*Лекция 15: Использование шаблонов в PHP*

### **Что такое шаблоны и зачем они нужны**

Что такое *шаблон* в языке программирования? Можно сказать, что ***шаблон*** - это текст с *переменными* внутри него. При обработке *шаблона* происходит замена *переменных* на их значения.

В одной из лекций мы уже рассматривали пример *шаблона*. Это был *шаблон* для отображения документов. *Пользователь* создавал строку текста, размеченного с помощью html-тегов, и вставлял в нее специальные *метасимволы* (вида <!имя элемента> ), которые наша *программа* впоследствии заменяла на значения соответствующих элементов. Для чего нам был нужен такой *шаблон*? Чтобы, например, можно было изменить стиль отображения документа, не меняя кода программы.

Наиболее распространенный ответ на вопрос, зачем нужны *шаблоны*, звучит примерно так: *шаблоны* нужны для того, чтобы отделить логику работы приложения от способа представления данных, т. е. от дизайна.

Приведенный пример *шаблона* - один из самых простых. Для его обработки используется только *функция* *подстановки* *str\_replace()*. Чаще всего для того, чтобы работать с *шаблонами*, создают библиотеки классов. В принципе создавать свою библиотеку не обязательно, поскольку существует множество свободно распространяемых библиотек *шаблонов*, над функциональностью которых трудятся большие коллективы разработчиков, стараясь сделать их универсальными, мощными и быстрыми. Некоторые из таких библиотек мы и рассмотрим. Но для начала сформулируем задачу, на примере решения которой будем демонстрировать использование различных *шаблонов*.

Итак, задача:

Требуется сгенерировать web-страницу со списком статей, имеющихся в базе данных. Для простоты считаем, что статья имеет название title, автора author, краткое содержание abstract и полное содержание fulltext, которое представлено либо в виде текста в базе данных, либо в виде ссылки на *файл*. *Список* должен быть организован так, чтобы при щелчке мышью на названии статьи ее полное содержание появлялось в новом окне.

### **Шаблоны подстановки**

Как можно решить такую задачу способом простой *подстановки*, т.е. тем *методом*, которым мы решили задачу отображения документов?

Нужно придумать *шаблон* для этой страницы и где-то его хранить (в файле или в базе данных). Очевидно, что мы не можем придумать *шаблон* для всей страницы, потому что не знаем, сколько статей в базе данных. В *шаблоне* же мы договорились использовать только html и *метасимволы* <!имя элемента>. Поэтому мы можем написать только *шаблон* для одной строки списка, который уже программно надо преобразовать в нужное количество строк.

<li><a href="<!fulltext>"

target=new><!title></a>

(<!author>)<br><p><!abstract></p>

Кроме того, здесь есть еще одна загвоздка - с отображением ссылки на полный текст статьи. Если мы будем действовать по правилу *подстановки* (менять все *метасимволы* на их значения из *базы данных*), то может получиться, что вместо <!fulltext> вставим не ссылку на текст, а сам текст. То есть для этого элемента нужна дополнительная проверка перед заменой и какие-то дополнительные действия в случае, если в *поле* fulltext содержится текст статьи, а не *ссылка* на *файл*. Не будем усложнять себе жизнь и договоримся, что в *поле* fulltext всегда содержится только *ссылка* на *файл*. Тогда задачу можно решить следующим образом:

<?

$li\_tmpl = file\_get\_contents("tmpl.html");

// считываем шаблон строки из файла

// устанавливаем соединение и выбираем

// базу данных

$conn = mysql\_connect("localhost",

"nina","123")

or die("Can't connect");

mysql\_select\_db("book");

$sql = "SELECT \* FROM Articles";

$q = mysql\_query($sql,$conn);

// отправляем запрос

$num = mysql\_num\_rows($q);

for($i=0; $i<$num; $i++){

$tmpl .= $li\_tmpl;

$tmpl = str\_replace("<!title>",

mysql\_result($q,$i,"title"),$tmpl);

$tmpl = str\_replace("<!author>",

mysql\_result($q,$i,"author"),$tmpl);

$tmpl = str\_replace("<!abstract>",

mysql\_result($q,$i,"abstract"),$tmpl);

$tmpl = str\_replace("<!fulltext>",

mysql\_result($q,$i,"fulltext"),$tmpl);

}

echo $tmpl;

?>

Если *шаблон* был такой, как приведен выше, то получим примерно следующее.

|  |  |
| --- | --- |
| **1.** | **Введение в PHP** (Савельева Н.В.) |
| Лекция дает представление о том, что такое язык PHP, для чего он создавался изначально и как используется теперь, какими возможностями обладает. |
| **2.** | **Установка и настройка ПО** (Иванов Иван) |
| Рекомендации по *установке* и настройке web-сервера и интерпретатора PHP |

В принципе *метод* достаточно прост и удобен, но требует дополнительных усилий программиста при возникновении задач более сложных, чем простая *подстановка* значений. Для решения задач, где требуется делать *подстановку* целых блоков или даже проверять условия, создают классы *шаблонов*, такие как *FastTemplate* и *Smarty*. Обсудим их подробнее.

