Руководство по Smarty

Monte Ohrt <monte at ohrt dot com> Andrei Zmievski <andrei@php.net>

Руководство по Smarty

Опубликовано 14-12-2005

Содержание

Предисловие

Несомненно, один из наиболее часто задаваемых вопросов в списках рассылки PHP - "Как мне сделать свои PHP-скрипты независимыми от дизайна?". Хотя PHP называют "скриптовым языком, встраиваемым в HTML", после написания нескольких проектов, в которых PHP и HTML свободно перемешиваются, многие понимают, что отделение формы от содержания - это Хорошая Вещь [TM]. Кроме того, во многих компаниях должности дизайнера и программиста разделены между собой. Так начинается поиск обработчика шаблонов...

Например, в нашей компании разработка приложения идёт таким образом: после того, как готова вся проектная документация, дизайнер интерфейса создаёт макеты и передаёт их программисту. Программист реализовывает логику приложения на PHP и использует макеты интерфейса для создания базовых шаблонов. Затем проект передаётся HTML-дизайнеру/верстальщику, который доводит шаблоны до совершенства. Проект может несколько раз переходить из этапа HTML-вёрстки к этапу программирования и обратно. Таким образом, важно иметь хорошую поддержку шаблонов, потому что программисты не хотят иметь дела с HTML и не хотят, чтобы HTML-дизайнеры копались в PHP-коде. Дизайнерам нужна поддержка конфигурационных файлов, динамических блоков и прочих интерфейсных нюансов, но они не хотят иметь дела со сложностями языка программирования PHP.

Глядя на множество обработчиков шаблонов, доступных сегодня для PHP, большинство из них предоставляет базовые возможности подстановки переменных в шаблоны и имеет ограниченную поддержку динамических блоков. Но нам требовалось нечто большее. Мы хотели, чтобы программисты ВООБЩЕ не имели дела с HTML, но это было практически неизбежно. К примеру, если дизайнер хотел, чтобы два фоновых цвета чередовались при отображении динамических блоков, эту задачу необходимо было решать вместе с программистом. Нам также требовалось, чтобы дизайнеры могли использовать собственные конфигурационные файлы и вставлять переменные из этих файлов в шаблоны. И так далее.

Мы начали написание спецификации для обработчика шаблонов ещё в 1999 году. Когда мы закончили спецификацию, мы начали работать над обработчиком шаблонов, написанным на Си, которому, как мы надеялись, разрешат стать частью PHP. Мы не только наткнулись на множество технических барьеров, но было и большое количество споров относительно того, что должен и не должен делать обработчик шаблонов. Благодаря этому опыту мы решили, что обработчик шаблонов должен быть написан на PHP в виде класса, чтобы каждый мог использовать его так, как хочет. Затем мы написали движок, который соответствовал этим требованиям и SmartTemplate™ появился на свет (замечание: этот класс никогда не был опубликован). Это был класс, который делал практически всё, что нам требовалось: обыкновенная подстановка переменных, поддержка подключения других шаблонов, интеграция с конфигурационными файлами, встраивание PHP-кода, ограниченная поддержка условий 'if' и улучшенная поддержка вложенных динамических блоков. Всё это достигалось использованием регулярных выражений и в итоге у нас получился код, который, скажем так, не позволял вносить в себя какие-либо изменения. Кроме того, он прилично тормозил в крупных приложениях из-за большого количества парсинга и регулярных выражений, которые обрабатывались при каждом запросе. Наибольшей проблемой с программистской точки зрения была та работа, которую нужно было провести над PHP-скриптом для настройки и обработки шаблонов и динамических блоков. Как же мы можем упростить это?

Затем пришло видение того, что в последствии переросло в Smarty. Мы знали, как быстр PHP-код, если его не перегружать обработкой шаблонов. Мы также знали, как всеобъемлюще и непонятно может выглядить язык PHP для среднестатистического дизайнера, и что это можно замаскировать при помощи более простого синтаксиса шаблонов. А почему бы нам не объединить две эти силы? Так и родился Smarty... :-)

Часть I. Приступая к работе

Содержание

Глава 1. Что такое Smarty?

Smarty - это компилирующий обработчик шаблонов для PHP. Говоря более четко, он предоставляет один из инструментов, которые позволяет добиться отделения прикладной логики и данных от представления. Это очень удобно в ситуациях, когда программист и верстальщик шаблона - различные люди.

Например, скажем, вы создаете страницу, которая показывает газетную статью. Название статьи, автор и сама статья - элементы, которые не содержат никакой информации о том, как они будут представлены. Их передают в Smarty из приложения, а верстальщик шаблона редактирует шаблоны и использует комбинацию тэгов HTML и тэгов шаблона, чтобы отформатировать представление этих элементов (таблицы HTML, фоновые цвета, размеры шрифта, стиля и т.д.). Однажды программист захочет изменить способ хранения статьи (сделать изменения в логике приложения). Это изменение не вызовет изменений в шаблонах. Содержание будет все еще передаваться в шаблон таким же самым способом. Аналогично, если верстальщик захочет полностью перепроектировать шаблоны, это не потребует никаких изменений в прикладной логике.

Одно из предназначений Smarty - это отделение логики приложения от представления. Конечно же, шаблоны могут содержать в себе логику, но лишь при условии, что эта логика необходима для правильного представления данных. Такие задачи, как подключение других шаблонов, чередующаяся окраска строчек в таблице, приведение букв к верхнему регистру, циклический проход по массиву для его отображения и т.д. - всё это является примером логики представления. Не следует думать, что Smarty заставляет вас разделять логику приложения и представление. Smarty не видит разницы между этими вещами, так что помещать или не помещать логику приложения в шаблоны - решать вам. Если же вы считаете, что в шаблоне *вообще* не должно быть логики, вы можете ограничиться использованием чистого текста и переменных.

Одна из уникальных возможностей Smarty - компилирование шаблонов. Это означает, что Smarty читает файлы шаблонов и создает PHP-код на их основе. Код создаётся один раз и потом только выполняется. Поэтому нет необходимости обрабатывать файл шаблона для каждого запроса и каждый шаблон может пользоваться всеми преимуществами таких кэширующих решений, как Zend Accelerator (http://www.zend.com/) или PHP Accelerator (http://www.php-accelerator.co.uk).

Некоторые особенности Smarty:

- Он очень быстр.
- Он эффективен, так как обработчик PHP делает за него грязную работу.
- Никакой лишней обработки шаблонов, они компилируются только один раз.
- Перекомпилируются только те шаблоны, которые изменились.
- Вы можете создавать пользовательские функции и модификаторы, что делает язык шаблонов чрезвычайно расширяемым.
- Настраиваемые разделители тэгов шаблона, то есть вы можете использовать {}, {{}}, <!--{}--> и т.д.
- Конструкции if/elseif/else/endif передаются обработчику PHP, так что синтаксис выражения {if ...} может быть настолько простым или сложным, насколько вам угодно.
- Допустимо неограниченное вложение секций, условий и т.д.

- Существует возможность включения PHP-кода прямо в ваш шаблон, однако обычно в этом нет необходимости (и это не рекомендуется), так как движок весьма гибок и расширяем.
- Встроенный механизм кэширования.
- Произвольные источники шаблонов.
- Пользовательские функции кэширования.
- Компонентная архитектура.

Глава 2. Установка

Содержание

Требования

Для установки и работы Smarty необходим веб-сервер с установленным PHP версии 4.0.6 или выше.

Базовая установка

Скопируйте файлы Smarty, которые находятся в субдиректории /libs/ дистрибутива. Редактировать эти PHP-файлы НЕ СЛЕДУЕТ. Они должны использоваться всеми приложениями и изменяться только при обновлении Smarty до новой версии.

Пример 2.1. Необходимые файлы библиотеки Smarty

```
Smarty.class.php
Smarty_Compiler.class.php
Config_File.class.php
debug.tpl
/internals/*.php (все файлы)
/plugins/*.php (все файлы, но отдельному сайту могут понадобиться не все)
```

Smarty использует константу [http://php.net/define] PHP SMARTY_DIR, которая указывает **полный путь** к директории 'libs/' Smarty. Обычно, если ваше приложение может найти файл Smarty.class.php, то нет необходимости устанавливать SMARTY_DIR - Smarty сам во всём разберётся. Однако, если Smarty.class.php не может быть найден в вашем include_path или вы не указывали абсолютный путь к нему в приложении, то вы должны определить SMARTY_DIR вручную. SMARTY_DIR **должен включать завершающий слэш**.

Вот как следует создавать экземпляр объекта Smarty в ваших PHP-скриптах:

Пример 2.2. Создание обекта Smarty

```
<?php
// Обратите внимание: в слове Smarty буква 'S' должна быть заглавной
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

Попробуйте выполнить вышеуказанный код. Если Вы получаете ошибку о том, что Smarty.class.php не найден, попробуйте следующие варианты действий:

Пример 2.3. Ручная установка константы SMARTY_DIR

```
<?php
// стиль *nix (не забывайте о заглавной 'S')
define('SMARTY_DIR', '/usr/local/lib/php/Smarty-v.e.r/libs/');

// стиль windows
define('SMARTY_DIR', 'c:/webroot/libs/Smarty-v.e.r/libs/');

// пример хака для работы одновременно с *nix и windows
// предполагается, что Smarty находится в директории 'includes/' относительно текущего скрипта
define('SMARTY_DIR', str_replace("\\", "/", getcwd()).'/includes/Smarty-v.e.r/libs/');

require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
?>
```

Пример 2.4. Передача абсолютного пути к файлам библиотеки

```
<?php
// стиль *nix (не забывайте о заглавной 'S')
require_once('/usr/local/lib/php/Smarty-v.e.r/libs/Smarty.class.php');

// стиль windows
require_once('c:/webroot/libs/Smarty-v.e.r/libs/Smarty.class.php');

$smarty = new Smarty();
?>
```

Пример 2.5. Добавление файлов библиотеки к include_path PHP

```
<?php
// Отредактируйте ваш файл php.ini, добавив директорию
// библиотеки Smarty в include_path и перезапустите веб-сервер.
// затем, следующий код должен работать:
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

Теперь, когда все файлы находятся на своих местах, пришло время установки директорий Smarty в вашем приложении. Smarty требует четыре директории, которые по умолчанию называются 'templates/', 'templates_c/', 'configs/' и 'cache/'. Каждая из них определяется свойствами класса Smarty: $template_dir, $compile_dir, $config_dir

и $cache_dir соответственно. Настойчиво рекомендуется использовать разные наборы этих директорий для каждого приложения, использующего Smarty.

Убедитесь, что вы знаете расположение корневой директории для документов вашего веб-сервера. В нашем примере это /web/www.example.com/docs/. Так как доступ к директориям Smarty получает только библиотека Smarty и они никогда не запрашиваются напрямую веб-браузером, вам рекомендуется вынести директории Smarty *за пределы* корневой директории для документов, чтобы избежать лишнего беспокойства относительно безопасности.

В нашем примере мы будем устанавливать Smarty для некоторой гостевой книги. Приложение было выбрано только для того, чтобы использовать его имя в именах директорий. Вы можете использовать те же настройки с любым другим приложением, просто меняя "guestbook" на имя вашего приложения. Мы же разместим наши директории Smarty тут: web/www.example.com/smarty/guestbook/

Вам понадобится как минимум один файл в корневой директории для документов - это скрипт, к которому обращается веб-браузер. Мы назовём наш скрипт *'index.php'* и поместим его в поддиректорию /guestbook/ корневой директории для документов.

> **Техническое замечание:** Бывает удобно настроить веб-сервер так, чтобы 'index.php' расценивался как индексный файл дирекотрии по умолчанию, чтобы при запросе страницы http://www.example.com/guestbook/, вызывался скрипт 'index.php' без 'index.php' в конце адресной строки. В веб-сервере Apache вы можете настроить это, добавив "index.php" в конец директивы *DirectoryIndex* (записи разделяются пробелами), как в примере из httpd.conf
>
> *DirectoryIndex index.htm index.html index.php index.php3 default.html index.cgi*

Давайте взгляним на текущую файловую структуру:

Пример 2.6. Файловая структура примера

```
/usr/local/lib/php/Smarty-v.e.r/libs/Smarty.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/Smarty_Compiler.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/Config_File.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/debug.tpl
/usr/local/lib/php/Smarty-v.e.r/libs/internals/*.php
/usr/local/lib/php/Smarty-v.e.r/libs/plugins/*.php

/web/www.example.com/smarty/guestbook/templates/
/web/www.example.com/smarty/guestbook/templates_c/
/web/www.example.com/smarty/guestbook/configs/
/web/www.example.com/smarty/guestbook/cache/

/web/www.example.com/docs/guestbook/index.php
```

Smarty понадобятся **права на запись** (пользователей Windows это не касается) в *$compile_dir* и *$cache_dir*, так что убедитесь, что у веб-сервера есть эти права. Обычно сервер запущен от имени пользователя "nobody" группы "nobody". Для пользователей OS X пользователем по умолчанию является "www" группы "www". Если вы используете Apache, вы можете заглянуть в ваш файл httpd.conf (который обычно расположен в "/usr/local/apache/conf/") чтобы узнать, какой пользователь и группа используются.

Пример 2.7. Установка прав доступа к файлам

```
chown nobody:nobody /web/www.example.com/smarty/guestbook/templates_c/
chmod 770 /web/www.example.com/smarty/guestbook/templates_c/

chown nobody:nobody /web/www.example.com/smarty/guestbook/cache/
chmod 770 /web/www.example.com/smarty/guestbook/cache/
```

> **Техническое замечание:** chmod 770 даёт достаточно жесткую защиту - разрешает только пользователю "nobody" и группе "nobody" доступ на чтение и запись в эти директории. Если вы хотите открыть доступ на чтение для всех (обычно для собственного удобства при просмотре этих файлов), вы можете использовать значение 775.

Нам необходимо создать файл 'index.tpl', которы будет загружаться Smarty. Он будет расположен в $template_dir.

Пример 2.8. Редактирование /web/www.example.com/smarty/guestbook/templates/index.tpl

```
{* Smarty *}

Привет, {$name}! Добро пожаловать в Smarty!
```

> **Техническое замечание:** {* Smarty *} - это комментарий шаблона. Он не является обязательным, но его размещение в начале каждого шаблона является хорошим тоном. Это позволяет проще различать файлы независимо от их расширения. К примеру, текстовые редакторы могут узнавать этот файл и включать особенную подсветку синтаксиса.

Теперь давайте отредактируем 'index.php'. Мы создадим экземпляр Smarty, присвоим значение переменной шаблона и отобразим файл 'index.tpl'.

Пример 2.9. Редактироение /web/www.example.com/docs/guestbook/index.php

```
<?php

// загружаем библиотеку Smarty
require_once(SMARTY_DIR . 'Smarty.class.php');

$smarty = new Smarty();

$smarty->template_dir = '/web/www.example.com/smarty/guestbook/templates/';
$smarty->compile_dir = '/web/www.example.com/smarty/guestbook/templates_c/';
$smarty->config_dir = '/web/www.example.com/smarty/guestbook/configs/';
$smarty->cache_dir = '/web/www.example.com/smarty/guestbook/cache/';

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

```
$smarty->display('index.tpl');
?>
```

> **Техническое замечание:** В нашем примере мы устанавливаем абсолютные пути ко всем директориям Smarty. Если /web/www.example.com/smarty/guestbook/ находится в include_path вашего PHP, то эти настройки не обязательны. Тем не менее, более эффективным и (из опыта) менее глюкоопасным является использование абсолютных путей. Это придаст уверенность в том, что Smarty получает файлы из тех директорий, из которых вы хотите.

Теперь перейдите к файлу index.php при помощи вашего веб-браузера. Вы должны увидеть надпись "Привет, Кристина! Добро пожаловать в Smarty!"

Вы закончили базовую установку Smarty!

Расширенная установка

Эта глава является продолжением базовой установки; пожалуйста, сперва прочитайте её.

Немного более гибким способом установки Smarty является наследование класса и инициализация вашего собственного окружения Smarty. Таким образом, вместо того, чтобы постоянно устанавливать пути директорий, присваивать одни и те же переменные и т.д., мы можем всё это сделать в одном месте. Давайте создадим новую директорию "/php/includes/guestbook/",а в ней - новый файл, который назовем setup.php. По условиям нашего примера, "/php/includes" находится в include_path. Убедитесь, чтобы то же самое было и у вас, или используетй абсолютные пути.

Пример 2.10. Редактирование /php/includes/guestbook/setup.php

```
<?php

// загружаем библиотеку Smarty
require('Smarty.class.php');

// Файл setup.php - это хорошее место для
// подключения библиотечных файлов вашего приложения,
// вы можете сделать это прямо здесь. Пример:
// require('guestbook/guestbook.lib.php');

class Smarty_GuestBook extends Smarty {

   function Smarty_GuestBook()
   {

        // Конструктор класса.
        // Он автоматически вызывается при создании нового экземпляра.

        $this->Smarty();

        $this->template_dir = '/web/www.example.com/smarty/guestbook/templates/';
        $this->compile_dir  = '/web/www.example.com/smarty/guestbook/templates_c/';
        $this->config_dir   = '/web/www.example.com/smarty/guestbook/configs/';
```

```
        $this->cache_dir    = '/web/www.example.com/smarty/guestbook/cache/';

        $this->caching = true;
        $this->assign('app_name', 'Guest Book');
    }

}
?>
```

Теперь давайте изменим index.php, чтобы он использовал setup.php:

Пример 2.11. Редактирование /web/www.example.com/docs/guestbook/index.php

```
<?php

require('guestbook/setup.php');

$smarty = new Smarty_GuestBook;

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

$smarty->display('index.tpl');
?>
```

Теперь вы видите, что создать экземпляр Smarty довольно просто - нужно лишь использовать Smarty_GuestBook, который автоматически инициализирует все настройки для нашего приложения.

Часть II. Smarty для дизайнеров шаблонов

Содержание

Глава 3. Базовый синтаксис

Содержание

Все тэги шаблонов Smarty располагаются между специальными разделителями. По умолчанию это { и }, но они могут быть изменены.

Для наших примеров мы будем использовать стандартные разделители. Smarty все содержимое вне разделителей отображает как статический контент, без изменений. Когда Smarty встречает тэги шаблона, то пытается интерпретировать их и вывести вместо них соответствующий результат.

Комментарии

Комментарии в шаблонах заключаются в звездочки (*) окруженные разделителями, например: {* это комментарий *}. Комментарии не отображаются в выводе шаблона, в отличие от <!-- комментариев HTML -->. Они используются для внутренних примечаний в шаблонах.

Пример 3.1. Комментарии

```
<body>
{* однострочный комментарий *}

{* этот многострочный комментарий
   не отправляется в бразуер
*}

{* здесь включаем заголовок *}
{include file="header.tpl"}


{* Примечание разработчика:  $includeFile назначается в скрипте foo.php *}
<!-- этот HTML-комментарий будет отправлен браузеру -->
{include file=$includeFile}

{include file=#includeFile#}

{* этот блок <select> ненужен *}
{*
```

```
<select name="company">
  {html_options options=$vals selected=$selected_id}
</select>
*}
</body>
```

Переменные

Переменные шаблона начинаются со знака $доллара. Они могут состоять из цифр, букв, знаков подчёркивания - как и обычные PHP variable [http://php.net/language.variables]. Вы можете обращаться к массивам, имеющим числовые и нечисловые индексы. Вы также можете обращаться к свойствам и методам объектов. Переменные конфигурационного файла - это исключения из долларового синтаксиса. К ним можно обращаться, окружив их #решетками# или воспользовавшись специальной переменной $smarty.config.

Пример 3.2. Переменные

```
{$foo}        <-- отображение простой переменной (не массив и не объект)
{$foo[4]}     <-- отображает 5-й элемент числового массива
{$foo.bar}    <-- отображает значение ключа "bar" ассоциативного массива, подобно PHP $foo['bar']
{$foo.$bar}   <-- отображает значение переменного ключа массива, подобно PHP $foo[$bar]
{$foo->bar}   <-- отображает свойство "bar" объекта
{$foo->bar()} <-- отображает возвращаемое значение метода "bar" объекта
{#foo#}       <-- отображает переменную "foo" конфигурационного файла
{$smarty.config.foo} <-- синоним для {#foo#}
{$foo[bar]}   <-- синтаксис доступен только в цикле section, см. {section}
{assign var=foo value="baa"}{$foo} <--  отображает "baa", см. {assign}

Также доступно множество других комбинаций

{$foo.bar.baz}
{$foo.$bar.$baz}
{$foo[4].baz}
{$foo[4].$baz}
{$foo.bar.baz[4]}
{$foo->bar($baz,2,$bar)} <-- передача параметра
{"foo"}       <-- статические значения также разрешены
```

См. также Зарезервированная переменная $smarty и Переменные файлов конфигурации.

Функции

Каждый тэг Smarty либо выводит значение переменной, либо вызывает некоторую функцию. Для вызова функции надо заключить в разделители название функции и ее параметры, например: {funcname attr1="val" attr2="val"}.

Пример 3.3. Синтаксис функций

```
{config_load file="colors.conf"}

{include file="header.tpl"}

{if $highlight_name}
Welcome, <font color="{#fontColor#}">{$name}!</font>
{else}
Welcome, {$name}!
{/if}

{include file="footer.tpl"}
```

И встроенные, и пользовательские функции используются с одинаковым синтаксисом.

Встроенные функции обеспечивают **внутреннюю** работу Smarty, например {if}, {section} и {strip}. Они не могут быть модифицированы.

Пользовательские функции являются **дополнительными** и реализуются через плагины. Они могут быть изменены по вашему желанию, также вы можете добавить новые. Примерами пользовательских функций могут быть {html_options} и {popup}.

Параметры

Большинство функций принимают аргументы, которые уточняют или изменяют ее поведение. Аргументы в Smarty очень похожи на параметры в HTML. Статические значения не обязательно заключать в кавычки, но это рекомендуется для текстовых строк. Переменные также могут быть использованы в качестве параметров, и не должны заключаться в кавычки.

Некоторые параметры принимают логические значения (true или false). Они могут быть указаны словами true, on и yes, или false, off и no без кавычек.

Пример 3.4. синтаксис параметров функции

```
{include file='header.tpl'}

{include file='header.tpl' attrib_name='attrib value'}

{include file=$includeFile}

{include file=#includeFile# title='Smarty is cool'}

{html_select_date display_days=yes}

{mailto address='smarty@example.com'}

<select name='company_id'>
  {html_options options=$companies selected=$company_id}
</select>
```

Внедренные переменные в двойных кавычках

Smarty распознает присвоенные переменные, если они встречаются в строках, заключенных в двойные кавычки, если переменные состоят из цифр, букв, знака подчёркивания и квадратных скобок. В случае, если переменная содержит другие символы (точки, ссыки на объекты и т.д.), переменную необходимо заключить в обратные кавычки. В данном случае вы не можете использовать модификаторы, их следует применять вне кавычек.

Пример 3.5. Синтаксис внедренных переменных

```
Пример синтаксиса:
{func var="test $foo test"}       <-- ищет $foo
{func var="test $foo_bar test"}   <-- ищет $foo_bar
{func var="test $foo[0] test"}    <-- ищет $foo[0]
{func var="test $foo[bar] test"}  <-- ищет $foo[bar]
{func var="test $foo.bar test"}   <-- ищет $foo (не $foo.bar)
{func var="test `$foo.bar` test"} <-- ищет $foo.bar
{func var="test `$foo.bar` test"|escape} <-- модификатор вне кавычек!

Практические примеры:
{include file="subdir/$tpl_name.tpl"} <-- заменит $tpl_name на ее значение
{cycle values="one,two,`$smarty.config.myval`"} <-- надо заключать в обратные кавычки
```

См. также escape.

Арифметические операции

Арифметические операции могут совершаться непосредственно над значениями переменных.

Пример 3.6. Примеры арифметики

```
{$foo+1}

{$foo*$bar}

{* несколько более сложных примеров *}

{$foo->bar-$bar[1]*$baz->foo->bar()-3*7}

{if ($foo+$bar.test%$baz*134232+10+$b+10)}

{$foo|truncate:"`$fooTruncCount/$barTruncFactor-1`"}

{assign var="foo" value="`$foo+$bar`"}
```

См. также функцию {math} для сложных вычислений.

Предотвращение обработки Smarty

Иногда необходимо, чтобы Smarty не обрабатывал часть шаблона, которая должна по умолчанию обрабатываться. Классическим примером такой ситуации является встраивание Javascript или CSS-кода в шаблон. Проблема появляется из-за того, что эти языки используют символы { и }, которые так же используются в качестве разделителей для Smarty.

Самым простым решением является избежание этой ситуации путём выноса Javascript'а и CSS-кода в отдельные файлы и использования стандартных методов HTML для доступа к ним.

Дословное включение контента возможно при помощи блоков {literal} .. {/literal}. Подобно тому, как вы используете HTML-сущности (и т.п.), вы можете использовать {ldelim},{rdelim} или {$smarty.ldelim} для отображения текущих разделителей.

Порой бывает удобно просто изменить свойства $left_delimiter и $right_delimiter в объекте Smarty.

Пример 3.7. Изменение разделителей

```
<?php

$smarty = new Smarty;
$smarty->left_delimiter = '<!--{';
$smarty->right_delimiter = '}-->';
$smarty->assign('foo', 'bar');
$smarty->display('example.tpl');

?>
```

Шаблон example.tpl:

```
<script language="javascript">
  var foo = <!--{$foo}-->;
  function dosomething() {
    alert("foo is " + foo);
  }
  dosomething();
</script>
```

См. также модификатор escape

Глава 4. Переменные

Содержание

Smarty имеет несколько различных типов переменных. Он зависит от символа, с которого начинается, или в какой заключена переменная.

Variables in Smarty can be either displayed directly or used as arguments for function attributes and modifiers, inside conditional expressions, etc. To print a variable, simply enclose it in the delimiters so that it is the only thing contained between them. Examples: Переменные в Smarty могут быть отображены или использованы как аргументы функций и модификаторов, внутри выражений условных операторов и т.д. Для вывода значения переменной необходимо указать имя переменной между разделителями. Примеры:

```
[
{$Name}

{$Contacts[row].Phone}

<body bgcolor="{#bgcolor#}">
```

Переменные, установленные в PHP

Переменные, установленные в PHP, употребляются со знаком доллар $ перед ним. Переменные, установленные в шаблоне с помощью функции assign употребляются аналогичным образом.

Пример 4.1. Установленные переменные

```
Привет {$firstname}, мы рады снова тебя видеть.
<p>
Последний раз ты посещал нас {$lastLoginDate}.

