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

Вл. Пономарев

Технологии программирования распределенных приложений

Учебно-методическое пособие по дисциплине
«Современные технологии программирования»

Озерск, 2015

УДК 681.3.06
П56

Пономарев В.В. Технологии программирования распределенных приложений. Учебно-методическое пособие. Озерск: ОТИ НИЯУ МИФИ, 2015. — 74 с., ил. Редакция 2015-10-10.

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 09.05.01 — «Применение и эксплуатация автоматизированных систем специального назначения» и направлению подготовки 09.03.01 — «Информатика и вычислительная техника».

Электронная версия пособия находится по адресу revol.ponocom.ru.

Рецензенты:
1) Начальник отдела ПО ИВЦ «ФГУП ПО «Маяк» А.Ю. Малышев.
2) Ст. преподаватель кафедры ПМ ОТИ НИЯУ МИФИ А.Ф. Зубаиров.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

1. Введение в язык PHP

1.1. Средства разработки

Для разработки web-приложений и выполнения практических работ требуются средства разработки. Они в значительной мере зависят от того, на какой основе строится web-приложение.

Web-приложение — это приложение, построенное по архитектуре «клиент-сервер», в которой в качестве сервера выступает web-сервер, такой, как Apache, а в качестве клиента — обозреватель web-страниц (иначе называемый браузером), такой, как Google Chrome.

Существует несколько распространенных языков, используемых на стороне сервера. Это PHP, Perl, Python, Ruby, Java, C, языки .NET и другие. В большинстве своем эти языки являются интерпретируемыми. Это означает, что на сервере находятся тексты на этих языках, называемые сценариями (или скриптами), которые в определенные моменты времени транслируются с помощью соответствующего интерпретатора с получением некоторого выходного текста. Этот текст, отправляется обозревателю для отображения. Интерпретатор выбирается web-сервером.

На стороне клиента, как правило, используется язык JavaScript, хотя существуют и другие (Visual Basic, ASP). Основой клиентской части приложения является так называемая объектная модель документа DOM. С ее помощью можно управлять объектами HTML-страниц, придавая web-приложению некоторую интерактивность и видимость непосредственного взаимодействия с пользователем.

Выбор языка программирования на стороне сервера в большой степени влияет на используемые средства разработки. Так, если предполагается писать web-приложение на языке .NET, то требуется использовать web-сервер Microsoft Internet Information Services, а языком программирования сценариев на стороне сервера должен быть ASP .NET.

Нужно понимать, что разработка серверной части приложения непосредственно на сервере сети интернет сопряжена с определенными трудностями, поэтому обычно разработка ведется на локальном компьютере, на котором устанавливаются web-сервер и соответствующие ему другие средства (интерпретаторы и СУБД).

В современном мире web-приложений наибольшую популярность для написания сценариев на стороне сервера приобрел язык PHP. Он используется обычно в сочетании с web-сервером Apache и СУБД MySQL. Для разработки таких web-приложений существует бесплатный пакет под названием Denwer, включающий в себя все необходимое. Этот пакет должен быть установлен на локальный компьютер для выполнения работ по данному курсу.

1.2. Модель «клиент-сервер» и сценарии PHP

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

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

Рисунок 1 — Клиент и сервер

На стороне клиента пользователь в обозревателе вводит в строку адреса так называемый URL (*Universal Resource Locator*, универсальный указатель местоположения ресурса) интересующей его страницы HTML. 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 здесь не рассматриваются).

На рисунке 1 в качестве ресурса выступает сценарий (скрипт). Будем понимать под сценарием программу, выполняющуюся на сервере и формирующую ответ обозревателю 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 сценарии, потому что в этом случае страница просто отправляется обозревателю.

На стороне сервера запрос клиента на получение сценария вызывает запуск интерпретатора сценария (интерпретатора PHP). Интерпретатор использует инструкции сценария для формирования выходного текста на языке HTML, который отправляется клиенту.

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

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

1.3. 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 клиентом и сервером. Страница HTML-запроса показана в листинге 1.

Листинг 1. Форма запроса

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

Как видно из рисунка, код HTML содержит описание формы, состоящей из двух элементов: текстового поля с именем name, равным user (строка 5) и кнопки отправки формы на сервер с надписью «Отправить» (строка 6). Атрибутами тега FORM являются метод GET и сценарий назначения form-answer.php (строка 4).

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

Примерный вид страницы в обозревателе приведен на рисунке 2. Здесь в поле введено имя «Петя».

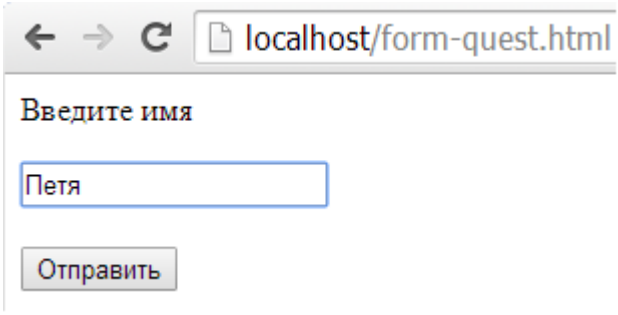

Рисунок 2 — Страница запроса

Ответный сценарий PHP form-ans.php показан в листинге 2.

Листинг 2. Сценарий ответа

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

Внешне сценарий ответа выглядит как страница HTML, за одним небольшим исключением. В строке 4 внутри «скобок» "<?" и "?>" располагается 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 обозревателем во время отправки формы.

Рассмотрим, что происходит при отправке формы (рисунок 3).

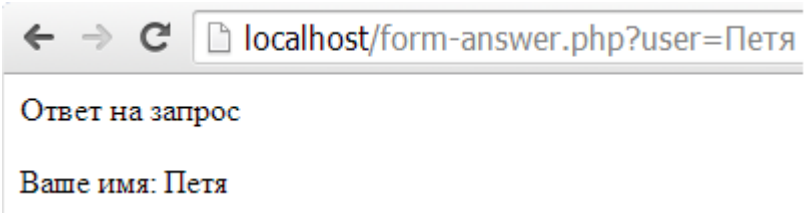

Рисунок 3 — Страница ответа

После ввода имени «Петя» и нажатия кнопки «Отправить» обозреватель сформировал запрос, который имеет вид (рисунок 3):

```
http://localhost/form-answer.php?user=Петя
```

Здесь после URL сценария стоит знак вопроса, за которым следует строка параметров запроса, в данном случае присутствует только один параметр с именем user и значением «Петя». Все параметры, передаваемые методом GET, в сценарии PHP доступны через ассоциативный массив с именем $_REQUEST. Доступ к элементам этого массива осуществляется по имени параметра, как показано в листинге 2, строка 4.

Цель данного простейшего примера — показать, как происходит взаимодействие запроса обозревателя со сценарием PHP.

1.4. Особенности языка PHP

Коротко рассмотрим основные особенности языка PHP:

- код на языке PHP должен быть заключен в «скобки» "<?" и "?>";
- код на языке PHP размещается в сценарии произвольным образом по отношению к другому тексту; этот другой текст воспринимается обозревателем как код HTML;
- любая переменная PHP начинается со знака "$" (доллар);
- простейший способ вставить в страницу HTML значение выражения PHP — это использовать конструкцию **<?=выражение?>**;
- переменные не требуют объявления и имеют переменный тип;

Параметры, посылаемые методом GET, попадают в ассоциативный массив $_REQUEST. Параметры, посылаемые методом POST, попадают в ассоциативные массивы $_REQUEST и $_POST. Дополнительную информацию содержит массив $_SERVER.

Чтобы узнать, какие элементы содержит массив $_SERVER, можно использовать следующий сценарий.

```
<? # массив $_SERVER
print_r($_SERVER);
?>
```

Здесь используется функция PHP с именем print_r, которая выводит значение массива. Содержание массива достаточно велико и зависит от сервера. Часто используются элементы HTTP_HOST, HTTP_USER_AGENT, REQUEST_URI, QUERY_STRING, SCRIPT_NAME.

Язык PHP использует синтаксис, схожий с такими языками, как Си и Java, хотя есть и некоторые отличия.

В дополнение к Си комментарии в этом языке могут иметь вид:

```
# строчный комментарий
```

Операторы и операции языка PHP в основном такие же, как и в языке Си. Некоторые отличия будут показаны в примерах.

Строки могут быть записаны внутри одинарных или двойных кавычек. Строки в двойных кавычках могут включать в себя переносы строк.

Вставить в строку значение выражения можно конкатенацией, а переменные можно вставлять в строки как есть; примеры:

```
$ii = "строка".$jj."строка";
$ii .= "строка $kk строка";
```

Оператором конкатенации является точка ".". Операция ".=" заменяет конструкцию вида $a = $a."строка" на конструкцию $a .= "строка".

1.5. 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 в листингах 1 и 2, можно легко объединить с помощью условного оператора языка PHP.

Смысл заключается в том, чтобы каким-то образом выяснить, является ли запрос обозревателя к сценарию первым или вторым.

При помощи функции isset(переменная) можно определить наличие или отсутствие некоторой переменной. 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.

В листинге 3 приведен код сценария, который использует данное свойство, используя в качестве переменной X значение ассоциативного массива $_REQUEST['user'].

Листинг 3. Запрос и ответ в одном файле

```
<!-- файл f2.php -->
<?php
echo "<HTML><HEAD><TITLE>Сценарий 2</TITLE></HEAD><BODY>";
if (isset($_REQUEST['user'])) {
  // это второй запрос
  echo "<P>Ответ на запрос</P>";
  echo "<P>Ваше имя: ".$_REQUEST['user']."</P>";
} else {
  // это первый запрос
  echo "<P>Введите имя";
  echo "<FORM method='GET' action='".$_SERVER['SCRIPT_NAME']."'>";
  echo "  <P><INPUT type='text' name='user'></P>";
  echo "  <P><INPUT type='submit' value='Отправить'></P>";
  echo "</FORM>";
}
echo "</BODY></HTML>";
?>
```

•

Данный сценарий показывает, как сформировать код HTML при помощи оператора echo. Заметим также, что непосредственно после открывающей скобки "<?" здесь приписано слово php. Это сделано с целью уточнения интерпретатора языка.

1.6. Альтернативный синтаксис

В листинге 4 приведен код сценария, выполняющий те же самые действия, что и код в листинге 3.

Листинг 4. Альтернативный синтаксис

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

•

Здесь, очевидно, основная часть текста — код HTML, содержащий несколько «вкраплений» инструкций PHP. Это возможно вследствие использования «альтернативного синтаксиса» операторов PHP.

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.

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

1.7. PHP и MySQL

MySQL — 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 сценарием — подключение к базе данных. Для этой цели будем использовать специальный дополнительный сценарий conn.php (листинг 5).

Листинг 5. Сценарий для подключения к базе данных

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

•

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.

Программно создадим таблицу пользователей с именем t_user, в которой есть два поля: id_us для уникального идентификатора, us_pw для пароля. Пример сценария t_create.php показан в листинге 6.

Листинг 6. Создание таблицы базы данных

```
01: <?php # файл t_create.php
02: include_once "conn.php";
03: if ($dben != 1) die("Нет подключения: $dben <BR>");
05: $qs = "drop table if exists t_user";
06: $qi = mysql_query($qs, $link_id);
07: if (!$qi) die("Ошибка удаления таблицы<BR>");
08: $qs = "create table t_user (
09:   id_us varchar(32) not null default '',
10:   us_pw varchar(32) not null default '',
11:   primary key (id_user)
12: )";
13: $qi = mysql_query($qs, $link_id);
14: if (!$qi) { echo "Ошибка: "; die(mysql_error()); }
15: $qs = "INSERT t_user (id_us, us_pw)
16:        VALUES ('admin', '".md5("123456")."')";
17: $qi = mysql_query($qs, $link_id);
18: mysql_close();
19: ?>
```

•

В строке 2 используется инструкция для однократного включения файла подключения к базе данных. В строке 3 проверяется переменная $dben, и при отсутствии подключения сценарий завершается функцией die, при этом выводится некоторое сообщение. В строке 6 с помощью функции mysql_query выполняется SQL-запрос на удаление таблицы. В строчках 8-12 формируется SQL-запрос на создание таблицы. В конце сценария формируется SQL-запрос на добавление новой записи.

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

После того, как таблица сформирована, можно приступить к разработке сценария, осуществляющего авторизацию. Вопросы безопасности здесь для простоты не рассматриваются.

Сценарий auth.php приведен в листинге 7.

Листинг 7. Авторизация

```
01: <?php # файл auth.php
02: include "conn.php";
03: if ($dben != 1) die("Нет подключения: $dben ");
04: $login = 0;
05: if (isset($_POST['user'])) {
06:   $user = substr($_POST['user'], 0, 32);
07:   $pass = substr($_POST['pass'], 0, 32);
08:   $qs = "SELECT us_pw FROM t_user WHERE id_us='$user';";
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_pw']) $login = 1;
14:   }
15: }
16: ?>
17: <HTML><HEAD><TITLE>Авторизация</TITLE></HEAD><BODY>
18: <H2>Авторизация</H2>
19: <? if ($login == 1): ?>
20:   <P>Ваше имя: <?=$user?></P>
21: <? else: ?>
22:   <FORM method="POST" action="<?$_SERVER['SCRIPT_NAME']?>">
23:     <P>Введите имя</P>
24:     <P><INPUT type="text" name="user" size="32"></P>
25:     <P>Введите пароль</P>
26:     <P><INPUT type="password" name="pass" size="32"></P>
27:     <P><INPUT type="submit" value="Отправить"></P>
28:   </FORM>
29: <? endif ?>
30: </BODY></HTML>
```

•

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.

1.8. Классы и PHP

Язык PHP является объектно-ориентированным, и, соответственно, в нем можно создавать классы. В листинге 8 приведен пример простого класса для работы с базой данных.

Листинг 8. Класс базы данных

```
<?php # файл db_sql.php
class db_sql {
  public $host = "",
  $db = "",
  $us = "",
  $pw = "",
  $error = "";
  $link_id = 0,
  public function connect(){
    $this->link_id = mysql_connect($this->host, $this->us, $this->pw);
    if (!$this->link_id) {
      $this->error = mysql_error();
      return false;
    }
    if (!mysql_select_db($this->db, $this->link_id)){
      $this->error = mysql_error();
      mysql_close();
      return false;
    }
    return true;
  }
  // другие методы
}
?>
```

•

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 в листинге 9.

Листинг 9. Использование класса базы данных

```
<?php # файл db_use.php
include_once "db_sql.php";
$db = new db_sql;
$db->host = "localhost";
$db->dbname = "database_name";
$db->user = "user_name";
$db->pass = "password";
if ($db->connect()) {
} else {
}
?>
```

•

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 в листинге 10.

Листинг 10. Наследование класса

```
<?php # файл db_admin.php
require_once "db_sql.php";
class db_admin extends db_sql {
  $host = "localhost";
  $db->dbname = "database_name";
  $db->user = "user_name";
  $db->pass = "password";
  function __construct() {
    parent::__construct();
    . . .
  }
}
?>
```

•

Классы PHP используют одиночное наследование классов, но множественное наследование интерфейсов. Методы интерфейсов объявляются с ключевым словом public и не имеют тела. Наследование интерфейсов указывается с помощью ключевого слова implements.

Пример объявления и наследования интерфейсов:

Листинг 11. Наследование интерфейсов

```
interface IFigure {
  public function draw();
}

interface IColor {
  public function setColor($color);
  public function getColor();
}

class Circle implements IFigure, IColor {
  public function draw() { . . . }
  public function setColor($color) { . . . }
  public function getColor() { . . . }
}
```

•

1.9. Сессии

Одна из частых задач, которая возникает при программировании web-приложений, заключается в передаче данных при переходе от одной страницы приложения к другой.

Рассмотрим сценарий авторизации пользователя (листинг 7).

После того, как пользователь авторизовался, в сценарии сформировалось значение переменной $login, указывающее на вход в приложение. При этом страница авторизации меняет свой вид: форма авторизации исчезает, вместо нее появляется начальная страница. До тех пор, пока пользователь остается на данной странице, переменная $login доступна. Как только пользователь уходит с этой страницы (загружается другая страница), переменная $login теряет свое значение.

Для сохранения данных при переходах между страницами используется механизм *сессий*. Переменные хранятся на сервере и доступны с помощью глобального ассоциативного массива $_SESSION.

Рассмотрим пример авторизации пользователя, в котором используются переменные сессии (листинг 12).

Листинг 12. Авторизация с переменными сессии

```
01: <?php # авторизация
02: session_start();
03: include "conn.php";
04: include "api.php";
05: $login = 0;
06: if (!isset($_SESSION['user'])) $_SESSION['user'] = "";
07: if (!isset($_SESSION['pass'])) $_SESSION['pass'] = md5("");
08: if (isset($_POST['user'])) $_SESSION['user'] = $_POST['user'];
09: if (isset($_POST['pass'])) $_SESSION['pass'] = md5($_POST['pass']);
10: $login = isvalidus($link_id, $_SESSION['user'], $_SESSION['pass']);
11: ?>
•
```

Первой строкой сценария должен быть вызов функции session_start.

