Введение в шаблонизатор Smarty

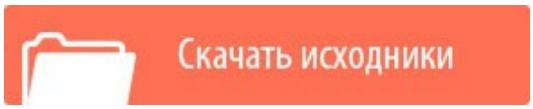
Дата публикации: 2012-02-13

От автора: *Одной из важнейших задач любого хорошего веб-разработчика является разделение логики сайта от его представления (дизайна). Это разделение обеспечивает несколько преимуществ:*

а) гибкость приложения – четкое разделение логики от дизайна позволяет программистам вносить изменения в логику и структуру сайта, не затрагивая его оформления, и в тоже время дает возможность дизайнерам изменять внешний вид сайта и при этом не затрагивать логическую часть веб – приложения;

б) код веб-приложения становится чистым и элегантным;

Думаю многие веб-разработчики, сталкивались с этой проблемой – решением данной задачи является применение шаблонов. В этом уроке мы будем изучать применение шаблонов с использованием шаблонизатора Smarty.



1. Формирование задачи

Давайте для начала определимся, что же такое шаблон? Шаблон — это довольно широкое понятие, но если говорить простым языком – это файл (документ), обычно в html или tpl формате, который содержит в себе html – теги (оформление и структуру веб-приложения), а также специальные метки, вместо которых подставляются данные из логической части данных. Подстановкой данных и формированием окончательного вида веб-приложения занимается шаблонизатор.

Существует огромное множество готовых шаблонизаторов для PHP. Конечно, шаблонизатор можно написать и самому – придумать свои метки для шаблона и разработать алгоритм замены этих меток. Но этот вариант не очень удобен, если Ваше веб-приложение будут обслуживать другие программисты или дизайнеры, так как в этом случае их придется посвящать в нюансы Вашего шаблонизатора. Поэтому лучше всего использовать готовое проверенное решение с хорошим функционалом и качественной документацией.

Smarty – наиболее популярная и широко распространенная система шаблонов на PHP. Работа Smarty заключается в компилировании шаблонов. Это означает, что Smarty последовательно считывает разметку файла шаблона и создает на их основе готовый PHP сценарий, этот сценарий создается один раз, а далее просто выполняется. Smarty содержит в себе большое количество функций, которые позволяют создавать в шаблонах сложную логику (если она нужна для правильного отображения данных): подключение других шаблонов, циклический разбор массива и т.д. Конечно, Вы можете и не создавать сложную логику шаблона, а ограничиться лишь использованием чистого текста и переменных.
На этом вводная часть закончена и можно приступать к изучению.

2. Установка Smarty

Скачать Smarty можно с официального сайта http://www.smarty.net/, перейдя на вкладку Download. На момент создания урока последняя стабильная версия это Smarty 3.1.7, ее и скачиваем. При распаковке архива мы видим папку Smarty 3.1.7, а в ней еще файлы и папки:

demo – демонстрационный пример использования шабонизатора;

libs – папка с дистрибутивом Smarty;

различные текстовые файлы(readme, описание условий копирования, описания отличий от более старых версий).
Для установки Smarty необходимо скопировать папку libs (из архива) в корневой каталог Вашего веб-приложения. Перечень файлов и папок каталога libs должен быть таким:

libs/
Smarty.class.php

SmartyBC.class.php
debug.tpl
sysplugins
plugins

Затем создать четыре директории, из которых Smarty будет читать свои конфигурационные файлы и файлы шаблонов. По умолчанию эти директории имеют название: *templates/, templates_c/, configs/, cache/(эти каталоги Вы можете назвать так, как Вам захочется, но при этом нужно будет указать шаболнизатору на их названия – это мы рассмотрим далее)*. Таким образом каталог с Вашим веб-приложением должен быть следующего вида:

www.example.com/(или папка с Вашим веб-приложением)
libs/
Smarty.class.php
SmartyBC.class.php
debug.tpl
sysplugins
plugins
templates
templates_c
configs
cache
index.php

Давайте рассмотрим, для чего нужны созданные четыре каталога:

- *templates* – здесь хранятся Ваши созданные шаблоны (**шаблоны для Smarty создаются в формате tpl**);
- *templates_c* – в этот каталог шаблонизатор записывает скомпилированный шаблон, на основе шаблона в каталоге templates;
- *configs* – каталог для хранения файлов конфигурации;
- *cache* – каталог для хранения кэшированных файлов шаблона.