### **Шаблоны FastTemplate**

***FastTemplate*** - это набор классов, позволяющих реализовать работу с *шаблонами* . Логику добавить в *шаблон FastTemplate* нельзя, вся она должна находиться в коде программы. Идея работы *шаблонов FastTemplate* заключается в том, что любая большая страница состоит из *множества* кусочков, самые маленькие из которых - обычные строки текста, и они получают имя и *значение*.

Что представляет собой *файл* *шаблона FastTemplate*? Это обычный html-*файл*, в котором могут встречаться *переменные* особого вида, впоследствии обрабатываемые *методами* *класса FastTemplate*.

*Синтаксис* *переменных* в *шаблонах FastTemplate* описывается следующим выражением: {([A-Z0-9\_]+)}

Это значит, что *переменная* должна начинаться с фигурной скобки "{". Второй и последующие символы должны быть буквами верхнего регистра от A до Z, цифрами или символами подчеркивания. *Переменная* вычисляется с помощью закрывающей фигурной скобки "}".

{TITLE}

{AUTH20}

{TOP\_OF\_PAGE}

Пример 15.1. Вычисление переменной с помощью закрывающей фигурной скобки "}"

Как уже было сказано, основная идея *FastTemplate* - создание страницы с помощью вложенных *шаблонов*. Например, для решения нашей задачи можно создать три файла *шаблона*:

1. **main.tpl** (Этот *шаблон* будет выводить страницу в целом)
2. <html>
3. <head><title>{TITLE\_}</title>
4. </head>
5. <body>
6. {MAIN}
7. </body>

</html>

1. **list.tpl** (будет описывать, как выводить список в целом)
2. <ul>
3. {LIST\_ELEMENT}

</ul>

1. **list\_element.tpl** (описывает непосредственно элемент списка)
2. <li><a href="{FULLTEXT}">{TITLE}</a>
3. ({AUTHOR})

<br> <p> {ABSTRACT}

*Шаблоны* мы создали - работу дизайнера выполнили. Теперь нужно научиться их обрабатывать, т.е. выполнить работу программиста. Сейчас создадим программу для обработки приведенных выше *шаблонов*.

Перед началом работы с *шаблонами FastTemplate* нужно подключить этот набор классов к нашей программе. В реальной жизни набор *классов FastTemplate* записан в один *файл*, как правило, с названием *class.FastTemplate.php3*, поэтому подключить его можно, например, с помощью команды:

include("class.FastTemplate.php3");

Следующий важный шаг - это создание *объекта класса FastTemplate*, с которым впоследствии мы будем работать:

$tpl = new FastTemplate(

"/path/to/templates");

В качестве параметра передается *путь* к месту, где находятся наши *шаблоны*.

#### **Методы FastTemplate**

Далее необходимо изучить *методы*, которые можно применять к созданному *объекту класса FastTemplate*. Параллельно обратим внимание, как их можно использовать для решения нашей задачи.

Для работы с *FastTemplate* нужно знать четыре основных *метода*: *define*, *assign*, *parse* и *FastPrint*.

##### Метод define

Синтаксис:

define( array ( ключ => значение,

ключ1 => значение1, ... ))

*Метод* *define()* связывает имя файла *шаблона* с более коротким именем, которое можно будет использовать в программе. То есть "ключ" - это имя, которое мы будем использовать в программе для ссылки на файл *шаблона*, имя которого записано в строке "значение". Реальные имена файлов *шаблонов* не рекомендуется использовать нигде, кроме *метода* *define*. При вызове *метода* *define()*происходит загрузка всех определенных в нем *шаблонов*.

$tpl->define( array (main => "main.tpl",

list\_f => "list.tpl",

list\_el=> "list\_element.tpl" ));

Пример 15.2. Использование метода define()

Здесь мы задаем псевдонимы именам файлов *шаблонов*. Эти псевдонимы, т.е. *переменные* main, list\_f и list\_el, будут использоваться в программе вместо соответствующих имен файлов main.*tpl*, list.*tpl* и list\_element.*tpl*.