В строчках 6 и 7 проверяется, существуют или нет переменные сессии с именами "user" и "pass". Если переменных нет, им присваиваются некоторые значения по умолчанию. В строчках 8 и 9 проверяется, были посланы параметры формы "user" и "pass". Если да, то переменным сессии присваивается значение этих переменных, причем значение пароля шифруется функцией md5. В строке 10 функция isvalidus проверяет логин и пароль, и возвращает значение 0 или 1 в переменную $login.

Включаемый файл api.php содержит описание функции isvalidus, которая выполняет запрос к базе данных. Для проверки входа в другом сценарии достаточно включить строку 10.

Для удаления переменной сессии используется функция unset.

1.10. Использование куки

Куки (*cookie*), — это поименованная порция информации небольшого объема, сохраняемая на стороне клиента. Куки используются обозревателями для сохранения данных и персональных настроек пользователя в перерывах между сеансами работы с web-приложениями.

Для установки куки используется функция setcookie, которая посылает обозревателю *заголовок*. В связи с этим перед применением этой функции сценарий не должен посылать обозревателю никакого другого текста, даже пробела или пустой строки. В противном случае при интерпретации сценария будет получено сообщение о невозможности вывода заголовка. Функция setcookie имеет следующий вид:

```
int setcookie($name [, $value, $expire, $path, $domain, $secure]);
```

Обязательным является только название куки. Другие параметры:

$value — значение;

$expire — время жизни;

$path — расположение (путь в домене, например, "/coo/");

$domain — домен (например, "example.com");

$secure — использовать защищенный канал для передачи куки;

В новых версиях функции есть еще параметр $httponly. Его значение true указывает, что куки будут доступны только через протокол http. При этом куки будут недоступны скриптовым языкам обозревателя.

Время жизни куки задается в секундах от начального времени. Чтобы, например, задать время жизни 1 час, можно использовать вызов функции time следующим образом:

```
$expire = time() + 3600;
```

Если задать время жизни 0 секунд, куки сохраняются только до конца сессии обозревателя.

Примеры установки куки:

```
setcookie("cookie_1", "some info"); # 0 секунд
setcookie("cookie_2", "some info", time() + 86400); # 1 сутки
setcookie("cookie_3", "some info", 0, "", "", false, true); # httponly
setcookie("cookie_3"); # удалить куки cookie_3
```

Для получения куки можно использовать глобальный ассоциативный массив $_COOKIE, а также $_REQUEST.

Пример получения значения куки cookie_2:

```
echo $_COOKIE['cookie_2'];
```

1.11. Загрузка файлов на сервер

Одной из задач web-приложения часто является получение файлов пользователя, таких, как изображения или файлы данных. В этом случае используется механизм загрузки файлов на сервер.

На стороне клиента для загрузки файлов используется HTML элемент формы INPUT типа file. В листинге 13 приведен пример сценария, в котором элемент для загрузки файла имеет параметр name, равный upload.

Листинг 13. Загрузка файла на сервер

```
01: <?php
02: $loaded = 0;
03: $err = "Выберите файл для загрузки";
04: if (isset($_FILES['upload'])) {
05:   $tmp = $_FILES['upload']['tmp_name'];
06:   $name = $_FILES['upload']['name'];
07:   if (file_exists($tmp)) {
08:     move_uploaded_file($tmp, $name);
09:     $loaded = 1;
10:   } else {
11:     $err = "Ошибка загрузки";
12:   }
13: }
14: ?>
15: <HTML><HEAD><TITLE>Загрузка файла</TITLE></HEAD>
16: <BODY>
17: <?if($loaded==0):?>
18:   <FORM method="POST" enctype="multipart/form-data">
19:     <P><?=$err?></P>
20:     <P><INPUT type="file" name="upload"></P>
21:     <P><INPUT type="submit" value="Загрузить"></P>
22:   </FORM>
23: <?else:?>
24:   <P>Файл загружен</P>
25:   <? print_r($_FILES['upload']); ?>
26: <?endif?>
27: </BODY></HTML>
```

•

После отправки формы на стороне сервера формируется элемент upload массива $_FILES, содержащий информацию о загруженном файле.

Доступна следующая информация:

name — оригинальное наименование файла;

type — mime-тип загруженного файла;

tmp_name — временное размещение загруженного файла;

error — состояние загрузки (признак ошибки);

size — размер загруженного файла.

В примере содержимое массива $_FILES['upload'] выводится с помощью функции print_r в строке 25.

Таким образом, массив $_FILES является многомерным, и получить элемент, относящийся к загруженному файлу, можно при помощи двукратной индексации этого массива, например $_FILES['upload']['name'].

После загрузки сценарий перемещает временный файл в требуемое место расположения с помощью функции move_uploaded_file (в строке 8). Можно использовать также функцию copy. При этом файлу понадобится составить подходящее название, оставив имеющееся расширение. В примере используется оригинальное наименование файла.

При необходимости ограничить размер файла следует проверять размер загруженного файла $_FILES['upload']['size'].

Параметр error может принимать следующие значения:

UPLOAD_ERR_OK — загрузка без ошибок;

UPLOAD_ERR_NO_FILE — файл не выбран;

UPLOAD_ERR_INI_SIZE — размер файла превышает значение, заданное директивой файла ini.php;

UPLOAD_ERR_FORM_SIZE — размер файла превышает значение, заданное полем UPLOAD_ERR_FORM_SIZE формы;

UPLOAD_ERR_PARTIAL — файл закачан частично.

1.12. Отправление почтовых сообщений

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

```
bool mail(кому, тема, сообщение [, заголовки]);
```

Параметр «кому» — это строка, содержащая список адресов получателей, разделенных пробелами. Параметр «тема» содержит тему сообщения. Параметр «сообщение» содержит тело сообщения в виде строки или нескольких строк, разделенных символом перевода строки "\n".

С помощью параметра «заголовки» можно добавить в электронное письмо дополнительные заголовки, такие, как "From", "Reply-To".

Пример отправки сообщения:

```
mail("123@mail.ru 456@mail.ru", "Test",
    "Test message.\nDon't reply.",
    join("\r\n", array("From: abc@mail.ru", "Reply-To: abc@mail.ru")));
```

Чаще используются не простые почтовые сообщения, а сообщения с вложениями. Такие электронные письма имеют сложную структуру, используют кодирование сообщений, но позволяют отправлять не только текст, но и другую информацию, например, картинки. В листинге 14 приведен пример простого класса для отправки писем с вложениями.

Листинг 14. Класс для создания почтового сообщения с вложениями

```
<?php # файл mpmail.php класс для создания e-mail с вложениями
class mpmail {
  public $parts, $to, $from, $headers, $subject, $body;
  function __construct(){
    $this->parts=array();
    $this->to=""; $this->from="";
    $this->headers=""; $this->subject=""; $this->body="";
  }
  function attach($msg, $name="", $ctype="application/octetstream"){
    $this->parts[] = array("ctype" => $ctype, "message" => $msg,
      "encode" => "base64", "name" => $name );
  }
  private function build_part($part){
    $msg = $part['message'];
    $msg = chunk_split(base64_encode($msg));
    return "Content-Type: "
      .$part['ctype'].($part['name']? "; name = \""
      .$part['name']."\"" : "")
      ."\nContent-Transfer-Encoding: base64\n\n$msg\n";
  }
  private function build_multipart(){
    $bnd = "b".md5(uniqid(time()));
    $mp = "Content-Type: multipart/mixed; boundary = $bnd\n\n
      This is a MIME encoded message.\n\n--$bnd";
    for ($i = sizeof($this->parts) - 1; $i >= 0; $i--) {
      $mp .= "\n".$this->build_part($this->parts[$i])."--$bnd";
    }
    return $mp .= "--\n";
  }
  private function get_mail(){
    $mime = "";
    if (!empty($this->from)) $mime.="From: ".$this->from. "\n";
    if (!empty($this->headers)) $mime.=$this->headers. "\n";
    if (!empty($this->body)){
      $this->attach($this->body, "", "text/plain; charset=utf-8;");
    }
    $mime.="MIME-Version: 1.0\n".$this->build_multipart();
    return $mime;
  }
  function send(){
    $mime = $this->get_mail();
    return mail($this->to, $this->subject, "", $mime);
  }
}
?>
```

•

Для отправки сообщения также используется функция mail, при этом сообщение отсутствует (пустая строка), а вложения формируют блок заголовков. Вложения прикрепляются вызовом метода attach, письмо отсылается методом send.

Пример использования класса приведен в листинге 15.

Листинг 15. Отправка письма с вложением

```
01: <?php
02: include_once "mpmail.php";
03: $img = file_get_contents("lookatme.jpg");
04: $mail = new mpmail();
05: $mail->to = "abc@mail.ru";
06: $mail->subject = "Картинка";
07: $mail->attach($img, "lookatme.jpg", "image/jpg");
08: echo $mail->send();
?>
•
```

В строке 2 в сценарий включается приведенный выше файл класса mpmail.php. В строке 3 содержимое картинки считывается целиком в переменную $img. В строке 4 создается объект $mail, для которого далее указываются свойства to (получатель) и subject (тема). В строке 7 картинка прикрепляется к почтовому сообщению. В строке 8 почтовое сообщение отправляется, результат функции посылается обозревателю. Для простоты никакие другие свойства и заголовки не формируются.

Чтобы с помощью данного класса отправить простое текстовое письмо, нужно присвоить свойству body текст сообщения, при этом текст будет преобразован во вложение.

Во время отладки web-приложения в среде Denwer почтовое сообщение размещается в папке tmp\!sendmail.

Если в режиме отладки отправляется письмо для подтверждения регистрации в web-приложении, нужно найти его в указанной папке, открыть и самостоятельно выполнить действия, предписанные письмом. Письмо имеет расширение .eml, а наименование составляется из даты и времени отправки, например "2015-01-01_12-00-00.eml".

1.13. Функции PHP

Здесь приведены наиболее важные функции интерпретатора PHP для работы с основными объектами сценариев. Приведенные примеры были проверены в интерпретаторе PHP версии 5.5.25.

1.13.1. Работа со строками

При проверке пустой строки следует учитывать, что правильный результат получится в случае, если оба операнда являются строками. В случае если один из операндов не является строкой, результат может зависеть от версии интерпретатора PHP. Поэтому рекомендуется перед использованием сравнения проверить, как конкретный интерпретатор выполняет сравнение (листинг 16).

Листинг 16. Проверка пустой строки

```
01: <?php
02: $str = "";
03: echo (($str ==  false) ? "1" : "0")."<BR>"; # 1
04: echo (($str ==     "") ? "1" : "0")."<BR>"; # 1
05: echo (($str ==      0) ? "1" : "0")."<BR>"; # 1
06: echo (($str === false) ? "1" : "0")."<BR>"; # 0
07: echo (($str ===    "") ? "1" : "0")."<BR>"; # 1
08: echo (($str ===     0) ? "1" : "0")."<BR>"; # 0
09: ?>
```

•

При сравнении строк следует учитывать, что функции сравнения могут возвращать любые числа, — 0, положительные и отрицательные, результат может зависеть от версии интерпретатора. Для сравнения без учета регистра используется функция strcasecmp (листинг 17).

Листинг 17. Сравнение строк

```
01: <?php # кодировка сценария utf-8
02: # латиница
03: echo strcmp("abc", "abc" )."<BR>"; #  0
04: echo strcmp("abc", "ABC" )."<BR>"; #  1
05: echo strcmp("abc", "ab"  )."<BR>"; #  1
06: echo strcmp("abc", "abcd")."<BR>"; # -1
07: # кириллица
08: echo strcmp("абв", "абв" )."<BR>"; #  0
09: echo strcmp("абв", "АБВ" )."<BR>"; #  1
10: echo strcmp("абв", "аб"  )."<BR>"; #  2
11: echo strcmp("абв", "абвг")."<BR>"; # -2
12: ?>
```

•

При определении длины строки нужно учитывать кодировку, так как функции вычисления длины могут возвращать неожиданные результаты. Некоторые функции для строк имеют аналоги для многобайтных кодировок, они начинаются с префикса "mb_" (листинг 18).

Листинг 18. Длина строки

```
01: <?php # кодировка сценария utf-8
02: $a = "abc"; $b = "абв";
03: echo strlen($a)."<BR>";              # 3
04: echo strlen($b)."<BR>";              # 6
05: echo mb_strlen($a)."<BR>";           # 3
05: echo mb_strlen($b)."<BR>";           # 6
06: echo mb_strlen($b, "utf-8")."<BR>"; # 3
07: ?>
```

•

При поиске подстроки следует выяснить, что возвращает функция strpos в случае неудачи. Когда подстрока находится в начале строки, функция возвращает 0, который сравним с false (листинг 19).

Листинг 19. Поиск подстроки

```
<?php
$s = "abc";
echo "'".strpos($s, "a")."'<BR>";                          # '0'
echo ((strpos($s, "a") ==  false) ? "1" : "0")."<BR>"; # 1
echo ((strpos($s, "a") === false) ? "1" : "0")."<BR>"; # 0
echo ((strpos($s, "a") ==      0) ? "1" : "0")."<BR>"; # 1
echo ((strpos($s, "a") ===     0) ? "1" : "0")."<BR>"; # 1
echo "'".strpos($s, "z")."'<BR>";                          # ''
echo ((strpos($s, "z") ==  false) ? "1" : "0")."<BR>"; # 1
echo ((strpos($s, "z") === false) ? "1" : "0")."<BR>"; # 1
echo ((strpos($s, "z") ==      0) ? "1" : "0")."<BR>"; # 1
echo ((strpos($s, "z") ===     0) ? "1" : "0")."<BR>"; # 0
echo ((strpos($s, "z") ==     "") ? "1" : "0")."<BR>"; # 1
echo ((strpos($s, "z") ===    "") ? "1" : "0")."<BR>"; # 0
?>
```

•

Проблема поиска подстроки заключается также в том, что для кодировки utf-8 нужно использовать функцию mb_strpos. Для минимизации ошибок поиска подстроки можно использовать собственную функцию m_strpos, которая будет возвращать значение –1 в случае неудачи, или результат функции mb_strpos в случае удачи (листинг 20).

Листинг 20. Пользовательская функция поиска подстроки

```
<?php # кодировка сценария utf-8
function m_strpos($str, $match, $start = 0, $codepage = "utf-8") {
  $res = mb_strpos($str, $match, $start, $codepage);
  return ($res === false) ? -1 : $res;
}
# латиница
echo m_strpos("YZ", "Y")."<BR>"; #  0
echo m_strpos("YZ", "Z")."<BR>"; #  1
echo m_strpos("YZ", "F")."<BR>"; # -1
# кириллица
echo m_strpos("БГ", "Б")."<BR>"; #  0
echo m_strpos("БГ", "Г")."<BR>"; #  1
echo m_strpos("БГ", "Я")."<BR>"; # -1
?>
```

•

Для использования функции m_strpos в сценарии, кодировка которого, например, windows-1251, нужно указать эту кодировку как значение по умолчанию в функции m_strpos.

Подстроку возвращает функция substr для однобайтной кодировки или mb_substr для многобайтной кодировки. При использовании символов кириллицы здесь также нужно быть осторожным (листинг 21). Для минимизации ошибок в этом случае также можно использовать собственную функцию (листинг 22).

Листинг 21. Получение подстроки

```
01: <?php # кодировка сценария utf-8
02: # латиница
03: echo substr("abcde",  0   )."<BR>";            # abcde
04: echo substr("abcde",  0, 3)."<BR>";            # abc
05: echo substr("abcde",  2, 2)."<BR>";            # cd
06: echo substr("abcde", -3, 2)."<BR>";            # cd
07: # кириллица
08: echo substr("абвгд",  0   )."<BR>";            # абвгд
09: echo substr("абвгд",  0, 3)."<BR>";            # а□
10: echo substr("абвгд",  0, 6)."<BR>";            # абв
11: echo mb_substr("абвгд", 0, 6)."<BR>";          # абв
12: echo mb_substr("абвгд", 0, 3, "utf-8")."<BR>"; # абв
13: echo mb_substr("абвгд", 3, 2, "utf-8")."<BR>"; # гд
14: ?>
```

Листинг 22. Пользовательская функция получения подстроки

```
01: <?php # кодировка сценария utf-8
02: function m_substr($str, $start, $len = null, $codepage = "utf-8") {
03:   if ($len === null) $len = mb_strlen($str, $codepage);
04:   return mb_substr($str, $start, $len, $codepage);
05: }
06: # латиница
07: echo m_substr("abcde",  0   )."<BR>"; # abcde
08: echo m_substr("abcde",  0, 3)."<BR>"; # abc
09: echo m_substr("abcde",  2, 2)."<BR>"; # cd
10: echo m_substr("abcde", -3, 2)."<BR>"; # cd
11: # кириллица
12: echo m_substr("абвгд",  0   )."<BR>"; # абвгд
13: echo m_substr("абвгд",  0, 3)."<BR>"; # абв
14: echo m_substr("абвгд",  2, 2)."<BR>"; # вг
15: echo m_substr("абвгд", -3, 2)."<BR>"; # вг
16: ?>
```

Замену символов строк выполняет функция str_replace (листинг 23).