3. Создание простого скрипта на основе Smarty

Теперь когда Smarty установлен и созданы необходимые каталоги давайте создадим первую страницу. Для этого первым делом необходимо подключить класс Smarty к нашему скрипту и создать объект этого класса(вся логика нашего скрипта будет в файле index.php):

```
<?php
//Подключаем класс Smarty
require_once 'lib/Smarty.class.php';
//Создадим объект класса Smarty
$smarty = new Smarty();
?>
```

Теперь когда Smarty подключен и создан объект его класса, давайте создадим переменную $name, с произвольным значением, и передадим это значение в Smarty и дальше выведем наш шаблон на экран(шаблон мы создадим ниже).

Так выглядит код файла index.php:

```php
<?php
//Подключаем класс смарти
require_once 'lib/Smarty.class.php';
//Создадим обьект класса смарти
$smarty = new Smarty();
//Создадим переменную для примера
$name = 'Vasya';
//Передаем переменную в шаблонизатор Smarty
$smarty->assign('name',$name);
//Выводим шаблон на экран
$smarty->display('main.tpl');
?>
```

Для тех кто мало знаком с обьектно-ориентированным программированием на PHP, $smarty – это обьект нашего класса Smarty(),assign() и display() – это его методы(так называются функции класса). Для доступа к методу класса сначала указываем обьект класса, далее два символа ->, далее сам метод и в скобках его параметры:

$smarty->assign('name',$name);

В этой строке мы у обьекта класса вызываем метод assign с двумя параметрами 'name', $name

Переменные передаются в шаблонизатор с помощью метода assign() в который нужно передать два параметра. Как Вы наверное догадались, первый параметр это название переменной, второй это ее значение(этот метод можно вызывать несколько раз, передавать можно как простые переменные, так и массивы). С помощью метода display() мы вызываем отображение нашего шаблона, имя шаблона передается как параметр обычной строкой(название файла с Вашим шаблоном).

Теперь давайте создадим файл шаблона main.tpl (**в каталоге templates с расширением файла tpl**):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
 <p>
  Переданная переменная - {$name}
 </p>
</body>
</html>
```

Как Вы видите это обычный html документ(только с расширением tpl), только с специальной меткой *{$name}*, в Smarty в фигурных скобках *{}* обозначаются все управляющие конструкции(теги) в нашем примере это переменная *$name* (та которую мы передали методом *assign()*). Таким образом что бы отобразить в шаблоне переменную необходимо в фигурных скобках указать имя переменной {$name}.

Если запустить скрипт на выполнение то на экране браузера мы увидим:

Переданная переменная – Vasya

Это и есть Ваш первый скрипт на основе шаблонизатора Smarty, как Вы видите логика сайта у нас содержится в файле *index.php*, а внешний вид — в файле *templates/main.tpl*, если Вы посмотрите в каталог templates_c, то увидите, что Smarty создал там скомпилированный файл шаблона содержимое которого и отобразил на экране.

4. Основные управляющие конструкции шаблонов

Комментарии в шаблонах оформляются таким образом:

```
{* this is a comment *}
```

Отображение переменных в шаблонах:

- *{$name}* – отображение простой переменной переданной Smarty с именем name.
- *{$name[4]}* – отображение пятого элемента массива.
- *{$name.key}* – отображение значения элемента массива с ключем key, аналогично записи в PHP $name[‘key’].
- *{$name.$key}* – отображение значения элемента массива с ключом, значение которого хранится в переменной $key, аналогично записи PHP $name[$key].
- *{#name#}* – отображение переменной name значение которой хранится в конфигурационных файла(перед отображением такой переменой конфигурационный файл нужно подгрузить в шаблоне – мы это рассмотрим ниже).

Условные операторы в шаблонах.

Шаблоны так же как и язык PHP поддерживают условные операторы, однако синтаксис их немного отличается от привычного нам. Каждый тэг *{if}* должен иметь пару *{/if}*. *{else}* и *{elseif}* так же допустимы. Доступны все квалификаторы из PHP, такие как ||, *or*, &&, *and*, *is_array()* и т.д. (более расширенный список квалификаторов вы можете посмотреть в документации на официальном сайте). Пример:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
 <p>
 {if $name == 'Vasya'}
  Переданная переменная - {$name}
 {else}
  Другая переменная
 {/if}
 </p>
</body>
</html>
```