##### **Метод assign**

**Синтаксис:**

assign( (пары ключ/значение) или

( array(пары ключ/значение) )

*Метод* *assign()* присваивает *переменным* значения, "ключ" - это имя *переменной*, а "значение" - значение, которое ей нужно присвоить. Чтобы *переменная* в *шаблоне* была заменена значением, это значение нужно задать ей с помощью *метода* *assign()*. Согласно синтаксису этот *метод* можно использовать в двух различных формах. В *FastTemplate* есть только один массив, поэтому, если вы повторно задаете значение одному и тому же ключу, оно будет перезаписано.

$tpl->assign(array(

TITLE => "Установка и настройка ПО",

TITLE => "Введение в PHP" ));

Пример 15.3. Использование метода assign()

Здесь мы дважды устанавливаем значение *переменной*, доступной в файлах *шаблона* по имени TITLE. Эта *переменная* будет иметь последнее присвоенное ей значение, т.е. она равна строке "Введение в PHP".

##### **Метод parse**

Синтаксис:

parse (возвращаемая переменная,

источники шаблонов)

*Метод* *parse()* - самый основной в *FastTemplate*. Он устанавливает значение возвращаемой *переменной* равным обработанным *шаблонам* из указанных источников. *Метод* может использоваться в трех формах: простой, составной и присоединяющей.

$tpl->parse(MAIN, "main");

// простая форма

$tpl->parse(MAIN, array ("list\_f", "main"));

// составная форма

$tpl->parse(MAIN, ".list\_el");

// присоединяющая форма

В простой форме *шаблон* с псевдонимом "main" загружается (если еще не был загружен), все его *переменные* подставляются, и результат сохраняется как значение *переменной* MAIN. Если *переменная* {MAIN} появится в более поздних *шаблонах*, то вместо нее будет подставлено значение, полученное в результате обработки *шаблона* "main". Это позволяет создавать вложенные *шаблоны*.

Составная форма *метода* *parse()* создана для того, чтобы упростить вложение *шаблонов* друг в друга. Следующие записи эквивалентны:

$tpl->parse(MAIN, "list\_f");

$tpl->parse(MAIN, ".main");

это то же самое что и

$tpl->parse(MAIN, array("list\_f", "main"));

Когда используется составная форма, важно, чтобы каждый *шаблон*, идущий в списке после первого, содержал ту *переменную*, в которую передаются результаты обработки *шаблона*. В примере выше main должен содержать *переменную* {MAIN}, поскольку именно в нее передаются результаты обработки *шаблона* list\_f. Если main не содержит *переменной* {MAIN}, то результаты обработки *шаблона* list\_f будут потеряны.

Присоединяющий стиль позволяет добавлять результаты обработки *шаблона* к *переменной* результата. Точка перед псевдонимом файла *шаблона* говорит *FastTemplate* о том, что нужно присоединить результат обработки этого *шаблона* к возвращенным результатам, а не перезаписывать его. Такой стиль наиболее часто используется при построении таблиц с переменным числом рядов, получаемых, например, в результате запроса к базе данных.

##### **Метод FastPrint**

Синтаксис:

FastPrint(обработанная переменная)

*Метод* *FastPrint()* печатает содержимое переданной в него обработанной *переменной*. Если он вызван без параметров, то печатается последняя использованная *методом* *parse()* *переменная*.

$tpl->FastPrint();

/\* если продолжать предыдущий пример,

то эта функция напечатает значение

переменной MAIN \*/

$tpl->FastPrint("MAIN");

// эта функция сделает то же самое

Пример 15.4. Использование метода FastPrint()

Если нужно печатать не на экран, а, например, в файл, то получить ссылку на данные можно с помощью *метода* *fetch()*.

$data = $tpl->fetch("MAIN");

fwrite($fd, $data); // запись данных в файл

#### Решение задачи с помощью шаблонов FastTemplate

Теперь попробуем собрать воедино все изученные *методы*, чтобы решить нашу задачу.

<?php

include("class.FastTemplate.php3"); //подключаем класс

//шаблонов FastTemplate

$tpl = new FastTemplate("c:/users/nina/tasks/"); //создаем

//объект FastTemplate

//задаем псевдонимы для имен файлов шаблонов

$tpl->define( array( main => "main.tpl",

list\_f => "list.tpl",

list\_el=> "list\_element.tpl" ));

// Присваиваем переменной TITLE\_ значение "List of articles"

$tpl->assign(TITLE\_, "List of articles");

/\* далее, как и раньше, устанавливаем соединение с базой

и получаем из нее значения нужных элементов \*/

$conn = mysql\_connect("localhost","nina","123")

or die("Can't connect");

mysql\_select\_db("book");

$sql = "SELECT \* FROM Articles";

$q = mysql\_query($sql,$conn);