Листинг 23. Замена символов

```
01: <?php
02: $s = "abcdef";
03: echo str_replace("cd", "123", $s)."<BR>";  # ab123ef
04: $s = "абвгде";
05: echo str_replace("вг", "123", $s)."<BR>";  # аб123де
06: ?>
```

Без учета регистра *замену символов* выполняет функция str_ireplace.

Подстановку символов выполняет функция strtr (листинг 24). Символы из второго параметра заменяются символами из третьего параметра в соответствующих позициях. С помощью этой функции можно выполнить транслитерацию символов кириллицы в латинские символы.

Листинг 24. Подстановка символов

```
01: <?php
02: $s = "aabb";
03: echo strtr($s, "ab", "ba")."<BR>"; # bbaa
04: $s = "ааббб";
05: echo strtr($s, "аб", "ба")."<BR>"; # ббаа
06: ?>
```

•

Отрезание пробелов выполняют функции:
trim (пробелы с обеих сторон),
ltrim (пробелы слева),
chop (пробелы справа).

Под пробелами здесь подразумеваются символы пробела, табуляции "\t", перевода строки "\n", возврата каретки "\r".

Изменение регистра выполняют функции:
strtoupper (в верхний регистр),
strtolower (в нижний регистр),
ucfirst (переводит в верхний регистр только первую букву).

Функции правильно работают с символами латиницы. При использовании символов кириллицы в кодировке WINDOWS-1251 лучше использовать многобайтные аналоги первых двух функций (листинг 25). При использовании обычных функций в этом случае, возможно, потребуется установить локаль функцией setlocale (пример см. листинг 29).

Листинг 25. Изменение регистра

```
01: <?php # кодировка сценария windows-1251
02: echo strtoupper("абв")."<BR>";                    # АБВ
03: echo strtolower("АБВ")."<BR>";                    # абв
04: echo mb_strtoupper("абв", "windows-1251")."<BR>"; # АБВ
05: echo mb_strtolower("АБВ", "windows-1251")."<BR>"; # абв
06: echo ucfirst("абв")."<BR>";                       # Абв
07: ?>
```

•

Функция ucfirst не имеет многобайтного аналога и не работает с символами кириллицы в кодировке utf-8 (листинг 26).

Листинг 26. Изменение регистра символов кириллицы в utf-8

```
01: <?php # кодировка сценария utf-8
02: echo mb_strtoupper("абв", "utf-8")."<BR>"; # АБВ
03: echo mb_strtolower("АБВ", "utf-8")."<BR>"; # абв
04: echo ucfirst("абв")."<BR>";                # абв
05: ?>
```

•

Для изменения регистра строк с кириллицей в кодировке WINDOWS-1251, можно использовать свои функции (листинг 27). Для простоты в примере показана замена только первых трех символов алфавита.

Листинг 27. Изменение регистра символов кириллицы windows-1251

```
01: <?php # кодировка сценария windows-1251
02: function toupper($s) {
03:   return strtr($s, "абв", "АБВ");
04: }
05: function tolower($s) {
06:   return strtr($s, "АБВ", "абв");
07: }
08: echo toupper("абв")."<BR>"; # АБВ
09: echo tolower("АБВ")."<BR>"; # абв
10: ?>
```

•

Одиночный символ возвращает функция chr, параметром которой является код символа. Обратной является функция ord, параметром которой является символ. Эти функции не работают с utf-8.

Символ строки возвращает операция *индексирования*. Индексация символов идет от нуля, первый символ строки $s — $s[0].

Разбиение строки в массив выполняет функция explode. Преобразование массива в строку выполняет функция join или implode (листинг 28).

Листинг 28. Разбиение строки

```
01: <?php
02: $a = explode(" ", "aa bb cc"); # разбиение по пробелу
03: print_r($a); echo "<BR>";      # Array( [0] => aa [1] => bb [2] => cc )
04: echo join(" ", $a)."<BR>";     # aa bb cc
05: $b = explode("я", "ББяГГяДД"); # разбиение по букве я
06: print_r($b); echo "<BR>";      # Array( [0] => ББ [1] => ГГ [2] => ДД )
07: echo implode("я", $b)."<BR>"; # ББяГГяДД
08: ?>
```

•

Для кодирования и декодирования URL используются функции:
urlencode (кодирование в формат URL),
urldecode (декодирование формата URL).

Для вывода числовых значений используется функция sprintf. Формат вывода числа определяется строкой формата, примерно так же, как в языке Си (листинг 29).

Листинг 29. Форматирование вывода чисел

```
01: <?php
02: echo sprintf("%d <BR>", 1.25);     # 1
03: echo sprintf("%g <BR>", 1.25);     # 1.25
04: setlocale(LC_ALL, "");             # установка локали
05: echo sprintf("%g <BR>", 1.25);     # 1,25
06: echo sprintf("%1.1f <BR>", 1.25); # 1,2
07: ?>
```

•

Важно: установка локали может влиять на другие сценарии сессии.

1.13.2. Работа с массивами

Массивы в PHP — это списки и ассоциативные массивы. Списки индексируются целыми числами от нуля, а ассоциативные массивы индексируются ключами, заданными во время создания элементов.

Списки и массивы создает функция array (листинг 30). Функции count и sizeof возвращают число элементов массива (функции равноправны).

Листинг 30. Создание массива с помощью функции array

```
01: <?php
02: $a = array(1, 2);                 # список
03: $b = array("a" => 1, "b" => 2); # ассоциативный массив
04: print_r($a); echo "<BR>";         # Array ( [0] => 1 [1] => 2 )
05: print_r($b); echo "<BR>";         # Array ( [a] => 1 [b] => 2 )
06: echo count($a)."<BR>";            # 2
07: echo sizeof($b)."<BR>";           # 2
08: ?>
```

•

Задавать элементы массива можно с помощью операции индексирования (листинг 31), при этом элементы добавляются в конец массива (строки 3-5, 8-10, 13-15), а использование функции array не требуется.

Листинг 31. Элементы массива

```
01: <?php
02: # формирование ассоциативного массива
03: $a[1] = "1";
04: $a[5] = "5";
05: $a["e"] = "e";
06: print_r($a); echo "<BR>"; # Array ( [1] => 1 [5] => 5 )
07: # формирование ассоциативного массива
08: $b["one"] = "P";
09: $b["two"] = "V";
10: $b[0] = "V";
11: print_r($b); echo "<BR>"; # Array ( [one] => P [two] => V [0] => V )
12: # формирование списка автоматически
13: $c[] = "A";
14: $c[] = "B";
15: $c[] = "C";
16: print_r($c); echo "<BR>"; # Array ( [0] => A [1] => B [2] => C )
17: ?>
```

•

Массив $c является списком, создаваемым автоматически. Массивы $a и $b являются ассоциативными массивами (списки индексируются от нуля и не могут иметь нечисловых ключей, идущих не по порядку).

Перебор всех значений списка можно выполнить с помощью параметрического цикла for и переменной — параметра цикла. Перебор всех значений ассоциативного массива выполняется с помощью цикла foreach.

В следующем листинге показаны оба варианта.

Листинг 32. Перебор элементов массива

```
01: <?php
02: # список
03: $a[] = "A";
04: $a[] = "B";
05: $a[] = "C";
06: for ($ii = 0; $ii < count($a); $ii++) {
07:   echo $a[$ii];
08: } echo "<BR>"; # ABC
09: # ассоциативный массив
10: $b["one"] = "P";
11: $b["two"] = "V";
12: $b['three'] = "V";
13: foreach ($b as $k => $v) {
14:   echo "$k=$v ";
15: } echo "<BR>"; # one=P two=V three=V
16: ?>
```

•

Массивы можно объединять с помощью операции "+" (листинг 33).

Листинг 33. Сложение элементов массива

```
01: <?php
02: # список
03: $a[] = "A";
04: $a[] = "B";
05: $a[] = "C";
06: $b[] = "D";
07: $b[] = "E";
08: $c = $a + $b;
09: for ($ii = 0; $ii < count($c); $ii++) {
10:   echo $c[$ii];
11: } echo "<BR>"; # ABC
12: $d[3] = "D";
13: $d[4] = "E";
14: $e = $a + $d;
15: for ($ii = 0; $ii < count($e); $ii++) {
16:   echo $e[$ii];
17: } echo "<BR>"; # ABCDE
18: # ассоциативный массив
19: $x["name"] = "Peter";
20: $x["surname"] = "Ustinov";
21: $y["surname"] = "Vladimir";
22: $y["year"] = 1921;
23: $z = $x + $y;
24: foreach ($z as $k=>$v) {
25:   echo "$v ";
26: } echo "<BR>"; # Peter Ustinov 1921
27: ?>
```

•

При сложении, как видно, элементы с одинаковыми индексами берутся из первого массива.

Для соединения всех элементов нескольких списков используется функция array_merge. Она объединяет списки с одинаковыми индексами.

Для сортировки массивов используются функции:

asort(массив) (сортировка по значениям),

arsort(массив) (сортировка по значениям в обратном порядке),

ksort(массив) (сортировка по ключам),

krsort(массив) (сортировка по ключам в обратном порядке),

uasort(массив, "функция") (пользовательская сортировка по значениям),

uksort(массив, "функция") (пользовательская сортировка по ключам),

natsort(массив) (естественная сортировка),

natcasesort(массив) (естественная сортировка без учета регистра),

sort(массив) (сортировка списка),

rsort(массив) (обратная сортировка списка),

usort(массив, "функция") (пользовательская сортировка списка).

Пользовательские сортировки требуют указания функции сравнения элементов. Естественная сортировка упорядочивает строки с числами в порядке возрастания чисел, а не лексикографически (листинг 34).

Листинг 34. Естественная сортировка

```
01: <?php
02: $a = array("A2", "A1", "A20", "A10");
03: sort($a);
04: print_r($a); echo "<BR>"; # Array([0]=>A1 [1]=>A10 [2]=>A2 [3]=>A20)
05: $a = array("A2", "A1", "A20", "A10");
06: natsort($a);
07: print_r($a); echo "<BR>"; # Array([1]=>A1 [0]=>A2 [3]=>A10 [2]=>A20)
08: for ($ii = 0; $ii < count($a); $ii++) {
09:   echo "$a[$ii] ";
10: } echo "<BR>";              # A2 A1 A20 A10
11: foreach ($a as $k=>$v) {
12:   echo "$v ";
13: } echo "<BR>";              # A1 A2 A10 A20
14: ?>
```

•

Следует обратить внимание на то, что после сортировки соотношение между ключами и значениями не теряется (строки 4 и 7).

Другие функции для массивов:

in_array(элемент, массив) — возвращает признак наличия в массиве указанного значения «элемент».

array_reverse(массив) — переворачивает массив.

shuffle(массив) — перемешивает *список* случайным образом.

array_flip(массив) — меняет местами значения и ключи.

array_slice(список, старт [, длина]) — возвращает часть списка, начиная с элемента старт и размером длина (или до конца списка, если третий параметр не задан).

Для имитации списка или очереди используются функции:
array_push(массив, элемент) — добавляет элемент в массив справа.
array_pop(массив) — возвращает элемент массива справа, удаляет его.
array_unshift(массив) — добавляет элемент в массива слева.
array_shift(массив, элемент) — возвращает элемент слева и удаляет его.

Для имитации множеств используются функции:
array_intersect(массив, список ...) — возвращает «пересечение».
array_diff(массив, список ...) — возвращает «разность».
array_unique(массив) — возвращает массив с уникальными значениями.

Для создания диапазона чисел используется функция range. Она возвращает список, заполненный числами от значения первого параметра до значения второго параметра.

1.13.3. Математические функции

В интерпретатор PHP встроено множество *констант* для математических вычислений:

M_E — число e
M_LOG2E — $log_2(e)$
M_LOG10E — $log_{10}(e)$
M_LN2 — $ln(2)$
M_LN10 — $ln(10)$
M_SQRT2 — $\sqrt{2}$
M_SQRT3 — $\sqrt{3}$
M_SQRT1_2 — $1/\sqrt{2}$
M_EULER — постоянная Эйлера
M_PI — число π
M_PI_2 — $\pi/2$
M_PI_4 — $\pi/3$
M_1_PI — $1/\pi$
M_2_PI — $2/\pi$
M_SQRTPI — $\sqrt{\pi}$
M_2_SQRT_PI — $2/\sqrt{\pi}$
M_LNPI — $ln(\pi)$

Тригонометрические функции:
deg2rad($deg) — перевод градусов в радианы
rad2deg($rad) — перевод радианов в градусы
sin($rad) — синус
cos($rad) — косинус
tan($rad) — тангенс
atan2($y, $x) — арктангенс $y/$x
asin($arg) — арксинус
acos($arg) — арккосинус
atan($arg) — арктангенс

Степенные функции:
sqrt($x) — квадратный корень
log($x) — натуральный логарифм
pow($x, $y) — $x в степени $y
exp($x) — число e в степени $x

Функции *округления*:
abs($val) — абсолютная величина (модуль) $val
round($val) — округляет значение $val до ближайшего целого
ceil($val) — округляет значение $val до минимального целого
floor($val) — округляет значение $val до максимального целого

Функции *перевода из одной системы счисления в другую*:

string base_convert($num, $from, $to) — переводит число $num из системы счисления с основанием $from в систему счисления с основанием $to

Следующие функции преобразуют запись числа в двоичной, восьмеричной или шестнадцатеричной системе счисления в целое десятичное число (слева), или выполняют обратное преобразование (справа):

int bindec($num_str)	string decbin($num)
int octdec($num_str)	string decoct($num)
int hexdec($num_str)	string dechex($num)

Минимальное и максимальное значение из множества заданных значений возвращают функции min и max.

Для генерирования случайных чисел используются функции:

int mt_rand(int $min = 0, int $max = MAX_RAND) — возвращает случайное целое число в диапазоне от $min до $max.

void mt_srand($seed) — настраивает генератор случайных чисел на формирование новой последовательности с источником $seed. Применять эту функция в явном виде нет необходимости, т.к. она вызывается автоматически перед вызовом mt_rand.

1.13.4. Работа с файлами

Работа с файлами в PHP производится примерно так же, как в Си.

В следующем примере создается текстовый файл, который затем считывается построчно.

Листинг 35. Запись и чтение текстового файла

```
<?php
$f = fopen("t.txt", "wt") or die("Ошибка создания");
echo "$f<BR>";
for ($i = 0; $i < 3; $i++) {
  fputs($f, "строка $i\n");
}
fclose($f);
$f = fopen("t.txt", "rt") or die("Ошибка открытия");
echo "$f<BR>";
while (!feof($f)) {
  $s = fgets($f);
  echo "$s<BR>";
}
fclose($f);
?>
•
```

Интересной здесь является конструкция "or die()". В случае ошибки функции fopen вызывается функция die, которая завершает работу сценария с выводом сообщения (или без сообщения, если параметр не задан).

Следующие функции используются для работы с файлами:
string fread($f, int $numbytes) — читает блок $numbytes символов.
int fwrite($f, string $s) — записывает в файл строку $s.
string file_get_contents($path) — считывает файл целиком.
string file_put_contents($path, $data) — записывает файл целиком.
int fseek($f, $offset, $w) — указатель файла смещается на $offset от $w: SEEK_SET (от начала), SEEK_CUR (от текущей точки), SEEK_END (от конца).
int ftell($f) — возвращает текущую позицию в файле.
void fflush($f) — принудительно записывает файл на диск из буфера.
bool file_exists($path) — проверяет существование файла
int filesize($path) — возвращает размер файла.
int filetime($path) — возвращает время модификации файла.
string filetype($path) — возвращает тип файла ("file", "dir", "link", и др.).
Для манипулирования файлами используются функции:
bool copy($path, $new_path) — копирует файл.
bool rename($path, $new_path) — переименовывает файл.
bool unlink($path) — удаляет файл (если на него нет *ссылок*).
string basename($path) — выделяет имя файла из полного пути.
string dirname($path) — выделяет каталог из полного пути.
string realpath(string $path) — возвращает абсолютный путь.
Для манипулирования каталогами используются функции:
bool mkdir($path) — создает каталог.
bool rmdir($path) — удаляет каталог.
bool chdir($path) — меняет текущий каталог.
string getcwd() — возвращает полный путь к текущему каталогу.
Для чтения каталогов используются функции:
$handle = opendir($path) — открывает каталог для чтения.
string readdir($handle) — считывает очередное имя каталога.
void closedir($handle) — закрывает ранее открытый каталог.
void rewinddir($handle) — «перематывает» каталог на начало.