OUTPUT:

Привет Петя, мы рады снова тебя видеть.
<p>
Последний раз ты посещал нас January 11th, 2001.
```

Ассоциативные массивы

Чтобы использовать переменную из ассоциативного массива, надо указать ключ элемента после знака '.' (точка).

Пример 4.2. доступ к переменным ассоциативного массива

index.php:

```
$smarty = new Smarty;
$smarty->assign('Contacts',
  array('fax' => '555-222-9876',
        'email' => 'zaphod@slartibartfast.com',
        'phone' => array('home' => '555-444-3333',
                         'cell' => '555-111-1234')));
$smarty->display('index.tpl');
```

index.tpl:

```
{$Contacts.fax}<br>
{$Contacts.email}<br>
{* you can print arrays of arrays as well *}
{$Contacts.phone.home}<br>
{$Contacts.phone.cell}<br>
```

OUTPUT:

```
555-222-9876<br>
zaphod@slartibartfast.com<br>
555-444-3333<br>
555-111-1234<br>
```

Индексированные массивы

Можно использовать переменную из массива по е индексу. Синтаксис аналогичен PHP.

Пример 4.3. доступ к елементу массива по его индексу

index.php:

```
$smarty = new Smarty;
$smarty->assign('Contacts',
  array('555-222-9876',
        'zaphod@slartibartfast.com',
        array('555-444-3333',
              '555-111-1234')));
$smarty->display('index.tpl');
```

index.tpl:

```
{$Contacts[0]}<br>
```

```
{$Contacts[1]}<br>
{* you can print arrays of arrays as well *}
{$Contacts[2][0]}<br>
{$Contacts[2][1]}<br>

OUTPUT:

555-222-9876<br>
zaphod@slartibartfast.com<br>
555-444-3333<br>
555-111-1234<br>
```

Объекты

Чтобы использовать свойства обьектов, надо указать перед именем атрибута знак `->'.

Пример 4.4. доступ к свойствам объекта

```
name: {$person->name}<br>
email: {$person->email}<br>

OUTPUT:

name: Zaphod Beeblebrox<br>
email: zaphod@slartibartfast.com<br>
```

Переменные файлов конфигурации

Для использования переменных, полученных из файлов конфигураии, необходимо заключить их имя между знаками # или через переменную $smarty.config. Для употребления их в качестве внедренныых переменных можно использовать только второй способ.

Пример 4.5. Переменные из файлов конфигурации

foo.conf:

```
pageTitle = "This is mine"
bodyBgColor = "#eeeeee"
tableBorderSize = "3"
tableBgColor = "#bbbbbb"
rowBgColor = "#cccccc"
```

index.tpl:

```
{config_load file="foo.conf"}
<html>
```

```
<title>{#pageTitle#}</title>
<body bgcolor="{#bodyBgColor#}">
<table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
<tr bgcolor="{#rowBgColor#}">
    <td>First</td>
    <td>Last</td>
    <td>Address</td>
</tr>
</table>
</body>
</html>
```

index.tpl: (альтернативный синтаксис)

```
{config_load file="foo.conf"}
<html>
<title>{$smarty.config.pageTitle}</title>
<body bgcolor="{$smarty.config.bodyBgColor}">
<table border="{$smarty.config.tableBorderSize}" bgcolor="{$smarty.config.tableBgColor}">
<tr bgcolor="{$smarty.config.rowBgColor}">
    <td>First</td>
    <td>Last</td>
    <td>Address</td>
</tr>
</table>
</body>
</html>
```

результат выполнения обоих примеров:

```
<html>
<title>This is mine</title>
<body bgcolor="#eeeeee">
<table border="3" bgcolor="#bbbbbb">
<tr bgcolor="#cccccc">
<td>First</td>
<td>Last</td>
<td>Address</td>
</tr>
</table>
</body>
</html>
```

Переменные из файлов конфигурации не могут быть использованы, пока они не будут загружены. Эта процедура описана далее в данном руководстве (**config_load**).

Зарезервированная переменная {$smarty}

Зарезервированная переменная {$smarty} используется для доступа к нескольким специальным переменным. Далее следует полный их список.

Переменные запроса.

К переменным из таких массивов, как _GET, _POST, _COOKIES, _SERVER, _ENV и _SESSION, можно обращаться аналогично нижеприведенным примерам.

Пример 4.6. Вывод переменных запроса

```
{* Вывод значения $page из URL (GET) http://www.example.com/index.php?page=foo *}
{$smarty.get.page}

{* Вывод переменной "page" из формы (POST) *}
{$smarty.post.page}

{* Вывод значения cookie "username" *}
{$smarty.cookies.username}

{* Вывод переменное сервера "SERVER_NAME" *}
{$smarty.server.SERVER_NAME}

{* Вывод переменной окружения "PATH" *}
{$smarty.env.PATH}

{* Вывод переменной сессии "id" *}
{$smarty.session.id}

{* Вывод переменной "username" из объединенного массива get/post/cookies/server/env *}
{$smarty.request.username}
```

{$smarty.now}

К текущему timestamp (штам времени) можно обратиться через {$smarty.now}. Оно содержит число секунд с начала так называемой Эпохи (Epoch, 1 января 1970 года) и может быть передано прямо модификатору date_format для вывода текущей даты.

Пример 4.7. использование {$smarty.now}

```
{* выводим текущее время и дату с помощью модификатора date_format *}
{$smarty.now|date_format:"%Y-%m-%d %H:%M:%S"}
```

{$smarty.const}

Реализует доступ к константам PHP.

Пример 4.8. использование {$smarty.const}

```
{$smarty.const._MY_CONST_VAL}
```

{$smarty.capture}

Доступ к выводу, сохраненному с помощью тэгов {capture}..{/capture}, можно получить используя переменную {$smarty}. Смотрите раздел capture для примера.

{$smarty.config}

Переменная {$smarty} может быть использована для получения значений переменных из файлов конфигураций. {$smarty.config.foo} является синонимом для {#foo#}. Смотрите раздел config_load для примера.

{$smarty.section}, {$smarty.foreach}

Переменная {$smarty} может быть использована для исппользования свойств структур 'section' и 'foreach'. Смотри разделы по section и foreach.

{$smarty.template}

Эта переменная содержит имя текущего шаблона.

{$smarty.ldelim}

Эта переменная используется для вставки символа левого разделителя. См. также {ldelim},{rdelim}.

{$smarty.rdelim}

Эта переменная используется для вставки символа правого разделителя. См. также {ldelim},{rdelim}.

Глава 5. Модификаторы переменных

Содержание

Модификаторы переменных могут быть прмменены к переменным, пользовательским функциям или строкам. Для их применения надо после модифицируемого значения указать символ | (вертикальная черта) и название модификатора. Так же модификаторы могут принимать параметры, которые влияют на их поведение. Эти параметры следуют за названием модификатора и разделяются : (двоеточием).

Пример 5.1. Пример модификатора

```
{* применение модификатора к переменной *}
{$title|upper}
{* модификатор с параметрами *}
{$title|truncate:40:"..."}

{* применение модификатора к аргументу функции *}
{html_table loop=$myvar|upper}
{* с параметрами *}
{html_table loop=$myvar|truncate:40:"..."}

{* применение модификатора к строке *}
{"foobar"|upper}
```

```
{* использование date_format для форматирования текущей даты *}
{$smarty.now|date_format:"%Y/%m/%d"}

{* применение модификатора к функции *}
{mailto|upper address="me@domain.dom"}
```

Если модификатор применяется к переменной-массиву, то он будет применен к каждому элементу массива. Если же требуется применить модификатор к массиву, как к переменной, то необходимо перед именем модификатора указать символ @. Пример: {$articleTitle|@count} выведет количество елементов в массиве $articleTitle.

Модификаторы могут автоматически загружаться из вашей директории $plugins_dir (см. также: Соглашение об именах) или могут регистрироваться явно (см.: register_modifier).

К тому же, любая PHP-функция может быть явно использована в качестве модификатора. (Предыдущий пример с @count на самом деле использует функцию PHP, а не модификатор Smarty). Использование PHP-функций в качестве модификаторов имеет две маленькие "ловушки": Во-первых, иногда порядок аргументов функции не самый удобный ({"%2.f"|sprintf:$float} - это рабочий, но не совсем удобный вариант. Больше подойдет {$float|string_format:"%2.f"}, который обеспечивается дистрибутивом Smarty). Во-вторых, в случае включения $security, все PHP-функции, которые будут использованы как модификаторы, должны быть объявлены "безопасными" в массиве $security_settings['MODIFIER_FUNCS'].

См. также register_modifier(), register_function(), Плагины - расширение функциональности Smarty и модификаторы.

capitalize

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	boolean	Нет	false	Этот параметр определяет, распространяется ли действие модификатора на слова с цифрами

Первые буквы каждого слова преобразуются в заглавные.

Пример 5.2. capitalize

```
<?php

$smarty = new Smarty;
$smarty->assign('articleTitle', 'next x-men film, x3, delayed.');
$smarty->display('index.tpl');

?>
```

Шаблон index.tpl:

```
{$articleTitle}
{$articleTitle|capitalize}
{$articleTitle|capitalize:true}
```

Результат обработки:

```
next x-men film, x3, delayed.
Next X-Men Film, x3, Delayed.
Next X-Men Film, X3, Delayed.
```

См. также lower и upper

cat

Позиция параметра	Тип	Обязателен	cat	Описание
1	string	Нет	*пусто*	Данная строка добавляется к модифицируемому значению.

Данная строка добавляется к модифицируемому значению.

Пример 5.3. cat

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Psychics predict world didn't end");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle|cat:" yesterday."}

OUTPUT:

Psychics predict world didn't end yesterday.
```

count_characters

Возвращает количество символов в строке.

Пример 5.4. count_characters

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Cold Wave Linked to Temperatures.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_characters}

OUTPUT:

Cold Wave Linked to Temperatures.
32
```

См. также count_paragraphs, count_sentences и count_words

count_paragraphs

Возвращает количество абзацев в строке.

Пример 5.5. count_paragraphs

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "War Dims Hope for Peace. Child's Death Ruins
Couple's Holiday.\n\nMan is Fatally Slain. Death Causes Loneliness, Feeling of Isolation.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_paragraphs}

OUTPUT:

War Dims Hope for Peace. Child's Death Ruins Couple's Holiday.

Man is Fatally Slain. Death Causes Loneliness, Feeling of Isolation.
2
```

См. также count_characters, count_sentences и count_words

count_sentences

Возвращает количество предложений.

Пример 5.6. count_sentences

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Two Soviet Ships Collide - One Dies. Enraged Cow Injures Farmer with Axe.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_sentences}

OUTPUT:

Two Soviet Ships Collide - One Dies. Enraged Cow Injures Farmer with Axe.
2
```

См. также count_characters, count_paragraphs и count_words

count_words

Возвращает количество слов.

Пример 5.7. count_words

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Dealers Will Hear Car Talk at Noon.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|count_words}

OUTPUT:

Dealers Will Hear Car Talk at Noon.
7
```

См. также count_characters, count_paragraphs и count_sentences

date_format

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	string	Нет	%b %e, %Y	Формат вывода даты.
2	string	Нет	n/a	Если модифицируемое значение пусто, то используется это.

Формирует дату и время по заданному формату strftime(). Даты могут быть в виде unix timestamps, mysql timestamps или в любом другом виде, который поймет strtotime(). Проектировщики шаблонов могут использовать date_format для контроля над форматом выводимых дат. Если дата, переданная модификатору, пуста, то второй параметр используется как дата.

Пример 5.8. date_format

```
<?php

$smarty = new Smarty;
$smarty->assign('yesterday', strtotime('-1 day'));
$smarty->display('index.tpl');

?>
```

Где index.tpl:

```
{$smarty.now|date_format}
{$smarty.now|date_format:"%A, %B %e, %Y"}
{$smarty.now|date_format:"%H:%M:%S"}
{$yesterday|date_format}
{$yesterday|date_format:"%A, %B %e, %Y"}
{$yesterday|date_format:"%H:%M:%S"}
```

Результатом будет:

```
Feb 6, 2001
Tuesday, February 6, 2001
14:33:00
Feb 5, 2001
Monday, February 5, 2001
14:33:00
```

Указатели преобразования date_format:

- %a - сокращенное название дня недели, в зависимости от текущей локали

- %A - полное название дня недели, в зависимости от текущей локали
- %b - сокращенное название месяца, в зависимости от текущей локали
- %B - полное название месяца, в зависимости от текущей локали
- %c - формат даты и времени по умолчанию для текущей локали
- %C - номер века (год, деленный на 100, представленный в виде целого в промежутке от 00 до 99)
- %d - день месяца в десятичном формате (от 00 до 31)
- %D - синоним %m/%d/%y
- %e - день месяца в десятичном формате без ведущего нуля (от 1 до 31)
- %g - Week-based year within century [00,99]
- %G - Week-based year, including the century [0000,9999]
- %h - синоним %b
- %H - часы по 24-часовым часам (от 00 до 23)
- %I - часы по 12-часовым часам (от 01 до 12)
- %j - день года (от 001 до 366)
- %k - часы по 24-часовым часам без ведущего нуля (от 0 до 23)
- %l - часы по 12-часовым часам без ведущего нуля (от 1 до 12)
- %m - номер месяца (от 01 до 12)
- %M - минуты
- %n - символ новой строки
- %p - `am' или `pm', в зависимости от заданного формата времени и текущей локали.
- %r - time in a.m. and p.m. notation
- %R - time in 24 hour notation
- %S - секунды
- %t - символ табуляции
- %T - время в формате %H:%M:%S
- %u - номер дня недели [1,7], где 1-ый день - понедельник
- %U - номер недели в году, считая первое воскресенья года первым днем первой недели
- %V - номер недели в году (по ISO 8601:1988) в диапазоне от 01 до 53, где первая неделя та, у которой хо-

тя бы 4 дня находятся в данном году. Понедельник считается первым днем недели.

- %w - номер дня недели, где 0 - воскресенье
- %W - номер недели в году, считаю первый понедельник первым днем первой недели.
- %x - предпочтительное представление даты для текущих настроек locale без времени
- %X - предпочтительное представление времени для текущих настроек locale без даты
- %y - год в виде десятичного числа без века (от 00 до 99)
- %Y - год в виде десятичного числа включая век
- %Z - часовой пояс или имя или сокращение
- %% - буквальный символ `%'

Замечание для программистов: date_format - это просто обёртка функции PHP strftime() [http://php.net/strftime]. Вы можете иметь больше или меньше доступных указателей преобразования в зависимости от функции strftime() той системы, где был скомпилирован PHP. Обратитесь к руководству вашей системы для полного списка доступных указателей.

default

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	string	Нет	*пусто*	Значение по умолчанию для пустой переменной.

Используется для установки значения по умолчанию для переменной. Если переменная оказывается пустой, то выводиться значение по умолчанию. Модификатор принимает один параметр.

Пример 5.9. default

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Dealers Will Hear Car Talk at Noon.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle|default:"no title"}
{$myTitle|default:"no title"}

OUTPUT:
```

```
Dealers Will Hear Car Talk at Noon.
no title
```

escape

Позиция параметра	Тип	Обязателен	Возможные значения	По умолчанию	Описание
1	string	Нет	html,htmlall,url,quotes,hex,hexentity,javascript	html	Формат защиты (escape).

"Защищает" специальные символы в переменной. Используется для защиты специальных символов html, защиты специальных символов url, защиты одиночных кавычек, конвертации в шестандцатеричный вид (hex), конвертации каждого символа в шестандцатеричное html представление (hexentity), защита специальных символов javascript. По умолчанию используется защита спецсимволов html.

Пример 5.10. escape

```
<?php

$smarty = new Smarty;
$smarty->assign('articleTitle', "'Stiff Opposition Expected to Casketless Funeral Plan'");
$smarty->display('index.tpl');

?>
```

Где index.tpl:

```
{$articleTitle}
{$articleTitle|escape}
{$articleTitle|escape:"html"}    {* escapes  & " ' < > *}
{$articleTitle|escape:"htmlall"} {* escapes ALL html entities *}
{$articleTitle|escape:"url"}
{$articleTitle|escape:"quotes"}
<a href="mailto:{$EmailAddress|escape:"hex"}">{$EmailAddress|escape:"hexentity"}</a>
```

Результатом будет:

```
'Stiff Opposition Expected to Casketless Funeral Plan'
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;
%27Stiff+Opposition+Expected+to+Casketless+Funeral+Plan%27
\'Stiff Opposition Expected to Casketless Funeral Plan\'
<a href="mailto:%62%6f%62%40%6d%65%2e%6e%65%74">&#x62;&#x6f;&#x62;&#x40;&#x6d;&#x65;&#x2e;&#x6e;&#x65;&#x74;</a>
```

См. также Предотвращение обработки Smarty

indent

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	integer	Нет	4	Количество символов для вставки.
2	string	Нет	(один пробел)	Символ для вставки.

Вставляет в начало каждой строки заданное количество заданных символов. По умолчанию вставляется 4 пробела.

Пример 5.11. indent

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'NJ judge to rule on nude beach.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}

{$articleTitle|indent}

{$articleTitle|indent:10}

{$articleTitle|indent:1:"\t"}

OUTPUT:

NJ judge to rule on nude beach.
Sun or rain expected today, dark tonight.
Statistics show that teen pregnancy drops off significantly after 25.

    NJ judge to rule on nude beach.
    Sun or rain expected today, dark tonight.
    Statistics show that teen pregnancy drops off significantly after 25.

          NJ judge to rule on nude beach.
          Sun or rain expected today, dark tonight.
          Statistics show that teen pregnancy drops off significantly after 25.

        NJ judge to rule on nude beach.
        Sun or rain expected today, dark tonight.
 Statistics show that teen pregnancy drops off significantly after 25.
```

lower

Переводит текст в нижний регистр.

Пример 5.12. lower

```
<?php

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Two Convicts Evade Noose, Jury Hung.');
$smarty->display('index.tpl');

?>
```

Где index.tpl:

```
{$articleTitle}
{$articleTitle|lower}
```

Результатом будет:

```
Two Convicts Evade Noose, Jury Hung.
two convicts evade noose, jury hung.
```

См. также upper и capitalize.

nl2br

Заменяет все переносы строк на тэг
 в заданной переменной. Это эквивалент PHP функции nl2br().

Пример 5.13. nl2br

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Sun or rain expected\ntoday, dark tonight");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle|nl2br}

OUTPUT:

Sun or rain expected<br />today, dark tonight
```

regex_replace

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	string	Да	*n/a*	Регулярное выражение для замены.
2	string	Да	*n/a*	Строка для замены.

Выполняется поиск и замена по регулярному выражению в переменнной. Используется синтаксис для функции preg_replace() из руководства по PHP.

Пример 5.14. regex_replace

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Infertility unlikely to\nbe passed on, experts say.");
$smarty->display('index.tpl');

index.tpl:

{* replace each carriage return, tab & new line with a space *}

{$articleTitle}
{$articleTitle|regex_replace:"/[\r\t\n]/":" "}

OUTPUT:

Infertility unlikely to
 be passed on, experts say.
Infertility unlikely to be passed on, experts say.
```

replace

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	string	Да	*n/a*	Строка для поиска.
2	string	Да	*n/a*	Строка для замены.

Выполняеться простой поиск и замена строки.

Пример 5.15. replace

```
index.php:
```

```
$smarty = new Smarty;
$smarty->assign('articleTitle', "Child's Stool Great for Use in Garden.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|replace:"Garden":"Vineyard"}
{$articleTitle|replace:" ":"   "}

OUTPUT:

Child's Stool Great for Use in Garden.
Child's Stool Great for Use in Vineyard.
Child's  Stool  Great  for  Use  in  Garden.
```

spacify

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	string	Нет	*один пробел*	Строка вставялется между каждым символом переменной.

spacify позволяет вставить пробел между каждым символом переменной. Можно также указать другой символ (или строку) для вставки.

Пример 5.16. spacify

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Something Went Wrong in Jet Crash, Experts Say.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|spacify}
{$articleTitle|spacify:"^^"}

OUTPUT:

Something Went Wrong in Jet Crash, Experts Say.
S o m e t h i n g  W e n t  W r o n g  i n  J e t  C r a s h ,  E x p e r t s  S a y .
S^^o^^m^^e^^t^^h^^i^^n^^g^^ ^^W^^e^^n^^t^^ ^^W^^r^^o^^n^^g^^ ^^i^^n^^ ^^J^^e^^t^^ ^^C^^r^^a^^s^^h^^,^^ ^^E^^x^^p^^e^^r^^t^^s
```

string_format

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	string	Да	*n/a*	Формат. (sprintf)

Форматирует строку по указанному формату. Используется синтаксис форматирования PHP функции sprintf.

Пример 5.17. string_format

```
index.php:

$smarty = new Smarty;
$smarty->assign('number', 23.5787446);
$smarty->display('index.tpl');

index.tpl:

{$number}
{$number|string_format:"%.2f"}
{$number|string_format:"%d"}

OUTPUT:

23.5787446
23.58
24
```

strip

Заменяет все повторные пробелы, новые строки и знаки табуляции на одиночный пробел или на заданную строку.

> **Замечание:** Если вы хотите сделать аналогичную операцию над частью текста шаблона, то используйте функцию strip.

Пример 5.18. strip

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Grandmother of\neight makes\t    hole in one.");
$smarty->display('index.tpl');

index.tpl:
```

```
{$articleTitle}
{$articleTitle|strip}
{$articleTitle|strip:" "}

OUTPUT:

Grandmother of
eight makes      hole in one.
Grandmother of eight makes hole in one.
Grandmother of eight makes hole in one.
```

strip_tags

Вырезает HTML теги, обычно все между < и >.

Пример 5.19. strip_tags

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Blind Woman Gets <font face=\"helvetica\">New Kidney</font> from Dad she Hasn't Seen in <b>years</b>.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|strip_tags}

OUTPUT:

Blind Woman Gets <font face="helvetica">New Kidney</font> from Dad she Hasn't Seen in <b>years</b>.
Blind Woman Gets New Kidney from Dad she Hasn't Seen in years.
```

truncate

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	integer	Нет	80	Количество символов для обрезания to.
2	string	Нет	...	Текст, который добавляется, если произошло обрезание.
3	boolean	Нет	false	Указывает, надо ли обрезать по границе слова (false) или по

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
				символу (true).

Обрезает переменную по указанной длине (по умолчанию 80). Вторым параметром можно указать текст, который будет добавлен в конец обрезанной строки. По умолчанию truncate будет пытаться вырезать слово, которой лежит на вырезаемой границе, целиком. Можно указать третий параметр true, если надо обрезать строку точно по определенному символу.

Пример 5.20. truncate

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Two Sisters Reunite after Eighteen Years at Checkout Counter.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|truncate}
{$articleTitle|truncate:30}
{$articleTitle|truncate:30:""}
{$articleTitle|truncate:30:"---"}
{$articleTitle|truncate:30:"":true}
{$articleTitle|truncate:30:"...":true}

OUTPUT:

Two Sisters Reunite after Eighteen Years at Checkout Counter.
Two Sisters Reunite after Eighteen Years at Checkout Counter.
Two Sisters Reunite after...
Two Sisters Reunite after
Two Sisters Reunite after---
Two Sisters Reunite after Eigh
Two Sisters Reunite after E...
```

upper

Заменяет все маленькие буквы на большие.

Пример 5.21. upper

```
<?php

$smarty = new Smarty;
$smarty->assign('articleTitle', "If Strike isn't Settled Quickly it may Last a While.");
$smarty->display('index.tpl');
```

```
?>
```

Где index.tpl:

```
{$articleTitle}
{$articleTitle|upper}
```

Результатом будет:

```
If Strike isn't Settled Quickly it may Last a While.
IF STRIKE ISN'T SETTLED QUICKLY IT MAY LAST A WHILE.
```

См. также lower и capitalize.

wordwrap

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	integer	Нет	80	Количество столбцов для переноса.
2	string	Нет	\n	Строка, которая вставляется на месте переноса.
3	boolean	Нет	false	Указывает, переносить по словам (false) или нет (true).

Переносит строку по количеству столбцов (по умолчанию 80). Можно указать строку, которая будет вставлятся на месте переноса (по умолчанию символ новой строки). По умолчанию wordwrap пытается переносить по словам, но если указать третим параметром true, то переноситься будет по конкретному символу.

Пример 5.22. wordwrap

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Blind woman gets new kidney from dad she hasn't seen in years.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}

{$articleTitle|wordwrap:30}
```

```
{$articleTitle|wordwrap:20}

{$articleTitle|wordwrap:30:"<br>\n"}

{$articleTitle|wordwrap:30:"\n":true}

OUTPUT:

Blind woman gets new kidney from dad she hasn't seen in years.