$num = mysql\_num\_rows($q);

for($i=0; $i<$num; $i++){

$title = mysql\_result($q,$i,"title");

$author = mysql\_result($q,$i,"author");

$abs = mysql\_result($q,$i,"abstract");

$full = mysql\_result($q,$i,"fulltext");

// присваиваем полученные значения переменным,

// использованным внутри шаблонов

$tpl->assign(array(

TITLE => $title,

AUTHOR => $author,

ABSTRACT => $abs,

FULLTEXT => $full ));

/\* подставляем вместо переменных значения в шаблоне list\_el и

добавляем полученное к переменной LIST\_ELEMENT \*/

$tpl->parse(LIST\_ELEMENT,".list\_el");

}

//подставляем значения в шаблоны list\_f и main

$tpl->parse(MAIN, array("list\_f","main"));

Header("Content-type: text/plain");

$tpl->FastPrint(); // выводим обработанный шаблон на экран

exit;

?>

Листинг 15.4.1. Решение задачи с помощью шаблонов FastTemplate

Заметим, что решение задачи получилось несколько более сложным, чем в первом случае, когда использовалась только *функция*регулярной замены. Зато здесь мы можем изменять три различных *шаблона* (документа в целом, списка и элемента списка).

Этот класс *шаблонов* появился еще до выхода *PHP4* для работы с *PHP3*. Чтобы протестировать приведенные примеры, нужно скачать библиотеку *классов FastTemplate* и скопировать этот файл в свою рабочую директорию. Если вы работаете с *PHP4*, то в файл *class.FastTemplate.php3* нужно внести пару изменений, о которых написано в документации, поставляющейся вместе с этой библиотекой.

### **Шаблоны Smarty**

*Smarty* - один из действующих проектов *PHP*, его официальный *сайт* - <http://www.smarty.net/>. Там можно скачать как сам набор классов *Smarty*, так и документацию к нему. Этот набор классов для обработки *шаблонов* - гораздо более мощный и функциональный, чем *FastTemplate*.Чем отличается *Smarty* от классов *шаблонов* типа *FastTemplate*? Прежде всего, он не отделяет полностью логику от содержания. Логика, касающаяся отображения данных, может присутствовать в *шаблоне*, считают разработчики *Smarty*. Поэтому в *шаблоне Smarty* могут быть условные *операторы, операторы* вставки файлов, *операторы* изменения *переменных*, циклы и т.п. Другая особенность *Smarty* - это *компиляция* *шаблонов*. *Шаблоны* переводятся в *php*-код, и *интерпретатор* *PHP* производит все необходимые действия по *подстановке* значений. Для ускорения работы скомпилированные *шаблоны* кэшируются.

Рассмотрим некоторые основные конструкции механизма *шаблонов Smarty*.

#### **Установка**

Первое, с чего мы начнем, - это *установка* *Smarty*. Здесь все не так просто, как с *FastTemplate*. *Smarty* состоит не из одного php-файла с описанием класса, а из целого набора различных файлов-библиотек. Для того чтобы работать с *шаблонами Smarty*, нужно сделать эти библиотеки доступными для всех ваших программ. Находятся они в каталоге /libs/ дистрибутива *Smarty*. Файл, в котором содержится определение самого класса *Smarty*, называется Smarty.class.php. Чтобы проверить, *доступны ли библиотеки класса* *Smarty*, нужно написать такой скрипт:

<?

require('Smarty.class.php');

// подключаем файл с

// описанием класса Smarty

$smarty = new Smarty;

// создаем экземпляр класса Smarty

?>

Если при его исполнении появилась ошибка, то нужно попробовать один из перечисленных ниже вариантов.

1. Указать полный путь до *файла описания* класса.
2. <?
3. // подключаем файл с описанием класса
4. require('c:/users/my/Smarty/libs/
5. Smarty.class.php');
6. $smarty = new Smarty;
7. // создаем экземпляр класса Smarty

?>

1. Добавить директорию, где содержатся библиотеки, в *include\_path* (в этом случае код менять не нужно).
2. Установить константу SMARTY\_DIR.
3. <?
4. define("SMARTY\_DIR",
5. "c:/users/my/Smarty/libs/");
6. require(SMARTY\_DIR."Smarty.class.php");
7. $smarty = new Smarty;

?>

Теперь, после того как мы убедились, что библиотеки будут найдены, нужно создать *директории*, необходимые для работы *Smarty*, по умолчанию имеющие такие имена:

* *templates* - директория, где мы будем хранить созданные *шаблоны* ;
* *templates\_c* - директория, где *Smarty* будет хранить скомпилированные *шаблоны* ;
* *configs* - директория для хранения *конфигурационных файлов* ;
* *cache* - директория для кэша.