1.13.5. Дата и время

time() — возвращает число миллисекунд с 1.1.1970.
mktime($hour, $minute, $second, $month, $day, $year) — вычисляет timestamp.
date($format [, $timestamp]) — строковое представление даты timestamp.
Следующий сценарий выводит текущую дату.

```
<?php
echo date("d.m.Y H:i:s")."<BR>"; # 31.12.2014 23:59:59
echo date("d.M.y H:i:s")."<BR>"; # 31.Dec.14 23:59:59
?>
```

Дополнительно см. функцию strftime.

2. XML-документы

XML (eXtensible Markup Language, расширяемый язык разметки) — язык описания документов, использующий представление в виде иерархии элементов текста. Язык XML предназначен, например, для хранения структурированных данных, для обмена информацией между программами, для создания более специализированных языков. XML является упрощенным подмножеством языка SGML (Standard Generalized Markup Language).

2.1. Структура XML-документа

XML, как и HTML, для выделения частей текста использует *теги*, то есть *названия*, заключенные в угловые скобки. Однако в отличие от HTML, XML предъявляет *строгие правила* использования тегов и их атрибутов (параметров). Эти правила следующие.

1. Названия тегов не стандартизованы и выбираются *произвольно*, исходя из назначения описываемой информации.

2. Каждому тегу должен соответствовать *закрывающий* тег, имеющий символ косой черты перед названием. Например, если в документе есть тег `<root>`, то далее должен быть закрывающий тег `</root>`:

```
<root>какое-то содержание</root>.
```

Исключением является *пустой тег* (тег, не имеющий содержания).

Назначение пустого тега заключается в выделении некоторой части документа, выражаемой с помощью названия и, возможно, с помощью атрибутов. Пустой тег должен иметь косую черту перед закрывающей треугольной скобкой. Пример пустого тега: `<image src="path" />`.

3. Теги *не могут пересекаться*. Иначе говоря, должна соблюдаться правильная вложенность тегов. Закрывающий тег вложенного элемента должен располагаться перед закрывающим тегом элемента контейнера.

4. Значения атрибутов тегов должны заключаться в кавычки.

Название тега или атрибута начинается с буквы или знака подчеркивания, после чего могут следовать буквы, цифры, знаки подчеркивания, тире. Название тега не должно начинаться с символов `xml` в любом регистре.

Рассмотрим пример XML-документа, приведенный в листинге 36.

Первая строка, — это необязательное описание, указывающее, что это XML-документ определенной версии и в определенной кодировке. Если эта строка есть, перед ней *недопустимы никакие символы*.

Далее расположена секция описания типа документа `DOCTYPE`. Эта секция также не является обязательной. При необходимости эта секция описывает структуру, синтаксис и компоненты XML-документа.

Листинг 36. Пример XML-документа

```
01: <?xml version="1.0" encoding="utf-8" standalone="yes"?>
02: <!DOCTYPE расписание [
03: <!ENTITY prep "Акопян">
04: ]>
05: <расписание>
06:   <день ид="1">
07:     <группа ид="1ПО-33Д">
08:       <пара ид="1" ауд="326" тип="лекция">
09:         <предмет>Теория множеств</предмет>
10:         <препод>&prep;</препод>
11:       </пара>
12:     </группа>
13:   </день>
14:   <!-- comment -->
15:   <?php #PI?>
16:   <![CDATA[chars]]>
17: </расписание>
```

•

На рисунках 4 и 5 приведено дерево узлов документа в листинге 36.

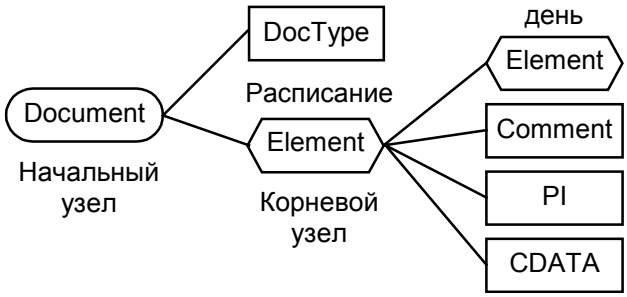

Рисунок 4 — Начальная часть дерева XML-документа

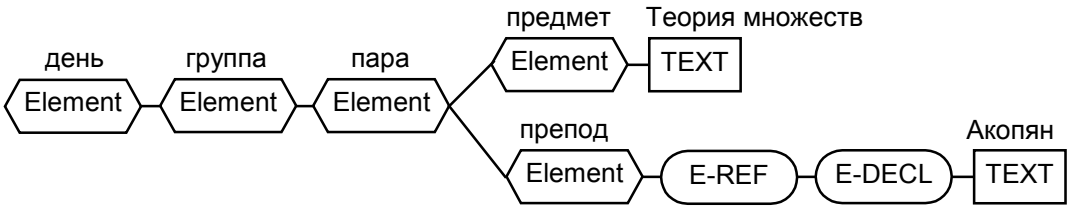

Рисунок 5 — Поддерево «день»

Основным понятием XML-документа является *узел* (node). Иерархия узлов представляет собой *дерево*, начинающееся с узла, который мы будем называть *начальным*. Начальный узел представляет документ в целом, и имеет тип Document.

Базовые узлы для построения дерева имеют тип Element.

Элементом XML-документа называется часть текст между открывающим и закрывающим тегами, вместе с самими этими тегами (или пустой тег целиком). Элементы могут быть вложены один в другой.

Значением элемента является конкатенация текста, который размещен в теге и во всех его внутренних тегах.

Начальный узел может содержать только один узел типа Element, называемый *корневым*. Корневому узлу в листинге 36 соответствует тег <расписание>, в документе не может быть двух таких тегов, а также других тегов этого уровня. Корневой тег представляет собственно содержание документа.

Конечными узлами (не имеющими подузлов) являются узлы типов:

- Text (текст внутри и вне тегов);
- Comment (комментарий);
- ProcessingInstruction (команды приложений, сокращенно PI);
- CDATASection (набор данных символьного типа, сокращенно CDATA).

Как видно из листинга 36, комментарий оформляется так же, как и в HTML, с помощью последовательностей символов "<!--" и "-->".

В XML-документ могут быть включены инструкции для обработки, при этом они заключаются в последовательности символов "<?" и "?>", причем первая последовательность должна указывать на тип приложения, которое эти инструкции может выполнить (в примере PHP).

Символы, которые не должны обрабатываться никаким образом, заключаются в блок символьных данных CDATASection (*Character Data*). Эти символы всегда отображаются так, как записаны, и не ведут к порождению каких-либо узлов (в том числе и теги). В этой секции можно записывать, например, теги HTML или код JavaScript.

Комментарии и команды приложений можно размещать в любом месте документа, но не первой строкой и не внутри тегов (в атрибутах).

Блоки символьных данных CDATASection можно размещать только после корневого тега.

На рисунке 5 есть также узлы, обозначенные E-REF и E-DECL. Для сокращения буквой E обозначен компонент ENTITY. Компоненты — это части документа, внешние или внутренние, описываемые в секции DOCTYPE.

2.2. Секция DOCTYPE

Документ XML называется *правильным* (*well-formed*), если выполнены правила его оформления. Анализатор может его разобрать, но не может проверить синтаксическую корректность.

С точки зрения синтаксиса тег <пара> может располагаться внутри тега <группа>, но не наоборот. С другой стороны, тег <группа> может содержать множество тегов <пара>. *Синтаксис* тегов описывается в секции DOCTYPE, которая содержит ссылки и объявления, в целом составляющие *описание типа документа* (DTD, *Document Type Definition*). В этой секции описываются также *компоненты* и *приложения*.

XML-документ, содержащий описание синтаксиса, и позволяющий проверить корректность следования тегов и правильность атрибутов, называется *состоятельным* или *действительным* (*valid*).

Наиболее общий вид секции DOCTYPE следующий:

```
<!DOCTYPE root-element uri-of-dtd [internal-subset]]>
```

Секция DOCTYPE начинается с литерала <!DOCTYPE. Далее следует название корневого тега root-element. Если с документом используется DTD, то после корневого тега следует URI для DTD. Затем следует так называемое *внутреннее подмножество* internal-subset, границами которого являются квадратные скобки. В конце закрывающая треугольная скобка.

Внутреннее подмножество содержит различные объявления. Они либо полностью описывают тип документа, либо являются дополнением *внешнего подмножества* (*external subset*), которое располагается вне документа (во внешнем DTD). Внешнее и внутреннее подмножества не являются обязательными, равно как и сама секция DOCTYPE.

Пример URI DTD можно найти, например, в HTML файле:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/strict.dtd">
```

Первый литерал после PUBLIC называется *открытым идентификатором* (*public identifier*). Второй литерал называется *системным идентификатором* (*system identifier*), в примере это URL.

Синтаксис внутреннего подмножества показан в листинге 37.

Листинг 37. Основные описатели DOCTYPE

```
<!DOCTYPE root-element [
<!ELEMENT тег описание-внутренних-тегов>
. . .
<!ATTLIST тег описание-атрибутов-тега>
. . .
<!ENTITY имя-внутреннего-компонента "строка">
. . .
<!ENTITY имя-внешнего-компонента SYSTEM адрес>
. . .
<!NOTATION имя SYSTEM "ресурс">
. . .
<!NOTATION имя PUBLIC "ресурс">
. . .
]>
```

•

Объявление ELEMENT описывает порядок следования тегов.

Объявление ATTLIST аналогичным образом описывает возможные атрибуты тегов, их значения, обязательные атрибуты и т.п.

В листинге 38 приведен примерный вид секции DOCTYPE для описания синтаксиса состоятельного XML-документа «расписание».

Листинг 38. Состоятельный XML-документ

```
01: <?xml version="1.0" encoding="utf-8" ?>
02: <!DOCTYPE расписание [
03: <!ELEMENT расписание (день+)            >
04: <!ELEMENT день       (группа+)          >
05: <!ELEMENT группа     (пара+)            >
06: <!ELEMENT пара       (предмет, препод) >
07: <!ELEMENT предмет    (#PCDATA)          >
08: <!ELEMENT препод     (#PCDATA)          >
09: <!ATTLIST день   ид  (1 | 2 | 3 | 4 | 5) #REQUIRED >
10: <!ATTLIST группа ид  CDATA                #REQUIRED >
11: <!ATTLIST пара   ид  (1 | 2 | 3 | 4 | 5) #REQUIRED
12:                  ауд CDATA                #REQUIRED
13:                  тип (лекция | практика) #REQUIRED >
14: ]>
15: <расписание>
16:   <день ид="1">
17:     <группа ид="1ПО-33Д">
18:       <пара ид="1" ауд="326" тип="лекция">
19:         <предмет>Теория множеств</предмет>
20:         <препод>Акопян</препод>
21:       </пара>
22:     </группа>
23:   </день>
24: </расписание>
```

Объявление ENTITY описывает *компоненты*. *Внутренние* компоненты указываются в виде строки здесь же, *внешние* располагаются в файле, адрес которого размещается после ключевого слова SYSTEM.

Внутренний компонент prep использован в листинге 36. В документ компонент вставляется с помощью конструкции &prep;. Компоненты могут содержать теги, их можно использовать в значениях атрибутов.

Пример объявления внешнего компонента:

```
<!ENTITY outer SYSTEM "some.xml">
```

Внешний компонент вставляется так же, как и внутренний.

Не анализируемый внешний компонент содержит данные, не являющиеся XML. Пример такого компонента (с нотацией):

```
<!NOTATION png SYSTEM "image/png">
<!ENTITY logo SYSTEM "img/logo.png" NDATA png>
```

Ключевое слово NDATA обозначает не анализируемые данные.

Существуют также *параметрические компоненты*. Они используются в основном во внешнем подмножестве и здесь не описываются.

Объявление NOTATION указывает имя приложения, которое может обрабатывать заданный в объявлении ресурс. Нотации используются, например, для описания не анализируемых данных, таких, как картинки, аудио или видео информация. Пример нотации не анализируемого компонента приведен выше.

Объявления DTD могут располагаться целиком во внешнем файле. В этом случае секция DOCTYPE может иметь вид

```
<!DOCTYPE root-element SYSTEM "path-to-DTD">
```

2.3. Пространства имен

Пространство имен (*namespace*) — это группа имен элементов и их атрибутов. С помощью пространства имен указывается, что элемент существует в некоторой области имен и должен проверяться относительно DTD этого пространства. Пространства имен позволяют устранять конфликты между одинаковыми именами разных контекстов.

Рассмотрим пример использования элемента <I>.

```
<предмет>Теория <I>множеств</I></предмет>
```

Здесь часть текста "Теория множеств" заключена в тег <I>. Аналогичный тег есть в HTML, он обозначает выделение части текста курсивом.

Анализ объектной модели XML-документа показывает, что этот тег ведет к формированию дополнительных узлов, так как тег <I> является с точки зрения XML обычным элементом. На следующем рисунке показано поддерево предмет приведенного выше фрагмента XML-документа.

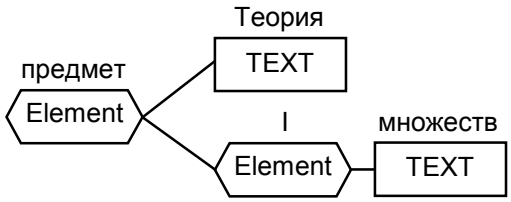

Рисунок 6 — Поддерево «предмет»

Если нужно использовать в данном месте именно тег <I>, соответствующий спецификации HTML, следует указать пространство имен:

```
<предмет>Теория <HTML:I xmlns:HTML="URL">множеств</HTML:I></предмет>
```

При этом в дереве узлов изменится название тега с I на HTML:I. Обратим внимание, что URL не обязательно должен указывать на существующий документ, такой, как "http://www.w3.org/1999/xhtml". Ничего не изменится, если в реальный документ подставить строку "URL".

Пространство имен отделяется от имени тега двоеточием, и, таким образом, является *префиксом*. Имя, состоящее из префикса пространства имен и собственно имени тега, называемся *квалифицированным*.

В одном документе можно использовать несколько пространств имен одновременно, важно, чтобы они были должным образом объявлены в том или ином теге. Если требуется, чтобы какой-то из префиксов использовался по умолчанию, нужно объявить пространство имен, не указывая двоеточие и префикс после xmlns:

```
<?xml version="1.0" encoding="utf-8" ?>
<расписание xmlns="URI-NS-XML" xmlns:HTML="URI-NS-HTML">
  . . .
        <предмет>Теория <HTML:I>множеств</HTML:I></предмет>
  . . .
</расписание>
```

В этом примере используется некое пространство имен по умолчанию, описываемое ссылкой "URI-NS-XML", и пространство имен HTML, описываемое ссылкой "URI-NS-HTML". Для указания, что тег <I> относится к пространству имен HTML, в теге используется квалифицированное имя, то есть "HTML:I". Для других тегов используется простое имя, следовательно, они относятся к пространству имен по умолчанию.

После указания пространства имен тег <HTML:I> не будет вести себя как тег HTML, даже если указать правильный URL. Он приведет к формированию узла "элемент XML" с именем HTML:I, и только. Этот пример показывает, что XML описывает структуру, но не оформление.

Для решения проблемы с тегом <I> можно применить следующий прием. Если результат обработки направляется обозревателю, и необходимо, чтобы часть текста была выделена курсивом на странице HTML, можно использовать секцию CDATA, которая не анализируется:

```
<предмет>Теория <![CDATA[<I>множеств</I>]]></предмет>
```

При этом узлы I и TEXT на рисунке 6 будут заменены узлом CDATA.

2.4. XML-схемы

XML-схемы являются наиболее современным способом описания структуры XML-документа, более гибким и мощным, нежели DTD.

Рассмотрим следующий пример. Предположим, нужно сформировать документ XML, описывающий чек покупки в магазине. Чек содержит информацию о магазине, дате покупки и приобретенных товарах.

Товар описывается наименованием, ценой и количеством.

В листинге 39 приведен пример подобного XML-документа с указанием XML-схемы для поверки правильности.

Листинг 39. XML-документ «чек покупки»

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <чек ид="1"
03:   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04:   xsi:noNamespaceSchemaLocation="cheque.xsd">
05:   <магазин>Три богатыря</магазин>
06:   <дата>2015-01-31</дата>
07:   <товар ид="1">
08:     <название>Молоко</название>
09:     <цена>41.5</цена>
10:     <количество>2</количество>
11:   </товар>
12:   <товар ид="2">
13:     <название>Салат</название>
14:     <цена>300</цена>
15:     <количество>120.5</количество>
16:   </товар>
17: </чек>
```

•

В строке 3 задается пространство имен xsi. В строке 4 указана схема документа — файл cheque с расширением xsd (XML Schema Definition). Атрибут xsi:noNamespaceSchemaLocation показывает, что пространство имен документа чек не задано. Файл cheque.xsd приведен в листинге 40.