Blind woman gets new kidney
from dad she hasn't seen in
years.

Blind woman gets new
kidney from dad she
hasn't seen in
years.

Blind woman gets new kidney<br>
from dad she hasn't seen in years.

Blind woman gets new kidney fr
om dad she hasn't seen in year
s.
```

Глава 6. Комбинирование модификаторов

Можно применять любое количество модификаторов к переменной. Они будут применяться в порядке их упоминания слева направо. Модификаторы должны быть разделены символом | (вертикальная черта).

Пример 6.1. Комбинирование модификаторов

```
<?php

$smarty->assign('articleTitle', 'Капля никотина убивает лошадь, хомячка разрывает на куски.');

?>
```

Содержимое шаблона:

```
{$articleTitle}
{$articleTitle|upper|spacify}
{$articleTitle|lower|spacify|truncate}
{$articleTitle|lower|truncate:30|spacify}
{$articleTitle|lower|spacify|truncate:30:". . ."}
```

Результат выполнения данного примера:

```
Капля никотина убивает лошадь, хомячка разрывает на куски.
К А П Л Я  Н И К О Т И Н А ...вырезано... В А Е Т  Н А  К У С К И .
к а п л я  н и к о т и н а ...вырезано... х о м я ч к а...
к а п л я  н и к о т и н а  у б и в а е т  л о ш а д ь . . .
к а п л я  н и к о т и н . . .
```

Глава 7. Встроенные функции

Содержание

В smarty включены несколько встроенных функций. Встроенные функции интегрированы в язык шаблонов. Нельзя создавать пользовательские функции с такими же названиями или как-либо модифицировать встроенные функции..

capture

{capture} используется для того, чтобы собрать результат работы шаблона в какую-то переменную, вместо того, чтобы вывести результат браузеру. Любое содержимое между {capture name="foo"} и {/capture} сохраняется в переменную, указанную в атрибуте name. Затем его можно использовать в шаблоне при помощи специальной переменной $smarty.capture.foo, где "foo" - значение, переданное атрибуту name. Если атрибут name не указан, то используется "default". Каждая комманда {capture} должна иметь пару {/capture}. Команда capture поддерживает вложенность.

Предостережение

Будте осторожны, сохраняя вывод команды {insert}. Если вы используете кэширование и в области кэширования встречаются команды **insert**, то не сохраняйте данный вывод.

Пример 7.1. Сохранение вывода шаблона

```
{* мы не хотим выводить строку таблицы, если содержимое не отображается *}
{capture name=banner}
  {include file='get_banner.tpl'}
{/capture}
{if $smarty.capture.banner ne ''}
<table>
```

```
   <tr>
      <td>
                        {$smarty.capture.banner}
                </td>
         </tr>
</table>
{/if}
```

Пример 7.2. сохранение содержимого в переменную

Этот пример также демонстрирует функцию {popup}

```
{capture name=some_content assign=popText}
   .... some content ....
{/capture}

<a href="#" {popup caption='Help' text=$popText}>help</a>
```

См. также $smarty.capture, {eval}, {fetch}, fetch() и {assign}.

{config_load}

{config_load} используется для загрузки конфигурационных переменных (#variable#) из конфигурационных файлов в шаблон.

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
file	string	Да	*n/a*	Имя config файла для загрузки
section	string	Нет	*n/a*	Имя секции для загрузки
scope	string	Нет	*local*	Способ обработки области видимости загруженных переменных. Должен быть одинм из local, parent или global. local означает, что переменные загружены в контекст локального шаблона. parent означает, что переменные загружены в контекст как локального, так и родительского шаблона. global означа-

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				ет, что переменные доступны из любого шаблона.
global	boolean	Нет	*No*	Доступны ли переменные из родительского шаблона. Аналогичен scope=parent. ЗАМЕЧАНИЕ: Этот атрибут перекрывается атрибутом scope, но все еще поддерживается. Если scope указан, то это значение игнорируется.

Пример 7.3. {config_load}

example.conf

```
#это комментарий конфигурационного файла

# глобальные переменные
pageTitle = "Main Menu"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

#секция переменных customer
[Customer]
pageTitle = "Customer Info"
```

и шаблон

```
{config_load file="example.conf"}

<html>
 <head>
  <title>{#pageTitle#|default:"No title"}</title>
 </head>
 <body bgcolor="{#bodyBgColor#}">
  <table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
        <tr bgcolor="{#rowBgColor#}">
                <td>First</td>
                <td>Last</td>
                <td>Address</td>
        </tr>
  </table>
 </body>
```

```
</html>
```

Конфигурационные файлы могут также содержать секции. Вы можете загружать переменные из определенной секции, указав атрибут *'section'*.

> **Замечание:** *Секции файлов конфигурации* и встроенная функция *{section}* не имеют ничего общего, кроме схожего названия.

Пример 7.4. функция {config_load} с секцией

```
{config_load file='example.conf' section='Customer'}

<html>
 <head>
  <title>{#pageTitle#|default:"No title"}</title>
 </head>
 <body bgcolor="{#bodyBgColor#}">
  <table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
        <tr bgcolor="{#rowBgColor#}">
              <td>First</td>
              <td>Last</td>
              <td>Address</td>
        </tr>
  </table>
 </body>
</html>
```

См. $config_overwrite для массивов конфигурационных переменных.

См. также Конфигурационные файлы, Конфигурационные переменные, $config_dir, get_config_vars() и config_load().

{foreach},{foreachelse}

Циклы *{foreach}* являются альтернативой циклам *{section}*. *{foreach}* используется для прохода по **единственному ассоциативному массиву**. Синтаксис *{foreach}* намного проще синтаксиса *{section}*, но с другой стороны **его можно использовать только для одного массива**. Каждый тег *{foreach}* должен иметь пару *{/foreach}*. Обязательными параметрами являются *from* и *item*. Имя цикла {foreach} может быть любым, состоящим из букв, цифр и знаков подчеркивания. Циклы *{foreach}* могут быть вложенными и имена вложенных {foreach} должны быть уникакльными между собой. Параметр *from* (обычно - массив значений) определяет количество итераций цикла *{foreach}*. *{foreachelse}* выполняется в том случае, если параметр *from* не содержит значений.

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
from	array	Да	*n/a*	Массив, по которому надо пройтись
item	string	Да	*n/a*	Имя переменной,

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				которая будет выступать в качестве значения текущего елемента
key	string	Нет	*n/a*	Имя переменной, которая будет выступать в качестве ключа текущего елемента
name	string	Нет	*n/a*	Название цикла foreach для доступа к его свойствам

Пример 7.5. {foreach} - предмет

```
<?php
$arr = array( 1001,1002,1003);
$smarty->assign('custid', $arr);
?>
```

```
{* этот пример напечатает все переменные массива $custid *}
{foreach from=$custid item=curr_id}
   id: {$curr_id}<br />
{/foreach}
```

Результат выполнения данного примера:

```
id: 1000<br />
id: 1001<br />
id: 1002<br />
```

Пример 7.6. {foreach} - предмет и ключ

```
// Ключ содержит ключ для каждого значения из цикла
// назначение выглядит примерно так:
<?php
$smarty->assign('contacts', array(
                        array('phone' => '1',
                              'fax' => '2',
                               'cell' => '3'),
                        array('phone' => '555-4444',
                              'fax' => '555-3333',
                              'cell' => '760-1234')
                        ));
?>
```

```
{foreach name=outer item=contact from=$contacts}
    <hr />
    {foreach key=key item=item from=$contact}
        {$key}: {$item}&lt;br&gt;
        {$key}: {$item}<br />
    {/foreach}
{/foreach}
```

Результат выполнения данного примера:

```
<hr />
    phone: 1<br />
    fax: 2<br />
    cell: 3<br />
<hr />
    phone: 555-4444<br />
    fax: 555-3333<br />
    cell: 760-1234<br />
```

Пример 7.7. {foreach} - базы данных (к примеру, PEAR или ADODB)

```
<?php
  $sql = 'select contact_id, name, nick from contacts order by contact';
  $smarty->assign("contacts", $db->getAssoc($sql));
?>
```

```
{foreach key=cid item=con from=$contacts}
    <a href="contact.php?contact_id={$cid}">{$con.name} - {$con.nick}</a><br />
{/foreach}
```

Циклы {foreach} также имеют собственные переменные, которые содержат свойства {foreach}. Они обозначаются так: {$smarty.foreach.foreachname.varname} {$smarty.foreach.foreachname.varname}, где foreachname - это название цикла (значение атрибута *name*).

См. {section} для примеров следующих свойств, так как они идентичны.

iteration

iteration используется для отображения текущего номера итерации цикла. Итерации всегда начинаются с 1 и увеличиваются на одну при каждом прохождении цикла.

first

first устанавливается в true, если текущая итерация первая.

last

last устанавливается в true, если текущая итерация последняя.

show

Атрибут *show* может принимать логические значения (истина или ложь). Если ложь, то цикл foreach не будет отображаться. Если присутствует тэг foreachelse, то он будет отображен.

total

total хранит количество итераций цикла. Может быть использовано как в цикле, так и вне его.

См. также {section} и $smarty.foreach.

{if},{elseif},{else}

Конструкция *{if}* в Smarty такая же гибкая, как и конструкция **if** [http://php.net/if] в PHP, только с несколькими дополнительными возможностями для шаблонов. Каждый тэг *{if}* должен иметь пару *{/if}*. *{else}* и *{elseif}* так же допустимы. Досутпны все квалификаторы и функции из PHP, такие как *||*, *or*, *&&*, *and*, *is_array()* и т.д.

Ниже следует список распознаваемых квалификаторов, которые должны быть отделены от окружающих элементов пробелами. Обратите внимания, что объекты в [квадратных скобках] являются необязательными. Иногда указаны эквиваленты в PHP.

Квалификатор	Альтернативы	Пример синтаксиса	Описание
==	eq	$a eq $b	равно
!=	ne, neq	$a neq $b	не равно
>	gt	$a gt $b	больше
<	lt	$a lt $b	меньше
>=	gte, ge	$a ge $b	больше или равно
<=	lte, le	$a le $b	меньше или равно
===		$a === 0	проверка идентичности
!	not	not $a	отрицание
%	mod	$a mod $b	остаток от деления
is [not] div by		$a is not div by 4	возможно деление без остатка
is [not] even		$a is not even	[не]чётно
is [not] even by		$a is not even by $b	[не]чётно значению
is [not] odd		$a is not odd	[не]нечётно
is [not] odd by		$a is not odd by $b	[не]нечётно значению

Пример 7.8. {if} statements

```
{if $name eq "Fred"}
        Welcome Sir.
{elseif $name eq "Wilma"}
    Welcome Ma'am.
{else}
    Welcome, whatever you are.
{/if}

{* пример с логикой "или" *}
{if $name eq "Fred" or $name eq "Wilma"}
        ...
{/if}

{* то же самое, что и выше *}
{if $name == "Fred" || $name == "Wilma"}
        ...
{/if}

{*
  следующий пример НЕ будет работать, квалификаторы условий следует
  отделять от окружающих элементов пробелами
*}
{if $name=="Fred" || $name=="Wilma"}
        ...
{/if}


{* скобки разрешены *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}
        ...
{/if}

{* вы также можете использовать функции php *}
{if count($var) gt 0}
        ...
{/if}

{* проверяет чётность значений *}
{if $var is even}
        ...
{/if}
{if $var is odd}
        ...
{/if}
{if $var is not odd}
        ...
{/if}

{* проверяет, делится ли $var на 4 без остатка *}
{if $var is div by 4}
        ...
{/if}

{*
  проверяет, является ли $var чётным двум, например
```

```
 0=чётно, 1=чётно, 2=нечётно, 3=нечётно, 4=чётно, 5=чётно и т.д.
*}
{if $var is even by 2}
      ...
{/if}

{* 0=чётно, 1=чётно, 2=чётно, 3=нечётно, 4=нечётно, 5=нечётно и т.д. *}
{if $var is even by 3}
      ...
{/if}

{* ------- if с функциями PHP ------- *}
{* check for array. *}
{if is_array($foo) }
   .....
{/if}

{* проверка на существование *}
{if isset($foo) }
   .....
{/if}
```

{include}

Тэги {include} используются для включения других шаблонов в текущий. Любые переменные, доступные в текущем шаблоне, доступны и во включаемом. Тэг {include} должен иметь атрибут "file", который указывает путь к ресурсу шаблона.

Опциональный атрибут *assign* указывает, что результат выполнения {include} будет присвоен переменной вместо отображения.

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
file	string	Да	*n/a*	Имя файла шаблона для включения
assign	string	Нет	*n/a*	Имя переменной, которой присвоится вывод шаблона
[var ...]	[var type]	Нет	*n/a*	Переменные, переданные в локальную область включаемого шаблона

Пример 7.9. Функция {include}

```
<html>
 <head>
  <title>{$title}</title>
```

```
 </head>
 <body>
  {include file='page_header.tpl'}
  {* тут идёт тело шаблона *}
  {include file="$tpl_name.tpl"} <-- заменит $tpl_name его значением
  {include file='page_footer.tpl'}
 </body>
</html>
```

Вы также можете передать переменные в подключаемый шаблон в виде атрибутов. Любая переменная, переданная в подключаемый шаблон, доступны только в области видимости подключаемого файла. Переданные переменные имеют преимущество перед существующими переменными с аналогичными именами.

Пример 7.10. передача переменных в {include}

```
{include file='header.tpl' title='Main Menu' table_bgcolor='#c0c0c0'}

{* тут идёт тело шаблона *}

{include file='footer.tpl' logo='http://my.example.com/logo.gif'}
```

где header.tpl может быть

```
<table border='1' width='100%' bgcolor='{$table_bgcolor|default:"#0000FF"}'>
 <tr>
  <td>
   <h1>{$title}</h1>
  </td>
 </tr>
</table>
```

Пример 7.11. {include} и присвоение переменной

Этот пример присвоит содержимое nav.tpl переменной $navbar, которая затем выводится сверху и снизу страницы.

```
<body>
{include file='nav.tpl' assign=navbar}
{include file='header.tpl' title='Main Menu' table_bgcolor='#effeef'}
{$navbar}

{* тут идёт тело шаблона *}

{include file='footer.tpl' logo='http://my.example.com/logo.gif'}
{$navbar}
</body>
```

Для подключения файлов вне папки $template_dir можно указывать файл с помощью ресурсов.

Пример 7.12. Примеры ресурсов шаблонов в {include}

```
{* абсолютные пути *}
{include file='/usr/local/include/templates/header.tpl'}

{* абсолютные пути (то же самое) *}
{include file='file:/usr/local/include/templates/header.tpl'}

{* абсолютные пути в windows (ОБЯЗАТЕЛЬНО используйте префикс "file:") *}
{include file='file:C:/www/pub/templates/header.tpl'}

{* подключение шаблона из ресурса с именем "db" *}
{include file='db:header.tpl'}

{* подключение шаблона с переменным именем - например, $module = 'contacts' *}
{include file="$module.tpl"}
{* не будет работать, т.к. в одинарных кавычках не работает подстановка переменных *}
{include file='$module.tpl'}
```

См. также {include_php}, {php}, Ресурсы and Составные шаблоны.

{include_php}

Техническое замечание: {include_php} достаточно устарела в Smarty, вы можете достичь этой функциональности при помощи собственных функций шаблона. Единственная причина для использования {include_php} - это серьёзная необходимость отделить PHP-функцию от директории plugins или кода вашего приложения. См. примеры составных шаблонов для дополнительной информации.

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
file	string	Да	*n/a*	Имя подключаемого php файла
once	boolean	Нет	*true*	Указывает подключать файл или нет, если он уже был однажды подключен
assign	string	Нет	*n/a*	Название переменной, которой будет присвоен вывод include_php

Тэги {include_php} используются для подключения PHP-скрипта в шаблон. Если режим security включен, то PHP-скрипт должен быть расположен в директории $trusted_dir. Тэг {include_php} должен иметь атрибут "file", который указывает путь к подключаемому PHP-файлу, либо относительный к $trusted_dir, либо абсолютный путь.

По умолчанию, PHP-файлы подключаются только один раз, даже если вызываются несколько раз в шаблоне. Можно указать, что файл должен быть подключен каждый раз, указав атрибут *once*. Установив once в ложь (false) указывает, что файл должен быть подключен вне зависимости от того, был ли он подключен раньше.

Можно указать опциональный атрибут *assign*, который указывает имя переменной, которой будет присвоен вывод *{include_php}*, вместо отображения.

Объект smarty доступен в подключаемом PHP-файле как $this.

Пример 7.13. Функция {include_php}

load_nav.php

```
<?php

// загружает переменные из БД MySQL и присваивает их шаблону
require_once("MySQL.class.php");
$sql = new MySQL;
$sql->query("select * from site_nav_sections order by name",SQL_ALL);
$this->assign('sections',$sql->record);

?>
```

index.tpl

```
{* абсолютный путь, либо относительный к $trusted_dir *}
{include_php file="/path/to/load_nav.php"}
{foreach item="curr_section" from=$sections}
    <a href="{$curr_section.url}">{$curr_section.name}</a><br />
{/foreach}
```

См. также {include}, {php}, {capture}, Ресурсы и Составные шаблоны

{insert}

Тэг {insert} очень похож на тэг {include}, за исключением того, что {insert} НЕ кэшируется, когда кэширование включено. Он будет выполнен при каждом обращении к шаблону.

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Да	*n/a*	Имя функции вставки (insert_name)
assign	string	Нет	*n/a*	Имя переменной, которой будет присвоен вывод
script	string	Нет	*n/a*	Имя php файла, который будет под-

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				ключен перед вызовом функции вставки
[var ...]	[var type]	Нет	*n/a*	Переменные, передаваемые в функцию вставки

Допустим, вы имеете шаблон с баннером вверху страницы. Баннер может содержать любую смесь HTML, изображений, flash и т.д., то есть нельзя использовать просто статическую ссылку, и мы не хотим, чтобы код баннера кэшировался с остальной страницей. Тогда используем тэг {insert}: шаблон знает значения #banner_location_id# и #site_id# (взяты из конфигурационного файла) и должен вызвать функцию, чтобы получить код баннера.

Пример 7.14. функция {insert}

```
{* пример вставки баннера *}
{insert name="getBanner" lid=#banner_location_id# sid=#site_id#}
```

В этом примере мы используем имя "getBanner" и передаем параметры #banner_location_id# и #site_id#. Smarty попробует вызвать функцию insert_getBanner() в вашей PHP программе, передав значения #banner_location_id# и #site_id# первым параметром в виде ассоциативного массива. Все имена функций вставки должны начинаться с "insert_" для предотвращения возможных конфликтов имен. Функция insert_getBanner() должна обработать переданные переменные и вернуть результат. Он будет отображен в шаблоне вместо тэга {insert}. В данном случае Smarty вызовет функцию insert_getBanner(array("lid" => "12345","sid" => "67890")); и выведет результат на месте тэга {insert}.

Если указан атрибут "assign", то вывод функции вставки будет присвоен указанной переменной вместо отображения. ЗАМЕЧАНИЕ: присвоение вывода тэга {insert} переменной шаблона не очень полезно, когда кеширование включено.

Если указан атрибут "script", то указанный PHP-файл будет подключен (только однажды) перед вызовом функции вставки. Это удобно, когда функция может не сущетсвовать, и должен быть подключен PHP-файл, чтобы определить функцию. Путь к файлу должен быть либо абсолтным, либо относительным относительно $trusted_dir. Когда security активирована, то PHP-файл должен быть в папке $trusted_dir.

Обьект Smarty передается в функцию как второй параметр. Так вы можете использовать и модифицировать информацию из объекта Smarty в функциях вставки.

> **Техническое Замечание:** Некоторые части шаблона можно не кэшировать. Если активировано кэширование, то тэг {insert} все равно не будет кэширован. Он будет вызван каждый раз при генерации страницы, даже из кешированных страниц. Это полезно для таких вещей, как баннеры, опросы, прогнозы погоды, результаты поиска, области обратной связи и т.д.

{ldelim},{rdelim}

{ldelim} и {rdelim} используются для предотвращения обработки разделителей, по-умолчанию "{" и "}". Вы

также можете использовать блок {literal}{/literal} для предотвращения обработки блоков текста. См. также {$smarty.ldelim}

Пример 7.15. {ldelim}, {rdelim}

```
{* будут выведены разделители в шаблоне *}

{ldelim}funcname{rdelim} is how functions look in Smarty!
```

Результат выполнения данного примера:

```
{funcname} is how functions look in Smarty!
```

Другой пример и немного javascript'a

```
<script language="JavaScript">
function foo() {ldelim}
    ... code ...
{rdelim}
</script>
```

выведет

```
<script language="JavaScript">
function foo() {
    .... code ...
}
</script>
```

См. также Предотвращение обработки Smarty

{literal}

Тэги {literal} позволяют воспринимать блоки данных буквально. Обычно они используются вместе с javascript или таблицами стилей, в которых фигурные скобки конфликтуют с синтаксисом разделителей. Весь текст внутри тэгов {literal}{/literal} не интерпретируется, а выводится "как есть". Если вам нужно вставить тэги шаблонов в блок {literal}, вам следует пойти по другому пути и использовать {ldelim}{rdelim} для экранирования отдельных разделителей.

Пример 7.16. тэги literal

```
{literal}
<script type="text/javascript">
<!--
  function isblank(field) {
    if (field.value == '')
      { return false; }
    else
```

```
    {
    document.loginform.submit();
    return true;
  }
 }
// -->
</script>
{/literal}
```

См. также Предотвращение обработки Smarty

{php}

Тэг {php} позволяет вставлять PHP-код прямо в шаблон. Он не будет как-либо изменен, независимо от $php_handling настроек. Этот тэг только для продвинутых пользователей, так как обычно не требуется.

Пример 7.17. тэги {php}

```
{php}
   // подключение php скрипта прямо
            // из шаблона
            include('/path/to/display_weather.php');
{/php}
```

> **Техническое замечание:** Для доступа к переменным PHP внутри блоков {php}, вам может понадобится использовать ключевое слово PHP global [http://php.net/global]

Пример 7.18. Тэги {php} с глобальными переменными

```
{php}
  global $foo, $bar;
  if($foo == $bar){
    // что-нить делаем
  }
{/php}
```

См. также $php_handling, {include_php}, {include} и Компонентные шаблоны.

{section},{sectionelse}

Секции используются для обхода **массивов данных** (так же, как и {foreach}). Каждый тег *{section}* должен иметь пару *{/section}*. Обязательными параметрами являются *name* и *loop*. Имя цикла {section} может быть любым, состоящим из букв, цифр и знаков подчеркивания. Циклы *{section}* могут быть вложенными и имена вложенных {section} должны быть уникакльными между собой. Переменная *loop* (обычно - массив значе-

ний) определяет количество итераций цикла. При печати переменных внутри секции, имя секции должно быть указано рядом с именем переменной внутри квадратных скобок []. *{sectionelse}* выполняется в том случае, если параметр *loop* не содержит значений.

Имя атрибута	**Тип**	**Обязателен**	**По умолчанию**	**Описание**
name	string	Да	*n/a*	Название секции
loop	mixed	Да	*n/a*	Значение, определяющее количество итераций цикла.
start	integer	Нет	*0*	Индекс позиции, с которой будет начинаться цикл. Если значение отрицательное, то начальная позиция вычисляется от конца массива. Например, если в переменной цикла 7 элементов и значение атрибута start равно -2, то начальный индекс будет 5. Неверные значения (значения, вне массива) автоматически обрезаются до ближайшего верного значения.
step	integer	Нет	*1*	Значение шага, которое используется для прохода по массиву. Например, step=2 указывает обход массива по элементам 0,2,4... Если шаг отрицателен, то обход массива будет производится в обратном направлении.
max	integer	Нет	*1*	Максимальное количество итераций цикла.
show	boolean	Нет	*true*	Указывает, показывать или нет эту секцию

Пример 7.19. {section}

```
<?php

$data = array(1000,1001,1002);
$smarty->assign('custid',$data);

?>
```

```
{* этот пример напечатает все значения массива $custid *}
{section name=customer loop=$custid}
    id: {$custid[customer]}<br />
{/section}
<hr />
{* этот пример напечатает все значения массива $custid в обратном порядке *}
{section name=foo loop=$custid step=-1}
    {$custid[foo]}<br />
{/section}
```

Результат выполнения данного примера:

```
id: 1000<br />
id: 1001<br />
id: 1002<br />
<hr />
id: 1002<br />
id: 1001<br />
id: 1000<br />
```

Ещё немного примеров без присвоенного массива.