Эти имена задаются свойствами *$template\_dir*, *$compile\_dir*, *$config\_dir*, *$cache\_dir* класса *Smarty*, поэтому их можно переопределить. Рекомендуется использовать различные наборы *директорий* для каждого приложения, работающего с *шаблонами Smarty*. Доступ к перечисленным *директориям* осуществляется библиотекой *Smarty* и никогда не выполняется напрямую через web-браузер. Поэтому, чтобы избежать проблем с безопасностью, лучше всего разместить эти *директории* там, куда нет доступа через www.

Создадим перечисленные *Smarty директории* по адресу c:/smarty\_dirs/book/. Заметим, что прочитать отсюда данные через браузер нельзя. Пусть наша программа (или приложение) находится по адресу /~my/tasks/book/. Для *директорий* *$compile\_dir* и *$cache\_dir**Smarty* потребуется доступ на запись, так что нужно установить соответствующие им права для сетевого пользователя, с которым ваш сервер работает по умолчанию (обычно это www или nobody).

Чтобы протестировать сделанные настройки, создадим простой *шаблон* и программу, обрабатывающую его с использованием механизма *Smarty*.

index.*tpl* (является *Smarty шаблоном* и находится в *директории* *шаблонов* c:/smarty\_dirs/book/templates/)

{\* Шаблон Smarty \*}

Привет, {$name}!

index.php (является кодом нашей программы и находится в *директории* /~my/tasks/book/ или, что то же самое, в *директории*c:/users/my/tasks/book/ )

<?

// загружаем Smarty-библиотеку и создаем экземпляр класса

define("SMARTY\_DIR","c:/users/my/Smarty/libs/");

require(SMARTY\_DIR."Smarty.class.php");

$smarty = new Smarty;

// указываем, где находятся Smarty-директории

$smarty->template\_dir = "c:/smarty\_dirs/book/templates/";

$smarty->compile\_dir = "c:/smarty\_dirs/book/templates\_c/";

$smarty->config\_dir = "c:/smarty\_dirs/book/configs/";

$smarty->cache\_dir = "c:/smarty\_dirs/book/cache/";

$smarty->assign("name","Вася"); // присваиваем переменной

// name значение Вася

$smarty->display("index.tpl"); // выводим обработанный

// шаблон

?>

Листинг 15.4.2. index.php

В результате должны получить:

Привет, Вася!

Все настройки, необходимые для работы нашего приложения, можно вынести в отдельный файл и организовать их в качестве *расширения класса* *Smarty*.

Далее более подробно рассмотрим, из каких элементов могут состоять *шаблоны Smarty* и как их обрабатывать внутри php-скрипта. Начнем с синтаксиса *шаблонов*.

#### **Основной синтаксис**

*Smarty* - не просто класс для обработки *шаблонов*, он определяет целый язык построения *шаблонов*. Мы коснемся только основных его элементов. Итак, что представляет собой *шаблон Smarty*? Это набор специальных конструкций ( *переменных*, вызовов *функций* и *методов* и т.п) и html-тегов. Все элементы (теги) языка *шаблонов Smarty* заключаются между символами-ограничителями. По умолчанию это символы фигурных скобок "{" и "}", но их можно изменить. Все, что не заключено в такие ограничители, *Smarty*рассматривает как константы, не требующие обработки. В *шаблоне* index.*tpl*, приведенном выше, {$name} - это *переменная*, а строки "Привет," и "!" - не изменяющиеся в процессе обработки *шаблона* константы.

*Комментарии* в *Smarty* записываются между двумя звездочками:

{\* Это комментарий. После обработки шаблона

он на экране не отображается \*}

Каждый *Smarty* тег либо выводит значение *переменной*, либо вызывает какую-либо *функцию*. *Функция* записывается следующим образом:

{имя\_функции атрибут1="значение1"

атрибут2="значение2"}

*Переменные* в *шаблоне* могут быть нескольких типов:

* *Переменные*, значение которым присваивается в *php-скрипте* пользователя, должны иметь перед именем знак доллара.

Например: {$first\_name}

* Элементы массива, значения которых были присвоены в *php-скрипте* пользователя, доступны в *шаблоне* с помощью синтаксиса {$имя\_массива.ассоциативный\_ключ}.

Например: {$person.last\_name}

Элементы не ассоциативного массива доступны с помощью *синтаксиса квадратных скобок*: {$имя\_массива[числовой\_индекс]}

Например: {$person[2]}

* Свойства объектов, заданные в *php-скрипте*, доступны в *шаблоне* с помощью такого синтаксиса: {$имя\_объекта->имя\_свойства}

Например: {$person->email}

* *Переменные*, загруженные из *конфигурационных файлов* (что это такое, мы расскажем чуть позже), заключаются между символами #. Также они доступны как элементы ассоциативного массива $smarty.config.