Листинг 40. XML-схема документа «чек покупки»

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
03: <xs:element name="чек">
04:   <xs:complexType>
05:     <xs:sequence>
06:       <xs:element name="магазин" type="xs:string"/>
07:       <xs:element name="дата" type="xs:date"/>
08:       <xs:element name="товар" maxOccurs="unbounded">
09:         <xs:complexType>
10:           <xs:sequence>
11:             <xs:element name="название" type="xs:string"/>
12:             <xs:element name="цена" type="xs:decimal"/>
13:             <xs:element name="количество" type="xs:decimal"/>
14:           </xs:sequence>
15:           <xs:attribute name="ид" type="xs:string" use="required"/>
16:         </xs:complexType>
17:       </xs:element>
18:     </xs:sequence>
19:     <xs:attribute name="ид" type="xs:integer" use="required"/>
20:   </xs:complexType>
21: </xs:element>
22: </xs:schema>
```

•

Оба документа были проверены на одном из сайтов, предоставляющих услугу валидации ("http://www.utilities-online.info/xsdvalidation/").

XML-схема является XML-документом, использующим теги пространства имен, заданным URL "http://www.w3.org/2001/XMLSchema" в строке 2. Префикс xs может быть другим, например, xsd. Схема должна быть состоятельным XML-документом.

Корневым элементом XML-схемы является тег schema (строка 2).

Тег (элемент) целевого документа описывается тегом xs:element. При этом атрибут name задает имя целевого тега, атрибут type задает тип данных. Тег xs:element может иметь другие атрибуты, описывающие разные аспекты применения целевого тега.

Различают *простые* и *составные* целевые элементы (теги).

Простой целевой элемент не имеет вложенных элементов. Пример описания простого целевого элемента приведен в строке 6:

```
06:       <xs:element name="магазин" type="xs:string"/>
```

Составной элемент имеет вложенные элементы. В этом случае тег xs:element содержит тег xs:complexType, описывающий вложенные элементы целевого элемента. Если вложенные элементы должны следовать в определенном порядке, то тег xs:complexType содержит тег xs:sequence. Внутри тега xs:sequence описываются вложенные целевые элементы, которые могут быть как простыми, так и составными. Пример описания составного целевого элемента приведен в строчках 8-17:

```
08:       <xs:element name="товар" maxOccurs="unbounded">
09:         <xs:complexType>
10:           <xs:sequence>
11:             <xs:element name="название" type="xs:string"/>
12:             <xs:element name="цена" type="xs:decimal"/>
13:             <xs:element name="количество" type="xs:decimal"/>
14:           </xs:sequence>
15:           <xs:attribute name="ид" type="xs:string" use="required"/>
16:         </xs:complexType>
17:       </xs:element>
```

Здесь атрибут maxOccurs со значением unbounded указывает, что целевой элемент товар может встречаться сколь угодно раз.

Атрибуты целевых элементов описываются тегом xs:attribute. Пример описания целевого атрибута приведен в строке 15:

```
15:           <xs:attribute name="ид" type="xs:string" use="required"/>
```

Атрибут ид элемента товар имеет тип string и является обязательным.

Следует отметить, что XML-схемы более наглядны, легко читаются человеком, позволяют выполнить контроль значений элементов и атрибутов. Этим они выгодно отличаются от DTD, использующих нотацию, похожую на форму Бэкуса-Наура (БНФ).

Приведенная в листинге 40 XML-схема документа чек не является единственно возможным вариантом. В листинге 41 приведена схема, построенная иначе, с разделением описания структуры на части.

Листинг 41. Разделенная XML-схема документа «чек покупки»

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
03: <!-- простые элементы -->
04: <xs:element name="магазин" type="xs:string"/>
05: <xs:element name="дата" type="xs:date"/>
06: <xs:element name="название" type="xs:string"/>
07: <xs:element name="цена" type="xs:decimal"/>
08: <xs:element name="количество" type="xs:decimal"/>
09: <!-- атрибуты -->
10: <xs:attribute name="ид" type="xs:integer"/>
11: <!-- составные элементы -->
12: <xs:element name="товар">
13:   <xs:complexType>
14:     <xs:sequence>
15:       <xs:element ref="название"/>
16:       <xs:element ref="цена"/>
17:       <xs:element ref="количество"/>
18:     </xs:sequence>
19:     <xs:attribute name="ид" type="xs:string" use="required"/>
20:   </xs:complexType>
21: </xs:element>
22: <!-- корневой элемент целевого документа -->
23: <xs:element name="чек">
24:   <xs:complexType>
25:     <xs:sequence>
26:       <xs:element ref="магазин"/>
27:       <xs:element ref="дата"/>
28:       <xs:element ref="товар" maxOccurs="unbounded"/>
29:     </xs:sequence>
30:     <xs:attribute ref="ид" use="required"/>
31:   </xs:complexType>
32: </xs:element>
33: </xs:schema>
```

•

Здесь сначала описываются простые элементы (строки 4-8), атрибуты (строка 10) и составные элементы (строки 12-21), а затем описывается корневой элемент целевого документа (строки 23-32). Атрибут ref тега xs:element указывает на выполненное ранее описание. Следует обратить внимание на использование атрибутов maxOccurs и use.

В листинге 42 приведен еще один вариант схемы документа чек.

В этом варианте элементы xs:simpleType определяются простые типы, такие, как строки и числа (строки 3-15). С помощью простых типов описываются составные типы товар (строки 17-25) и чек (строки 26-34). В конце схемы определяется корневой элемент (строка 36).

Листинг 42. Типизированная XML-схема документа «чек покупки»

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
03: <!-- простые типы -->
04: <!-- строковый -->
05: <xs:simpleType name="strtype">
06:   <xs:restriction base="xs:string"/>
07: </xs:simpleType>
08: <!-- целочисленный -->
09: <xs:simpleType name="inttype">
10:   <xs:restriction base="xs:integer"/>
11: </xs:simpleType>
12: <!-- десятичный -->
13: <xs:simpleType name="dectype">
14:   <xs:restriction base="xs:decimal"/>
15: </xs:simpleType>
16: <!-- составные типы -->
17: <!-- товар -->
18: <xs:complexType name="товар-тип">
19:   <xs:sequence>
20:     <xs:element name="название" type="strtype"/>
21:     <xs:element name="цена" type="dectype"/>
22:     <xs:element name="количество" type="dectype"/>
23:   </xs:sequence>
24:   <xs:attribute name="ид" type="strtype" use="required"/>
25: </xs:complexType>
26: <!-- чек -->
27: <xs:complexType name="чек-тип">
28:   <xs:sequence>
29:     <xs:element name="магазин" type="strtype"/>
30:     <xs:element name="дата" type="xs:date"/>
31:     <xs:element name="товар" type="товар-тип" maxOccurs="100"/>
32:   </xs:sequence>
33:   <xs:attribute name="ид" type="inttype" use="required"/>
34: </xs:complexType>
35: <!-- корневой элемент целевого документа -->
36: <xs:element name="чек" type="чек-тип"/>
37: </xs:schema>
```

•

Здесь в строке 31 указано другое значение для максимального количества элементов товар в целевом документе.

При составлении XML-схем можно комбинировать показанные приемы в любом сочетании. Это особенно следует учитывать при моделировании иерархий схем.

Рассмотренные примеры XML-схем показывают применение лишь небольшого арсенала средств, используемых для описания элементов и типов данных. Более подробную информацию можно получить, например, изучив страницы сайта "http://www.w3schools.com/schema/default.asp".

В листингах 39 и 40 показано, как связать схему с документом, когда пространство имен элементов целевого документа не задано. Следующий пример показывает, как указать это пространство имен.

В листинге 43 приведен еще один пример схемы XML (note.xsd).

Листинг 43. Схема документа с указанием пространства имен

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
03:     targetNamespace="revol.ponocom.ru"
04:     xmlns="revol.ponocom.ru"
05:     elementFormDefault="qualified">
06: <xs:element name="заметка">
07:   <xs:complexType>
08:     <xs:sequence>
09:       <xs:element name="кому" type="xs:string"/>
10:       <xs:element name="от-кого" type="xs:string"/>
11:       <xs:element name="содержание" type="xs:string"/>
12:     </xs:sequence>
13:   </xs:complexType>
14: </xs:element>
15: </xs:schema>
```

•

Здесь строка 2 описывает пространство имен xs, которое использует элементы и типы данных этого пространства для описания схемы, и зафиксировано строкой "http://www.w3.org/2001/XMLSchema":

Строка 3 указывает, что элементы целевого документа (чек, товар, магазин, дата, ид и др.) относятся к пространству имен, зафиксированному строкой "revol.ponocom.ru".

Строка 4 указывает пространство имен по умолчанию, описываемое строкой "revol.ponocom.ru".

Строка 5 указывает, что элементы конкретного документа, использующего объявленные в схеме элементы, должны сопоставляться с пространством имен.

В листинге 44 приведен XML-документ, в котором указывается пространство имен и приведенная выше XML-схема.

Листинг 44. XML-документ с пространством имен

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <заметка xmlns="revol.ponocom.ru"
03:     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04:     xsi:schemaLocation="revol.ponocom.ru note.xsd">
05:   <кому>ВВ</кому>
06:   <от-кого>РР</от-кого>
07:   <содержание>встреча 1 числа</содержание>
08: </заметка>
```

•

В строке 2 объявляется пространство имен по умолчанию.

В строке 3 объявляется пространство имен *XML Schema Instance* для использования атрибута schemaLocation. Значение этого атрибута состоит из двух частей, разделенных *пробелом* (строка 4). Первая часть указывает, какое пространство имен следует использовать, а вторая часть задает местоположение XML-схемы.

2.5. Язык XHTML

Язык XHTML (*eXtensible HyperText Markup Language*) сочетает в себе возможности языка HTML со строгостью описания структур, присущей XML. В этом языке введены ограничения на использование тегов HTML и атрибутов.

1. Каждый тег XHTML должен иметь закрывающий. В HTML это правило не является обязательным.

2. Несимметричные теги типа
 должны быть записаны как пустые теги XML, то есть
.

3. Теги должны быть правильным образом вложены один в другой.

4. Значения атрибутов записываются в кавычках.

5. Не допускается минимизированная форма записи атрибутов.

Например, тег

```
<INPUT type="CHECKBOX" CHECKED>
```

должен быть записан следующим, например, образом

```
<INPUT type="CHECKBOX" CHECKED="CHECKED"/>    .
```

Эти правила дают возможность анализировать XHTML документы как документы XML.

2.6. Обработка XML-документов

Сами по себе XML-документы не используют, хотя они легко читаются человеком, обычно их обрабатывают программами.

Для обработки XML-документов предусмотрены стандарты, описывающие объектную модель документа DOM (*Document Object Model*).

Для выборки из XML-документа отдельных его частей используется язык XPath. Для преобразования XML-документа в другие документы используется язык XSLT. Для подготовки документа к печати используется стандарт XSL-FO. Существуют и другие стандарты и языки.

В языке PHP есть соответствующие расширения для указанных стандартов и языков. Они описываются в последующих разделах.

3. Объектная модель документа (DOM)

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

Есть три уровня этого стандарта, обозначаемые DOM Level 1, DOM Level 2, DOM Level 3, или просто DOM1, DOM2, DOM3.

Стандарт DOM1 определяет общую структуру XML-документа, типы узлов. Узлы описываются в виде классов, свойств, методов. Последняя версия этого стандарта датируется 2000 годом, она имеет название *Document Object Model* (*DOM*) *Level 1 Specification* (*Second Edition*).

Стандарт DOM2 определяет использование пространств имен документов XML и их взаимодействие, а также события, представления, стили, вводит новые свойства и методы.

Стандарт DOM3 определяет загрузку и сохранение, проверку, события, XPath, добавляет новые свойства и методы.

Заметим, что стандарты DOM описывают построение объектной модели документов не только в формате XML, но также и в форматах HTML и XHTML (построение объектной модели в обозревателях).

В этом разделе рассматривается реализация DOM в языке PHP.

Описываются только несколько важнейших классов, для которых приводятся только важнейшие свойства и методы. При необходимости дополнительную информацию легко найти в сети интернет, например, по адресу "http://php.net/manual/ru/book.dom.php".

3.1. Кодировки XML-документов

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 функция PHP5 iconv:

```
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 можно использовать функцию:

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

Пример перекодировки из кодовой страницы 1251 в UTF-8:

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

Существуют и другие функции для работы с кодировками.

3.2. Базовый класс DOMNode

Класс DOMNode является базовым для классов всех других типов узлов дерева объектов, таких, как DOMDocument, DOMElement, DOMAttr, DOMText, DOMProcessingInstruction, DOMCharacterData и других. Он определяет свойства и методы, которые используются чаще всего для работы с объектной моделью. Заметим, что префикс DOM можно писать в любом регистре.

Рассмотрим свойства данного класса.

nodeType — возвращает тип узла в виде целого числа.

Существует множество констант, описывающих различные узлы, некоторые из них приведены в следующей таблице.

Таблица 1. Константы типов узлов

Константа	*Значение*	*Узел*
XML_ELEMENT_NODE	1	Element
XML_ATTRIBUTE_NODE	2	Attribute
XML_TEXT_NODE	3	Text
XML_CDATA_SECTION_NODE	4	CDATASection
XML_ENTITY_REF_NODE	5	Entity Reference
XML_ENTITY_NODE	6	Entity
XML_PI_NODE	7	ProcessingInstruction
XML_COMMENT_NODE	8	Comment
XML_DOCUMENT_NODE	9	Document
XML_DOCUMENT_TYPE_NODE	10	DOCTYPE
XML_DOCUMENT_FRAG_NODE	11	DocumentFragment
XML_NOTATION_NODE	12	Notation
XML_ENTITY_DECL_NODE	17	Entity Declaration

nodeName — возвращает имя узла в виде строки.

Возвращаемое значение зависит от типа узла.

В следующей таблице приведены примеры.

Таблица 2. Имена различных узлов

Узел	*Возвращаемое значение*
Element	имя соответствующего тега
Attribute	имя атрибута
Text	#text
CDATASection	#cdata-section
Entity Reference	имя компонента, указанное в ENTITY
Entity Declaration	имя компонента, указанное в ENTITY
ProcessingInstruction	имя приложения
Comment	#comment
Document	#document
DOCTYPE	имя корневого тега

nodeValue — возвращает значение узла. Значением узла обычно является весь текст, который встречается в узле, включая текст, содержащийся во вложенных узлах. Некоторые типы узлов не имеют значения.

ownerDocument — начальный узел (DOMDocument).
parentNode — родительский узел (DOMNode).
childNodes — список дочерних узлов (DOMNodeList).
firstChild — первый дочерний узел (DOMNode).
lastChild — последний дочерний узел (DOMNode).

attributes — список атрибутов (DOMNamedNodeMap).

previousSibling — предыдущий узел данного уровня (DOMNode).
nextSibling — следующий узел данного уровня (DOMNode).

Класс DOMNode предоставляет следующие методы.

hasChildsNodes — признак наличия дочерних узлов.
hasAttributes — признак наличия атрибутов.

isSameNode(DOMNode) — признак того же самого узла, что и параметр.

Следующие методы используются для модификации объектной модели. С их помощью можно добавлять и удалять узлы.

DOMNode appendChild(DOMNode) — добавляет новый узел (в конец).
DOMNode insertBefore(DOMNode newNode [, DOMNode refNode]) — добавляет новый узел newNode перед refNode, или в конец, если refNode не задан.

DOMNode removeChild(DOMNode) — удаляет указанный узел.
DOMNode replaceChild(DOMNode newNode, DOMNode oldNode) — заменяет узел oldNode узлом newNode.

DOMNode cloneNode([bool $deep]) — создает копию узла. Параметр deep указывает, нужно ли копировать потомков узла. По умолчанию этот параметр имеет значение false.

Некоторые свойства класса DOMNode возвращают множества объектов. Так, свойство childNodes возвращает список типа DOMNodeList, а свойство attributes возвращает список типа DOMNamedNodeMap.

3.3. Вспомогательные классы списков

Класс NodeList — это упорядоченный список узлов. Доступ к узлу списка возможен только по его порядковому номеру. Класс имеет одно свойство length (количество узлов), и один метод item(index), возвращающий узел с номером index. Первый узел имеет номер 0.

Класс NamedNodeMap — это неупорядоченный список узлов с уникальными именами (список атрибутов). Этот класс также имеет свойство length и метод item. Класс определяет также следующие методы.

getNamedItem(string) — возвращает узел по имени (DOMNode).

setNamedItem(DOMNode) — добавляет узел в список.

removeNamedItem(string) — удаляет узел из списка по имени.

Есть версии этих методов для операций с пространством имен.

3.4. Класс DOMDocument

Класс DOMDocument представляет начальный узел XML-документа.

Сценарий, работающий с DOM, начинаются с получения объекта именно этого класса. Далее, используя свойства и методы, унаследованные от базового класса DOMNode, сценарий обходит дерево узлов.

Рассмотрим свойства класса DOMDocument.

Следующие свойства возвращают специфичные узлы XML.

documentElement — возвращает корневой узел.

doctype — возвращает узел DOCTYPE.