```
{section name=foo start=10 loop=20 step=2}
    {$smarty.section.foo.index}
{/section}
<hr />
{section name=bar loop=21 max=6 step=-2}
    {$smarty.section.bar.index}
{/section}
```

Результат выполнения данного примера:

```
10 12 14 16 18
<hr />
20 18 16 14 12 10
```

Пример 7.20. Переменная loop команды {section}

```
<?php
```

```
$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 N 45th', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);

?>
```

```
{*
  переменная loop определяет только количество итераций.
  вы можете получать доступ к любой переменной из шаблона внутри секции.
  Этот пример предполагает, что $custid, $name и $address все являются
  массивами, содержащими одинаковое количество значений
*}
{section name=customer loop=$custid}
<p>
    id: {$custid[customer]}<br />
    name: {$name[customer]}<br />
    address: {$address[customer]}
</p>
{/section}
```

Результат выполнения данного примера:

```
<p>
    id: 1000<br />
    name: John Smith<br />
    address: 253 N 45th
</p>
<p>
    id: 1001<br />
    name: Jack Jones<br />
    address: 417 Mulberry ln
</p>
<p>
    id: 1002<br />
    name: Jane Munson<br />
    address: 5605 apple st
</p>
```

Пример 7.21. именование {section}

```
{*
  имя секции может быть любым, так как оно используется для обращения к
  данным в пределах секции
*}
{section name=anything loop=$custid}
```

```
<p>
   id: {$custid[anything]}<br />
   name: {$name[anything]}<br />
   address: {$address[anything]}
</p>
{/section}
```

Пример 7.22. вложенные секции

```
<?php

$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 N 45th', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);

$types = array(
           array( 'home phone', 'cell phone', 'e-mail'),
           array( 'home phone', 'web'),
           array( 'cell phone')
         );
$smarty->assign('contact_type', $types);

$info = array(
           array('555-555-5555', '666-555-5555', 'john@myexample.com'),
           array( '123-456-4', 'www.example.com'),
           array( '0457878')
        );
$smarty->assign('contact_info', $info);
?>
```

```
{*
 секции могут иметь вложенность любой глубины. Используя вложенные секции,
 вы можете обращаться к сложным структурам данных, таким как многомерные
 массивы. В этом примере $contact_type[customer] - это массив
 типов контактов для текущего клиента.
*}
{section name=customer loop=$custid}
<hr>
   id: {$custid[customer]}<br />
   name: {$name[customer]}<br />
   address: {$address[customer]}<br />
          {section name=contact loop=$contact_type[customer]}
       {$contact_type[customer][contact]}: {$contact_info[customer][contact]}<br />
          {/section}
{/section}
```

Результат выполнения данного примера:

```
<hr>
   id: 1000<br />
   name: John Smith<br />
   address: 253 N 45th<br />
      home phone: 555-555-5555<br />
      cell phone: 666-555-5555<br />
      e-mail: john@myexample.com<br />
<hr>
   id: 1001<br />
   name: Jack Jones<br />
   address: 417 Mulberry ln<br />
      home phone: 123-456-4<br />
      web: www.example.com<br />
<hr>
   id: 1002<br />
   name: Jane Munson<br />
   address: 5605 apple st<br />
      cell phone: 0457878<br />
```

Пример 7.23. секции и ассоциативные массивы

```
<?php

$data = array(
          array('name' => 'John Smith', 'home' => '555-555-5555',
               'cell' => '666-555-5555', 'email' => 'john@myexample.com'),
          array('name' => 'Jack Jones', 'home' => '777-555-5555',
               'cell' => '888-555-5555', 'email' => 'jack@myexample.com'),
          array('name' => 'Jane Munson', 'home' => '000-555-5555',
               'cell' => '123456', 'email' => 'jane@myexample.com')
       );
$smarty->assign('contacts',$data);

?>
```

```
{*
  Это пример вывода ассоциативного массива
  данных внутри секции
*}
{section name=customer loop=$contacts}
<p>
   name: {$contacts[customer].name}<br />
   home: {$contacts[customer].home}<br />
   cell: {$contacts[customer].cell}<br />
   e-mail: {$contacts[customer].email}
</p>
{/section}
```

Результат выполнения данного примера:

```
<p>
    name: John Smith<br />
    home: 555-555-5555<br />
    cell: 666-555-5555<br />
    e-mail: john@myexample.com
</p>
<p>
    name: Jack Jones<br />
    home phone: 777-555-5555<br />
    cell phone: 888-555-5555<br />
    e-mail: jack@myexample.com
</p>
<p>
    name: Jane Munson<br />
    home phone: 000-555-5555<br />
    cell phone: 123456<br />
    e-mail: jane@myexample.com
</p>
```

Базы данных (например, PEAR или ADODB)

```
<?php

$sql = 'select id, name, home, cell, email from contacts';
$smarty->assign('contacts',$db->getAll($sql) );

?>
```

```
{*
   выводим результат запроса к БД в таблицу
*}
<table>
<tr><th> </th><th>Name</th><th>Home</th><th>Cell</th><th>Email</th></tr>
{section name=co loop=$contacts}
  <tr>
    <td><a href="view.php?id={$contacts[co].id}">view<a></td>
    <td>{$contacts[co].name}</td>
    <td>{$contacts[co].home}</td>
    <td>{$contacts[co].cell}</td>
    <td>{$contacts[co].email}</td>
  <tr>
{/section}
</table>
```

Пример 7.24. {sectionelse}

```
{* sectionelse будет выполнена в том случае, если $custid не содержит значений *}
{section name=customer loop=$custid}
    id: {$custid[customer]}<br />
{sectionelse}
    there are no values in $custid.
```

```
{/section}
```

Секции так же имеют собственные переменные, которые содержат свойства секций. Они обозначаются так: {$smarty.section.sectionname.varname}

> **Замечание:** Начиная с версии Smarty 1.5.0, синтаксис переменных свойств сессий был изменен с {%sectionname.varname%} на {$smarty.section.sectionname.varname}. Старый синтаксис всё ещё поддерживается, но вы увидите лишь примеры нового синтаксиса.

index

index используется для отображения текущего индекса массива, начиная с нуля (или с атрибута start, если он был указан) и увеличиваясь на единицу (или на значение атрибута step, если он был указан).

> **Техническое Замечание:** Если атрибуты step и start не указаны, то index аналогичен атрибуту секции iteration, кроме того, что начинается с 0, а не с 1.

Пример 7.25. свойства {section} index

```
{* к вашему сведению, $custid[customer.index] и $custid[customer] означают одно и то же *}

{section name=customer loop=$custid}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
```

Результат выполнения данного примера:

```
0 id: 1000<br />
1 id: 1001<br />
2 id: 1002<br />
```

index_prev

index_prev используется для отображения предыдущего индекса цикла На первой итерации он установлен в -1.

index_next

index_next используется для отображения следующего индекса цикла На последней итерации он всё же на единицу больше текущего (или на другое значение, если указан атрибут step).

Пример 7.26. свойства {section} index_next и index_prev

```
<?php
```

```
$data = array(1001,1002,1003,1004,1005);
$smarty->assign('custid',$data);

?>
```

```
{* к вашему сведению, $custid[cus.index] и $custid[cus] означают одно и то же *}

<table>
  <tr>
    <th>index</th><th>id</th>
    <th>index_prev</th><th>prev_id</th>
    <th>index_next</th><th>next_id</th>
  </tr>
{section name=cus loop=$custid}
  <tr>
    <td>{$smarty.section.cus.index}</td><td>{$custid[cus]}</td>
    <td>{$smarty.section.cus.index_prev}</td><td>{$custid[cus.index_prev]}</td>
    <td>{$smarty.section.cus.index_next}</td><td>{$custid[cus.index_next]}</td>
  </tr>
{/section}
</table>
```

Результатом выполнения этого примера будет таблица, содержащая следующее:

```
index  id    index_prev prev_id index_next next_id
0      1001  -1              1         1002
1      1002  0         1001    2         1003
2      1003  1         1002    3         1004
3      1004  2         1003    4         1005
4      1005  3         1004    5
```

iteration

iteration используется для отображения текущего номера итерации цикла.

> **Замечание:** Это значение не зависит от свойств start, step и max, в отличие от свойства index. Кроме того, итерации начинаются с единицы, а не с нуля, как индексы. rownum - это синоним к свойству iteration, они работают одинаково.

Пример 7.27. свойство {section} iteration

```
<?php

// array of 3000 to 3015
$id = range(3000,3015);
$smarty->assign('custid',$id);

?>
```

```
{section name=cu loop=$custid start=5 step=2}
  iteration={$smarty.section.cu.iteration}
  index={$smarty.section.cu.index}
  id={$custid[cu]}<br />
{/section}
```

Результат выполнения данного примера:

```
iteration=1 index=5 id=3005<br />
iteration=2 index=7 id=3007<br />
iteration=3 index=9 id=3009<br />
iteration=4 index=11 id=3011<br />
iteration=5 index=13 id=3013<br />
iteration=6 index=15 id=3015<br />
```

Этот пример использует свойство iteration для вывода заголовка таблицы через каждые пять строчек (использует {if} с оператором mod - остаток от деления).

```
<table>
{section name=co loop=$contacts}
  {if $smarty.section.co.iteration % 5 == 1}
    <tr><th> </th><th>Name</th><th>Home</th><th>Cell</th><th>Email</th></tr>
  {/if}
  <tr>
    <td><a href="view.php?id={$contacts[co].id}">view<a></td>
    <td>{$contacts[co].name}</td>
    <td>{$contacts[co].home}</td>
    <td>{$contacts[co].cell}</td>
    <td>{$contacts[co].email}</td>
  <tr>
{/section}
</table>
```

first

Параметр first установлен в true, если текущая итерация секции является первой.

last

Параметр last установлен в true, если текущая итерация секции является последней.

Пример 7.28. свойства {section} first и last

Этот пример проходит циклом по массиву $customers, выводит заголовок на первой итерации и футер на последней (использует свойство {section} total)

```
{section name=customer loop=$customers}
  {if $smarty.section.customer.first}
    <table>
```

```
  <tr><th>id</th><th>customer</th></tr>
 {/if}

 <tr>
  <td>{$customers[customer].id}</td>
  <td>{$customers[customer].name}</td>
 </tr>

 {if $smarty.section.customer.last}
  <tr><td></td><td>{$smarty.section.customer.total} customers</td></tr>
  </table>
 {/if}
{/section}
```

rownum

rownum используется для отображения текущего номера итерации цикла, начиная с единицы. Это синоним свойства iteration, они работа идентично.

loop

loop используется для отображения последнего номера индекса, по которому проходила итерация секции. Это свойство может быть использовано как внутри, так и вне секции.

Пример 7.29. свойство {section} index

```
{section name=customer loop=$custid}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}

There were {$smarty.section.customer.loop} customers shown above.
```

Результат выполнения данного примера:

```
0 id: 1000<br />
1 id: 1001<br />
2 id: 1002<br />

There were 3 customers shown above.
```

show

show используется в качестве параметра секции. *show* является булевым значением, true или false. Если false, секция не будет отображена. Если присутствует секция {sectionelse}, вместо этого будет отображена она.

Пример 7.30. атрибут {section} show

```
{*
  $show_customer_info (true/false) может быть передан из приложения PHP,
  чтобы определить, необходимо ли отображать секцию
*}
{section name=customer loop=$custid show=$show_customer_info}
  {$smarty.section.customer.rownum} id: {$custid[customer]}<br />
{/section}

{if $smarty.section.customer.show}
  the section was shown.
{else}
  the section was not shown.
{/if}
```

Результат выполнения данного примера:

```
1 id: 1000<br />
2 id: 1001<br />
3 id: 1002<br />

the section was shown.
```

total

total используется для отображения количества итераций, через которые пройдет эта секция. Это свойство может быть использовано как внутри, так и вне секции.

Пример 7.31. свойство {section} total

```
{section name=customer loop=$custid step=2}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}

    There were {$smarty.section.customer.total} customers shown above.
```

Результат выполнения данного примера:

```
0 id: 1000<br />
2 id: 1002<br />
4 id: 1004<br />

There were 3 customers shown above.
```

См. также {foreach} и $smarty.section.

{strip}

Часто вебдизайнеры сталкиваются с проблемой, что пробелы и переносы строк влияют на отображение HTML в броузере ("фишки" броузера), то есть может понадобится склеить все теги в шаблоне вместе, чтобы получить желаемый результат. Но в результате получается нечитаемый или труднoредактируемый шаблон.

В выводимом тексте, заключенном между тэгами {strip} и {/strip}, удаляются повторные пробелы и переносы строк, перед отображением. Так вы можете сохранив шаблон читаемым не волноваться насчет лишних пробелов.

> **Техническое Замечание:** {strip}{/strip} не влияет на содержимое переменных шаблона. См. модификатор strip.

Пример 7.32. тэги {strip}

```
{* следующее будет выведено в виде одной строки *}
{strip}
<table border='0'>
	<tr>
		<td>
			<a href="{$url}">
			<font color="red">This is a test</font>
			</a>
		</td>
	</tr>
</table>
{/strip}
```

Результат выполнения данного примера:

```
<table border='0'><tr><td><a href="http://...snipped...</a></td></tr></table>
```

Заметьте, что в данном примере все строки начинаются и заканчиваются HTML тэгами. Учтите, что все строки склеиваются вместе. Если в начале или в конце строки присутствует обычный текст, то вы можете не получить желаемый результат.

Глава 8. Пользовательские Функции

Содержание

Smarty поставляется с несколькими пользовательскими функциями, которые вы можете использовать в шаблонах.

assign

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
var	string	Да	*n/a*	Имя переменной, значение которой будет устанавливаться
value	string	Да	*n/a*	Устанавливаемое значение

assign используется для установки значения переменной в процессе выполнения шаблона.

Пример 8.1. assign

```
{assign var="name" value="Bob"}
```

```
Значение $name - {$name}.

OUTPUT:

Значение $name - Bob.
```

counter

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Нет	*default*	Имя счетчика
start	number	Нет	*1*	Начальное значение счетчика
skip	number	Нет	*1*	Шаг счетчика
direction	string	Нет	*up*	Направление (вверх - up/вниз - down)
print	boolean	Нет	*true*	выводить значение счетчика или нет
assign	string	Нет	*n/a*	Имя переменной, которой будет присвоен вывод

counter используется для управления счетчиком. counter запоминает количество итераций. Можно регулировать начало, интервал и направление отсчета, а также указать, выводить ил значение счетчика или нет. Можно запустить несколько счетчиков одновременно, указав уникальное имя для каждого. Если имя счетчика не указано, будет использовано по умолчанию 'default'.

Если указан атрибут "assign", то вывод тэга counter будет присвоен переменной шаблона, вместо отображения.

Пример 8.2. counter

```
{* инициализируем счетчик *}
{counter start=0 skip=2 print=false}

{counter}<br>
{counter}<br>
{counter}<br>
{counter}<br>

OUTPUT:

2<br>
4<br>
6<br>
8<br>
```

cycle

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Нет	*default*	Название цикла
values	mixed	Да	*N/A*	Значения, по которым будет производиться цикл. Либо список, разделеный запятыми (либо другим указанным разделителем), либо массив значений.
print	boolean	Нет	*true*	Выводить значение, или нет
advance	boolean	Нет	*true*	Переключаться или нет на следующее значение
delimiter	string	Нет	,	Разделитель, используемый в атрибуте values.
assign	string	Нет	*n/a*	Имя переменной, которой будет присвоен вывод тэга

Cycle используется для прохода через множество значений. С его помощью можно легко реализовать переключение между двумя и более цветами в таблице, или пройти цикл через массив.

Можно проходить через несколько множеств значений одновременно, указав атрибут name. Имена должны быть уникальными.

Можно не отображать данный элемент, установив атрибут print в ложь (false). Удобно для пропуска значения, без его вывода.

Атрибут advance используется для повтора значения. Если установлен в истина (true), то при следующем вызове cycle будет выведено то же значение.

Если указан специальный атрибут "assign", то вывод cycle присваивается переменной, вместо отображения.

Пример 8.3. cycle

```
{section name=rows loop=$data}
<tr bgcolor="{cycle values="#eeeeee,#d0d0d0"}">
   <td>{$data[rows]}</td>
</tr>
{/section}

OUTPUT:
```

```
<tr bgcolor="#eeeeee">
  <td>1</td>
</tr>
<tr bgcolor="#d0d0d0">
  <td>2</td>
</tr>
<tr bgcolor="#eeeeee">
  <td>3</td>
</tr>
```

debug

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
output	string	Нет	*html*	Тип вывода (html или javascript)

{debug} выводит консоль отладки. Это работает независимо от значения опции debug. Так как этот тэг обрабатывается в процесе выполнения, то возможно вывести только присвоенные переменные, но не используемые шаблоны. Но вы видите все переменные, доступные в области видимости текущего шаблона.

eval

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
var	mixed	Да	*n/a*	Переменная (или строка) для обработки
assign	string	Нет	*n/a*	Имя переменной, которой будет присвоен вывод

eval используется для обработки переменной, как шаблона. Можно использовать для таких вещей, как хранение шаблонных тэгов/переменных в переменной или в файлах конфигруации.

Если указан специальный атрибут "assign", то вывод тэга eval присваивается переменной, вместо отображения.

> **Техническое Замечание:** Переменные шаблоны обрабатываются так же, как и обычные шаблоны. Они подвластны тем же правилам и ограничениям безопасности.

> **Техническое Замечание:** Переменные шаблоны компилируются при каждом обращении. Откомпилированные версии не сохраняются! Однако, если кэширование включено, то вывод будет закэширован с остальной частью шаблона.

Пример 8.4. eval

```
setup.conf
----------

emphstart = <b>
emphend = </b>
title = Welcome to {$company}'s home page!
ErrorCity = You must supply a {#emphstart#}city{#emphend#}.
ErrorState = You must supply a {#emphstart#}state{#emphend#}.


index.tpl
---------

{config_load file="setup.conf"}

{eval var=$foo}
{eval var=#title#}
{eval var=#ErrorCity#}
{eval var=#ErrorState# assign="state_error"}
{$state_error}

OUTPUT:

This is the contents of foo.
Welcome to Foobar Pub & Grill's home page!
You must supply a <b>city</b>.
You must supply a <b>state</b>.
```

fetch

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
file	string	Да	*n/a*	файл, http или ftp сайт для отображения
assign	string	Нет	*n/a*	Имя переменной, которой будет присвоен вывод

fetch используется для отображения локальных файлов, http или ftp страниц. Если файл начинается с "http://", то вебстраница будет получена и выведена. Если файл начинается с "ftp://", то файл будет получен с ftp сервера и выведен. Для локальных файлов должен быть указан либо абсолютный путь, либо путь относительно выполняемого php файла.

Если указать специалнный атрибут "assign", то вывод функции fetch будет присвоен переменной вместо отображения. Добавлено в Smarty версии 1.5.0.

> **Техническое Замечание:** HTTP переадресация не поддерживается. Убедитесь, что указываете завершающие слэши, где это необходимо.

Техническое Замечание: Если включена security и указан файл из локальной файловой системы, то отобразятся лишь файлы, который находятся в указаных безопасных папках ($secure_dir).

Пример 8.5. fetch

```
{* включаем javascript в шаблон *}
{fetch file="/export/httpd/www.example.com/docs/navbar.js"}

{* Добавляем немного прогноза погоды с сервера погоды *}
{fetch file="http://www.myweather.com/68502/"}

{* новостную ленту берем с ftp сервера *}
{fetch file="ftp://user:password@ftp.example.com/path/to/currentheadlines.txt"}

{* присваиваем полученный файл переменной *}
{fetch file="http://www.myweather.com/68502/" assign="weather"}
{if $weather ne ""}
        &lt;b&gt;{$weather}&lt;/b&gt;
{/if}
```

html_checkboxes

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Нет	*checkbox*	название списка флажков
values	array	Да, если не указан атрибут options	*n/a*	Массив значений для флажков
output	array	Да, если не указан атрибут options	*n/a*	массив названий флажков
selected	string/array	Нет	*пусто*	выбранный флажок(флажки)
options	associative array	Да, если не указаны атрибуты values и output	*n/a*	Ассоциативнй массив значений и названий
separator	string	Нет	*пусто*	строка разделяющая каждый флажок
labels	boolean	Нет	*true*	добавляет <label>-тэги к выводу

Пользовательская функция html_checkboxes генерирует группу HTML флажков по указанной информации. Также заботится о флажках, которые выбраны по умолчанию. Параметры values и output являются обязательными, если не указан атрибут options. Весь вывод совместим с XHTML.

Все параметры, которые не указаны в списке, выводятся в виде пар name/value в каждом созданном тэге

<input>.

Пример 8.6. html_checkboxes

Где PHP код:

```
<?php

require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array('Joe Schmoe','Jack Smith','Jane Johnson','Charlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');

?>
```

и шаблон index.tpl:

```
{html_checkboxes name="id" values=$cust_ids selected=$customer_id output=$cust_names separator="<br />"}
```

или где PHP код:

```
<?php

require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_checkboxes', array(
                        1000 => 'Joe Schmoe',
                        1001 => 'Jack Smith',
                        1002 => 'Jane Johnson',
                        1003 => 'Charlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');
?>
```

и шаблон index.tpl:

```
{html_checkboxes name="id" options=$cust_checkboxes selected=$customer_id separator="<br />"}
```

оба примера выведут:

```
<label><input type="checkbox" name="id[]" value="1000" />Joe Schmoe</label><br />
<label><input type="checkbox" name="id[]" value="1001" checked="checked" />Jack Smith</label><br />
<label><input type="checkbox" name="id[]" value="1002" />Jane Johnson</label><br />
<label><input type="checkbox" name="id[]" value="1003" />Charlie Brown</label><br />
```

html_image

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
file	string	Да	*n/a*	название/путь к изображению
height	string	Нет	*реальная высота изображения*	высота изображения
width	string	Нет	*реальная ширина изображения*	ширина изображения
basedir	string	Нет	*корень веб сервера*	папка, от которой указаны относительные пути
alt	string	Нет	""	альтернативное описание изображения
href	string	Нет	*n/a*	значение href, куда ссылается картинка

Пользовательская функция html_image генерирует HTML для изображения. Ширина и высота автоматически вычисляются из файла изображения, если не указаны явно.

basedir - базовая папка для относительных путей. Если не указана, то используется корень веб сервер (переменная окружени DOCUMENT_ROOT). Если security включено, то путь к изображения должен быть в пределах безопасных папок.

Атрибут link указывает, куда ссылается изображение. Атрибут link устанавливает значение атрибута href тэга A. Если указан атрибут link, то изображение окружается выражениями <a href="LINKVALUE"> и <a>.

> **Техническое Замечание:** html_image требует обращение к диску для чтения изображения и вычисления его размеров. Если не используется кэширование шаблонов, то тогда лушче не пользоваться тэгом html_image и вставлять статические тэги изображений, для достижения оптимального быстродействия.

Пример 8.7. html_image

```
index.php:

require('Smarty.php.class');
$smarty = new Smarty;
$smarty->display('index.tpl');

index.tpl:

{html_image file="pumpkin.jpg"}
{html_image file="/path/from/docroot/pumpkin.jpg"}
{html_image file="../path/relative/to/currdir/pumpkin.jpg"}

OUTPUT: (possible)

<img src="pumpkin.jpg" width="44" height="68">
```

```
<img src="/path/under/docroot/pumpkin.jpg" width="44" height="68">
<img src="../path/relative/to/currdir/pumpkin.jpg" width="44" height="68">
```

html_options

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
values	массив	Да, если не указан атрибут options	*n/a*	массив значений для выпадающего списка
output	массив	Да, если не указан атрибут options	*n/a*	массив названий для выпадающего списка
selected	string/array	Нет	*пусто*	Выбранный элемент(ы)
options	ассоциативный массив	Да, если не указаны атрибуты values и output	*n/a*	ассоциативный массив значений и названий
name	string	Нет	*пусто*	Название выпадающего списка

пользовательская функция html_options генерирует группу html тэгов option по указанной информации. Также заботится о выбранных по умолчанию элементах. Атрибуты values и output обязательны, если не указан атрибут options.

Если данное значение - массив, то оно будет представлено в виде html OPTGROUP. Рекурсия с OPTGROUP поддерживается. Весь вывод совместим с XHTML.

Если указан необязательный атрибут *name*, то группа опций заключится в тэг <select name="groupname"> и </select>, иначе сгенерируется только группа опций.

Все параметры, которые не указаны выше, выводятся в виде пары name/value внутри тэга <select>. Они игнорируются, если атрибут *name* не указан.

Пример 8.8. html_options

```
ПРИМЕР 1
--------

index.php:

require('Smarty.php.class');
$smarty = new Smarty;
$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array('Joe Schmoe','Jack Smith','Jane
Johnson','Charlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');
```

```
index.tpl:

<select name=customer_id>
 {html_options values=$cust_ids selected=$customer_id output=$cust_names}
</select>

ПРИМЕР 2
--------

index.php:

require('Smarty.php.class');
$smarty = new Smarty;
$smarty->assign('cust_options', array(
   1001 => 'Joe Schmoe',
   1002 => 'Jack Smith',
   1003 => 'Jane Johnson',
   1004 => 'Charlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');

index.tpl:

<select name=customer_id>
 {html_options options=$cust_options selected=$customer_id}
</select>


РЕЗУЛЬТАТ: (оба примера)
-----------------------

<select name=customer_id>
 <option label="Joe Schmoe" value="1000">Joe Schmoe</option>
 <option label="Jack Smith" value="1001" selected="selected">Jack Smith</option>
 <option label="Jane Johnson" value="1002">Jane Johnson</option>
 <option label="Charlie Brown" value="1003">Charlie Brown</option>
</select>
```

html_radios

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Нет	*radio*	название элементов выбора
values	массив	Да, если не указан атрибут options	*n/a*	массив значений элементов выбора
output	массив	Да, если не указан атрибут options	*n/a*	массив названий элементов выбора
checked	string	Нет	*пусто*	Значение выбранного элемента

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
options	ассоциативный массив	Да, если не указаны атрибуты values и output	*n/a*	ассоциативный массив значений и названий элементов выбора
separator	string	Нет	*пусто*	текст, разделяющий элементы выбора

Пользовательсякая функция html_radios генерирует HMTL код группы элементов выбора (radio button group). Автоматически устанавливает выбранное значение, если оно укзано. Требует наличия атрибутов values и output или атрибута options. Сгенерированный HMTL код совместим с XHTML.

Все параметры, которые не указаны в таблице выше, передаются и выводятся внутри каждого созданного тэга <input>.

Пример 8.9. html_radios

index.php:

```
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array('Joe Schmoe','Jack Smith','Jane
Johnson','Charlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');
```

index.tpl:

```
{html_radios name="id" values=$cust_ids selected=$customer_id output=$cust_names separator="<br />"}
```

index.php:

```
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_radios', array(
                    1000 => 'Joe Schmoe',
                    1001 => 'Jack Smith',
                    1002 => 'Jane Johnson',
                    1003 => 'Charlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');
```

index.tpl:

```
{html_radios name="id" options=$cust_radios selected=$customer_id separator="<br />"}
```

OUTPUT: (both examples)

```
<label for="id_1000"><input type="radio" name="id" value="1000" id="id_1000" />Joe Schmoe</label><br />
<label for="id_1001"><input type="radio" name="id" value="1001" id="id_1001" checked="checked" />Jack Smith</label><br />
<label for="id_1002"><input type="radio" name="id" value="1002" id="id_1002" />Jane Johnson</label><br />
<label for="id_1003"><input type="radio" name="id" value="1003" id="id_1003" />Charlie Brown</label><br />
```

html_select_date

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
prefix	string	Нет	Date_	префикс названий переменных
time	timestamp/ГГ-ГГ-ММ-ДД	Нет	текущее время в формате unix timestamp или ГГГГ-ММ-ДД	используемое время
start_year	string	Нет	текущий год	Начальный год в выпадающем списке. Либо указывается явно, либо относительно текущего года (+/- N)
end_year	string	Нет	аналогично start_year	Конечный год в выпадающем списке. Либо указывается явно, либо относительно текущего года (+/- N)
display_days	boolean	Нет	true	выводить ли список дней
display_months	boolean	Нет	true	выводить ли список месяцев
display_years	boolean	Нет	true	выводить ли список лет
month_format	string	Нет	%B	Формат названия месяцев (strftime)
day_format	string	Нет	%02d	формат названия дней (sprintf)
day_value_format	string	Нет	%d	формат значения дней (sprintf)
year_as_text	boolean	Нет	false	Выводить ли значение года текстом
reverse_years	boolean	Нет	false	Выводить года в обратном порядке

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
field_array	string	Нет	null	название переменной (name), которая будет содержать выбранные значения в виде массива: name[Day], name[Year], name[Month].
day_size	string	Нет	null	Устанавливает атрибут size тэга select для дней
month_size	string	Нет	null	Устанавливает атрибут size тэга select для месяцев
year_size	string	Нет	null	Устанавливает атрибут size тэга select для лет
all_extra	string	Нет	null	Устанавливает дополнительные атрибуты для всех тэгов select/input
day_extra	string	Нет	null	Устанавливает дополнительные атрибуты тэгов select/input для дней
month_extra	string	Нет	null	Устанавливает дополнительные атрибуты тэгов select/input для месяцев
year_extra	string	Нет	null	Устанавливает дополнительные атрибуты тэгов select/input для лет
field_order	string	Нет	MDY	Порядок следования полей (МДГ)
field_separator	string	Нет	\n	текст, разделяющий поля
month_value_format	string	Нет	%m	формат значения месяца (strftime). По умолчанию - %m (номер месяца).

пользовательская функция html_select_date генерирует поля выбора даты.

Пример 8.10. html_select_date

```
{html_select_date}


ВЫВОД:

<select name="Date_Month">
<option value="1">January</option>
<option value="2">February</option>
<option value="3">March</option>
<option value="4">April</option>
<option value="5">May</option>
<option value="6">June</option>
<option value="7">July</option>
<option value="8">August</option>
<option value="9">September</option>
<option value="10">October</option>
<option value="11">November</option>
<option value="12" selected="selected">December</option>
</select>
<select name="Date_Day">
<option value="1">01</option>
<option value="2">02</option>
<option value="3">03</option>
<option value="4">04</option>
<option value="5">05</option>
<option value="6">06</option>
<option value="7">07</option>
<option value="8">08</option>
<option value="9">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13" selected="selected">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
```

```
</select>
<select name="Date_Year">
<option value="2001" selected="selected">2001</option>
</select>
```

Пример 8.11. html_select_date

```
{* год начала и конца могут быть заданы относительно текущего года *}
{html_select_date prefix="StartDate" time=$time start_year="-5" end_year="+1" display_days=false}

ВЫВОД: (текущий год - 2000)

<select name="StartDateMonth">
<option value="1">January</option>
<option value="2">February</option>
<option value="3">March</option>
<option value="4">April</option>
<option value="5">May</option>
<option value="6">June</option>
<option value="7">July</option>
<option value="8">August</option>
<option value="9">September</option>
<option value="10">October</option>
<option value="11">November</option>
<option value="12" selected="selected">December</option>
</select>
<select name="StartDateYear">
<option value="1995">1995</option>
<option value="1996">1996</option>
<option value="1997">1997</option>
<option value="1998">1998</option>
<option value="1999">1999</option>
<option value="2000" selected="selected">2000</option>
<option value="2001">2001</option>
</select>
```

html_select_time

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
prefix	string	Нет	Time_	префикс названий переменных
time	timestamp	Нет	текущее время	используемое время
display_hours	boolean	Нет	true	выводить часы
display_minutes	boolean	Нет	true	выводить минуты
display_seconds	boolean	Нет	true	выводить секунды
display_meridian	boolean	Нет	true	выводить меридиан

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				(am/pm)
use_24_hours	boolean	Нет	true	использовать 24-часовой формат времени
minute_interval	integer	Нет	1	интервал элементов выпадающего списка минут
second_interval	integer	Нет	1	интервал элементов выпадающего списка секунд
field_array	string	Нет	n/a	название переменной, в которую передадутся выбранные значения в виде массива
all_extra	string	Нет	null	указывает дополнительные атрибуты для всех тэгов select/input
hour_extra	string	Нет	null	указывает дополнительные атрибуты для тэгов select/input для выбора часов
minute_extra	string	Нет	null	указывает дополнительные атрибуты для тэгов select/input для выбора минут
second_extra	string	Нет	null	указывает дополнительные атрибуты для тэгов select/input для выбора секунд
meridian_extra	string	Нет	null	указывает дополнительные атрибуты для тэгов select/input для выбора меридиан

Пользовательская функция html_select_time генерирует HTML поля для выбора времени. Она может выборочно отображать поля для часов, минут, секунд и меридиана.

Аттрибут time имеет несколько разных форматов: он моэет быть представлен в виде уникальной временной метки (UNIX timestamp) или строки в формате ГГГГММДДЧЧММСС или любо другой строки, которую

может обработать PHP-функция strtotime().

Пример 8.12. html_select_time

```
{html_select_time use_24_hours=true}


ВЫВОД:

<select name="Time_Hour">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09" selected="selected">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
</select>
<select name="Time_Minute">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
```

```
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20" selected="selected">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
<option value="32">32</option>
<option value="33">33</option>
<option value="34">34</option>
<option value="35">35</option>
<option value="36">36</option>
<option value="37">37</option>
<option value="38">38</option>
<option value="39">39</option>
<option value="40">40</option>
<option value="41">41</option>
<option value="42">42</option>
<option value="43">43</option>
<option value="44">44</option>
<option value="45">45</option>
<option value="46">46</option>
<option value="47">47</option>
<option value="48">48</option>
<option value="49">49</option>
<option value="50">50</option>
<option value="51">51</option>
<option value="52">52</option>
<option value="53">53</option>
<option value="54">54</option>
<option value="55">55</option>
<option value="56">56</option>
<option value="57">57</option>
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Second">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
```

```
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23" selected="selected">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
<option value="32">32</option>
<option value="33">33</option>
<option value="34">34</option>
<option value="35">35</option>
<option value="36">36</option>
<option value="37">37</option>
<option value="38">38</option>
<option value="39">39</option>
<option value="40">40</option>
<option value="41">41</option>
<option value="42">42</option>
<option value="43">43</option>
<option value="44">44</option>
<option value="45">45</option>
<option value="46">46</option>
<option value="47">47</option>
<option value="48">48</option>
<option value="49">49</option>
<option value="50">50</option>
<option value="51">51</option>
<option value="52">52</option>
<option value="53">53</option>
<option value="54">54</option>
<option value="55">55</option>
<option value="56">56</option>
<option value="57">57</option>
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Meridian">
<option value="am" selected="selected">AM</option>
<option value="pm">PM</option>
</select>
```

html_table

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
loop	array	Да	*n/a*	массив данных, по которому будет произведен обход
cols	integer	Нет	*3*	количество колонок в таблице. если атрибут cols не указан, но указан rows, тогда кол-во колонок вычисляется исходя из кол-ва строк и кол-ва элементов, которые необходимо отобразить. если оба атрибута, rows и cols, не указаны, cols по умолчанию равен 3.
rows	integer	Нет	*empty*	количество строк в таблице. если атрибут rows не указан, но указан cols, тогда кол-во строк вічисляется исходя из кол-ва колонок и кол-ва элементов, которые необходимо отобразить.
inner	string	No	*cols*	направление последовательного отображения элементов из массива loop. *cols* означает, что элементы отображаются колонка за колонкой. *rows* означает, что элементы отображаются строка за строкой.
table_attr	string	Нет	*border="1"*	дополнительные атрибуты тэга table
tr_attr	string	Нет	*пусто*	дополнительные атрибуты тэга tr (если указан массив, то его элементы ци-

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				клически повторяются)
td_attr	string	Нет	*пусто*	дополнительные атрибуты тэга td (если указан массив, то его элементы циклически повторяются)
trailpad	string	Нет	* *	значение остаточных ячеек на последней строке табилцы
hdir	string	Нет	*right*	направление отображения каждого ряда. допустимые значения: *left* (слева направо), *right* (справа налево)
vdir	string	Нет	*down*	направление отображение каждой колонки. допустимые значения: *down* (сверху вниз), *up* (снизу вверх)

Пользовательская функция *html_table* выводит массив в виде HTML таблицы. Атрибут *cols* указывает количество колонок. Атрибуты *table_attr*, *tr_attr* и *td_attr* указывают дополнительные атрибуты тэгов table, tr и td. Если значение *tr_attr* или *td_attr* - массив, то кго значения циклический повторяются. Атрибут *trailpad* устанавливает значения для остаточных ячеек на последней строке таблицы.

Пример 8.13. html_table

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('data',array(1,2,3,4,5,6,7,8,9));
$smarty->assign('tr',array('bgcolor="#eeeeee"','bgcolor="#dddddd"'));
$smarty->display('index.tpl');
?>
```

```
{html_table loop=$data}
{html_table loop=$data cols=4 table_attr='border="0"'}
{html_table loop=$data cols=4 tr_attr=$tr}
```

Результат выполнения данного примера:

```
<table border="1">
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td><td>6</td></tr>
<tr><td>7</td><td>8</td><td>9</td></tr>
</table>
<table border="0">
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr><td>9</td><td> </td><td> </td><td> </td></tr>
</table>
<table border="1">
<tr bgcolor="#eeeeee"><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr bgcolor="#dddddd"><td>5</td><td>6</td><td>7</td><td>8</td></tr>
<tr bgcolor="#eeeeee"><td>9</td><td> </td><td> </td><td> </td></tr>
</table>
```

mailto

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
address	string	Да	*n/a*	адрес e-mail
text	string	Нет	*n/a*	название ссылки. По умолчанию: адрес e-mail
encode	string	Нет	*none*	Способ кодирования e-mail. Может быть none, hex, javascript или javascript_charcode.
cc	string	Нет	*n/a*	адреса e-mail для точной копии. Адреса разделяются запятыми.
bcc	string	Нет	*n/a*	адреса e-mail для "слепой" копии. Адреса разделяются запятыми.
subject	string	Нет	*n/a*	тема письма.
newsgroups	string	Нет	*n/a*	в какие конференции передавать. конференции разделяются запятыми.
followupto	string	Нет	*n/a*	адреса для дальнейшего перенаправления. Адреса разделяются запятыми.
extra	string	Нет	*n/a*	Дополнительный

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				атрибуты, передаваемые в ссылку такие как стили (style)

пользовательская функция mailto автоматизирует создание ссылок на e-mail адреса с возможностью кодирования их. Кодирование усложняет работу web-пауков, которые собирают e-mail адреса с вашего сайта.

Техническое Замечание: javascript - скорее всего наиболее полная форма кодирования, хотя вы так же можете использовать шестнадцатиричное кодирование.

Пример 8.14. mailto

```
{mailto address="me@example.com"}
{mailto address="me@example.com" text="send me some mail"}
{mailto address="me@example.com" encode="javascript"}
{mailto address="me@example.com" encode="hex"}
{mailto address="me@example.com" subject="Hello to you!"}
{mailto address="me@example.com" cc="you@example.com,they@example.com"}
{mailto address="me@example.com" extra='class="email"'}
{mailto address="me@example.com" encode="javascript_charcode"}

OUTPUT:

<a href="mailto:me@example.com" >me@example.com</a>
<a href="mailto:me@example.com" >send me some mail</a>
<script type="text/javascript" language="javascript">eval(unescape('%64%6f%63%75%6d%65%6e%74%2e%77%72%6
9%74%65%28%27%3c%61%20%68%72%65%66%3d%22%6d%61%69%6c%74%6f%3a%6d%65%40%64%6f%6d%
61%69%6e%2e%63%6f%6d%22%20%3e%6d%65%40%64%6f%6d%61%69%6e%2e%63%6f%6d%3c%2f%61%3e
%27%29%3b'))</script>
<a href="mailto:%6d%65@%64%6f%6d%61%69%6e.%63%6f%6d" >&#x6d;&#x65;&#x40;&#x64;&
#x6f;&#x6d;&#x61;&#x69;&#x6e;&#x2e;&#x63;&#x6f;&#x6d;</a>
<a href="mailto:me@example.com?subject=Hello%20to%20you%21" >me@example.com</a>
<a href="mailto:me@example.com?cc=you@example.com%2Cthey@example.com" >me@example.com</a>
<a href="mailto:me@example.com" class="email">me@example.com</a>
<script type="text/javascript" language="javascript">
<!--
{document.write(String.fromCharCode(60,97,32,104,114,101,102,61,34,109,97,105,108,116,111,58,109,101,64,101,120,97,109,112,108,101,46,99,111,10
//-->
</script>
```

math

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
equation	string	Да	*n/a*	математической выражение
format	string	Нет	*n/a*	формат рузультата (sprintf)

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
var	numeric	Да	*n/a*	переменная выражения
assign	string	Нет	*n/a*	переменная шаблона, которой будет присвоен вывод
[var ...]	numeric	Да	*n/a*	дополнительные переменные выражения

пользовательская функция math позволяет дизайнерам шаблонов вычислять математические выражения в шаблоне. Любая численная переменная шаблона может быть использована в выражении. Переменные, используемые в выражении, передаются в качестве параметров, которыми могут быть как и переменные шаблона, так и статические значения. Допустимые операторы: +, -, /, *, abs, ceil, cos, exp, floor, log, log10, max, min, pi, pow, rand, round, sin, sqrt, srans и tan. Ознакомтесь с php документацией под данным функциям.

Если указан атрибут "assign", то вывод будет присвоен переменной, вместо отображения.

> **Техническое Замечание:** использование функции math значительно сказывается на времени выполнения программы, так как реализована с помощью php функции eval(). Выполнение математических операций в php программе более эффективно, то есть везде, где возможно, следует делать вычисления в программе и передавать результат в шаблон. Следует также избегать повторяющегося вызова функции math (например, в циклах).

Пример 8.15. math

```
{* $height=4, $width=5 *}

{math equation="x + y" x=$height y=$width}

OUTPUT:

9


{* $row_height = 10, $row_width = 20, #col_div# = 2, assigned in template *}

{math equation="height * width / division"
     height=$row_height
     width=$row_width
     division=#col_div#}

OUTPUT:

100


{* можно использовать скобки *}

{math equation="(( x + y ) / z )" x=2 y=10 z=2}
```

```
OUTPUT:

6


{* можно указать формат результата (sprintf) *}

{math equation="x + y" x=4.4444 y=5.0000 format="%.2f"}

OUTPUT:

9.44
```

popup

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
text	string	Да	*n/a*	HTML код (текст), который будет выводиться в высплывающем окне.
trigger	string	Нет	*onMouseOver*	Способ вызова окна. Может быть либо onMouseOver, либо onclick
sticky	boolean	Нет	*false*	Makes the popup stick around until closed
caption	string	Нет	*n/a*	устанавливает заголовок
fgcolor	string	Нет	*n/a*	цвет внутри окна
bgcolor	string	Нет	*n/a*	цвет границы окна
textcolor	string	Нет	*n/a*	цвет текста в окне
capcolor	string	Нет	*n/a*	цвет заголовка окна
closecolor	string	Нет	*n/a*	цвет текста "Close" (Закрыть)
textfont	string	Нет	*n/a*	шрифт текста в окне
captionfont	string	Нет	*n/a*	шрифт заголовка окна
closefont	string	Нет	*n/a*	шрифт текста "Close" (Закрыть)
textsize	string	Нет	*n/a*	размер шрифта в окне

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
captionsize	string	Нет	*n/a*	размер шрифта заголовка
closesize	string	Нет	*n/a*	размер шрифта текста "Close" (Закрыть)
width	integer	Нет	*n/a*	ширина окна
height	integer	Нет	*n/a*	высота окна
left	boolean	Нет	*false*	создавать окно слева от курсора мыши
right	boolean	Нет	*false*	создавать окно справа от курсора мыши
center	boolean	Нет	*false*	создавать окно на месте мыши
above	boolean	Нет	*false*	создает окно выше курсора мыши. ЗАМЕЧАНИЕ: возможно только если указан атрибут height
below	boolean	Нет	*false*	создает окно под курсором мыши
border	integer	Нет	*n/a*	толщина границы окна
offsetx	integer	Нет	*n/a*	горизонтальное смещение окна от курсора мыши.
offsety	integer	Нет	*n/a*	вертикальное смещение окна от курсора мыши
fgbackground	url to image	Нет	*n/a*	определяет фоновое изображение текста, вместо fgcolor.
bgbackground	url to image	Нет	*n/a*	определяет фоновое изображение для границ окна, вместо bgcolor. ЗАМЕЧАНИЕ: Может понадобиться установить bgcolor в "" или будет показан цвет, а не зиображение. ЗАМЕЧАНИЕ: При наличии ссылки "Close" Netscape перерисовывает ячейки таблицы, что

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				приводит к некорректному отображению.
closetext	string	Нет	*n/a*	устанвливает текст вместо "Close"
noclose	boolean	Нет	*n/a*	не отображать текст "Close"
status	string	Нет	*n/a*	установить значение строки статуса в браузере
autostatus	boolean	Нет	*n/a*	установить значение строки статуса в браузере в текст окна. ЗАМЕЧАНИЕ: отменяет значение status
autostatuscap	string	Нет	*n/a*	установить значение строки статуса в браузере в текст заголовка. ЗАМЕЧАНИЕ: отменяет начение autostatus
inarray	integer	Нет	*n/a*	указывает, что текст окна следует взять из указанного элемента массива ol_text, расположенного в overlib.js. Этот параметр может использоваться вместо text
caparray	integer	Нет	*n/a*	указывает, что заголовок окна следует взять из указанного элемента массива ol_caps
capicon	url	Нет	*n/a*	выводит изображение перед заголовком окна.
snapx	integer	Нет	*n/a*	ровняет окно к горизонтальной сетке
snapy	integer	Нет	*n/a*	ровняет окно к вертикальной сетке
fixx	integer	Нет	*n/a*	закрепляет горизонтальное положение

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				окна. ЗАМЕЧАНИЕ: отменяет все остальные параметры горизонтального положения
fixy	integer	Нет	*n/a*	закрепляет вертикалнное положение окна. ЗАМЕЧАНИЕ: отменяет все остальные параметры вертикального положения
background	url	Нет	*n/a*	указывает фоновое изображение окна
padx	integer,integer	Нет	*n/a*	дополняет фоновое изображение горизонтальными отступами к тексту. ЗАМЕЧАНИЕ: этот параметр принимает два значения
pady	integer,integer	Нет	*n/a*	дополняет фоновое изображение вертикальными отступами к тексту. ЗАМЕЧАНИЕ: этот параметр принимает два значения
fullhtml	boolean	Нет	*n/a*	Позволяет полностью контролировать HTML над фоновым изображением. HTML код ожидается в параметре text
frame	string	Нет	*n/a*	Контролировать выспливающие окна в различных фрэймах. См. сайт overlib для дополнительной информации по этой функции
timeout	string	Нет	*n/a*	вызывает указанную javascript функцию и использует результат как текст окна

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
delay	integer	Нет	*n/a*	время жизни окна в милисекундах. Позволяет реализовывать высплывающие подсказки.
hauto	boolean	Нет	*n/a*	автоматическое опредление горизонтального местоположения окна относительно мыши.
vauto	boolean	Нет	*n/a*	автоматическое опредление вертикального местоположения окна относительно мыши.

функция popup используется для генерации javascript кода, который создаст высплывающее окно.

Пример 8.16. popup

```
{* popup_init должен быть вызван один раз в начале страницы *}
{popup_init src="/javascripts/overlib.js"}

{* создаем ссылку с всплывающим окном, которое появляеться при наведении мыши *}
<a href="mypage.html" {popup text="This link takes you to my page!"}>mypage</A>

{* можно использовать html, ссылки и т.п. в высплывающем окне *}
<a href="mypage.html" {popup sticky=true caption="mypage contents"
text="<ul><li>links</li><li>pages</li><li>images</li></ul>" snapx=10 snapy=10}>mypage</A>
```

popup_init

функция popup реализует интеграцию с библиотекой overLib, которая используется для создания всплывающих окон. Они могут использоваться для вывода контекстно-зависимой информации, такой как контекстная помощь или всплывающие подсказки. Функция popup_init должна быть вызвана один раз в начале страницы, где планируется вызов функции popup. Библиотеку overLib написал Эрик Босрап (Erik Bosrup). Домашняя страница расположена по адресу http://www.bosrup.com/web/overlib/.

Начиная со Smarty версии 2.1.2, библиотека overLib не включается в релиз. Скачайте библиотеку overLib, поместите файл overlib.js в корень документов (DOCUMENT_ROOT) или глубже. При вызове функции popup_init передайте относительный путь к этому файлу в качестве параметра src.

Пример 8.17. popup_init

```
{* popup_init должен быть вызван один раз в начале страницы *}
 {popup_init src="/javascripts/overlib.js"}
```

textformat

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
style	string	Нет	*n/a*	предустановленный стиль
indent	number	Нет	*0*	отступ строки
indent_first	number	Нет	*0*	отступ первой строки
indent_char	string	Нет	*(single space)*	символ, которым заполняется отступ строк.
wrap	number	Нет	*80*	количество символов в строке
wrap_char	string	Нет	*\n*	текст, разделяющий каждую строку
wrap_cut	boolean	Нет	*false*	Переносить текст по символам (то есть точно по указанной длине строки) (true), или по границам слов (false)
assign	string	Нет	*n/a*	переменная шаблона, которой будет присвоен вывод

функция textformat используется для форматирования текста, заключенного внтури ее. В основном, убирает лишние пробелы и специальные символы, а так же форматирует абзацы, делает отступы, переносит слова.

Можно указывать параметры явно, или использовать предустановленные стили. На данный момент существует только стиль "email".

Пример 8.18. textformat

```
{textformat wrap=40}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
```

```
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}
```

OUTPUT:

```
This is foo. This is foo. This is foo.
This is foo. This is foo. This is foo.

This is bar.

bar foo bar foo foo. bar foo bar foo
foo. bar foo bar foo foo. bar foo bar
foo foo. bar foo bar foo foo. bar foo
bar foo foo. bar foo bar foo foo.


{textformat wrap=40 indent=4}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}
```

OUTPUT:

```
    This is foo. This is foo. This is
    foo. This is foo. This is foo. This
    is foo.

    This is bar.
```

```
   bar foo bar foo foo. bar foo bar foo
   foo. bar foo bar foo foo. bar foo
   bar foo foo. bar foo bar foo foo.
   bar foo bar foo foo. bar foo bar
   foo foo.

{textformat wrap=40 indent=4 indent_first=4}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}

OUTPUT:

       This is foo. This is foo. This
   is foo. This is foo. This is foo.
   This is foo.

       This is bar.

       bar foo bar foo foo. bar foo bar
   foo foo. bar foo bar foo foo. bar
   foo bar foo foo. bar foo bar foo
   foo. bar foo bar foo foo. bar foo
   bar foo foo.

{textformat style="email"}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
```

```
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}

OUTPUT:

This is foo. This is foo. This is foo. This is foo. This is foo. This is
foo.

This is bar.

bar foo bar foo foo. bar foo bar foo foo. bar foo bar foo foo. bar foo
bar foo foo. bar foo bar foo foo. bar foo bar foo foo. bar foo bar foo
foo.
```

Глава 9. Конфигурационные файлы

С помощью конфигурационных файлов дизайнеру удобно управлять глобальными переменными из одного файла. Например, цветами в шаблонах. Обычно, если вы хотите сменить цветувую схему, то необходимо просмотреть каждый шаблон и в каждом изменить цвета. С помощью файла конфигурации все цвета могут быть вынесены в отдельный файл и только один файл надо будет исправлять.

Пример 9.1. Пример файла конфигурации

```
# глобальные переменные
pageTitle = "Main Menu"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

[Customer]
pageTitle = "Customer Info"

[Login]
pageTitle = "Login"
focus = "username"
Intro = """Значение, которое занимает больше
        чем одну строку должно быть заключено
        в тройные кавычки."""

# спрятанная секция
[.Database]
host=my.example.com
db=ADDRESSBOOK
user=php-user
pass=foobar
```

Значения переменных в конфигурационных файлах могут заключаться в кавычки, но это не обязательно. Можно использовать как двойные, так и одинарные кавычки. Если у вас есть значение, которое занимает больше, чем одну строку, необходимо заключить его в тройные кавычки (""""). Можно включать комментарии в файл конфигурации используя любой синтакис, который не является допустимым синтаксисом файлов конфигурации. Для этих целей рекомендуется использовать символ # (hash) в начале строки.