В данном примере если переменная *$name* равна *Vasya*, то на экран выведется *Переданная*

переменная – Vasya, если не равна то — *Другая переменная*.

Цикл foreach в шаблонах.

Очень часто в шаблон необходимо передать массив данных(например выборку из базы даннх), в этом случае очень удобно использовать цикл foreach для последовательного перебора всех элементов массива. Синтаксис функции foreach также похож на синтаксис условного оператора *{if}*. Каждый тэг *{foreach}* должен иметь закрывающую пару *{/foreach}* (напомню, что все теги в Smarty записываются в фигурных скобках).

Синтаксис оператора foreach:

```
{foreach $arrayvar as $itemvar}
 ..действие..
{/foreach}
{foreach $arrayvar as $keyvar=>$itemvar}
..действие..
{/foreach}
```

Как Вы видите синтаксис очень похож на PHP: *$arrayvar* – передаваемый в шаблонизатор массив данных, *as* – ключевое слово для оператора foreach (как и в PHP), *$itemvar* – переменная в которую будет записано значение первого элемента массива, $keyvar – переменная в которую будет записан ключ первого элемента массива(как и в PHP).

Оператор {capture}.

Оператор *{capture}* используется для того, чтобы сохранить результат обработки части шаблона, между тэгами, в какую-то переменную, вместо того, чтобы отобразить результат. Любое содержимое между *{capture name='var'}* и *{/capture}* сохраняется в переменную, указанную в атрибуте *name*.

Чтобы вывести на экран сохраненные данные необходимо использовать специальную переменную специальной переменной *$smarty.capture.var*, где *var* — значение, переданное атрибуту *name*. Например давайте сохраним результат работы цикла *{foreach}* в переменную, а уже потом выведем результат на экран.

```
{capture name='var'}
 {foreach $arrayvar as $itemvar}
 {*..действие..*}
{/foreach}
{/capture}
{* ..действие.. *}
{* вывод на экран переменной var *}
{$smarty.capture.var}
```

Подключение вспомогательных шаблонов.

Очень часто необходимо в шаблон подключать дополнительные шаблоны, например подключение шапки сайта или футера. Для этого используется оператор *{include}*. Любые переменные, доступные в текущем шаблоне, доступны и в подключаемом. Тэг *{include}* должен иметь атрибут *'file'*, который указывает путь к подключаемому шаблону. Например:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
{*подключаем шапку шаблона*}
  {include file='page_header.tpl'}

 {*основной шаблон*}

{*подключаем футер шаблона*}
  {include file='page_footer.tpl'}
 </body>
</html>
```

В примере к основному шаблону мы подключаем файл шапки *‘page_header.tpl’* и файл футера *‘page_footer.tpl’*.

Отображение переменных из конфигурационных файлов.

Как Вы помните одним из обязательных каталогов в шаблонизаторе, есть каталог *configs*, который хранит в себе конфигурационные файлы шаблонов (эти файлы мы создаем сами). Так вот, если у нас есть переменные, которые не относятся к логике скрипта, а отвечают только за оформление веб-приложения(цвет, различные заголовки и т.д), мы можем создать файл конфигурации(**в папке** *configs* **в формате** *conf*), и прописать в нем все нужные нам переменные. Затем подгрузить этот файл в шаблоне и отобразить нужные нам переменные.

Для начала давайте рассмотрим, какой вид должен иметь файл конфигурации.

Для примера я создал файл *configs/my_conf.conf* , с таким содержанием:

#это комментарий конфигурационного файла

# глобальные переменные
pageTitle = «Hello world»
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

#секция переменных customer
[Customer]

pageTitle = «Customer Info»

Как видите синтаксис файла очень прост: # — решеткой обозначаются комментарии, далее с новой строки пишется имя переменной и после знака равно ее значение. Можно также переменные объединять в секции, для этого имя секции нужно прописать в скобках *[имя сесии]*, но переменные из секции будут доступны только после подгружения всей секции, но об этом ниже.

После того как файл создан давайте подгрузим его в шаблон (с помощью конструкции {config_load file='имя файла'}):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
 <p>
  {config_load file='my_conf.conf'}
  Переменная из конфигурационного файла - {#pageTitle#}
 </p>
</body>
</html>
```