Следующие свойства управляют вводом и выводом:

formatOutput — формирует выходной XML-документ с отступами.

preserveWhiteSpace — задает режим обработки пустых текстовых полей. Если значение свойства равно false, пустые текстовые поля не формируют текстовых узлов. Пустым текстовым полем называется текст, содержащий только *пробельные символы*: пробел, символ табуляции, перевод строки, возврат каретки. Такой текст образуется в документе XML в результате разбиения его на строки с отступами.

resolveExternals — подключать внешние XML-документы;

substituteEntities — подставлять компоненты;

validateOnParse — проверять соответствие DTD.

Следующие свойства имеют отношение к строке <?xml version= ... ?>.

xmlVersion — версия XML.

xmlEncoding — кодировка.

xmlStandalone — признак независимого документа.

Рассмотрим методы класса DOMDocument.

Следующая группа методов создает узлы разных типов.

createAttribute(name) — создает узел атрибута.

createCDATASection(string) — создает узел CDATA.

createComment(string) — создает узел Comment.

createElement(name [, value]) — создает узел Element.

createEntityReference(name) — создает ссылку на компонент.

createProcessingInstruction(target [, data]) — создает узел инструкций.

createTextNode(string) — создает текстовый узел.

Эти методы возвращают объекты соответствующих классов.

Следующие методы возвращают узлы.

DOMElement getElementById(id) — возвращает элемент *по значению* уникального идентификатора, который устанавливает функция setIdAttribute.

DOMNodeList getElementsByTagName(string name) — возвращает список узлов по имени тега (элемента). Имя "*" — все элементы.

Следующие методы создают внутреннее представление DOM.

load(filename) — загружает XML-документ из файла.

loadXML(string source) — загружает XML-документ из строки.

loadHTML(string source) — загружает HTML-документ из строки.

loadHTMLFile(filename) — загружает HTML-документ из файла.

Следующие методы преобразуют внутреннее представление DOM.

save(filename) — сохраняет XML-документ в файл.

saveHTMLFile(filename) — сохраняет HTML-документ в файл.

saveXML(DOMNode = NULL) — возвращает строку XML.

saveHTML(DOMNode = NULL) — возвращает строку HTML.

Если при вызове двух последних метода задан параметр (узел), то выводится часть документа, начиная с этого узла.

Следующие методы проверяют состоятельность XML-документа с использованием XML-схемы.

schemaValidate(filename) — сопоставляет документ со схемой в файле.

schemaValidateSource(string) — схема задана в виде строки.

Эти методы возвращают "1" в случае успеха. В случае неудачи генерируется предупреждение или ошибка, возвращаемое значение не определено. Работа методов зависят также от версии PHP.

Экземпляр класса DOMDocument создается следующим образом:

```
$doc = new DOMDocument([string version[, string encoding]]);
```

Первый параметр задает версию XML-документа, второй — выходную кодировку.

3.5. Загрузка и сохранение XML-документов

Следующий сценарий показывает, как создать XML-документ из строки, построить дерево объектов и вывести документ на страницу обозревателя.

Листинг 45. Загрузка XML из строки и вывод в виде HTML

```
01: <?php
02: # создать XML
03: $doc = new DOMDocument();
04: # текст XML
05: $xml = "<?xml version='1.0' encoding='utf-8'?>
06: <заметка><текст>Пара в час</текст></заметка>";
07: # загрузить текст XML
08: $doc->loadXML($xml);
09: # сохранить XML как строку
10: echo $doc->saveXML(); # <заметка><текст>Пара в час
11: ?>
```

•

В строке 3 создается объект XML-документа. В строчках 5 и 6 формируется текст XML. В строке 8 текст XML загружается и формируется дерево объектов. В строке 10 дерево объектов преобразуется в строку XML методом loadXML. Результат вывода приведен в комментарии в строке 10. Видно, что выводятся начальные теги.

Если имена тегов заменить английскими аналогами, теги выводиться не будут, — обозреватель воспримет их как неверные теги HTML.

Чтобы вывести текст XML в обозревателе, нужно послать заголовок CGI, указывающий на тип mime содержания. Следующий листинг полностью аналогичен предыдущему, за исключением того, что в строке 3 выводится заголовок CGI, указывающий, что посылается простой текст.

Листинг 46. Вывод XML в виде текста

```
01: <?php
02: # выводить как текст
03: header("Content-Type: text/plain; charset=utf-8");
04: # создать XML
05: $doc = new DOMDocument();
06: # текст XML
07: $xml = "<?xml version='1.0' encoding='utf-8'?>
08: <заметка><текст>Пара в час</текст></заметка>";
09: # загрузить текст XML
10: $doc->loadXML($xml);
11: # сохранить XML как строку
12: echo $doc->saveXML();
13: ?>
```

•

Результат вывода — текст переменной $xml так, как он записан в сценарии (две строки).

Если задать заголовок CGI, указывающий, что выводится XML-документ, результат вывода будет совершенно другим, таким, как если бы обозревателем был открыт файл типа .xml. Следующий сценарий показывает, как вывести такой заголовок (строка 3).

Листинг 47. Вывод XML в виде XML

```
01: <?php
02: # выводить как XML
03: header("Content-type: application/xml");
04: # создать XML
05: $doc = new DOMDocument();
06: # текст XML
07: $xml = "<?xml version='1.0' encoding='utf-8'?>
08: <заметка><текст>Пара в час</текст></заметка>";
09: # загрузить текст XML
10: $doc->loadXML($xml);
11: # сохранить XML как строку
12: echo $doc->saveXML();
13: ?>
```

•

Вывод в обозревателе приведен на рисунке 7.

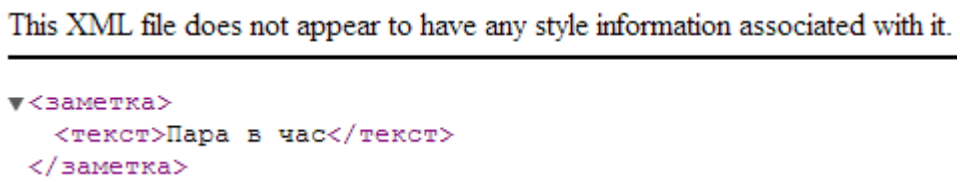

Рисунок 7 — Вывод сценария в листинге 47.

Следующий сценарий выводит XML-документ в файл. Для правильного форматирования выходного документа в строке 11 свойство formatOutput установлено в значение true.

Листинг 48. Вывод XML в файл

```
01: <?php
03: # создать XML
04: $doc = new DOMDocument();
05: # текст XML
06: $xml = "<?xml version='1.0' encoding='utf-8'?>
07: <заметка><текст>Пара в час</текст></заметка>";
08: # загрузить текст XML
09: $doc->loadXML($xml);
10: # вывод форматировать
11: $doc->formatOutput = true;
12: # сохранить XML в файл
13: echo $doc->save("note.xml"); # 120
14: ?>
```

•

Сценарий выводит размер записанного файла. Результат вывода:

```
<?xml version="1.0" encoding="utf-8"?>
<заметка>
  <текст>Пара в час</текст>
</заметка>
```

Следующий сценарий загружает полученный файл.

Листинг 49. Загрузка XML из файла

```
01: <?php
02: # выводить как текст
03: header("Content-Type: text/plain; charset=utf-8");
04: # создать XML
05: $doc = new DOMDocument();
06: # загрузить XML из файла
07: $doc->load("note.xml");
08: # сохранить XML как строку
09: echo $doc->saveXML();
10: ?>
```

•

Следующий сценарий сравнивает документ со схемой.

Листинг 50. Сравнение XML-документа со схемой

```
01: <?php
02: # управление выводом ошибок
03: #error_reporting(0);
04: # создать схему XML в виде строки
05: $xsd = "<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
06: <xs:element name='заметка'>
07:   <xs:complexType>
08:     <xs:sequence>
09:       <xs:element name='текст' type='xs:string'/>
10:     </xs:sequence>
11:   </xs:complexType>
12: </xs:element>
13: </xs:schema>";
14: # создать XML
15: $doc = new DOMDocument();
16: # загрузить XML из файла
17: $doc->load("note.xml");
18: # сравнить XML со схемой
19: if (!$doc->schemaValidateSource($xsd)) die("Invalid");
10: echo "Valid";
21: ?>
```

•

Схема XML формируется в строчках 5-13. В строке 19 внутреннее дерево сравнивается со схемой методом schemaValidateSource.

Сценарий выводит Valid. Если изменить тип элемента текст со значения string на значение integer (строка 9), сценарий выведет Invalid и предупреждение. Отключить предупреждение можно с помощью строки 3, которая управляет выводом ошибок (удалив символ комментария).

3.6. Создание XML-документа методами классов

Методы вида createXXXX объекта класса DOMDocument создают новые узлы, которые затем можно добавлять в дерево объектов методами класса DOMNode.

В следующем сценарии создается документ заметка с одним внутренним элементом текст, содержащим текст «Пара в час».

Листинг 51. Создание XML-документа методами классов

```
01: <?php
02: # выводить как текст
03: header("Content-Type: text/plain; charset=utf-8");
04: # создать XML
05: $doc = new DOMDocument("1.0", "utf-8");
06: # создать корневой элемент
07: $root = $doc->createElement("заметка");
08: # добавить элемент в узел документа
09: $root = $doc->appendChild($root);
10: # создать внутренний элемент
11: $node = $doc->createElement("текст", "Пара в час");
12: # добавить элемент в узел корневого элемента
13: $root->appendChild($node);
14: # форматировать вывод
15: $doc->formatOutput = true;
16: # сохранить XML в файл
17: $doc->save("note-2.xml");
18: # сохранить XML как строку
19: echo $doc->saveXML();
20: ?>
```

•

В строке 7 создается узел заметка, который запоминается в переменной $root. В строке 9 узел добавляется в объект Document, а возвращаемый корневой *узел дерева* вновь присваивается переменной $root.

В строке 11 создается узел текст со значением "Пара в час". В строке 13 новый узел добавляется в объект корневого узла.

В строке 15 устанавливается форматирование вывода. В строке 17 документ выводится в файл note-2.xml, а в строке 19 документ отправляется обозревателю.

Сценарий выводит в файл и в обозревателе одно и то же:

```
<?xml version="1.0" encoding="utf-8"?>
<заметка>
  <текст>Пара в час</текст>
</заметка>
```

Заметим, что метод createElement позволяет задать значение узла, то есть одновременно создать текстовый узел. Если не задавать значение, то текстовый узел необходимо создавать методом createTextNode.

В следующем сценарии текстовый узел создается отдельно. Кроме того, в этом сценарии создается атрибут, а также комментарий.

Листинг 52. Создание атрибутов и других узлов

```
01: <?php
02: # выводить как текст
03: header("Content-Type: text/plain; charset=utf-8");
04: # создать XML
05: $doc = new DOMDocument("1.0", "utf-8");
06: # создать корневой элемент
07: $root = $doc->createElement("документ");
08: # добавить элемент в узел документа
09: $root = $doc->appendChild($root);
10: # создать внутренний элемент
11: $node = $doc->createElement("параграф");
12: # создать текстовый узел
13: $txt = $doc->createTextNode("текст");
14: # добавить текстовый узел
15: $node->appendChild($txt);
16: # создать атрибут
17: $attr = $doc->createAttribute("ид");
18: # задать значение атрибута
19: $attr->value = "1";
20: # добавить атрибут во внутренний узел
21: $node->appendChild($attr);
22: # добавить элемент в узел корневого элемента
23: $root->appendChild($node);
24: # создать узел комментария
25: $cmt = $doc->createComment("комментарий");
26: # добавить комментарий первым узлом корневого элемента
27: $root->insertBefore($cmt, $root->firstChild);
28: # форматировать вывод
29: $doc->formatOutput = true;
30: # сохранить XML как строку
31: echo $doc->saveXML();
32: ?>
```

•

В строке 13 создается текстовый узел, который прикрепляется к узлу элемента в строке 15. В строке 17 создается узел атрибута, в строке 19 атрибуту задается значение, а в строке 21 атрибут прикрепляется к узлу элемента. В строке 25 создается узел комментария, который в строке 27 добавляется перед узлом элемента методом insertBefore.

Сценарий выводит следующий текст XML:

```
<?xml version="1.0" encoding="utf-8"?>
<документ>
  <!--комментарий-->
  <параграф ид="1">текст</параграф>
</документ>
```

3.7. Обход дерева объектной модели

Следующий листинг показывает пример рекурсивной функции, которая выводит номер узла, его тип, имя и значение, а также показывает атрибуты узла, если они есть (выводится имя и значение).

Листинг 53. Обход дерева

```
01: <?php
02: # выводить как HTML
03: header("Content-Type: text/html; charset=utf-8");
04: # вспомогательные функции
05: include_once "xmlapi.php";
06: # рекурсивный вывод узла
07: function showNode($e, $u, $n) {
08:   # выводим номер узла, тип, имя, значение
09:   $s = "N$n : ".nt($e)." : $e->nodeName : \"$e->nodeValue\"";
10:   disp($u, $s);
11:   # проверяем наличие атрибутов
12:   if ($e->hasAttributes()) {
13:     # список атрибутов узла
14:     $attr = $e->attributes;
15:     # выводим атрибуты
16:     foreach ($attr as $b => $a) {
17:       # $a - узел атрибута
18:       disp($u + 1, "$a->name=\"$a->value\"");
19:     }
20:   }
21:   # проверяем наличие дочерних узлов
22:   if ($e->hasChildNodes()) {
23:     # список дочерних узлов
24:     $list = $e->childNodes;
25:     # количество дочерних узлов
26:     $nn = $list->length;
27:     # выводим дочерние узлы
28:     for ($i = 0; $i < $nn; $i++) {
29:       # рекурсивно вызывая функцию для каждого потомка
30:       showNode($list->item($i), $u + 1, $i);
31:     }
32:   }
33: }
34: # создать XML
35: $dom = new DOMDocument();
36: # предотвратить создание пустых текстовых узлов
37: $dom->preserveWhiteSpace = false;
38: # загрузить XML из файла
39: $dom->load("shed.xml");
40: # показать узлы
41: showNode($dom, 0, 0);
42: ?>
```

•

Пример использует несколько вспомогательных функций, описанных в файле xmlapi.php.

Параметрами функции showNode являются исследуемый узел $e, номер уровня $u и номер узла $n.

Функция сначала выводит информацию об узле $e. В строке 9 листинга формируется строка для вывода. Она состоит из буквы N, обозначающей узел, за которой следует номер узла $n. Далее к этому значению присоединяется результат функции nt($e), которая определяет тип узла и формирует соответствующее значение. Затем в строку добавляются имя узла (свойство nodeName) и значение узла (свойство nodeValue).

Сформированная строка передается вспомогательной функции disp, первым параметром которой является номер уровня $u. Номер уровня позволяет вывести строки с отступами для облегчения чтения.

В строке 12 проверяется наличие атрибутов исследуемого узла $e с помощью свойства hasAttributes. Если атрибуты есть, они записываются из свойства attributes в переменную $attr, получающую тип DOMNamedNodeMap (строка 14). Затем в цикле foreach формируется переменная $a, представляющая собой отдельный атрибут, и имя и значение атрибута выводятся функцией disp.

В строке 22 проверяется наличие дочерних узлов узла $e. Если дочерние узлы есть, то их список в строке 24 записывается из свойства childNodes в переменную $list, получающую тип DOMNodeList. В строке 26 с помощью свойства length определяется количество дочерних узлов и запоминается в переменной $nn. Затем цикл for используется для вывода дочернего узла с помощью рекурсивного вызова функции showNode.

В строке 37 свойству preserveWhiteSpace присваивается значение, равное false, которое предотвращает создание пустых текстовых узлов. Изменение значения на true приведет к созданию дополнительных узлов.

Вспомогательная функция nt имеет примерно следующий вид:

```
function nt($e) {
  switch ($e->nodeType) {
  case XML_ELEMENT_NODE:        return "ELEMENT";     break; #1
  case XML_ATTRIBUTE_NODE:      return "ATTR";        break; #2
  case XML_TEXT_NODE:           return "TEXT";        break; #3
  case XML_CDATA_SECTION_NODE: return "CDATA";        break; #4
  case XML_ENTITY_REF_NODE:     return "ENTITY_REF";  break; #5
  case XML_ENTITY_NODE:         return "ENTITY";      break; #6
  case XML_PI_NODE:             return "PI";          break; #7
  case XML_COMMENT_NODE:        return "COMMENT";     break; #8
  case XML_DOCUMENT_NODE:       return "DOCUMENT";    break; #9
  case XML_DOCUMENT_TYPE_NODE: return "DOCTYPE";     break; #10
  case XML_DOCUMENT_FRAG_NODE: return "FRAGMENT";    break; #11
  case XML_NOTATION_NODE:       return "NOTATION";    break; #12
  case XML_ENTITY_DECL_NODE:    return "ENTITY_DECL"; break; #17
  }
}
```

Вспомогательная функция disp имеет следующий простой вид:

```
function disp($u, $s) {
  echo str_repeat(" ", $u * 3)."$s<BR/>";
}
```

Функция формирует некоторое количество неразрывных пробелов с помощью стандартной функции str_repeat, добавляет к ним выводимую строку и тег
 для формирования новых строк на странице HTML.