Конфигурационный файл в примере имеет две секции. Названия секций заключены в квадратные скобки []. Названия секций могут быть произвольными строками, не содержащими символов [или]. Четыре переменные вначале - глобальные переменные или переменные вне секций. Эти переменные всегда загружаются из файла конфигурации. Если загружается определенная секция, то глобальные переменные и переменные из этой секции становятся доступными. Если переменная существует как глобальная, так и внутри секции, то используется версия из секции. Если есть две одинаковые переменные в пределах одной секции, то используеться последний встретившийся вариант.

Файлы конфигурации загружаются в шаблон в помощью функции **{config_load}** (см. также **config_load()**).

Можно спрятать отдельные переменные или целые секции, добавив к названию точку в начале. Это полез-

но, когда ваше приложение берет некоторые переменные, ненужные в шаблоне, из файла конфигурации. Если шаблоны могут редактировать третьи лица, то вы можете быть спокойны за ценную информацию из файлов конфигураций: они не смогут ее загрузить в шаблон.

См. также {config_load}, $config_overwrite, get_config_vars(), clear_config() и config_load()

Глава 10. Отладочная консоль

В Smarty включена консоль для отладки. Консоль позволяет узнать все включенные шаблоны, присвоенные переменные и настройки из конфинурационных файлов для текущего экземпляра Smarty. Шаблон "debug.tpl", поставляемый вместе со Smarty, управляет видом консоли. Установите опцию Smarty $debugging в true и, если необходимо, укажите в $debug_tpl путь к шаблону debug.tpl (по умолчанию это SMARTY_DIR). Когда вы загружаете страницу, должно появиться всплывающие окно и вывести консоль отладки. Для вывода доступных переменных из конкретного шаблона, см. функцию {debug}. Для отключения консоли отладки, установите $debugging в false. Можно также опционально включить консоль отладки, добавив SMARTY_DEBUG в URL, если включена опция $debugging_ctrl.

> **Техническое Примечание:** Консоль отладки не работает, когда используется функция API fetch(). Необходимо использовать только функцию display(). Она генерирует javascript код вначале каждой сгенерированной страницы. Если вам не нравится javascript, можно отредатировать debug.tpl для изменения способа отображения по вашему вкусу. Отладочная информация не кэшируется и в отладочную информацию не включается информация о debug.tpl.

> **Замечание:** Время загрузки каждого шаблона и файла конфигурации выводятся в секундах или в миллисекундах.

См. также Решение проблем, $error_reporting и trigger_error().

Часть III. Smarty для программистов

Содержание

Глава 11. Константы

Содержание

SMARTY_DIR

Эта константа должна содержать полный путь к файлам Smarty. Если константа не определена, Smarty будет пытаться определить путь самостоятельно. При определении данной константы, слэш в конце строки обязателен.

Пример 11.1. SMARTY_DIR

```
<?php
// установка пути к директории Smarty
define("SMARTY_DIR","/usr/local/lib/php/Smarty/");

require_once(SMARTY_DIR."Smarty.class.php");
?>
```

SMARTY_CORE_DIR

Это полный путь к файлам ядра (core) Smarty. Если он не установлен, Smarty будет использовать значение по умолчанию - путь к поддиректории internals/ директории SMARTY_DIR. Если константа определена, путь должен заканчиваться слэшем. Используйте эту константу, когда вручную подключаете любой из core.* файлов.

Пример 11.2. SMARTY_CORE_DIR

```
<?php
// загрузка core.get_microtime.php

require_once(SMARTY_CORE_DIR."core.get_microtime.php");
?>
```

Глава 12. Переменные

Содержание

$template_dir

Это название директории шаблонов по умолчанию. Если вы не передадите тип ресурса во время подключения файлов, они будут искаться здесь. Значение по умолчанию - "./templates", а это значит, что движок бу-

дет искать шаблоны в поддиректории templates той директории, в которой выполняется php-скрипт.

Техническое замечание: Не рекоммендуется размещать эту директорию в пределах корневой директории веб-сервера.

$compile_dir

Имя каталога, в котором хранятся компилированные шаблоны. По умолчанию установлено в "./templates_c", т.е. поиск каталога с компилированными шаблонами будет производиться в том же каталоге, в котором выполняется скрипт.

Техническое замечание: При установке этого параметра можно использовать как относительные, так и абсолютные пути. Для создаваемых файлов include_path не используется.

Техническое замечание: Не рекомендуется помещать этот каталог внутри корневого каталога документов веб-сервера.

$config_dir

Каталог для хранения конфигурационных файлов, используемых в шаблонах. По умолчанию установлено в "./configs", т.е. поиск каталога с конфигурационными файлами будет производиться в том же каталоге, в котором выполняется скрипт.

Техническое замечание: Не рекомендуется помещать этот каталог внутри корневого каталога документов веб-сервера.

$plugins_dir

Это директория (или директории), в которых Smarty будет искать необходимые ему плагины. По умолчанию это поддиректория "plugins" директории SMARTY_DIR. Если вы укажете относительный путь, Smarty будет в первую очередь искать относительно SMARTY_DIR, затем оносительно текущей рабочей директории (cwd, current working directory), а затем относительно каждой директории в PHP-директиве include_path. Если $plugins_dir является массивом директорий, Smarty будет искать ваш плагин в каждой директории плагинов в том порядке, в котором они указаны.

Техническое замечание: Для улучшения производительности, не нужно настраивать plugins_dir для использования include_path. Используйте абсолютные пути или относительные пути от SMARTY_DIR или текущей рабочей директории.

$debugging

Активирует debugging console - порожденное при помощи javascript окно браузера, содержащее информацию о подключенных шаблонах и загруженных переменных для текущей страницы.

$debug_tpl

Имя файла шаблона, используемого для панели отладки (debugging console). По умолчанию это файл debug.tpl, расположенный в SMARTY_DIR.

$debugging_ctrl

Позволяет активировать режим отладки альтернативными путями. Значение NONE запрещает использовать альтернативные методы. При значении переменной URL, режим отладки будет активирован для данного вызова скрипта в случае, если в QUERY_STRING будет обнаружено ключевое слово SMARTY_DEBUG. Этот параметр игнорируется, если $debugging установлено в true.

$autoload_filters

При необходимости загрузки при каждом вызове шаблонов некоторого количества фильтров, вы можете определить их, используя эту переменную, и Smarty автоматически их загрузит. Переменная представляет из себя ассоциативный массив, ключи в котором являются типами фильтров, а значения - массивами имен фильтров. Например:

```
<?php
$smarty->autoload_filters = array('pre' => array('trim', 'stamp'),
                                  'output' => array('convert'));
?>
```

$compile_check

При каждом вызове PHP-приложения Smarty проверяет, изменился или нет текущий шаблон с момента последней компиляции. Если шаблон изменился, он перекомпилируется. В случае, если шаблон еще не был скомпилирован, его компиляция производится с игнорированием значения этого параметра. По умолчанию эта переменная установлена в true. В момент, когда приложение начнет работать в реальных условиях (шаблоны больше не будут изменяться), этап проверки компиляции становится ненужным. В этом случае проверьте, чтобы переменная $compile_check была установлена в "false" для достижения максимальной производительности. Учтите, что если вы присвоите этой переменной значение "false", и файл шаблона будет изменен, вы *НЕ* увидите изменений в выводе шаблона до тех пор, пока шаблон не будет перекомпилирован. Если caching и compile_check активированы, файлы кэша будут регенерированы при обновлении связанных с ним шаблонов или конфигурационных файлов. См. $force_compile или clear_compiled_tpl.

$force_compile

Указывает Smarty (пере)компилировать шаблоны при каждом вызове. Этот параметр перекрывает действие $compile_check и по умолчанию не активирован. Действие параметра удобно использовать в процессе разработки и отладки, однако никогда не используйте его в условиях реальной эксплуатации: если кэширование активировано, файл(ы) кэша будут каждый раз перезаписываться.

$caching

Сообщает Smarty, будет или нет кэшироваться вывод шаблонов. По умолчанию этот параметр установлен в 0, т.е. не активирован. Если ваши шаблоны генерируют большие объемы кода, рекомендуется активировать кэширование - это даст ощутимый прирост в производительности. Вы также можете использовать множественный кэш шаблонов. Значение 1 или 2 активирует кэширование. При задании значения 1, для определения времени жизни кэша используется текущее значение переменной $cache_lifetime. Значение 2 задает Smarty использовать значение cache_lifetime во время окончания генерации кэша. В этом случае вы можете

устанавливать cache_lifetime непосредственно перед обработкой шаблона для осуществления гибкого контроля за истечением времени жизни конкретного экземпляра кэша. См. также is_cached.

Если параметр $compile_check активирован, кэш будет обновляться в случае, когда любой из шаблонов или конфигурационных файлов, являющихся частью этого кэша, был изменен. Если активирован $force_compile, кэш будет обновляться во всех случаях.

$cache_dir

Имя каталога, в котором хранится кэш шаблонов. По умолчанию установлено в "./cache". Это означает, что поиск каталога с кэшем будет производиться в том же каталоге, в котором выполняется скрипт. Вы также можете использовать собственную функцию-обработчик для управления файлами кэша, которая будет игнорировать этот параметр.

> **Техническое замечание:** При установке этого параметра можно использовать как относительные, так и абсолютные пути. Для создаваемых файлов include_path не используется.

> **Техническое замечание:** Не рекомендуется помещать этот каталог внутри корневого каталога документов веб-сервера.

$cache_lifetime

Задает длительность времени в секундах, в течение которого кэш шаблона будет актуальным. По истечении этого времени кэш будет регенерирован. Переменная $caching должна быть установлена в "true" при использовании $cache_lifetime. Значение переменной -1 задает неограниченное время жизни кэша. Значение переменной 0 вызовет постоянную его регенерацию (подходит только для тестирования, для отключения кэширования более целесообразно устанавливать $caching = false.)

Если $force_compile активировано, файлы кэша каждый раз будут регенерироваться, отключая таким образом кэширование. Вы можете очистить сразу все файлы кэша, используя функцию clear_all_cache(), или в случае с конкретными файлами (группами) кэша - при помощи функции clear_cache().

> **Техническое замечание:** Если вы хотите назначить конкретным шаблонам собственное время жизни их кэша, вы можете сделать это путем установки $caching = 2, затем установкой $cache_lifetime в нужное значение перед вызовом display() или fetch().

$cache_handler_func

Вы можете добавить собственную функцию для управления файлами кэша вместо вызовов встроенного метода, используя $cache_dir. За дополнительной информацией обратитесь к разделу Управление кэшированием.

$cache_modified_check

Если установлено в true, Smarty будет учитывать If-Modified-Since заголовок, посланный клиентом. Если время создания кэшированного файла не изменилось с момента последнего посещения, то взамен его содержимого будет послан заголовок "304 Not Modified". Это работает только в случае, если кэшированное содержимое не содержит тэгов **insert**.

$config_overwrite

Если установлено в true, переменные, полученные из конфигурационных файлов, будут перекрывать все остальные. В лбом случае, перемнные будут помещены в массив. Это удобно, когда вы хотите хранить массив данных в конфигурационном файле - просто задавайте каждый элемент много раз. По умолчанию true.

$config_booleanize

Если установлено в true, значения параметров конфигурационных файлов on/true/yes и off/false/no будут конвертированы в булевы значения автоматически. В этом случае вы можете использовать в шаблоне конструкции, подобные этой: {if #foobar#} ... {/if}. Если foobar равно on, true или yes, будет осуществлен переход по условию {if}. По умолчанию равно true.

$config_read_hidden

Если установлено в true, скрытые разделы (имеющие имя, начинающиеся с точки) в конфигурационных файлах могут быть доступными из шаблонов. Как правило, следует устанавливать значение этого параметра в false: в этом случае вы можете хранить важные данные (например, параметры подключения к базе данных) в конфигурационных файлах, не беспокоясь, что они будут загружены в шаблон. По умолчанию false.

$config_fix_newlines

Если установлено в true, переводы строк в стиле mac и dos (\r и \r\n) в конфигурационных файлах будут конвертированы в \n при синтаксическом разборе. По умолчанию равно true.

$default_template_handler_func

Функция, вызываемая в случае, если шаблон не был получен из своего источника.

$php_handling

Этот параметр говорит Smarty, как обращаться с PHP-кодом, встроенным в шаблоны. Существует четыре возможных значения, значением по умолчанию является SMARTY_PHP_PASSTHRU. Обратите внимание, что это НЕ влияет на PHP-код внутри тэгов {php}{/php} в шаблоне.

- SMARTY_PHP_PASSTHRU - Smarty показывает тэги без обработки.
- SMARTY_PHP_QUOTE - Smarty превращает спецсимволы тэгов в HTML-сущности.
- SMARTY_PHP_REMOVE - Smarty удаляет тэги из шалона.
- SMARTY_PHP_ALLOW - Smarty будет выполнять теги как PHP-код.

Замечание: Встраивать PHP-код в шаблоны весьма не рекоммендуется. Вместо этого, используется пользовательские функции или модификаторы.

$security

$security true/false, по умолчанию false. Безопасность (security) полезна в ситуациях, когда ваши шаблоны редактируют лица, не заслуживающе вашего доверия (например, через ftp) и вы хотите сократить риски взлома системы с помощью языка шаблонов. Включение безопасного режима накладывает следующие ограничения на язык шаблонов, если только они не изменяются параметром $security_settings:

- Если $php_handling установлен в SMARTY_PHP_ALLOW, это неявно меняет его на SMARTY_PHP_PASSTHRU
- PHP-функции запрещены в условиях IF, кроме тех, которые указаны в $security_settings
- Шаблоны могут быть подключены только из директорий, перечисленных в массиве $secure_dir
- Локальные файлы могут быть прочитаны при помощи {fetch} только из директорий, перечисленных в массиве $secure_dir
- Тэги {php}{/php} запрещены
- PHP-функции запрещено использовать в виде модификаторов, кроме тех, которые указаны в $security_settings

$secure_dir

Это массив всех локальных директори, которые считаются безопасными. {include} и {fetch} используют этот параметр при влюченном безопасном режиме.

$security_settings

Это используется для изменения или указания настроек безопасности когда безопасносить (security) включена. Допустимые значения:

- PHP_HANDLING - true/false. Если установлено в true, параметр $php_handling не проверяется на безопасность.
- IF_FUNCS - Это массив имён PHP-функций, разрешенных к использованию в условиях IF.
- INCLUDE_ANY - true/false. Если установлено в true, любой шаблон может быть подключен из файловой системы, независимо от списка $secure_dir.
- PHP_TAGS - true/false. Если установлено в true, тэги {php}{/php} разрешены к использованию в шаблонах.
- MODIFIER_FUNCS - Это массив имён PHP-функций, разрешенных к использованию в качестве модификаторов переменных.
- ALLOW_CONSTANTS - true/false. Если установлено в true, разрешается использование констант вида {$smarty.const.name}. По умолчанию равно "false" из соображений безопасности.

$trusted_dir

$trusted_dir используется только при включенном параметре $security. Это массив всех директорий, которые считаются надёжными. Надёжные директории - это директории, в которых вы храните свои php-скрипты, которые включаются прямо в шаблоны при помощи директивы {include_php}.

$left_delimiter

Левый разделитель, используемый в языке шаблонов. По умолчанию равно "{".

См. также $right_delimiter.

$right_delimiter

Правый разделитель, используемый в языке шаблонов. По умолчанию равно "}".

См. также $left_delimiter.

$compiler_class

Задает имя класса-компилятора, который Smarty будет использовать для компиляции шаблонов. По умолчанию это 'Smarty_Compiler'. Только для продвинутых пользователей.

$request_vars_order

Порядок, в котором будут регистрироваться переменные запроса, наподобие variables_order из php.ini

$request_use_auto_globals

Определяет должен ли Smarty использовать $HTTP_*_VARS[] ($request_use_auto_globals=false - значением по умолчанию) или $_*[] ($request_use_auto_globals=true). Это влияет на поведение шаблонов, которые используют {$smarty.request.*}, {$smarty.get.*} и т.д. Внимание: если вы установите $request_use_auto_globals в true, variable.request.vars.order не учитывается, а вместо него используется значение gpc_order из настроек php.

$error_reporting

Если это свойство имеет ненулевое значние, то оно используется в качестве значения error_reporting внутри display() и fetch(). При включенном режиме отладки это значение игнорируется и уровень обработки ошибок не меняется.

$compile_id

Постоянный идентификатор компиляции. Как альтернативу использованию одного и того же compile_id при каждом вызове функции, вы можете самостоятельно задавать этот идентификатор, и в этом случае будет безусловно автоматически это значение.

С помощью compile_id вы можете обойти ограничение, из-за которого вы не можете использовать один compile_dir для разных template_dir. Если вы установите уникальный compile_id для каждого template_dir, Smarty сможет различать компилированные шаблоны по их compile_id.

К примеру, если у вас есть префильтр, локализирующий ваш ваши шаблоны (проще говоря, переводит части шаблонов на другой язык) во время компиляции, то вам следует использовать текущий язык в качестве compile_id и вы получите по набору скомпилированных шаблонов для каждого используемого языка.

Другим примером может быть использование одной компиляционной директории для нескольких доменов / нескольких vhost'ов, к примеру:

Пример 12.1. compile_id

```
$smarty->compile_id = $_SERVER['SERVER_NAME'];
  $smarty->compile_dir = 'path/to/shared_compile_dir';
```

$use_sub_dirs

Установите это в false если ваше окружение PHP не разрешает создание директорий от имени Smarty. Под-директории более эффективны, так что используйте их, если можете.

> **Техническое замечание:** Начиная с версии Smarty 2.6.2, значением по умолчанию для use_sub_dirs является false.

$default_modifiers

Массив модификаторов, неявно применяемых ко всем переменным шаблона. Например, для HTML-экранирования каждой переменной по умолчанию, используется конструкция array('escape:"htmlall"'); Для исключения действия таких модификаторов на какую-либо переменную, применяйте специальный "smarty" модификатор с параметром "nodefaults", например {$var|smarty:nodefaults}.

$default_resource_type

Это свойство говорит Smarty, какой тип ресурсов использовать по умолчанию. Значением этого свойства по умолчанию является 'file', так что $smarty->display('index.tpl'); и $smarty->display('file:index.tpl'); имеют одинаковый смысл. Обратитесь к главе ресурсы для дополнительной информации.

Глава 13. Методы

Содержание

append

append

append

void **append** (mixed var)
void **append** (string varname, mixed var [, bool merge])

Функция используется для добавления элемента в назначенный (assigned) массив. Если вы добавляете к строковому значению, оно конвертируется в значение массива, и, затем добавляется. Вы можете передавать пары имя/значение явно, или в виде ассоциативного массива, состоящего из пар имя/значение. Если вы устанавливаете необязательный третий параметр в true, то значение будет не добавлено, а слито с текущим массивом.

> **Техническое замечание:** Параметр *merge* учитывает ключи массива, поэтому если вы объединяете массивы с числовыми индексами, то они могут наложиться друг на друга или привести к непоследовательному порядку ключей. Результат отличается от действия функции PHP array_merge() [http://php.net/array_merge], которая заново нумерует элементы в массиве с числовыми ключами.

Пример 13.1. append

```
<?php
// передача пар имя/значение
$smarty->append("Name","Fred");
$smarty->append("Address",$address);

// передача ассоциативного массива
$smarty->append(array("city" => "Lincoln","state" => "Nebraska"));
?>
```

append_by_ref

append_by_ref

append_by_ref

void **append_by_ref** (string varname, mixed var [, bool merge])

Эта функция используется для добавления значений к шаблону по ссылке. Если вы добавляете переменную по ссылке то, соответсвенно, можете изменять значение переменной, на которую она ссылается. Для объектов, append_by_ref() так же помогает избежать их копирования в памяти. Смотрите руководство PHP на предмет ссылок на переменные для более глубокого пояснения. Если вы устанавливаете необязательный третий параметр в true, то значение не добавляется, а сливается с текущим массивом.

> **Техническое замечание:** Параметр *merge* учитывает ключи массива, поэтому если вы объединяете массивы с числовыми индексами, то они могут наложиться друг на друга или привести к непоследовательному порядку ключей. Результат отличается от действия функции PHP array_merge() [http://php.net/array_merge], которая заново нумерует элементы в массиве с числовыми ключами.

Пример 13.2. append_by_ref

```
<?php
// добавление пар имя/значение
$smarty->append_by_ref("Name",$myname);
$smarty->append_by_ref("Address",$address);
?>
```

assign

assign

assign

void **assign** (mixed var)
void **assign** (string varname, mixed var)

Функция используется для присвоения значений в шаблонах. Вы можете явно передавать пары имя/значение, или ассоциативные массивы, содержащие пары имя/значение.

Пример 13.3. assign

```
<?php
// передача пар имя/значение
$smarty->assign('Name', 'Fred');
$smarty->assign('Address', $address);

// передача ассоциативного массива
$smarty->assign(array("city" => "Lincoln", "state" => "Nebraska"));
?>
```

assign_by_ref

assign_by_ref

assign_by_ref

void **assign_by_ref** (string varname, mixed var)

Эта функция используется для передачи значения переменной в шаблон по ссылке, вместо создания ее копии. Смотрите руководство PHP на предмет ссылок на переменные для более глубокого пояснения.

> **Техническое замечание:** Функция используется для передачи значения в шаблон по ссылке. Если вы назначаете переменную по ссылке, то имеете возможность менять значение переменной, на которую она ссылается. Что касается объектов, то assign_by_ref() помогает избежать копирования переданных в шаблон объектов в памяти. Смотрите руководство PHP на предмет ссылок на переменные.

Пример 13.4. assign_by_ref

```
<?php
// передача пар имя/значение
$smarty->assign_by_ref('Name', $myname);
$smarty->assign_by_ref('Address', $address);
?>
```

clear_all_assign

clear_all_assign

clear_all_assign

void **clear_all_assign** (void)

Очищает все ранее присвоенные значения переменных.

Пример 13.5. clear_all_assign

```
<?php
// очистить значения переменных
$smarty->clear_all_assign();
?>
```

clear_all_cache

clear_all_cache

clear_all_cache

void **clear_all_cache** ([int expire_time])

Полностью очищает кэш шаблонов. В качестве необязательного параметра, вы можете указать минимальный возраст файлов кэша в секундах, по достижению которого они будут очищены.

Пример 13.6. clear_all_cache

```
<?php
// очистить кэш целиком
$smarty->clear_all_cache();
?>
```

clear_assign

clear_assign

clear_assign

void **clear_assign** (mixed var)

Очищает присвоенное ранее значение переменной. Это может быть как обычная переменная, так и массив некоторых значений.

Пример 13.7. clear_assign

```
<?php
// очистить значение переменной
$smarty->clear_assign("Name");

// очистить значения несколько переменных
$smarty->clear_assign(array("Name", "Address", "Zip"));
?>
```

clear_cache

clear_cache

clear_cache

void **clear_cache** (string template [, string cache_id [, string compile_id [, int expire_time]]])

Очищает кэш указанного шаблона. Если у вас есть несколько кэшированных копий шаблона, вы можете очистить конкретную копию, указав *cache_id* в качестве второго параметра. Вы так же можете передать compile id в качестве третьего параметра или "сгруппировать" шаблоны вместе и удалить группу. Для получения дополнительной информации, смотрите раздел "Кэширование". В качестве необязательного четвертого параметра вы можете указать минимальное время в секундах, в течение которого кэш будет существовать перед очисткой.

Пример 13.8. clear_cache

```
<?php
// очистить кэш шаблона
$smarty->clear_cache("index.tpl");

// очистить кэшированную копию с конкретным cache id
// при множественном кэшировании шаблона
$smarty->clear_cache("index.tpl", "CACHEID");
?>
```

clear_compiled_tpl

clear_compiled_tpl

clear_compiled_tpl

void **clear_compiled_tpl** ([string tpl_file [, string compile_id [, int exp_time]]])

Очищает указанную откомпилированную версию шаблона, или все откомпилированные файлы шаблона если ничего не указано. Если вы передадите аргумент compile_id, только откомпилированные версии шаблонов с этим compile_id будут очищены. Если вы передадите аргумент exp_time, только откомпилированные версии шаблонов старше exp_time секунд будут очищены. По умолчанию очищаются все откомпилированные версии шаблонов независимо от их возраста. Эта функция только для опытных пользователей и обычно в ее использовании необходимости нет.

Пример 13.9. clear_compiled_tpl

```
<?php
// очистить откомпилированный шаблон
$smarty->clear_compiled_tpl("index.tpl");

// очистить всю директорию с откомпилированными шаблонами
$smarty->clear_compiled_tpl();
?>
```

clear_config

clear_config

clear_config

void **clear_config** ([string var])

Функция очищает все назначенные конфигурационные переменные. Если передано имя переменной, значит очищается только она.

Пример 13.10. clear_config

```
<?php
// очищаем все переменные.
$smarty->clear_config();

// очищаем только одну переменную
$smarty->clear_config('foobar');
?>
```

config_load

config_load

config_load

void **config_load** (string file [, string section])

Функция загружает данные из файла конфигурации (config file) и помещает их в шаблон. Это аналог функции шаблона config_load.

> **Техническое замечание:** Начиная с версии Smarty 2.4.0, переменные шаблона подставляются при вызове fetch() или display(). Кофигурационные переменные, загружаемые при вызове config_load() всегда глобальны. Для повышения скорости работы, конфигурационные файлы тоже компилируются и к ним применительны настройки force_compile и compile_check.