Напомню что, если вы хотите отобразить значение переменной из конфигурационного файла то синтаксис будет выглядеть таким образом:

{#имя переменной#}

Если запустить скрипт на выполнение то на экране мы увидим — Переменная из конфигурационного файла — *Hello world*, что означает что файл конфигураций успешно подрузился и значение переменной было взято из него. Для отображение переменных из секции файла конфигураций необходимо при загрузке файла указать в теге *config_load* атрибут *section*, после этого переменные в секции будут доступны для отображения:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
 <p>
  {config_load file='my_conf.conf' section='Customer'}
  Переменная из конфигурационного файла - {#pageTitle#}
 </p>
</body>
</html>
```

После выполнения скрипта на экране мы увидим — *Переменная из конфигурационного файла — Customer Info*, обратите внимание, что теперь значение переменной *{#pageTitle#}* **взялось из секции**.

5. Обзор основных методов Smarty

В случае если Вы хотите изменить имена четырех системных директорий шаблонизатора, Вам потребуется указать имена этих каталогов с помощью следующих методов (имя каталога передаем параметром метода):

```
//определим основные директории
//каталог с конфигурационными файлами Smarty
$smarty->setConfigDir('configs');
//каталог с Вашими шаблонами в формате tpl Smarty
$smarty->setTemplateDir('templates');
//каталог с компилированными шаблонами Smarty
$smarty->setCompileDir('templates_c');
//каталог в котором хранится кеш шаблонов
$smarty->setCacheDir('cache');
```

Указывать эти каталоги нужно сразу же после создания обьекта Smarty.

Методы для работы с переменными.

Напомню, что для передачи переменной в шаблон используем метод *assign()*, с таким синтаксисом:

```
$smarty->assign('имя переменной', 'значение');
```

Что бы очистить переданную в шаблон переменную применяется метод *clearAssign('name')*, имя переменной передаем методу как параметр(в данном случае очищаем переменную *name*). Пример:

```
//Очищаем переменную $name
 $smarty->clearAssign('name');
```

Можно также параметром к методу, передать массив переменных, которые нужно очистить:

```
//очищаем несколько переменных
 $smarty->clearAssign(array('Name', 'Address', 'Zip'));
```

Для очищения всех переданных переменных используем метод *clearAllAssign()* (без параметров):

```
//Очищаем все переданные переменные
 $smarty->clearAllAssign();
```

Методы для работы с шаблонами.

Напомню, что выводит шаблон на экран метод *display()*, параметром передаем имя шаблона, например (отображаем шаблон *main.tpl*):

```
//Выводим шаблон на экран
 $smarty->display('main.tpl');
```

Как только мы вызываем метод *display()*, шаблонизатор проверяет изменился ли наш шаблон, если изменился, или мы вызываем метод первый раз – происходит компиляция нашего шаблона и вывод его на экран браузера. Если шаблон не изменился то он просто выводится на экран. За эту проверку отвечает свойство шаблонизатора *$compile_check*, которое по умолчанию равно *TRUE*, если его значение изменить на *FALSE* то проверки не будет, а после вызова метода *display()*, на экране будет отображаться ранее скомпилированный шаблон (это нужно в том случае, когда Вы точно знаете, что изменения в Ваш шаблон больше вноситься не будут). Пример:

```
//Отменяем проверку на изменения в шаблоне
$smarty->compile_check = FALSE;
```

Удалить компилированный шаблон можно при помощи метода *clearCompiledTemplate()*, параметром передаем имя шаблона, например удалим компилированный шаблон *main.tpl*:

```php
//удаляем компилированный шаблон
 $smarty->clearCompiledTemplate('main.tpl');
```

Иногда требуется не вывести шаблон на экран, а сохранить его отработанное значение в переменную. Для этого применяется метод *fetch('имя шаблона')*, параметром передаем имя файла шаблона. В следующем примере мы сохраняем отработку шаблона в переменную, и затем выводим его через привычную функцию *echo*:

```php
//Сохраняем отработку шаблона в переменную
$var = $smarty->fetch('main.tpl');
//Выводим на экран
echo $var;
```

Что бы получить переменные, которые уже переданы в шаблон, или получить значение одной конкретной переменной, переданной в шаблон, нужно воспользоваться методом *getTemplateVars()*, если вызвать этот метод без параметров то он вернет массив всех переданных шаблону переменных, если же в параметре передать имя переменной то метод вернет ее значение:

```php
//Для примера выведем на экран переданные переменные шаблону
print_r($smarty->getTemplateVars());
//Выведем на экран значение переменной name
echo $smarty->getTemplateVars('name');
```

Если Вы разрабатываете сложное веб-приложение с большим количеством шаблонов, то перед выводом шаблона на экран(во избежание ошибок), необходимо сделать проверку существует ли выводимый шаблон или нет, это позволяет сделать метод *templateExists('имя шаблона')*, который возвращает *TRUE* если шаблон существует и *FALSE* в противном случае(параметром передаем имя шаблона). Пример:

```php
//Проверяем существует ли шаблон
if( !$smarty->templateExists('main.tpl') ){
    exit('Такого шаблона не существует');
}
```

В примере если шаблона *main.tpl* не существует, то происходит завершение работы скрипта, но в реальном примере нужно бы было вызывать шаблон страницы, которая выводит на экран ошибки приложения – это уже на Ваше усмотрение от того что Вы делаете.

6. Кэширование в Smarty

Давайте рассмотрим последнюю тему, сегодняшнего урока — кэширование в Smarty. Кэширование применяется для ускорения работы методов *display()* или *fetch()*, при помощи сохранения результатов их работы в специальный файл, который сохраняется в папке cache. Если кэширование разрешено и файл доступен (уже сохранен ранее), то вместо повторной обработки шаблона, выводится кэшированная версия вызова. Кэширование может значительно ускорить работу, особенно в случае длительно обрабатываемых шаблонов. Так как результат работы методов *display()* или *fetch()* кэшируется, один кэшированный файл может состоять из нескольких файлов шаблонов, конфигурационных файлов и т.д.

Но кэшированием нужно пользоваться осторожно, нужно четко понимать, что Вы кэшируете. Например, если у Вас есть главная страница скрипта, содержимое которой меняется редко, в этом случае ее можно кэшировать на час и более. Если же у Вас страница, на которой содержимое меняется очень часто, то

смысла в кэшировании такой страницы нет.

Для начала кэширование нужно включить (по умолчанию оно выключено). Включить можно в двух режимах: первый – кэширование с кэшированием страниц на один час (время этого режима установлено по умолчанию), и второй – кэшированием с заданным временем кэширования (вы этот время задаете сами).

Сначала давайте рассмотрим первый вариант, и в наш самый первый пример после создания объекта Smarty, допишем следующее:

```
//Включаем кеширование страницы(по умолчанию на один час)
$smarty->setCaching(Smarty::CACHING_LIFETIME_CURRENT);
```

Вот как выглядит файл index.php. Шаблон остался без изменений.