Пример вывода сценария из листинга 53:

```
N0 : DOCUMENT : #document : ""
   N0 : COMMENT : #comment : "комментарий"
   N1 : PI : php : "// PHP code "
   N2 : ELEMENT : расписание : "АлгебраАкопян"
      N0 : ELEMENT : день : "АлгебраАкопян"
         ид="1"
         N0 : ELEMENT : группа : "АлгебраАкопян"
            ид="1ПО-33Д"
            N0 : ELEMENT : пара : "АлгебраАкопян"
               ид="1"
               ауд="326"
               тип="лекция"
               N0 : ELEMENT : предмет : "Алгебра"
                  N0 : CDATA : #cdata-section : "Алгебра"
               N1 : ELEMENT : препод : "Акопян"
                  N0 : TEXT : #text : "Акопян"
```

Исследуемый файл имеет следующий вид:

```
<?xml version="1.0" encoding="utf-8"?>
<!--комментарий-->
<?php // PHP code ?>
<расписание>
  <день ид="1">
    <группа ид="1ПО-33Д">
      <пара ид="1" ауд="326" тип="лекция">
        <предмет><![CDATA[<I>Алгебра</I>]]></предмет>
        <препод>Акопян</препод>
      </пара>
    </группа>
  </день>
</расписание>
```

3.8. Класс DOMElement

Класс базового строительного узла DOMElement добавляет в основном методы для работы с атрибутами, так как атрибуты могут быть только у узлов типа Element. Неунаследованное свойство tagName возвращает то же, что и свойство nodeName (имя тега).

Следующие методы предназначены для работы с именем атрибута.
getAttribute(name) — возвращает значение атрибута name.
setAttribute(name, value) — устанавливает значение атрибута name.
hasAttribute(name) — возвращает признак наличия атрибута name.
removeAttribute(name) — удаляет атрибут name.
Следующие функции работают с узлами атрибутов.
getAttributeNode(name) — возвращает узел атрибута name.
setAttributeNode(DOMAttr) — добавляет узел атрибута.
removeAttributeNode(DOMAttr) — удаляет узел атрибута.
Следующие функции нужны для работы с ключевым атрибутом.
setIdAttribute(name, isid) — объявляет атрибут с именем name ключевым.
Атрибут с именем name должен существовать.
Ключевой атрибут используется методом getElementById.
setIdAttributeNode(DOMNode, isid) — объявляет узел атрибута ключевым.
Следующий сценарий использует методы setIdAttribute и getElementById.

Листинг 54. Создание ключевого атрибута и поиск узла по нему

```
01: <?php
02: # создать XML
03: $doc = new DOMDocument("1.0", "utf-8");
04: # создать корневой элемент
05: $root = $doc->createElement("документ");
06: # добавить элемент в узел документа
07: $root = $doc->appendChild($root);
08: # создать внутренний элемент
09: $node = $doc->createElement("параграф", "текст");
10: # добавить атрибут ид
11: $node->setAttribute("номер", "1");
12: # сделать атрибут ид ключевым
13: $node->setIdAttribute("номер", true);
14: # добавить элемент в узел корневого элемента
15: $root->appendChild($node);
16: # форматировать вывод
17: $doc->formatOutput = true;
18: # сохранить XML как строку
19: echo $doc->saveXML();
20: # получить элемент по ключевому атрибуту
21: $temp = $doc->getElementById("1");
22: # вывести значение полученного элемента
23: echo "\"".$temp->nodeValue."\"";
24: ?>
```

•

В строке 11 создается атрибут элемента параграф с именем номер со значением "1". В строке 13 этот атрибут устанавливается как ключевой.

В строке 21 выбирается элемент параграф, имеющий ключевой атрибут со значением "1", в строке 23 выводится значение элемента параграф, равное "текст".

4. Язык XPath

Во время работы с XML-документом наиболее часто требуется выбрать определенную группу узлов. Язык XPath предоставляет для этой цели удобный и эффективный синтаксис. Версия стандарта 2.1 называется XML Path Language (XPath) 2.1, стандарт расположен по адресу "http://www.w3.org/TR/2009/WD-xpath-21-20091215/".

Выборка узлов происходит в результате выполнения запроса. Запрос представляет собой строку, описывающую путь к искомым узлам примерно так, как спецификация файла описывает путь к файлу в файловой системе. В качестве примера будем рассматривать следующий документ XML (листинг 55, файл testbook.xml).

Листинг 55. Документ XML для выборки

```
<?xml version="1.0" encoding="utf-8"?>
<книга>
  <!--комментарий-->
  <ГЛ за="PHP" ид="1">
    <ЧА за="PHP-1" ид="1">
      <ПА ид="1">П1.</ПА>
      <ПА ид="2">П2.</ПА>
      <ТА ид="1">Т1.</ТА>
    </ЧА>
    <ЧА за="PHP-2" ид="2">
      <ПА ид="1">П3.</ПА>
      <ПА ид="2">П4.</ПА>
      <ТА ид="2">Т2.</ТА>
    </ЧА>
  </ГЛ>
  <ГЛ за="XML" ид="2">
    <ЧА за="XML-1" ид="1">
      <ПА ид="1">П5.</ПА>
      <ПА ид="2">П6.</ПА>
      <ТА ид="3">Т3.</ТА>
    </ЧА>
    <ЧА за="XML-2" ид="2">
      <ПА ид="1">П7.</ПА>
      <ПА ид="2">П8.</ПА>
      <СП ид="1">
        <Э ид="1">Э1.</Э>
        <Э ид="2">Э2.</Э>
      </СП>
    </ЧА>
  </ГЛ>
</книга>
```

•

Для простоты здесь имена ГЛАВА, ЧАСТЬ, ПАРАГРАФ, ТАБЛИЦА и другие сокращены до двух букв. Для тестирования выборок в этом документе используется сценарий, приведенный в следующем листинге.

Листинг 56. Сценарий для выборки

```
01: <?php
02: # выводить как HTML
03: header("Content-Type: text/html; charset=utf-8");
04: # создать XML
05: $doc = new DOMDocument();
06: # предотвратить пустые текстовые узлы
07: $doc->preserveWhiteSpace = false;
08: # загрузить XML
09: $doc->load("testbook.xml");
10: # создать XPath
11: $dxp = new DOMXpath($doc);
12: # найти контекстный узел
13: $ctx = $dxp->query("//ГЛ[1]/ЧА[1]/ТА[1]//@ид")->item(0);
14: # найти узлы, удовлетворяющие запросу
15: $list = $dxp->query("../../* | //ГЛ[2]/ЧА[2]/СП//text()", $ctx);
16: # количество узлов
17: $nn = $list->length;
18: echo "Найдено узлов: $nn <BR/>";
19: for ($i = 0; $i < $nn; $i++) {
20:   $node = $list->item($i);
21:   echo ($i+1).": $node->nodeName=\"$node->nodeValue\"<BR/>";
22: }
23: ?>
```

•

Для работы с запросами в PHP есть класс DOMXpath, поддерживаемый библиотекой libxml2. Экземпляр Xpath создается следующим образом:

```
$dxp = new DOMXpath($domdocument);
```

В листинге 56 объект DOMXpath создается в строке 11.

Запрос выполняет метод DOMXpath:query:

```
DOMNodeList query($string xpath[, DOMNode $context])  .
```

Первым параметром является обязательная строка запроса, соответствующая синтаксису XPath. Второй параметр, называемый контекстом, задается в случае, если запрос содержит относительный адрес, который в этом случае вычисляется от узла контекста.

В листинге 56 выполняется два запроса. В строке 13 выполняется запрос на выборку контекстного узла $ctx. Поскольку запрос всегда возвращает список узлов DOMNodeList, здесь выбирается первый узел списка методом item. В строке 15 выполняется выборка интересующих нас узлов. В листинге 55 контекстный узел подчеркнут двойной линией, а интересующая нас выборка подчеркнута одинарной линией.

Рассмотрим запрос, выполняющий выборку контекстного узла:

```
//ГЛ[1]/ЧА[1]/ТА[1]//@ид  .
```

По аналогии с файловой системой, данный запрос выбирает путь "глава 1—часть 1—таблица-1", после чего выбирается узел атрибута ид таблицы. Знак @ указывает на то, что выбирается атрибут.

Рассмотрим основной запрос:

```
../../* | //ГЛ[2]/ЧА[2]/СП//text()  .
```

Этот запрос состоит из двух частей (двух подзапросов), соединенных знаком вертикальной черты.

Рассмотрим первый подзапрос:

```
../../*  .
```

По аналогии с файловой системой, этот запрос ссылается на узел, который является родительским узлом родительского узла контекстного узла (две точки обозначают родительский узел). Звездочка, как и везде, имеет значение «все». Таким образом, этот подзапрос выбирает *все узлы родителя родителя*. Родительским узлом атрибута является узел элемента ТА (таблица), а родительским узлом таблицы является часть. В выборку запроса войдут параграфы 1 и 2, а также таблица.

Рассмотрим второй подзапрос:

```
//ГЛ[2]/ЧА[2]/СП//text()  .
```

Здесь выбирается путь "глава-2—часть-2—список", а функция text предписывает выбрать текстовые узлы списка. Таким образом, результат второй выборки — текст внутри элементов списка.

Выполнение сценария дает следующий результат в обозревателе:

```
Найдено узлов: 5
1: ПА="П1."
2: ПА="П2."
3: ТА="Т1."
4: #text="Э1."
5: #text="Э2."
```

Очевидно, что результат совпадает с нашими ожиданиями.

4.1. Структура запроса

Язык XPath использует два способа адресации:
- относительная (*RelativeLocationPath*);
- абсолютная (*AbsoluteLocationPath*).

Относительный адрес задается в виде "Шаг/Шаг/...".
Абсолютный адрес задается в виде "/Шаг/Шаг/...".
Каждый из шагов может состоять из трех частей:

ОСЬ::ШАБЛОН-УЗЛА[ПРЕДИКАТ]

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

Ось (*Axis*) задает направление, в котором выбираются узлы. Всего есть 13 направлений, они описаны далее.

Шаблон (*NodeTest*) задает шаблон выбора узла по типу или имени.

Шаблон узла может принимать следующие основные значения:

* — все узлы, соответствующие типу оси;

имя — все узлы с указанным именем;

node() — указывает на все узлы оси;

text() — указывает на текстовые узлы;

comment() — указывает на узлы комментариев;

processing-instuction() — указывает на узлы приложений.

Предикат (*Predicate*) задает дополнительные условия для выбора узла, например, его порядковый номер. В предикатах используются функции, которые описаны далее. Часто используемый предикат позиции, например, ГЛ[1], имеет полный вид [position()=1].

4.2. Оси XPath

В следующей таблице приведены описания осей XPath.

Таблица 3. 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::	Последующие 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	

Заметим, что узлы атрибутов не являются дочерними узлами соответствующего элемента (хотя имеют родителя), и они не попадают в выборку обычных узлов. Для выборки атрибутов нужно указывать ось "attribute::" или сокращение "@". То же самое касается узлов, описывающих пространства имен. Для них используется ось "namespace::".

4.3. Функции

Следующие функции XPath можно использовать в предикатах.

4.3.1. Функции для работы с узлами

position() — позиция узла в списке от единицы.
last() — количество узлов в списке (последний узел).
count(node-set) — количество узлов в выражении.
id(object) — выбирает элемент по уникальному идентификатору.
name(node-set) — полное имя первого элемента.

4.3.2. Строковые функции

string(object) — текстовое содержание объекта.
string-length(string) — длина строки.
concat(string, string, ...) — конкатенация строк.
contains(string, string) — истина, если первая строка содержит вторую.
substring(string, start [, length]) — часть строки, начиная с позиции start длиной length символов или до конца строки. Нумерация символов с 1.
substring-before(string, string) — часть строки 1 до позиции строки 2.
substring-after(string, string) — часть строки 1 после строки 2.
starts-with(string, string) — истина, если строка 2 — начало строки 1.
ends-with(string, string) — истина, если строка 2 — конец строки 1.
normalize-space(string) — удаляет начальные и конечные пробельные символы и заменяет их внутри строки одним пробелом.
translate(string, string, string) — в строке 1 заменяет символы из строки 2, символами из строки 3.

4.3.3. Математические функции

number(object) — преобразует объект в число.
sum(node-set) — сумма числовых значений узлов.
floor(number) — возвращает ближайшее целое сверху.
ceiling(number) — возвращает ближайшее целое снизу.
round(number) — возвращает ближайшее целое (округление).

4.3.4. Поддержка областей имен

namespace-uri(node-set) — область имен, связанная с первым узлом.
local-name(node-set) — локальная часть имени первого элемента.
С помощью данных функций можно сформировать запрос для выборки элементов с определенной областью имен:

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

registerNamespace(prefix, uri) — регистрирует префикс prefix.

4.3.5. Операторы

Операторы, используемые в предикатах:

or, and, not, <, >, <=, >=, =, +, –, *, div, mod .

4.4. Примеры запросов

Рассмотрим несколько примеров запросов в файле testbook.xml.

1. "/". Выбран центральный узел. Результат запроса:

```
Найдено узлов: 1
1: #document=""
```

2. "/node()" или "/*". Выбран узел книга. Результат запроса:

```
Найдено узлов: 1
1: книга="П1.П2.Т1.П3.П4.Т2.П5.П6.Т3.П7.П8.Э1.Э2."
```

3. "//node()". Выбраны все узлы (35 узлов). Результат запроса:

```
Найдено узлов: 35
1: книга="П1.П2.Т1.П3.П4.Т2.П5.П6.Т3.П7.П8.Э1.Э2."
2: #comment="комментарий"
3: ГЛ="П1.П2.Т1.П3.П4.Т2."
4: ЧА="П1.П2.Т1."
5: ПА="П1."
6: #text="П1."
7: ПА="П2."
8: #text="П2."
9: ТА="Т1."
10: #text="Т1."
11: ЧА="П3.П4.Т2."
12: ПА="П3."
13: #text="П3."
14: ПА="П4."
15: #text="П4."
16: ТА="Т2."
17: #text="Т2."
18: ГЛ="П5.П6.Т3.П7.П8.Э1.Э2."
19: ЧА="П5.П6.Т3."
20: ПА="П5."
21: #text="П5."
22: ПА="П6."
23: #text="П6."
24: ТА="Т3."
25: #text="Т3."
26: ЧА="П7.П8.Э1.Э2."
27: ПА="П7."
28: #text="П7."
29: ПА="П8."
30: #text="П8."
31: СП="Э1.Э2."
32: Э="Э1."
33: #text="Э1."
34: Э="Э2."
35: #text="Э2."
```

4. "//*". Выбраны узлы элементов (21 узел). Результат запроса:

```
Найдено узлов: 21
1: книга="П1.П2.Т1.П3.П4.Т2.П5.П6.Т3.П7.П8.Э1.Э2."
2: ГЛ="П1.П2.Т1.П3.П4.Т2."
3: ЧА="П1.П2.Т1."
4: ПА="П1."
5: ПА="П2."
6: ТА="Т1."
7: ЧА="П3.П4.Т2."
8: ПА="П3."
9: ПА="П4."
10: ТА="Т2."
11: ГЛ="П5.П6.Т3.П7.П8.Э1.Э2."
12: ЧА="П5.П6.Т3."
13: ПА="П5."
14: ПА="П6."
15: ТА="Т3."
16: ЧА="П7.П8.Э1.Э2."
17: ПА="П7."
18: ПА="П8."
19: СП="Э1.Э2."
20: Э="Э1."
21: Э="Э2."
```

5. "//node()[3]". Выбраны все третьи узлы всех элементов.
Результат запроса:

```
Найдено узлов: 5
1: ТА="Т1."                          4: ТА="Т3."
2: ТА="Т2."                          5: СП="Э1.Э2."
3: ГЛ="П5.П6.Т3.П7.П8.Э1.Э2."
```

6. "/*/node()[3]". Выбран третий узел первого уровня.
Результат запроса:

```
Найдено узлов: 1
1: ГЛ="П5.П6.Т3.П7.П8.Э1.Э2."
```

8. "/*/*". Выбирает все узлы второго уровня. Результат запроса:

```
Найдено узлов: 2
1: ГЛ="П1.П2.Т1.П3.П4.Т2."
2: ГЛ="П5.П6.Т3.П7.П8.Э1.Э2."
```

9. "/*/*/@*". Выбирает узлы атрибутов всех узлов второго уровня.
Результат запроса:

```
Найдено узлов: 4
1: за="PHP"                          3: за="XML"
2: ид="1"                            4: ид="2"
```

10. "//@за". Выбирает все узлы атрибутов за (заголовок).
Результат запроса:

```
Найдено узлов: 6
1: за="PHP"                          4: за="XML"
2: за="PHP-1"                        5: за="XML-1"
3: за="PHP-2"                        6: за="XML-2"
```

11. "//ПА//node()" или "//ПА/node()". Выбирает текстовые узлы элементов ПА (параграф). Результат запроса:

```
Найдено узлов: 8
1: #text="П1."                       5: #text="П5."
2: #text="П2."                       6: #text="П6."
3: #text="П3."                       7: #text="П7."
4: #text="П4."                       8: #text="П8."
```

12. "//ГЛ[1]/ЧА[2]/ПА[1]//following-sibling::node()". Выбирает сначала первый параграф второй части второй главы, а затем последующие потомки второй части. Результат запроса:

```
Найдено узлов: 2
1: ПА="П4."
2: ТА="Т2."
```

Дополнительную информацию по языку XPath и запросам см., например, ссылку "http://www.w3schools.com/xsl/xpath_intro.asp".