Пример 13.11. config_load

```
<?php
// загружаются и назначаются конфигурационные переменные
$smarty->config_load('my.conf');

// загрузка секции конфигурационного файла
$smarty->config_load('my.conf','foobar');
?>
```

display

display

display

void **display** (string template [, string cache_id [, string compile_id]])

Функция отоброжает шаблон. Укажите верный тип ресурса шаблонов и путь. В качестве необязательного второго параметра, вы можете передать cache id. Смотрите раздел Кэширование для дополнительной информации.

В качестве необязательного третьего аргумента вы можете передать *$compile_id*. Это полезно в случае, если вы хотите скомпилировать несколько различных версий одного шаблона, например несколько версий одного шаблона на разных языках. Другое применение *$compile_id* можно найти, если вы используете несколько $template_dir, но только одну $compile_dir. Устанавливайте свой *compile_id* для каждой $template_dir, иначе шаблоны с одинаковыми именами будут сохраняться поверх друг друга. Также вы можете один раз указать $compile_id, вместо того, чтобы каждый раз передавать его при вызове этой функции.

Пример 13.12. display

```
<?php
include("Smarty.class.php");
$smarty = new Smarty;
$smarty->caching = true;

// обращаемся к базе только в случае отсутствия кэша
if(!$smarty->is_cached("index.tpl"))
{

    // подставляем некоторые данные
    $address = "245 N 50th";
    $db_data = array(
            "City" => "Lincoln",
            "State" => "Nebraska",
            "Zip" = > "68502"
            );

    $smarty->assign("Name","Fred");
    $smarty->assign("Address",$address);
    $smarty->assign($db_data);

}

// отображаем результат
$smarty->display("index.tpl");
?>
```

Используйте синтаксис ресурсов шаблонов для отображения файлов, находящихся вне директории

$template_dir.

Пример 13.13. Примеры отображения шаблонов из различных ресурсов

```
<?php
// абсолютный файловый путь
$smarty->display("/usr/local/include/templates/header.tpl");

// абсолютный файловый путь (тоже самое)
$smarty->display("file:/usr/local/include/templates/header.tpl");

// абсолютный путь Windows (ОБЯЗАТЕЛЬНО используйте префикс "file:")
$smarty->display("file:C:/www/pub/templates/header.tpl");

// вставка из ресурса под названием "db"
$smarty->display("db:header.tpl");
?>
```

См. также fetch() и template_exists.

fetch

fetch

fetch

string **fetch** (string template [, string cache_id [, string compile_id]])

Функция возвращает вывод шаблона вместо его отображения на экран. Укажите верный тип ресурса шаблонов и путь. В качестве необязательного второго параметра можно передать cache id. Смотрите раздел Кэширование для получения дополнительной информации.

В качестве необязательного третьего аргумента вы можете передать *$compile_id*. Это полезно в случае, если вы хотите скомпилировать несколько различных версий одного шаблона, например несколько версий одного шаблона на разных языках. Другое применение *$compile_id* можно найти, если вы используете несколько $template_dir, но только одну $compile_dir. Устанавливайте свой *compile_id* для каждой $template_dir, иначе шаблоны с одинаковыми именами будут сохраняться поверх друг друга. Также вы можете один раз указать $compile_id, вместо того, чтобы каждый раз передавать его при вызове этой функции.

Пример 13.14. fetch

```
<?php
include("Smarty.class.php");
$smarty = new Smarty;

$smarty->caching = true;

// обращаемся к БД только если отсутствует кэш
if(!$smarty->is_cached("index.tpl"))
{

    // присваиваем некоторые значения
    $address = "245 N 50th";
    $db_data = array(
            "City" => "Lincoln",
            "State" => "Nebraska",
            "Zip" = > "68502"
            );

    $smarty->assign("Name","Fred");
    $smarty->assign("Address",$address);
    $smarty->assign($db_data);

}

// перехватываем вывод
$output = $smarty->fetch("index.tpl");

// здесь выполняем какие-либо действия с $output
```

```
echo $output;
?>
```

См. также display() и template_exists.

get_config_vars

get_config_vars

get_config_vars

array **get_config_vars** ([string varname])

Возвращает значение переданной конфигурационной переменной. Если аргумент не передан, то будет возвращен массив всех конфигурационных переменных.

Пример 13.15. get_config_vars

```
<?php
// получаем загруженную конфигурационную переменную 'foo'
$foo = $smarty->get_config_vars('foo');

// получаем все загруженные конфигурационные переменные шаблона
$config_vars = $smarty->get_config_vars();

// смотрим, что у нас получилось
print_r($config_vars);
?>
```

get_registered_object

get_registered_object

get_registered_object

array **get_registered_object** (string object_name)

Возвращает ссылку на зарегестрированный объект. Может быть полезно в случае необходимости получения прямого доступа к зарегестрированному объекту из пользовательской функции.

Пример 13.16. get_registered_object

```
<?php
function smarty_block_foo($params, &$smarty) {
  if (isset[$params['object']]) {
    // получаем ссылку на объект
      $obj_ref =& $smarty->&get_registered_object($params['object']);
    // теперь используем $obj_ref как ссылку на объект
  }
}
?>
```

get_template_vars

get_template_vars

get_template_vars

array **get_template_vars** ([string varname])

Возвращает значение переменной. Если аргумент не передан, будет возвращен массив всех назначенными переменными.

Пример 13.17. get_template_vars

```
<?php
// получаем назначенную переменную шаблона 'foo'
$foo = $smarty->get_template_vars('foo');

// получаем все назначенные переменные шаблона
$tpl_vars = $smarty->get_template_vars();

// поглядим, что из этого вышло
print_r($tpl_vars);
?>
```

is_cached

is_cached

is_cached

bool **is_cached** (string template [, string cache_id [, string compile_id]])

Возвращает true если существует кэш для указанного шаблона. Работает только в том случае, если значение caching установлено в true.

Пример 13.18. is_cached

```
<?php
$smarty->caching = true;

if(!$smarty->is_cached("index.tpl")) {
    // обращаемся к БД, назначаем переменные
}

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

Также вы можете передавать cache id в качестве необязательного второго параметра, если у вас используется множественное кэширование шаблона.

Также вы можете передавать compile id в качестве необязательного третьего параметра. Если вы не передадите этот параметр, будет использован текущий $compile_id.

Если вы не хотите передавать cache id, но хотите передать compile id, вы должны передать null в качестве cache id.

Пример 13.19. is_cached при множественном кэшировании шаблона

```
<?php
$smarty->caching = true;

if(!$smarty->is_cached("index.tpl", "FrontPage")) {
    // обращаемся к БД, назначаем переменные
}

$smarty->display("index.tpl", "FrontPage");
?>
```

Техническое замечание: Если is_cached возвращает true, при этом она загружает кэшированный вывод и хранит его в памяти. Любые последующие вызовы display() или fetch() будут возвращать этот

хранимый в памяти вывод и не будут пытаться перезагрузить файл кэша. Это предотвращает неприятную ситуацию, которая может возникнуть если другой процесс очищает кэш между вызовами is_cached и display в предыдущем примере. Это также означает, что clear_cache() и другие изменения настроек кэширования могут не вступить в силу после того, как is_cached вернула true.

load_filter

load_filter

load_filter

void **load_filter** (string type, string name)

Эта функция может быть использована для загрузки плагина фильтра. Первый аргумент определяет тип загружаемого фильтра и может быть одним из следующих: 'pre', 'post' или 'output'. Второй аргумент определяет имя плагина фильтра, к примеру 'trim'.

Пример 13.20. Загрузка плагинов фильтров

```
<?php
$smarty->load_filter('pre', 'trim'); // загружаем префильтр под названием 'trim'
$smarty->load_filter('pre', 'datefooter'); // загружаем еще один префильтр - 'datefooter'
$smarty->load_filter('output', 'compress'); // загружаем фильтр вывода 'compress'
?>
```

register_block

register_block

register_block

void **register_block** (string name, mixed impl, bool cacheable, mixed cache_attrs)

Используйте для динамической регистрации плагинов блоковых функций. В качестве аргументов передаются имя блоковой функции и имя функции, реализующей ее.

Коллбек-функцией php *impl* может быть (a) строка, содержащая имя функции, или (b) массив вида array(&$object, $method), где &$object является ссылкой на объект, а $method является строкой, содержащей имя метода, или (c) массив в форме array($class, $method), где $class является именем класса, а $method является методом этого класса.

cacheable и *cache_attrs* в большинстве случаев могут быть опущены. Смотрите Управление кэшированием результатов работы плагинов для получения информации об их правильном использовании.

Пример 13.21. register_block

```
<?php
$smarty->register_block("translate", "do_translation");

function do_translation ($params, $content, &$smarty, &$repeat)
{
  if (isset($content)) {
    $lang = $params['lang'];
    // выполняем перевод $content
    return $translation;
  }
}
?>
```

Содержимое шаблона:

```
{* шаблон *}
{translate lang="br"}
Hello, world!
{/translate}
```

register_compiler_function

register_compiler_function

register_compiler_function

bool **register_compiler_function** (string name, mixed impl, bool cacheable)

Используется для динамической регистрации плагина функции компилятора. Передается наименование функции компилятора, далее имя функции, реализующей ее.

Коллбек-функцией php *impl* может быть (a) строка, содержащая имя функции, или (b) массив вида array(&$object, $method), где &$object является ссылкой на объект, а $method является строкой, содержащей имя метода, или (c) массив в форме array($class, $method), где $class является именем класса, а $method является методом этого класса.

cacheable и *cache_attrs* в большинстве случаев могут быть опущены. Смотрите Управление кэшированием результатов работы плагинов для получения информации об их правильном использовании.

register_function

register_function

register_function

void **register_function** (string name, mixed impl [, bool cacheable [, mixed cache_attrs]])

Используется для динамической регистрации плагинов функций шаблона. Передается наименование функции шаблона и имя функции, реализующей ее.

Функция обратного вызова PHP *impl* может быть (a) строка, содержащая имя функции, или (b) массив вида array(&$object, $method), где &$object является ссылкой на объект, а $method является строкой, содержащей имя метода, или (c) массив в форме array($class, $method), где $class является именем класса, а $method является методом этого класса.

cacheable и *cache_attrs* в большинстве случаев могут быть опущены. Смотрите Управление кэшированием результатов работы плагинов для получения информации об их правильном использовании.

Пример 13.22. register_function

```
$smarty->register_function("date_now", "print_current_date");

function print_current_date($params)
{
  if(empty($params['format'])) {
    $format = "%b %e, %Y";
  } else {
    $format = $params['format'];
    return strftime($format,time());
  }
}

// теперь вы можете использовать ее в Smarty чтобы вывести текущую дату: {date_now}
// или  {date_now format="%Y/%m/%d"} чтобы задать формат.
?>
```

register_modifier

register_modifier

register_modifier

void **register_modifier** (string name, mixed impl)

Используйте функцию для динамической регистрации плагина модификатора. В функцию передаются имя модификатора и имя функции, реализующей его.

Коллбек-функцией php *impl* может быть (a) строка, содержащая имя функции, или (b) массив вида array(&$object, $method), где &$object является ссылкой на объект, а $method является строкой, содержащей имя метода, или (c) массив в форме array($class, $method), где $class является именем класса, а $method является методом этого класса.

cacheable и *cache_attrs* в большинстве случаев могут быть опущены. Смотрите Управление кэшированием результатов работы плагинов для получения информации об их правильном использовании.

Пример 13.23. register_modifier

```
<?php
// вносим функцию PHP stripslashes в модификатор Smarty.

$smarty->register_modifier("sslash"," stripslashes");

// теперь можно использовать {$var|sslash} чтобы вырезать слеши из переменной
?>
```

register_object

register_object

register_object

void **register_object** (string object_name, object object, array allowed_methods_properties, boolean format, array block_methods)

Функция регестрирует объект для использования в шаблоне. Обратитесь к разделу Объекты за примерами.

См. также unregister_object.

register_outputfilter

register_outputfilter

register_outputfilter

void **register_outputfilter** (mixed function)

Используйте функцию для динамической регистрации фильтров вывода, чтобы управлять выводом шаблона перед тем, как он будет отображен. Обратитесь к фильтрам вывода шаблонов для получения дополнительной информации.

Коллбек-функцией php *function* может быть (a) строка, содержащая имя функции, или (b) массив вида array(&$object, $method), где &$object является ссылкой на объект, а $method является строкой, содержащей имя метода, или (c) массив в форме array($class, $method), где $class является именем класса, а $method является методом этого класса.

register_postfilter

register_postfilter

register_postfilter

void **register_postfilter** (mixed function)

Используйте функцию для динамической регистрации постфильтров, в целях управления выводом шаблонов уже после их компиляции. Обратитесь к постфильтрам шаблонов для получения дополнительной информации.

Коллбек-функцией php *function* может быть (a) строка, содержащая имя функции, или (b) массив вида array(&$object, $method), где &$object является ссылкой на объект, а $method является строкой, содержащей имя метода, или (c) массив в форме array($class, $method), где $class является именем класса, а $method является методом этого класса.

register_prefilter

register_prefilter

register_prefilter

void **register_prefilter** (mixed function)

Используйте функцию для динамической регистрации префильтра, в целях управления содержимым шаблона перед его компиляцией. Обратитесь к префильтрам шаблонов для получения дополнительной информации.

Коллбек-функцией php *function* может быть (a) строка, содержащая имя функции, или (b) массив вида array(&$object, $method), где &$object является ссылкой на объект, а $method является строкой, содержащей имя метода, или (c) массив в форме array($class, $method), где $class является именем класса, а $method является методом этого класса.

register_resource

register_resource

register_resource

void **register_resource** (string name, array resource_funcs)

Используйте эту функцию, чтобы динамически зарегистрировать плагин ресурса в Smarty. Передается имя ресурса и массив php-функций. Обратитесь к ресурсам шаблонов для полученияодополнительной информации.

> **Техническое замечание:** Имя ресурса должно состоять минимум из двух букв. Однобуквенные имена ресурсов будут игнорироваться и испольщоваться как часть файлового пути, например $smarty->display('c:/path/to/index.tpl');

Массив php-функций *resource_funcs* должен содержать 4 или 5 элементов. В случае четырех элементов, элементы являются соответствующими коллбек-функциями: "source", "timestamp", "secure" и "trusted" функции ресурса. В случае пяти элементов, первый элемент должен быть ссылкой на объект или имя класса, объект или класс которого реализовывает ресурс, а 4 следующих элементов должны быть названиями методов, реализующимх "source", "timestamp", "secure" и "trusted".

Пример 13.24. register_resource

```
<?php
$smarty->register_resource("db", array("db_get_template",
                                       "db_get_timestamp",
                                       "db_get_secure",
                                       "db_get_trusted"));
?>
```

trigger_error

trigger_error

trigger_error

void **trigger_error** (string error_msg [, int level])

Эта функция может быть использована для вывода сообщения об ошибке средствами Smarty. Параметр *level* может быть равен одному из значений, используемых для PHP-функции trigger_error(), т.е. E_USER_NOTICE, E_USER_WARNING, и др. По умолчанию установлено значение E_USER_WARNING.

template_exists

template_exists

template_exists

bool **template_exists** (string template)

Эта функция проверяет, существует ли определенный шаблон. Здесь можно указать путь к шаблону в файловой системе или строку ресурса, соответствующую шаблону.

unregister_block

unregister_block

unregister_block

void **unregister_block** (string name)

Используйте функцию для динамической дерегистрации плагина блоковой функции. В качестве аргумента передается имя функции.

unregister_compiler_function

unregister_compiler_function

unregister_compiler_function

void **unregister_compiler_function** (string name)

Используйте функцию для динамической дерегистрации функции компиляции. В качестве аргумента передается имя функции компиляции.

unregister_function

unregister_function

unregister_function

void **unregister_function** (string name)

Используйте функцию для динамической деригистрации плагина функции шаблона. В качестве аргумента передается имя функции шаблона.

Пример 13.25. unregister_function

```
<?php
// мы не хотим давать доступ дизайнеру шаблонов к системным файлам

$smarty->unregister_function("fetch");
?>
```

unregister_modifier

unregister_modifier

unregister_modifier

void **unregister_modifier** (string name)

Используйте функцию для динамической дерегистрации плагина модификатора. В качестве аргумента передается имя модификатора.

Пример 13.26. unregister_modifier

```
<?php
// мы не хотим, чтобы дизайнер шаблонов очищал переменные от тегов

$smarty->unregister_modifier("strip_tags");
?>
```

unregister_object

unregister_object

unregister_object

void **unregister_object** (string object_name)

Используется для дерегистрации объекта.

См. также register_object и раздел Объекты

unregister_outputfilter

unregister_outputfilter

unregister_outputfilter

void **unregister_outputfilter** (string function_name)

Используйте функцию для динамической дерегистрации фильтра вывода.

unregister_postfilter

unregister_postfilter

unregister_postfilter

void **unregister_postfilter** (string function_name)

Используйте функцию для динамической дерегистрации постфильтра.

unregister_prefilter

unregister_prefilter

unregister_prefilter

void **unregister_prefilter** (string function_name)

Используйте для динамической дерегистрации префильтра.

unregister_resource

unregister_resource

unregister_resource

void **unregister_resource** (string name)

Используйте функцию для динамической дерегистрации плагина ресурса. В качестве аргумента передается имя ресурса.

Пример 13.27. unregister_resource

```
<?php
$smarty->unregister_resource("db");
?>
```

Глава 14. Кэширование

Содержание

Кэширование используется для ускорения вызовов display() или fetch() путем сохранения их результатов в файл. Кэшированная версия файла (если таковая будет найдена) отображается сразу, без перегенерации вывода. Кэширование может значительно ускорить работу, особенно если используются сложные шаблоны с большим количеством данных. Так как кэшируется вывод display() и fetch(), один файл в кэше может представлять из себя набор разных шаблонов, конфигурационных файлов - всего того, что использовалось при генерации этого вывода.

Так как шаблоны могут меняться со временем, очень важно следить за тем что вы кэшируете и на который срок. Например, если вы отображаете титульную страницу вашего сайта, которая не меняется слишком часто, то её кэшированную версию можно не обновлять в течение часа или больше. С другой стороны, если вы выводите страницу с прогнозом погоды, которая может меняться с каждой минутой, то кэшировать её не имеет смысла.

Настройка кэширования

Прежде всего, кэширование необходимо активировать. Это можно сделать, установив $caching = true (или 1).

Пример 14.1. Включение кэширования

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$smarty->display('index.tpl');
```

При включенном кэшировании, вызываемая функция display('index.tpl') интерпретирует шаблон как обычно, но также сохраняет копию вывода в файл (кэшированую копию) в $cache_dir. При следующем вызове display('index.tpl'), вместо повторной интерпретации шаблона, будет использована кешированая копия.

> **Техническое замечание:** Файлы в директории $cache_dir имеют те же имена, что и соответствующие шаблоны. Их имена оканчиваются расширением ".php", но на самом деле они не являются выполняемыми php-скриптами. Не редактируйте эти файлы!

Каждая кэшированая страничка существует на протяжении определенного времени, указанного в $cache_lifetime. Значение по умолчанию равно 3600 секундам или 1 часу. После того, как это время истекает, кэш обновляется. Существует возможность присвоить каждой кэшированой страничке собственное время жизни, установив $caching = 2. Смотрите документацию $cache_lifetime для получения подробных сведений.

Пример 14.2. Установка собственного cache_lifetime для кэшированой копии

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 2; // Срок действия только для этой копии

// устанавливаем cache_lifetime для index.tpl в 5 минут
$smarty->cache_lifetime = 300;
$smarty->display('index.tpl');

// устанавливаем cache_lifetime для home.tpl в 1 час
$smarty->cache_lifetime = 3600;
$smarty->display('home.tpl');

// Примечание: следующая $cache_lifetime настройка не будет работать, когда $caching = 2.
// Срок жизни кэша для home.tpl уже был установлен
// в 1 час, и Smarty больше не будет обращать внимание на значение $cache_lifetime.
// Время жизни кэша home.tpl по прежнему будет истекать по прошествию одного часа.
$smarty->cache_lifetime = 30; // 30 секунд
$smarty->display('home.tpl');
```

Если включен параметр $compile_check, то каждый файл шаблона и конфигурации, связанный с файлом кэша, проверяется на наличие изменений. Если один из этих файлов был модифицирован с тех пор, как кэш был создан, кэш немедленно обновляется. Это незначительно повышает нагрузку, поэтому, для оптимальной производительности оставьте значение $compile_check равным false.

Пример 14.3. Включение $compile_check

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;
$smarty->compile_check = true;

$smarty->display('index.tpl');
```

Если $force_compile активирован, файлы кэша всегда будут обновляться. Это средство можно использовать для отключения кэширования во время отладки. $force_compile обычно используется только в целях отладки, так как более правильным способом отключения кеширования является установка $caching = false (или 0).

Функция is_cached() может быть использована для определения, имеется ли у шаблона работоспособный

кэш. Если у вас есть кэшированый шаблон, которому необходимо, например, получить выборку из базы данных, вы можете использовать эту функцию, чтобы пропустить процесс обращения к базе.

Пример 14.4. Использование is_cached()

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

if(!$smarty->is_cached('index.tpl')) {
        // Кэш отсутствует, значит присваеваем значения переменным.
        $contents = get_database_contents();
        $smarty->assign($contents);
}

$smarty->display('index.tpl');
```

Вы можете сделать так, чтобы часть страницы оставалась динамической, даже если страница кэшируется, при помощи встроенной функции insert. Например, кэшироваться может вся страница, за исключением баннера. Используя функцию insert для баннера, вы можете сохранять этот элемент динамичным, внутри кэшированой странички. Смотрите документацию по insert для получения подробностей и примеров.

Очистить все файлы кэша можно при помощи функции clear_all_cache(), а конкретный файл кэша (или группу) - вызвав clear_cache() функцию.

Пример 14.5. Очистка кэша

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// очищаем все файлы кэша
$smarty->clear_all_cache();

// очищаем только кэш шаблона index.tpl
$smarty->clear_cache('index.tpl');

$smarty->display('index.tpl');
```

Множественное кэширование страниц

Вы можете создавать несколько кэшированных копий для одного вызова функции display() или fetch(). Предположим, что по вызову display('index.tpl') должны отображаться данные, содержимое которых зависит от определенных условий, и вы хотите иметь несколько вариантов соответствующих кэшированных копий. Для этого необходимо передать в функцию идентификатор кэша (cache_id) в качестве второго параметра.

Пример 14.6. Вызов display() с идентификатором кэша

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

$smarty->display('index.tpl',$my_cache_id);
```

В примере мы передали переменную $my_cache_id в функцию display() в качестве cache_id. Для каждого уникального значения $my_cache_id будет создана кэшированная копия вывода index.tpl. Здесь, значение "article_id" было передано в скрипт через URL, присвоено переменной $my_cache_id и использовано как cache_id.

> **Техническое замечание:** Будьте очень осторожными при передаче значений от клиента (браузера) в Smarty (как и в любое PHP-приложение). Хотя приведенный пример фактического использования article_id прямо из URL выглядит нормально, он может иметь неприятные последствия. Значение cache_id используется для создания директории в файловой системе, поэтому, если пользователь решит передать крайне большое значение article_id или написать скрипт, который посылает случайные article_id с огромной частотой, это может вызвать проблемы на уровне сервера. Поэтому вам необходимо в обязательном порядке проверять данные из форм, перед тем как использовать их. В нашем случае, мы заранее знаем, что значение article_id имеет длину в 10 символов, состоит только из букв и цифр, а так же должно являться реальным идентификатором в базе данных. Все это необходимо проверить!

Имейте ввиду, что тоже самое значение cache_id необходимо использовать как второй параметр в функциях is_cached() и clear_cache(), если вы хотите применить их к конкретному кэшу.

Пример 14.7. Передача cache_id в is_cached()

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

if(!$smarty->is_cached('index.tpl',$my_cache_id)) {
        // Кэша нет, поэтому присваиваем значение переменным.
        $contents = get_database_contents();
        $smarty->assign($contents);
}

$smarty->display('index.tpl',$my_cache_id);
```

Вы можете удалить все кэшированные копии с конкретным cache_id, передав null в качестве первого параметра clear_cache().