```
<?php
//Подключаем класс смарти
require_once 'lib/Smarty.class.php';
//Создадим объект класса смарти
$smarty = new Smarty();
//Включаем кэширование страницы(по умолчанию на один час)
$smarty->setCaching(Smarty::CACHING_LIFETIME_CURRENT);
//Создадим переменную для примера
$name = 'Vasya';
//Передаем переменную в шаблонизатор Smarty
$smarty->assign('name',$name);
//Выводим на экран
$smarty->display('main.tpl');
?>
```

Метод *setCaching()* включает кэширование страницы, параметром к нему необходимо передать – какой режим кэширования нужно активировать, в данном примере мы передаем константу класса *CACHING_LIFETIME_CURRENT*, которая включает первый вариант кэширования(кэширование на один час). Далее при обновлении страницы мы все также увидим *Переданная переменная – Vasya*, но обратите внимание, если мы в скрипте изменим значение переменной $name на любое другое значение, то изменений мы не увидим, сколько бы не обновляли страницу (только когда пройдет один час и время жизни кэшированной файла истечет) – страница закэширована. Ее кэшированный файл Вы можете увидеть в каталоге cache. Теперь давайте удалим кэшированный файл, с помощью метода *clearCache('имя шаблона')*, параметром к которому передаем имя шаблона (кэш которого мы хотим удалить). Допишем перед методом *display()* следующий код:

```
//Удалим кэш шаблона
 $smarty->clearCache('main.tpl');
```

А сам метод *display()* закомментируем. После обновления странички браузера, посмотрите в каталог cache, Вы увидите, что кэшированный файл удалился. Далее расскомментируем метод *display()*, и обновим браузер – видно что переменная отобразилась с изменениями, и появился файл кэша в папке cache.

Чтобы включить режим кэширования с заданным временем кэширования нужно в метод *setCaching()* передать параметром константу класса *CACHING_LIFETIME_SAVED*, а дальше с помощью метода *setCacheLifetime* (время кэширования) установить время кэширования (передать параметром время в секундах):

```php
<?php
//Подключаем класс смарти
require_once 'lib/Smarty.class.php';
//Создадим объект класса смарти
$smarty = new Smarty();
//Включаем кэширование страницы(по умолчанию на один час)
$smarty->setCaching(Smarty::CACHING_LIFETIME_SAVED);
//Время кэширования
$smarty->setCacheLifetime(20);
//Создадим переменную для примера
$name = 'Vasya';
//Передаем переменную в шаблонизатор Smarty
$smarty->assign('name',$name);
//Выводим на экран
$smarty->display('main.tpl');
?>
```

Так выглядит код файла файл *index.php*, с включенным кэшированием на 10 секунд. После обновления странички измените значение переменной name на любое другое значение, и обновите браузер, Вы увидите что на экране ничего не изменилось, но подождите 20 секунд (время кэширования), и обновите заново страничку – теперь изменения вступили в силу (время кэширования истекло).

Иногда бывает полезным знать кэширована страничка или нет, в этом нам поможет метод *isCached (имя шаблона)*, который возвращает *TRUE* или *FALSE*, в зависимости кэширована страница или нет.