Например: {#bodyBgColor#} или {$smarty.config.bodyBgColor}

Кроме того, существует *переменная* {$smarty}, зарезервированная для некоторых специальных *переменных* *шаблона*, таких как *переменные* HTTP запроса, даты и времени, и т.п.

В *шаблонах Smarty* определен ряд *модификаторов*, которые можно применять к *переменным*, пользовательским *функциям* или строкам с тем, чтобы модифицировать их значения. Чтобы применить *модификатор*, нужно указать его название после вертикальной черты, следующей за именем *переменной*, *функции* или строкой, к которой он применяется.

Например, чтобы перевести значение *переменной* {$title} в верхний регистр, нужно применить к ней *модификатор* upper, т.е. написать следующее: {$title|upper}

Можно использовать сразу несколько *модификаторов*, отделяя их друг от друга прямой вертикальной чертой. Например, {$title|upper|*truncate*} переведет значение *переменной* в верхний регистр и урежет до 80 символов.

Перечислять все имеющиеся *модификаторы* мы не будем. Их список можно найти в документации *Smarty*. Скажем только, что с их помощью можно посчитать число символов, слов и параграфов, дописать строку, задать *формат вывода* даты и времени, сделать регулярную замену и многое другое.

#### **Конфигурационные файлы**

*Конфигурационные файлы* используются для того, чтобы управлять *глобальными переменными*, используемыми в *шаблоне*, с помощью одного файла. Их идея очень похожа на таблицы стилей css. *Конфигурационный файл* содержит набор *переменных* и их значения. Перед именем *переменной* не ставится никаких дополнительных символов типа знака доллара. Значение *переменной* по желанию заключают в кавычки (двойные или одинарные), если оно состоит из нескольких строк, то его заключают в тройные кавычки.

# глобальные переменные

pageTitle = "List of documents"

bodyBgColor = #000000

tableBgColor = #000000

rowBgColor = #00ff00

[Customer]

pageTitle = "Список статей"

Intro = """Это значение состоит из

нескольких строк. Поэтому его нужно

заключить в тройные кавычки."""

# скрытая секция

[.Database]

host=localhost

db=book

user=nina

pass=123

Пример 15.5. Пример конфигурационного файла

*Конфигурационный файл* может состоять из нескольких разделов ( *секций* ), каждая из которых имеет имя и может загружаться отдельно от остальных *секций*. Имя *секции* заключается в квадратные скобки. Кроме *секций* в *конфигурационном файле* могут содержаться *глобальные переменные* - они не входят ни в одну *секцию* и всегда загружаются при загрузке *конфигурационного файла*. Если загружается какая-то одна *секция*, то загружаются ее *переменные* и *глобальные переменные*. Если *переменная* существует и как *глобальная переменная*, и как *переменная* *секции*, то используется *переменная* *секции*. Если вы одинаково назовете две *переменные*внутри одной *секции*, то будет использоваться последняя из них. В приведенном выше примере две *секции* - Customer и Database, а кроме них заданы *глобальные переменные* pageTitle, bodyBgColor, tableBgColor и rowBgColor.

Чтобы спрятать значение *переменной* или *секцию* целиком, нужно перед ее именем поставить точку. В таком случае при загрузке *конфигурационного файла* эти данные нельзя будет прочесть. В примере мы сделали *скрытой секцию* Database, чтобы нельзя было узнать пароль и имя пользователя, применяемые для установки соединения.

*Комментарии* в *конфигурационном файле* можно обозначать символом #.

Загрузка *конфигурационных файлов* производится с помощью *встроенной функции* или *метода* *config\_load*, подробнее об этом мы расскажем в следующей главе.

#### **Методы**

Для работы с *шаблонами* класс *Smarty* определяет набор *методов*. Рассмотрим несколько основных *методов*.

##### **Метод assign**

Синтаксис:

void assign (смешанное значение);

void assign (имя переменной,

смешанное значение);

*Метод* используется для того, чтобы присвоить значения *переменным* *шаблона*. Можно передавать ассоциативные массивы, содержащие пары имя/значение *переменных*, или передавать пары имя/значение для каждой *переменной* в отдельности.

