)` — преобразует документ в новое дерево для последующей корректировки.

Для вывода трансформированного документа в текстовом формате в инструкции `<xsl:output...>` необходимо указать значение `text`. При этом XSLT-программа не должна содержать теги HTML.