```php
//Проверяем кэширована ли страница
if($smarty->isCached('main.tpl')) {
 //Ваш код
}
```

Заключение

На этом все, данный урок завершен, надеюсь, Вам было интересно. Напомню, что на официальном сайте, есть подробная информация, о всех методах и свойствах шаблонизатора.

Всего доброго и удачного Вам кодирования!

Ваши пожелания и замечания буду рад увидеть в комментариях к статье!

Автор: Гавриленко Виктор. Команда webformyself.

Метки: шаблонизатор Smarty

Похожие статьи:


Yii2 – наиболее востребованный PHP-фрей...

15 и более самых лучших методов напис...


30+ лучших приемов PHP для начинающих...


Интеллектуальная страница 404 в Angul...


Оптимизация изображений для сайта...


Меньше кода, больше приложений: Vulca...

Комментарии Вконтакте:

Комментарии Facebook:

Комментарии (37)

2012-02-14 в 06:20 **Татьяна**

Спасибо за статью. Нужно подробно разобраться с шаблонзатором.

Ответить

2012-02-14 в 06:55 **Михаил**

Виктор, уважуха за труды. Все замечательно, инфа полезная и изложена довольно понятным стилем. Только вот есть одно НО в статье: В конфигурационных файлах хранятся скорее константы, нежели переменные. Это может ввести в заблуждение, особенно новичков.

Ответить

2012-02-14 в 09:41 **Viktor**

Логически, Михаил, Вы правы, и я с Вами согласен, но когда я записывал урок все термины называл так, как они названы в документации по Smarty.(в документации пишут, что в конфигурационных файлах хранятся переменные)

Ответить

2012-12-19 в 10:02 **Дарья**

Подскажите, если в конфигурационных файлах хранятся константы, а где тогда искать переменные?

Ответить

2012-02-14 в 08:08 **Svetlana Корнилова**

Хорошая статья, тем более с видео.

Когда планируется продолжение?

Ответить

2012-02-14 в 10:37 **Виктор Гавриленко**

Пока еще не знаю, но мы подумаем об этом.

Ответить

2012-02-19 в 00:20 **Очередной вебмастер**

Тоже хотелось бы продолжения,работаю в данный момент с CMS на Smarty. Хотелось бы получить по нему как можно более полную информацию.

Ответить

2012-02-14 в 08:21 **Евгений**

Спасибо Виктор за статью. Полезная информация.

Ответить

2012-02-14 в 08:23 **Александр**

Почему невозможно скачать видео?

Ответить

2012-02-14 в 10:22 **Андрей Кудлай**

Только что скачал видео без проблем. Попробуйте еще раз, возможно, в тот момент просто было слишком много подключений к файлообменнику.

Ответить

2012-02-14 в 09:20 **Николай**

Хорошо, а если меня интересует та же идея wordpress+denwer? Подскажите куда глянуть.

Ответить

2012-02-14 в 10:16 **Андрей Кудлай**

А зачем? Для WordPress Вам Smarty не потребуется. Для этого WordPress предлагает уже готовые методы (теги), при помощи которых можно решить практически любую задачу

Ответить

2012-02-14 в 11:59 **Александр**

Интересная статья и урок. Но зачем усложнять себе жизнь? Пора уже переходить на оф-лайн CMS, многофункциональные, неприхотливые и шаблонизатор не нужен. Все в них заложено. И глубоких знаний языков программирования не требуется. Многие вообще делают прекрасные сайты даже не зная что такое html и php.

Ответить

2012-02-14 в 13:20 **Андрей Кудлай**

А что такое «оф-лайн CMS»?

Александр, а что делать, если необходимо создать высоконагружаемое приложение? Здесь стандартными CMS можно не обойтись.

Ответить

2012-02-14 в 13:26 **Виктор Гавриленко**

Конечно,можно пользоваться готовыми системами управления контентом, но очень часто,приходится писать,что то свое,с нуля, и тогда Smarty очень полезен.

Ответить

2012-02-14 в 18:36 **Михаил**

Александр, отчасти вы правы. CMS — это хороший инструмент, но! Хорошее знание языков программирования — это универсальный инструмент, который всегда под рукой, и всегда сможет выручить, если не хватит возможностей той же CMS. Материал Виктора рассчитан как раз на программистов. ИМХО

Ответить

2012-02-14 в 18:38 **Михаил**

И да, меня тоже мучает вопрос: что же такое «оф-лайн CMS»?

Ответить

2012-02-14 в 13:45 **Владимир**

Спасибо! Очень интересный урок, все доступно объяснено.

Ответить

2012-02-16 в 13:03 **Михаил**

NoteSite — простая и удобная оффлайн CMS. NoteSite позволяет легко и просто, «на лету», менять структуру и содержание сайта.

Ответить

2012-02-16 в 17:48 **Никита Рябин**

Хорошая статья.

Ответить

2012-06-21 в 23:23 **Emil**