<?php

// передаем пары имя/значение для

// переменной Name и

// Address в отдельности

$smarty->assign("Name","Вася");

$smarty->assign("Address",$addr);

// здесь $addr может быть и массивом

// передаем ассоциативный массив

$smarty->assign(array(

"city" => "Новосибирск",

"street" => "Пирогова"));

// таким образом, переменные city и street

// получат соответствующие значения

?>

Пример 15.6. Использование метода assign()

##### **Метод append**

Синтаксис:

void append (смешанное значение);

void append (имя переменной,

смешанное значение);

void append (имя переменной,

смешанное значение, слияние);

Принцип действия этого примерно такой же, как и у *assign*. *Метод* *append* позволяет присоединить элемент к массиву. Если вы присоединяете значение к строковой *переменной*, то она преобразуется в массив, и значение добавляется уже в него. Так же, как и в *assign*, здесь можно передавать пары ключ/значение или ассоциативные массивы, содержащие эти пары. Если указать третий аргумент слияние равным TRUE, то значение будет не присоединено в качестве еще одного элемента, а слито воедино с текущим массивом.

$smarty->append(array(

title => $title,

author => $author))

Пример 15.7. Использование метода append()

Здесь если title была строкой, то она становится массивом и к нему добавляется еще один элемент со значением $title. То же самое происходит с *переменной* author.

##### **Метод config\_load**

void config\_load(имя файла, [имя секции]);

*Метод* загружает *конфигурационный файл* и встраивает его в *шаблон*. Аналогично этому *методу* действует *функция* *config\_load*.

Пример: $smarty->*config\_load*("task.conf","Database");

##### **Метод display**

void display(шаблон);

*Метод* отображает *шаблон*. У этого *метода* есть еще два опциональных параметра, о которых можно прочитать в документации.

##### **Метод fetch**

string fetch(шаблон);

Этот *метод* возвращает обработанный *шаблон* в строковую *переменную*, вместо того чтобы выводить его на экран. У этого *метода* есть еще два опциональных параметра, о которых можно прочитать в документации.

#### **Встроенные функции**

*Smarty* поставляется с набором *встроенных функций*, интегрированных в язык *шаблонов*. Нельзя создавать свои *функции* с такими же именами или модифицировать *встроенные функции*. Опишем некоторые из таких *функций*.

##### **Функция config\_load**

Синтаксис:

{config\_load file="имя\_файла" }

Эта *функция* используется для загрузки в *шаблон* *переменных* из *конфигурационных файлов*. Кроме имени загружаемого файла, у этой *функции* может быть еще несколько дополнительных параметров. Например, параметр *section*, в котором указывают имя *секции* для загрузки. Более подробную информацию об этих и других параметрах можно получить из документации *Smarty*.

Пример:

{config\_load file="task.conf"}

##### 

##### **Функция capture**

Синтаксис:

{capture name="имя\_блока"

assign="имя\_переменной"} ...

{/capture}

Эта *функция* предназначена для того, чтобы собирать в *переменную* выходные данные *шаблона* вместо того, чтобы выводить их на экран. Все, что находится между {capture name="varname"} и {/capture}, будет записано в *переменную* с именем varname. Захваченный таким образом контент может быть использован в *шаблоне* посредством специальной *переменной*$smarty.capture.varname, где varname - значение, переданное в атрибут name *функции* *capture*. Если имя *переменной* не задано, будет использовано имя default.

Второй параметр *assign* задает имя *переменной*, которой будет присвоено захваченное выходное значение. Этот параметр, как и name, не обязательный.

##### **Функция section**

Синтаксис:

{section name="имя\_секции"

loop="переменная\_для\_выч-ния\_числа\_итераций"

[,start="индекс\_начальной\_позиции"]

[, step="шаг"] [,max="максимум\_итераций"]

[,show="показывать\_ли\_секцию"] }...

{/section}

*Секция* *Section* - это цикл для обхода элементов массива. Обязательными являются параметры name, с помощью которого задается имя *секции*, и loop, который представляет собой *переменную*, определяющую число итераций цикла. Как правило, loop - это *переменная* типа массив, и число итераций *секции* равно числу элементов этого массива. Чтобы вывести *переменную* внутри цикла, нужно после имени *переменной* указать в квадратных скобках имя *секции*.

{section name=art loop=$title}

Название: {$title[art]}<br>

{/section}

Пример 15.8. Цикл для обхода элементов массива

##### **Функция foreach**

Синтаксис:

{foreach from="имя\_массива"

item="имя\_текущего\_элемента"}

... {/foreach}

Кроме того, можно использовать дополнительные атрибуты key - имя ключа для текущего элемента массива и name - имя цикла, с помощью которого можно будет получать доступ к его свойствам. Атрибуты from и item - обязательные.

Циклы ***foreach*** являются альтернативой циклам *section*. Действие *функции* *foreach* очень похоже на работу цикла *foreach* в языке PHP.