5. Язык XSLT

Этот язык базируется на XSL (*eXtensible Stylesheet Language*, расширяемый язык таблиц стилей). XSL — это семейство рекомендаций консорциума W3C, описывающее языки преобразования и визуализации документов XML. Включает в себя:

- язык XSLT — преобразование (*transformations*) XML-документов;
- язык XSL-FO — подготовка к печати XML-документов;
- язык XPath — навигация в XML-документе;
- язык XQuery — запросы к XML-документу.

Файл типа .xsl описывает преобразование XML-документа, — каким образом данные документа XML представить в виде XML, HTML, XHTML или в виде простого текста. Называется таблицей стилей, потому что к элементам XML применяются те или иные оформления (стили).

Документ XSL является документом XML. Он содержит так называемые *шаблоны* (*templates*), которые применяются к выбранным с помощью XPath элементам XML. По сути, шаблоны — это алгоритмы применения оформления к XML, как часть парадигмы отделения данных от их представления (парадигмы MVC). Документ XML содержит только данные, но не имеет описательных средств их оформления, как, например, HTML. Документ XSL, наоборот, не содержит данных, но указывает, как данные XML следует оформить.

В процессе выполнения XSL-преобразования используются:

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

В простом случае XSLT-процессор получает на входе документ XML и таблицу стилей XSL, и создает выходной документ (рисунок 8).

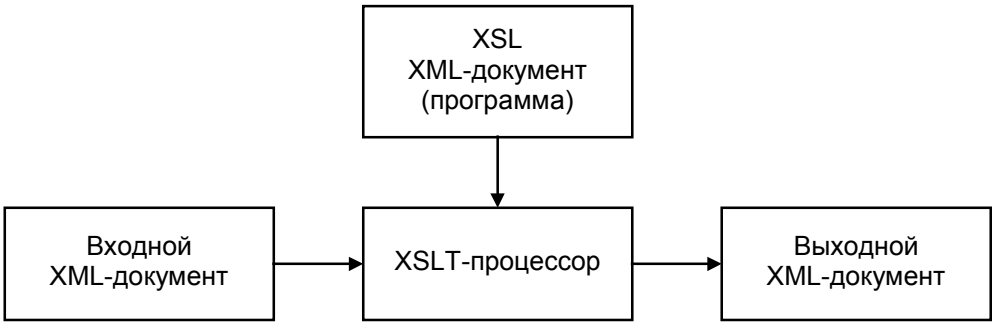

Рисунок 8 — Схема XSL преобразования

В качестве примера рассмотрим простой XML-документ, содержащий данные о книгах (листинг 57).

Листинг 57. XML-документ sbooks.xml

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <?xml-stylesheet type="text/xsl" href="sbooks.xsl"?>
03: <книги>
04:   <книга>
05:     <название>XML для программистов</название>
06:     <автор>Пономарев В.В.</автор>
07:     <год>2010</год>
08:   </книга>
09:   <книга>
10:     <название>PHP для программистов</название>
11:     <автор>Пономарев В.В.</автор>
12:     <год>2011</год>
13:   </книга>
14: </книги>
```

•

В строке 2 здесь указано, что с данным документом сопоставляется трансформация, описанная в файле sbooks.xsl (листинг 58).

Листинг 58. Преобразование XML-документа sbooks.xml

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <xsl:stylesheet version="1.0"
03:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
04: <xsl:template match="/">
05:   <HTML><BODY>
06:   <H3>Книги</H3>
07:   <TABLE border="1">
08:     <TR bgcolor="lightgreen">
09:       <TH>Название книги</TH>
10:       <TH>Автор</TH>
11:     </TR>
12:     <xsl:for-each select="книги/книга">
13:     <TR>
14:       <TD><xsl:value-of select="название"/></TD>
15:       <TD><xsl:value-of select="автор"/></TD>
16:     </TR>
17:     </xsl:for-each>
18:   </TABLE>
19:   </BODY></HTML>
20: </xsl:template>
21: </xsl:stylesheet>
```

•

Трансформация XSL — это состоятельный документ XML. Корневым элементом является stylesheet или transform. Указание пространства имен и версии является обязательным.

В строке 4 описывается шаблон для обработки корневого узла "/", который, можно сказать, соответствует функции main в языке Си, то есть это *главный* шаблон. Шаблон описывает инструкции по обработке элементов документа XML.

Как видно из листинга 58, главный шаблон описывает документ в формате HTML (строки 5-19). В строках 7-18 описывается таблица, в которую будет выводиться информация XML. Таблица состоит из строки заголовка (строки 8-11) и строк, соответствующим отдельным книгам (строки 13-16). Поскольку строк, описывающих книги, может быть произвольное количество, строки 13-16 заключены в инструкцию for-each, представляющую цикл XSL. Атрибут select этой инструкции есть не что иное, как запрос XPath "книги/книга". Хотя здесь использован относительная форма запроса, шаблон template в строке 4 указывает, что контекстным узлом является корневой (match="/").

В строке 14 используется инструкция value-of, которая извлекает из запроса XPath "книги/книга" очередной элемент название с помощью атрибута select, значением которого также является запрос XPath "название", причем контекстным узлом уже будет "книги/книга[j]", где предикат "[j]" условно обозначает итерацию. Аналогичным образом в строке 15 извлекается очередной элемент автор.

Если открыть файл sbooks.xml в обозревателе, мы увидим, как документ XML преобразован в документ HTML (рисунок 9).

Книги

Название книги	Автор
XML для программистов	Пономарев В.В.
PHP для программистов	Пономарев В.В.

Рисунок 9 — Результат трансформации XML-документа sbooks.xml

Для сравнения посмотрите на рисунок 7, который показывает, как отображается документ XML, не имеющий связанной с ним таблицей стилей (трансформацией).

5.1. Инструкции XSL

XSL-документ всегда является состоятельным XML-документом и имеет следующую структуру в наиболее общем виде:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ... >
  <!-- тело - инструкции XSL и выходной текст -->
</xsl:stylesheet>
```

Тело документа содержит инструкции в произвольном порядке, следует только учитывать уровень их расположения. Этим XSL отличается от алгоритма, который задает жесткий порядок действий.

5.1.1. Инструкции форматирования

Инструкции форматирования управляют выводом. Они всегда располагаются на верхнем уровне (дочерние узлы элемента stylesheet).

```
<xsl:output method="xml" encoding="utf-8" indent="yes"/>
```

Инструкция output форматирует вывод. Основные атрибуты (значения атрибутов показывают варианты, первый вариант по умолчанию):

method="xml | html | text | имя" — формат выходного документа.

version — версия (для HTML и XML).

encoding — кодировка.

indent="yes | no" — формировать структурный отступ.

omit-xml-declaration="no | yes" — подавить вывод строки <?xml ... ?>.

media-type="text/html" — MIME-тип.

cdata-section-elements="список-элементов-через-пробел" — список элементов, которые должны быть заключены в секцию CDATA.

Структурный отступ на самом деле формируется только для метода вывода XML. Чтобы вывести документ в формате HTML со структурными отступами, нужно выводить его методом XML, подавив вывод строки <?xml ... ?> атрибутом omit-xml-declaration="yes".

Указание версии для метода HTML формирует элемент DOCTYPE.

```
<xsl:strip-space elements="element element ..."/>
<xsl:preserve-space elements="element element ..."/>
```

Инструкция strip-space удаляет незначащие пробельные символы указанных элементов, а инструкция preserve-space, наоборот, оставляет их. По умолчанию пробельные символы не удаляются, поэтому инструкция сохранения используется в случае, если есть инструкция удаления.

```
<xsl:decimal-format name="имя" ... />
```

Инструкция decimal-format определяет именованный формат вывода чисел, который далее используется функцией format-number. Основные атрибуты (значения атрибутов — это значения по умолчанию):

decimal-separator="." — десятичный разделитель.

grouping-separator="," — разделитель групп.

minus-sign="-" — знак, используемый для отрицательных чисел.

percent="%" — знак, используемый для обозначения процента.

zero-digit="0" — знак, используемый для обозначения нуля.

digit="#" — знак, используемый для обозначения цифры в шаблоне.

pattern-separator=";" — знак, используемый для разделения положительного и отрицательного шаблонов.

Пример задания формата с именем ru:

```
<xsl:decimal-format name="ru"
     decimal-separator=","
     grouping-separator=" "
/>
```

Пример использования формата ru в функции format-number:

```
<xsl:value-of select="format-number(12345.6, '# ###,00', 'ru')"/>
```

Функция выводит "12 345,60".

5.1.2. Инструкции включения внешних трансформаций

Следующие две инструкции подключают к данной трансформации внешние файлы. Эти инструкции располагаются на верхнем уровне.

```
<xsl:include href="url"/>
<xsl:import href="url"/>
```

Шаблоны, подключаемые инструкцией include, имеют такой же приоритет, как и шаблоны файла, который их включает. Шаблоны, подключаемые инструкцией import, имеют меньший приоритет.

5.1.3. Переменные и параметры

Следующие две инструкции описывают переменные и параметры шаблонов. При расположении на верхнем уровне переменные и параметры являются глобальными, иначе локальными. Глобальный параметр передается главному шаблону в вызове метода трансформации в сценарии, который создает XSLT-процессор (см. далее).

```
<xsl:variable name="имя">выражение</xsl:variable>
<xsl:variable name="имя" select="xpath"/>
<xsl:param name="имя">выражение</xsl:param>
<xsl:param name="имя" select="xpath"/>
```

Как видно, каждая из инструкций имеет две формы. Первая форма задает значение переменной или параметра в содержании тега. Вторая форма задает значение с помощью атрибута select, значением которого является запрос XPath. Эти две формы являются взаимоисключающими.

Если шаблон использует параметры, они объявляются с помощью инструкции with-param. Эта инструкция может располагаться внутри инструкций apply-templates и call-template.

```
<xsl:with-param name="имя" select="xpath"/>
<xsl:with-param name="имя">выражение</xsl:with-param>
```

Как инструкция param, инструкция with-param задает значение одной из двух форм: либо атрибутом select, либо содержанием тега.

5.1.4. Инструкции шаблонов

Шаблоны (*templates*) описывают алгоритмы формирования выходного документа. Шаблон с атрибутом match выполняет действия над частью входного документа, выбранной запросом атрибута select. Шаблон с атрибутом name, — это именованный шаблон, который можно вызывать по имени. Шаблон должен иметь либо атрибут match либо атрибут name, либо оба атрибута. В листинге 59 приведены примеры шаблонов.

Листинг 59. Шаблоны (sbookst.xsl)

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <xsl:stylesheet version="1.0"
03:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
04: <!-- главный шаблон -->
05: <xsl:template match="/">
06:   <HTML><BODY>
07:   <xsl:apply-templates/>
08:   </BODY></HTML>
09: </xsl:template>
10: <!-- шаблон, выбирающий книгу -->
11: <xsl:template match="книга">
12:   <P>
13:   <xsl:apply-templates select="название"/>
14:   <xsl:apply-templates select="автор"/>
15:   </P>
16: </xsl:template>
17: <!-- шаблон, выбирающий название -->
18: <xsl:template match="название">
29:   Название: <span style="color:red">
20:   <xsl:value-of select="."/>
21:   </span><BR/>
22: </xsl:template>
23: <!-- шаблон, выбирающий автора -->
24: <xsl:template match="автор">
25:   Автор: <span style="color:blue">
26:   <xsl:value-of select="."/>
27:   </span><BR/>
28: </xsl:template>
29: </xsl:stylesheet>
```

•

Главный шаблон (строки 5-9) выбирает корневой узел. Тело этого шаблона описывает документ HTML в целом. Содержание HTML вставляется с помощью инструкции apply-templates (применить шаблоны).

Далее XSLT-процессор пытается применить другие шаблоны. Подходящим является шаблон, выбирающий элемент книга, доступный в корневом узле. Этот шаблон (строки 11-16) формирует параграф, содержание которого заполняется с помощью инструкций apply-templates. В отличие от первой инструкции apply-templates (в строке 7), эти инструкции задают названия выбираемых элементов с помощью атрибута select.

Два оставшихся шаблона формируют строки параграфа, выбирая значения текущего узла с помощью инструкций value-of запрос XPath которых имеет вид "." (точка выбирает текущий узел).

Применение этой трансформации к файлу sbooks.xml (листинг 57), дает вывод, показанный на рисунке 10.

Название: XML для программистов
Автор: Пономарев В.В.

Название: PHP для программистов
Автор: Пономарев В.В.

Рисунок 10 — Результат трансформации шаблонами sbookst.xml

5.1.5. Инструкции вызова шаблонов

Шаблон с атрибутом match вызывается инструкцией apply-templates.

```
<xsl:apply-templates select="xpath" mode="mode">
  <xsl:sort select="xpath" ... />
  <xsl:with-param name="имя" ... />
</xsl:apply-templates>
```

Внутри этой инструкции могут располагаться инструкции сортировки sort и параметров with-param.

Именованный шаблон вызывается инструкцией call-template.

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

Внутри этой инструкции могут располагаться только параметры.

5.1.6. Инструкция сортировки

С помощью инструкции sort выбираются сортируемые значения и дополнительные параметры сортировки.

```
<xsl:sort select="xpath" ... />
```

Атрибуты инструкции sort:

select — запрос, выбирающий узел или узлы для сортировки.

lang — код языка.

data-type="text | number | qname" — тип сортируемых данных.

order="ascending | descending" — порядок сортировки.

case-order="upper-first | lower-first" — указывает, какой регистр сортировать первым.

5.1.7. Управляющие инструкции

Следующие инструкции используются для описания алгоритмов обработки. Они соответствуют операторам в языках программирования.

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

Инструкция for-each выполняет указанные действия для всех дочерних узлов выборки, заданной атрибутом select.

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

Инструкция if выполняет действия при истинном условии. Пример условия приведен в листинге 60, строка 7.

Листинг 60. Инструкция if (sbooksc.xsl)

```
01: <?xml version="1.0" encoding="utf-8"?>
02: <xsl:stylesheet version="1.0"
03:   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
04: <xsl:template match="/">
05:   <HTML><BODY>
06:   <xsl:for-each select="книги/книга">
07:     <xsl:if test="starts-with(название, 'XML')">
08:       <P>
09:         <xsl:value-of select="название"/><BR/>
10:         <xsl:value-of select="автор"/>
11:       </P>
12:     </xsl:if>
13:   </xsl:for-each>
14:   </BODY></HTML>
15: </xsl:template>
16: </xsl:stylesheet>
```

•

Условие выбирает книги, название которых начинается с "XML".

Применение этой трансформации к файлу sbooks.xml (листинг 57), формирует параграф из двух строк с описанием только первой книги.

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

Инструкция 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.

5.2. XSLT-процессор

Листинг 61 показывает, как создать процессор XSLT и применить трансформацию к документу XML в сценарии на языке PHP.

Листинг 61. Процессор XSLT

```
01: <?php
02: # выводить как HTML
03: header("Content-Type: text/html; charset=utf-8");
04: # создать XML
05: $domxml = new DOMDocument();
06: # загрузить XML
07: $domxml->load("sbooks.xml");
08: # создать XSL
09: $domxsl = new DOMDocument();
10: # загрузить XSL
11: $domxsl->load("sbooks.xsl");
12: # процессор XSLT
13: $xslt = new XSLTProcessor();
14: # присоединить XSL
15: $xslt->importStylesheet($domxsl);
16: # преобразовать XML
17: echo $xslt->transformToXML($domxml)."<BR/>";
18: echo $xslt->transformToURI($domxml, "books.xml");
19: ?>
```

•

Сначала создаются объектные модели XML-документа и трансформации (строки 4-11). Затем создается процессор XSLT (строка 13). В строке 15 трансформация импортируется в процессор.

Методы transformXXXX применяют трансформацию к документу XML, объект которого передается первым параметром. В зависимости от метода трансформации, формируется либо строка (в строке 17), либо файл определенного типа (в строке 18).

Кроме показанных методов transformToXML и transformToURI есть еще метод transformToDOC. Этот метод возвращает результат трансформации в виде объекта DOMDocument.

Если главный шаблон трансформации требует параметры, каждый отдельный параметр задается с помощью метода setParameter.

```
bool setParameter($namespace, $name, $value)
```

Первый параметр метода задает пространство имен параметра, второй — имя параметра, третий — значение. Все параметры имеют строковый тип. Если пространство имен не задано или неизвестно, подставляется пустая строка.

Следующий пример показывает, как передать параметр главному шаблону и использовать включение других трансформаций.