statya ochen xoroshaya s neterpeniyem budu jdat prodoljeniya

Ответить

2012-07-21 в 19:28 **ALIGARX**

Привет! Все подключил как в уроке.. но при вызове шаблона, он не появляется в браузере, а также не компилится в папке templates_c .. в чем проблема может быть

Ответить

2012-07-21 в 20:00 **Андрей Кудлай**

Здравствуйте. Наверное, будет лучше, если Вы зададите этот вопрос на нашем форуме с немного более конкретным описанием — на каком этапе возникла проблема, что сделано и каков результат.

Ответить

2012-11-16 в 23:16 **Tolik**

norm manual. thx-)

Ответить

2013-03-14 в 10:46 **Анастасия**

Виктор, добрый день.

Помогите разобраться с doctype для шаблонов Smarty.

Сайт на базе ShopScript Free. Пытаюсь вставлять различные doctype в файл Index.tpl.php в самом верху перед . По всему сайту сразу меняется шрифт — текст становится нечитаемым.

Что я неправильно делаю?

Возможно нужен какой-то специальный doctype?

PS Заморочка с DOCTYPE связана с некорректным отображением онлайн-консульнатнта в IE.

Ответить

2013-03-14 в 11:27 **Андрей Кудлай**

Добрый день, Анастасия.

Пока отвечу вместо Виктора. Насколько я понимаю, под фразой «текст становится нечитаемым» подразумевается то, что текст превращается в т.н. кракозябры. Если это так, то DOCTYPE здесь совершенно не при чем. Эта проблема уже связана с кодировкой. Скорее всего, Ваш редактор при сохранении попросту перекодирует документ. Насколько я помню, кодировка документа для ShopScript Free должна быть ANSI.

Чтобы разобраться с кодировками рекомендую ознакомиться с уроком Кодировки сайта: проблемы, вопросы, решения.

Ответить

2013-03-14 в 11:51 **Анастасия**

Андрей, спасибо. Но кракозябры не появляются. Текст остается прежним, но ооочень мелким, меняются межстрочные интервалы, размер картинок, да даже четкость цветов (как кажется)…

Ответить

2013-03-14 в 12:54 **Андрей Кудлай**

Тогда более точный ответ можно дать, только посмотрев сайт. Создайте тему на нашем форуме, опишите там еще раз проблему и дайте ссылку на сайт. На форуме гораздо проще решать подобные вопросы.

Ответить

2014-01-13 в 17:04 **Евгений**

Помогите разобраться, вылезла ошибка Fatal error: Uncaught —> Smarty: Unable to load template file 'template.tpl' <— thrown in C:\home\site-doka.ru\www\blocks\libs\sysplugins\smarty_internal_templatebase.php on line 127, все делаю как на видео

Ответить

2014-01-13 в 19:16 **Евгений**

Разобрался, smarty был не в корне сайта

Ответить

2015-01-14 в 11:20 **Константин**

Здравствуйте, подскажите пожалуйста, как сделать условие на «если главная страница» то действие?

Ответить

2015-01-14 в 23:03 **Виктор Гавриленко**

Здравствуйте, Константин!

Можно в произвольную переменную, к примеру $home, сохранить TRUE если страница главная, далее выполнить проверку:

{if $home }

действие

{/if}

Ответить

2015-06-30 в 20:06 **Евгений**

до прочтения статьи использовал корявое самописное подобие шаблонизатора, думал нужно много времени для изучения и перехода на smarty… ошибался) хорошая статья, заставила заинтересоваться Smarty всерьёз… буду все свои поделки на него переводить

Ответить

2015-08-02 в 18:54 **Владимир**

Огромное спасибо за ваши труды. Материал очень доступен для понимания. Продолжайте в том же духе.

Ответить

2015-08-04 в 06:27 **Виктор Гавриленко**

Пожалуйста, Владимир!

Ответить

2015-11-03 в 14:57 **Александр**

Здравствуйте. instantcms использует smarty. хочу одеть шаблон с DLE на него. сможете помочь?

Ответить

2016-02-17 в 19:31 **Андрей**

Добрый день, а можно шаблон с кодом smarty как то привязывать к домену или по ключу чтобы не копировали шаблон спасбио

Ответить

Добавить комментарий

Ваш e-mail не будет опубликован. Обязательные поля помечены *

Имя *

E-mail *

Сайт

☐ Я не робот.

☑ Нажимая на кнопку "Отправить комментарий", я даю согласие на рассылку, согласие на обработку персональных данных и соглашаюсь с политикой конфиденциальности

Комментарий

Можно использовать следующие HTML-теги и атрибуты: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <em> <i> <q cite=""> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">`

Отправить комментарий

Уведомить меня об ответах на мой комментарий по e-mail.

красивое создание сайтов

jquery ползунки