{foreach from=$articles item=art}

Title: {$art}<br>

{/foreach}

Пример 15.9. Цикл foreach

Циклы *foreach* имеют свои собственные свойства. Получить доступ к ним можно таким образом: {$smarty.foreach.foreachname.varname}, где foreachname - это имя цикла, заданное его параметром name, а varname - имя свойства.

##### **Оператор if, elseif, else**

Синтаксис:

{if выражение} блок\_действий

{elseif выражение1} блок\_действий1

{else} блок\_действий2

{/if}

Действие оператора практически аналогично оператору *if*... *elseif*... *else* языка PHP. В выражениях могут использоваться следующие операторы сравнения: eq, ne, neq, gt, lt, *lte*, le, gte, ge, is even, is *odd*, is not even, is not *odd*, not, mod, div by, even by, *odd* by, ==, !=, >, <, <=, >=. Каждый из них обязательно должен быть отделен от окружающих его значений пробелами. В выражениях можно использовать круглые скобки и вызывать php- *функции*.

{if $name eq "Вася"}

Добро пожаловать, Вася.

{elseif $name eq "Петя"}

Добро пожаловать, Петя.

{else}

Добро пожаловать. А вы кто?

{/if}

Пример 15.10. Операторы if, elseif, else

{\* этот пример не будет работать,

поскольку не поставлены

пробелы вокруг операторов сравнения \*}

{if $name=="Вася" || $name=="Петя"}

...

{/if}

Пример 15.11. Неработающий пример

#### **Решение задачи с помощью шаблонов Smarty**

Теперь, после знакомства с основными конструкциями *Smarty*, мы можем попытаться решить задачу *отображения списка* документов. *Шаблон* списка будет выглядеть следующим образом:

{\* Smarty template index.tpl \*}

{config\_load file="task.conf" }

<html>

<head><title>{#pageTitle#}</title>

</head>

<body>

<ol>

{section name=art loop=$title}

<li><a href="{$fulltext[art]}">

{$title[art]}</a>

({$author[art]})

<br> <p> {$abstract[art]}

{/section}

</ol>

</body>

</html>

В *файле конфигурации* task.conf будем хранить название страницы и параметры для доступа к базе данных:

# глобальные переменные

pageTitle = "List of documents"

[Customer]

pageTitle = "Список статей"

[Database]

host=localhost

db=book

user=nina

pass=123

Скрипт ( index.php ), обрабатывающий написанный нами *шаблон*, может выглядеть таким образом:

<?

define("SMARTY\_DIR","c:/users/nina/Smarty/libs/");

require(SMARTY\_DIR."Smarty.class.php");

$smarty = new Smarty;

$smarty->template\_dir = "c:/smarty\_dirs/book/templates/";

$smarty->compile\_dir = "c:/smarty\_dirs/book/templates\_c/";

$smarty->config\_dir = "c:/smarty\_dirs/book/configs/";

$smarty->cache\_dir = "c:/smarty\_dirs/book/cache/";

// вышеприведенный блок лучше вынести в отдельный файл

$smarty->config\_load("task.conf","Database");

$host = $smarty->get\_config\_vars("host");

$user = $smarty->get\_config\_vars("user");

$pass = $smarty->get\_config\_vars("pass");

$db = $smarty->get\_config\_vars("db");

$conn = mysql\_connect($host, $user, $pass) or die("Can't connect");

mysql\_select\_db($db);

$sql = "SELECT \* FROM Articles";

$q = mysql\_query($sql,$conn);

$num = mysql\_num\_rows($q);

for($i=0; $i<$num; $i++){

$title = mysql\_result($q,$i,"title");

$author = mysql\_result($q,$i,"author");

$abs = mysql\_result($q,$i,"abstract");

$full = mysql\_result($q,$i,"fulltext");

$smarty->append(array(

title => $title,

author => $author,

abstract => $abs,

fulltext => $full ));

}

$smarty->display("index.tpl");

?>

Листинг 15.12. index.php

Как вы, скорее всего, заметили, программа получилась еще более громоздкой, чем в первых двух случаях, когда использовалась простая замена значений и *шаблоны FastTemplate*. Действительно, механизм *Smarty* гораздо более сложен, чем тот же *FastTemplate*, но зато и более функционален.

### ***Заключение***

Итак, в этой лекции было рассказано о том, что такое *шаблоны* и как их можно использовать при программировании web-приложений на языке *PHP*. Мы рассмотрели три способа решения задачи *отображения сложного списка* документов: с помощью *функции* регулярной замены, с помощью *класса шаблонов FastTemplate* и с помощью языка *шаблонов Smarty*. При этом мы познакомились с основными свойствами, *методами* и *функциями* классов *FastTemplate* и *Smarty*.