Пример 14.8. Удаление всех кэшированных копий с конкретным cache_id

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// удаляем все кэшированные копии со значением "sports" в качестве cache_id
$smarty->clear_cache(null,"sports");

$smarty->display('index.tpl',"sports");
```

Таким образом, вы можете группировать ваши кэшированные копии, назначая им одинаковые cache_id.

Групповое кэширование

Вы можете сделать группировку более продуманной, используя групповые значения cache_id. В таком случае, каждая подгруппа отделяется знаком вертикальной черты "|" в значении cache_id. Возможно создавать любое количество подгрупп.

Вы можете представить себе группы кэширования в виде иерархии каталогов. К примеру, группа кэширования "a|b|c" соответствует структуре каталогов "/a/b/c/". clear_cache(null,"a|b|c") - это всё равно, что удалить файлы из "/a/b/c/*". clear_cache(null,"a|b") соответствует удалению файлов "/a/b/*". Если вы укажете compile_id, например clear_cache(null,"a|b","foo"), он добавляется в конец группы кэширования: "/a/b/c/foo/". Если вы укажете имя шаблона, например clear_cache("foo.tpl","a|b|c"), то Smarty попытается удалить "/a/b/c/foo.tpl". Вы НЕ можете удалить определенный шаблон из нескольких групп кэширования, наподобие "/a/b/*/foo.tpl" - группы кэширования работают ТОЛЬКО слева направо. Вам нужно будет сгруппировать шаблоны под единой иерархией групп кэширования, чтобы иметь возможность очистить их как группу.

Групповое кэширование не следует путать с иерархией директорий шаблонов. Групповое кэширование не имеет представления о том, как структурированы ваши шаблоны. К примеру, если структура ваших шаблонов выглядит как "themes/blue/index.tpl" и вы хотите иметь возможность очистить все файлы кэша для темы "blue", вам нужно создать такую структуру групп кэширования, которая бы повторяла файловую структуру ваших шаблонов, например display("themes/blue/index.tpl","themes|blue"), а затем очистить их вот так: clear_cache(null,"themes|blue").

Пример 14.9. Группы в cache_id

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// Удалить все кэшированные копии подгруппы "sports|basketball"
$smarty->clear_cache(null,"sports|basketball");
```

```
// Удалить все кэшированные копии группы "sports",
// включая "sports|basketball", или "sports|(anything)|(anything)|(anything)|..."
$smarty->clear_cache(null,"sports");

$smarty->display('index.tpl',"sports|basketball");
```

Управление кэшированием результатов работы плагинов

Начиная с плагинов Smarty-2.6.0, кэшируемость плагинов может быть объявлена во время их регистрации. Третий аргумент у register_block, register_compiler_function и register_function называется *$cacheable* и имеет значение по умолчанию true, что соответствует поведению плагинов Smarty версии ранее 2.6.0

Если плагин регистрируется с $cacheable=false, плагин вызывается каждый раз, когда страница отображается, даже если сама страница кэширована. Поведение плагина немного похоже на функцию insert.

В отличие от {insert}, атрибуты плагина не кэшируются по умолчанию. Они могут быть объявлены как кэшируемые при помощи четвертого параметра - *$cache_attrs*. *$cache_attrs* это массив имен атрибутов, которые должны кэшироваться, чтобы функция плагина брала значение в том виде, в котором оно было в момент помещения страницы в кэш, каждый раз, когда страница запрашивается из кэша.

Пример 14.10. Предотвращение кэширования результата работы плагина

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->caching = true;

function remaining_seconds($params, &$smarty) {
    $remain = $params['endtime'] - time();
    if ($remain >=0)
        return $remain . " second(s)";
    else
        return "done";
}

$smarty->register_function('remaining', 'remaining_seconds', false, array('endtime'));

if (!$smarty->is_cached('index.tpl')) {
    // извлекаем $obj из БД и присваиваем...
    $smarty->assign_by_ref('obj', $obj);
}

$smarty->display('index.tpl');
?>
```

Шаблон index.tpl:

```
Оставшееся время: {remaining endtime=$obj->endtime}
```

Количество секунд до endtime объекта $obj изменяется при каждом обновлении страницы, даже если страница кэширована. Так как атрибут endtime кэширован, объект извлекается из базы данных в тот момент, когда страница помещается в кэш, но не во время последующих запросов к странице.

Пример 14.11. Предотвращение кэширования части страницы

index.php:

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->caching = true;

function smarty_block_dynamic($param, $content, &$smarty) {
    return $content;
}
$smarty->register_block('dynamic', 'smarty_block_dynamic', false);

$smarty->display('index.tpl');
?>
```

Шаблон index.tpl:

```
Страница кэширована: {"0"|date_format:"%D %H:%M:%S"}

{dynamic}

Текущее время: {"0"|date_format:"%D %H:%M:%S"}

... выполняем разные действия ...

{/dynamic}
```

Во время обновления страницы вы заметите, что даты отличаются. Одна является "динамической", другая - "статической". Вы можете поместить в конструкцию {dynamic}...{/dynamic} любой код и быть уверенным, что он не будет помещён в кэш вместе с остальной частью страницы.

Глава 15. Расширенные возможности

Содержание

Объекты

Smarty позволяет использовать в шаблонах объекты PHP. Существуют два способа их вызова. Первый - зарегистрировать объект для шаблона, затем вызвать его примерно так же, как и пользовательскую функцию. Второй - присвоить объект шаблону и использовать его, как любую другую присвоенную переменную. Первый метод гораздо аккуратнее и безопаснее, так как у зарегистрированного объекта можно ограничить свойства и методы. Но, в тоже время, зарегистрированный объект нельзя использовать в циклах, нельзя помещать в массив объектов и так далее. Выбор способа за вами, но используйте по возможности первый, чтобы максимально упростить синтаксис шаблона.

В безопасном режиме недоступны приватные методы и функции (имена которых начинаются с "_"). Если существует и метод, и свойство с одинаковыми именами, то будет использован метод.

Вы можете ограничить использование объекта только некоторыми методами и свойствами. Для этого перечислите их в массиве и укажите этот массив третьим параметром при регистрации объекта.

По умолчанию, параметры из шаблона передаются объекту точно так же, как и пользовательской функции. Первым параметром передаётся ассоциативный массив, вторым - объект Smarty. Если вы хотите передавать параметры по одному, как при традиционном обращении с объектами, установите четвёртый параметр вызова в false.

Пример 15.1. использование зарегистрированного или присвоенного объекта

```
<?php
// сам объект

class My_Object() {
        function meth1($params, &$smarty_obj) {
                return "это мой метод meth1";
        }
}

$myobj = new My_Object;
// регистрируем объект (будет доступен по ссылке)
```

```
$smarty->register_object("foobar",$myobj);
// если мы хотим ограничиться определёнными методами или свойствами, перечисляем их при регистрации
$smarty->register_object("foobar",$myobj,array('meth1','meth2','prop1'));
// если мы хотим использовать традиционный способ передачи параметров объекту, регистрируем объект с соответствующим
// флагом, установленным в false
$smarty->register_object("foobar",$myobj,null,false);

// Так же мы можем присвоить объект. Желательно присваивать объект по ссылке.
$smarty->assign_by_ref("myobj", $myobj);

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

Вот так нужно обращаться к вашим объектам в index.tpl:

```
{* вызываем зарегистрированный объект *}
{foobar->meth1 p1="foo" p2=$bar}

{* результат можно поместить в переменную *}
{foobar->meth1 p1="foo" p2=$bar assign="output"}
в результате получаем {$output}

{* вызываем присвоенный объект *}
{$myobj->meth1("foo",$bar)}
```

Префильтры

Префильтры шаблона - это функции PHP, которые обрабатывают шаблон перед его компиляцией. Это удобно для удаления лишних комментариев и прочих ненужных после компиляции данных.

Префильтры могут быть или заргистрированы или загружены из папки с плагинами с помощью функции load_filter() или с помощью установки переменной $autoload_filters.

Smarty передаёт фильтру исходный код шаблона в качестве первого аргумента и предполагает, что функция вернёт результат своей работы.

Пример 15.2. использование префильтра

```
<?php
// код в вашем скрипте
function remove_dw_comments($tpl_source, &$smarty)
{
    return preg_replace("/<!--#.*-->/U","",$tpl_source);
}

// регистрация префильтра
$smarty->register_prefilter("remove_dw_comments");
$smarty->display("index.tpl");
?>
```

Это удалит все комментарии из исходного текста шаблона.

Постфильтры

Постфильтры шаблона - это функции PHP, которые обрабатывают шаблон после его компиляции. Постфильтры могут быть или зарегистрированы или загружены из папки с плагинами при помощи функции load_filter(), или с помощью установки переменной $autoload_filters. Smarty передаёт фильтру скомпилированный код шаблона в качестве первого аргумента и предполагает, что функция вернёт результат своей работы.

Пример 15.3. использование постфильтра

```
<?php
// код в вашем скрипте
function add_header_comment($tpl_source, &$smarty)
{
    return "<?php echo \"<!-- Создано при помощи Smarty! -->;\n\"; ?>\n".$tpl_source;
}

// регистрация постфильтра
$smarty->register_postfilter("add_header_comment");
$smarty->display("index.tpl");
?>
```

Теперь скомпилированный шаблон Smarty index.tpl выглядит так:

```
<!-- Создано при помощи Smarty! -->
{* остальной код шаблона... *}
```

Фильтры вывода

Когда шаблон выводится через display() или fetch(), результат может быть пропущен через один или несколько фильтров вывода. Отличие их от постфильтров состоит в том, что постфильтры действуют на уже скомпилированный шаблон, перед его записью на диск, в то время как фильтры вывода обрабатывают шаблон в момент его исполнения.

Фильтры вывода могут быть или зарегистрированы или загружены из папки с плагинами с помощью функции load_filter(), или с помощью установки переменной $autoload_filters. Smarty передаёт фильтру результат обработки шаблона в качестве первого аргумента и предполагает, что функция вернёт результат своей работы.

Пример 15.4. Использование фильтра вывода

```
<?php
// код в вашем скрипте
function protect_email($tpl_output, &$smarty)
```

```
{
    $tpl_output =
        preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.([a-zA-Z]{2,3}|[0-9]{1,3}))!',
                    '$1%40$2', $tpl_output);
    return $tpl_output;
}

// регистрация фильтра вывода
$smarty->register_outputfilter("protect_email");
$smarty->display("index.tpl");

// теперь все адреса электронной почты в выводе шаблона будут
// обработаны несложной функцией защиты от спам-ботов
?>
```

Управление кэшированием

Вместо стандартного механизма кэширования, использующего файлы, вы можете использовать свои функции для чтения, записи и очистки кэшированных шаблонов.

Добавьте в ваше приложение функцию, которую Smarty сможет использовать для управления кэшем. Укажите её имя в переменной класса $cache_handler_func. Теперь Smarty будет использовать её для операций с кэшированным содержимым. Первый параметр вашей функции - действие, принимает значения 'read', 'write' или 'clear' (соответственно, 'прочитать', 'записать' или 'очистить'). Вторым параметром передаётся объект smarty. Третьим - данные для кэширования. Третий параметр используется только при чтении и записи. При записи Smarty передаёт через него кэшированный контент. При чтении предполагается, что через него передаётся ссылка на переменную, в которую контент будет загружен. При очистке значение третьего параметра не обрабатывается. Четвёртый параметр - имя файла с шаблоном (используется при чтении/записи), пятый - идентификатор кэша (опционально), шестой - идентификатор компиляции (опционально, используется для построения разных кэшей для одного шаблона), седьмой - срок годности кэша (опционально). Примечание: последний параметр ($exp_time) добавлен в Smarty 2.6.0.

Пример 15.5. Применение MySQL в качестве хранилища кэшированных данных

```
<?php
/*

пример использования:

include('Smarty.class.php');
include('mysql_cache_handler.php');

$smarty = new Smarty;
$smarty->cache_handler_func = 'mysql_cache_handler';

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


код для MySQL таблицы:
```

```
create database SMARTY_CACHE;

create table CACHE_PAGES(
CacheID char(32) PRIMARY KEY,
CacheContents MEDIUMTEXT NOT NULL
);

*/

function mysql_cache_handler($action, &$smarty_obj, &$cache_content, $tpl_file=null, $cache_id=null, $compile_id=null, $exp_time=null)
{
	// параметры подключения к базе данных - хост, логин, пароль, название базы
	$db_host = 'localhost';
	$db_user = 'myuser';
	$db_pass = 'mypass';
	$db_name = 'SMARTY_CACHE';
	// установите в true для использования gzip компрессии кэшированных данных
	$use_gzip = false;

	// создаём уникальный идентификатор кэша
	$CacheID = md5($tpl_file.$cache_id.$compile_id);

	if(! $link = mysql_pconnect($db_host, $db_user, $db_pass)) {
		$smarty_obj->_trigger_error_msg("cache_handler: не могу подключиться к базе данных");
		return false;
	}
	mysql_select_db($db_name);

	switch ($action) {
		case 'read':
			// чтение кэша из базы
			$results = mysql_query("select CacheContents from CACHE_PAGES where CacheID='$CacheID'");
			if(!$results) {
				$smarty_obj->_trigger_error_msg("ошибка кэша: неверный запрос.");
			}
			$row = mysql_fetch_array($results,MYSQL_ASSOC);

			if($use_gzip && function_exists("gzuncompress")) {
				$cache_content = gzuncompress($row["CacheContents"]);
			} else {
				$cache_content = $row["CacheContents"];
			}
			$return = $results;
			break;
		case 'write':
			// сохранение кэша в базе

			if($use_gzip && function_exists("gzcompress")) {
				// сжимаем контент чтобы сэкономить место
				$contents = gzcompress($cache_content);
			} else {
				$contents = $cache_content;
			}
			$results = mysql_query("replace into CACHE_PAGES values(
					'$CacheID',
```

```
                                        '".addslashes($contents)."')
                                    ");
                if(!$results) {
                    $smarty_obj->_trigger_error_msg("ошибка кэша: неверный запрос.");
                }
                $return = $results;
                break;
            case 'clear':
                // очистка кэша
                if(empty($cache_id) && empty($compile_id) && empty($tpl_file)) {
                    // clear them all
                    $results = mysql_query("delete from CACHE_PAGES");
                } else {
                    $results = mysql_query("delete from CACHE_PAGES where CacheID='$CacheID'");
                }
                if(!$results) {
                    $smarty_obj->_trigger_error_msg("ошибка кэша: неверный запрос.");
                }
                $return = $results;
                break;
            default:
                // ошибка, указан неизвестный метод
                $smarty_obj->_trigger_error_msg("ошибка кэша: неизвестный метод \"$action\"");
                $return = false;
                break;
        }
        mysql_close($link);
        return $return;

}

?>
```

Ресурсы

Шаблоны можно получать из самых разных источников. Когда вы отображаете, вызываете или подключаете один шаблон из другого, вы указываете тип ресурса, вместе с соответствующим путём и названием шаблона.

Шаблоны из папки $template_dir

Шаблоны, лежащие в папке $template_dir, не требуют при вызове указания типа ресурса, хотя вы можете использовать префикс file: для сохранения стиля. Для вызова просто укажите относительный от $template_dir путь к шаблону.

Пример 15.6. Вызов шаблона из папки $template_dir

```
<?php
$smarty->display("index.tpl");
```

```
$smarty->display("admin/menu.tpl");
$smarty->display("file:admin/menu.tpl"); // тоже самое, что и строкой выше

{* код в шаблоне *}
{include file="index.tpl"}
{include file="file:index.tpl"} {* тоже самое, что и строкой выше *}
```

Шаблоны из произвольной папки

Для вызова шаблонов из папки вне $template_dir необходимо использовать префикс file: с последующим указанием асболютного пути и имени шаблона.

Пример 15.7. Вызов шаблона из произвольной папки

```
// PHP скрипт
$smarty->display("file:/export/templates/index.tpl");
$smarty->display("file:/path/to/my/templates/menu.tpl");
?>
```

А изнутри шаблона Smarty:

```
{include file="file:/usr/local/share/templates/navigation.tpl"}
```

Файловые пути в Windows

Если вы работаете под Windows, то пути к файлам, как правило, начинаются с буквы логического диска (например, C:). Не забудьте указать префикс "file:" в начале пути, чтобы избежать конфликтов имён и достичь необходимого результата.

Пример 15.8. использование шаблонов с файловіми путями Windows

```
<?php
// PHP скрипт
$smarty->display("file:C:/export/templates/index.tpl");
$smarty->display("file:F:/path/to/my/templates/menu.tpl");
?>
```

А изнутри шаблона Smarty:

```
{include file="file:D:/usr/local/share/templates/navigation.tpl"}
```

Шаблоны из прочих источников

Вы можете вызывать шаблоны, используя любые доступные через PHP источники: базы данных, сокеты,

LDAP и так далее. Для этого нужно написать соответствующий плагин ресурса и зарегистрировать его в Smarty.

Смотрите раздел плагины ресурсов для более подробной информации о тех функциях, которые вы должны предоставить.

> **Замечание:** Обратите внимание на то, что вы не можете переопределить встроенный ресурс file, но в ваших силах написать и зарегистрировать ресурс с другим именем, который будет использовать другой способ вызова шаблонов из файловой системы.

Пример 15.9. Использование собственных ресурсов

```
// код в вашем скрипте
function db_get_template ($tpl_name, &$tpl_source, &$smarty_obj)
{
    // обращаемся к базе, запрашиваем код шаблона,
    // перегружаем его в $tpl_source
    $sql = new SQL;
    $sql->query("select tpl_source
                 from my_table
                where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_source = $sql->record['tpl_source'];
        return true;
    } else {
        return false;
    }
}

function db_get_timestamp($tpl_name, &$tpl_timestamp, &$smarty_obj)
{
    // обращаемся к базе, запрашиваем поле $tpl_timestamp.
    $sql = new SQL;
    $sql->query("select tpl_timestamp
                 from my_table
                where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_timestamp = $sql->record['tpl_timestamp'];
        return true;
    } else {
        return false;
    }
}

function db_get_secure($tpl_name, &$smarty_obj)
{
    // предполагаем, что наши шаблоны совершенно безопасны
    return true;
}

function db_get_trusted($tpl_name, &$smarty_obj)
{
    // не используется для шаблонов
```

```
}

// регистрируем ресурс под именем "db"
$smarty->register_resource("db", array("db_get_template",
                                       "db_get_timestamp",
                                       "db_get_secure",
                                       "db_get_trusted"));

// используем ресурс из PHP скрипта
$smarty->display("db:index.tpl");
?>
```

А изнутри шаблона Smarty:

```
{include file="db:/extras/navigation.tpl"}
```

Функция для обработки шаблона по умолчанию

Вы можете определить функцию, которая будет использована, если шаблон не может быть вызван из соответствующего ресурса. Это можно использовать, к примеру, для построения недостающего шаблона на лету.

Пример 15.10. использование функции для обработки шаблона по умолчанию

```
<?php
// код в вашем скрипте

function make_template ($resource_type, $resource_name, &$template_source, &$template_timestamp, &$smarty_obj)
{
        if( $resource_type == 'file' ) {
                if ( ! is_readable ( $resource_name )) {
                        // создаём и записываем файл шаблона.
                        $template_source = "Это новый шаблон.";
                        $template_timestamp = time();
                        $smarty_obj->_write_file($resource_name,$template_source);
                        return true;
                }
    } else {
                // не файл
                return false;
    }
}

// определение обработчика
$smarty->default_template_handler_func = 'make_template';
?>
```

Глава 16. Плагины - расширение функциональности Smarty

Содержание

Архитектура версии 2.0 позволяет внедрять плагины, которыми являются практически все настраиваемые элементы функционала Smarty. Сюда входят:

- функции
- модификаторы
- блоковые функции
- функции компилятора
- префильтры
- постфильтры
- фильтры вывода
- ресурсы
- вставки

За исключением ресурсов, в целях обратной совместимости с предыдущими версиями, сохранена возможность регистрации функций посредством register_* API. Если вы не используете API, а вместо этого модифицируете $custom_funcs, $custom_mods и некоторые другие переменные напрямую, тогда вам придется подогнать ваши скрипты под использование API или преобразовать добавленную вами функциональность в плагины.

Как работают плагины

Плагины загружаются только при необходимости. Загруженными окажутся только только те модификаторы, функции,ресурсы и т.п., которые определены в скрипте шаблона. Более того, каждый плагин загружается лишь один раз, даже если у вас имеется несколько различных экземпляров объекта Smarty, выполняемых внутри одного запроса.

Пре/постфильтры и фильтры вывода - это отдельный случай. Так как они не упоминаются в шаблонах, их необходимо зарегистрировать или явно загрузить с помощью API-функций перед обработкой шаблона. Порядок исполнения множественных фильтров зависит от порядка, в котором они были зарегистрированы или загружены.

В целях оптимизации производительности, под плагины отведена одна специальная директория. Чтобы установить плагин, просто поместите его в эту директорию и Smarty будет использовать его в автоматическом режиме.

Как работают плагины

Плагины загружаются только по необходимости. Только те модификаторы, функции, ресурсы и т.д., которые используются в шаблоне, будут загружены. К тому же, каждый плагин загружается только один раз, даже если у вас есть несколько экземпляров Smarty, работающих в пределах одного запроса.

Пре/постфильтры и фильтры вывода заслуживают отдельного упоминания. Так как они не упоминаются в шаблонах, они должны быть зарегистрированы и загружены неявно через API-функции ещё до обработки шаблона. Порядок выполнения нескольких фильтров одного типа зависит от порядка, в котором они регистрировались или загружались.

Директория плагинов может быть строкой, содержащей путь, или массивом, содержащим множество путей. Чтобы установить плагин, просто поместите его в одну из этих директорий и Smarty автоматически будет его использовать.

Соглашение об именах

При присвоении имен файлам и функциям плагинов, необходимо придерживаться определенных правил, чтобы Smarty находил и мог использовать эти плагины.

Имена файлов плагинов должны формироваться по следующей схеме:

type.name.php

Где type (тип) это один из следующих типов плагинов:

- function
- modifier
- block
- compiler
- prefilter
- postfilter
- outputfilter
- resource
- insert

и name (имя) соответствует правилам наименования идентификаторов в PHP (только буквы, цифры и знак подчеркивания).

Несколько примеров: function.html_select_date.php, resource.db.php, modifier.spacify.php.

Функции, находящиеся внутри файлов плагинов, должны именоваться следующим образом:

smarty_*type*, _*name*()

Значения type и name те же, что прежде.

Smarty выдаст сообщение об ошибке, если необходимый файл плагина не будет найден, или файл плагина, а так же функция плагина будут названы неправильно.

Написание плагинов

Smarty может подгружать плагины автоматически из файловой системы или регистрировать их во время выполнения (at runtime) посредством одной из register_* API функций. Их также можно дерегистрировать, используя unregister_* API функции.

Плагинам, которые регистрируются во время выполнения, могут присваиваться имена не соответствующие правилам соглашения об именах.

Если плагин зависит от некоторых функций другого плагина (как в некоторых случаях с плагинами, поставляемыми вместе со Smarty), то такой плагин можно загрузить следующим образом:

```
<?php
require_once $smarty->_get_plugin_filepath('function', 'html_options');
?>
```

Важно знать, что объект Smarty всегда передаётся в плагин последним параметром (за двумя исключениями: модификатором объект Smarty вообще не передаётся, а блоки получают *&$repeat* следом за объектом Smarty в целях обратной совместимости с ранними версиями Smarty).

Функции шаблона

void **smarty_function_*name*()**(*$params*, *&$smarty*);
array *$params*;
object *&$smarty*;

Все атрибуты, передаваемые в функции шаблона из самого шаблона, хранятся в *$params* в виде ассоциативного массива. Получить доступ к его значениям можно напрямую: $params['start'] или используя extract($params) для импорта в таблицу.

Вывод (возвращаемое значение) функции будет подставлен в место расположения тега функции в шаблоне (функция **fetch()** например). В качестве альтернативы, функция может выполнять какие либо действия без какого-либо вывода (**assign()** функция).

Если функция должна присвоить(assign) значения некоторым переменным в шаблоне или использовать иные возможности Smarty, то можно работать с объектом *$smarty* как обычно.

См. также: register_function(), unregister_function().

Пример 16.1. Функция-плагин с выводом

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     function.eightball.php
 * Type:     function
 * Name:     eightball
 * Purpose:  outputs a random magic answer
 * -------------------------------------------------------------
 */
function smarty_function_eightball($params, &amp;$smarty)
{
    $answers = array('Да',
                     'Нет',
                     'Никоим образом',
                     'Перспектива так себе...',
                     'Спросите позже',
                     'Все может быть');

    $result = array_rand($answers);
    return $answers[$result];
}
?>
```

которая может быть использована в шаблоне следующим образом:

```
Вопрос: Мы когда-нибудь найдем время для отпуска?
Ответ: {eightball}.
```

Пример 16.2. Функция-плагин без вывода

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     function.assign.php
 * Type:     function
 * Name:     assign
 * Purpose:  assign a value to a template variable
 * -------------------------------------------------------------
 */
function smarty_function_assign($params, &amp;$smarty)
{
    extract($params);

    if (empty($var)) {
        $smarty->trigger_error("assign: missing 'var' parameter");
        return;
    }
```

```
    if (!in_array('value', array_keys($params))) {
        $smarty->trigger_error("assign: missing 'value' parameter");
        return;
    }

    $smarty->assign($var, $value);
}
?>
```

Модификаторы

Модификаторы - это маленькие функции, которые воздействуют на переменные в шаблоне перед тем, как те будут выведены на экран или использованы в ином контексте. Для каждой переменной шаблона, одновременно могут быть использованы несколько модификаторов.
mixed **smarty_modifier_*name*()**(*$value*, *$param1*);
mixed *$value*;
[mixed *$param1*, ...];

Первый параметр плагина-модификатора это значение в отношении которого модификатор будет применен. Остальные параметры могут быть произвольными, в зависимости от операций, которые они осуществляют.

Модификатор должен возвращать результат, полученный в процессе своего выполнения.

Смотрите также: register_modifier(), unregister_modifier().

Пример 16.3. Простой плагин-модификатор

Этот плагин в своей основе является аналогом одной из PHP-функций. Он не имеет никаких дополнительных параметров.

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     modifier.capitalize.php
 * Тип:      modifier
 * Имя:      capitalize
 * Назначение:  Сделать первую букву каждого слова в
 * строке прописной
 * -------------------------------------------------------------
 */
function smarty_modifier_capitalize($string)
{
    return ucwords($string);
}
?>
```

Пример 16.4. Более сложный модификатор

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     modifier.truncate.php
 * Тип:     modifier
 * Имя:     truncate
 * Назначение:  Урезать строку до определенной длины,
 *           при необходимости обрезать слово на половине и присоеденить строку $etc.
 * -------------------------------------------------------------
 */
function smarty_modifier_truncate($string, $length = 80, $etc = '...',
                                  $break_words = false)
{
    if ($length == 0)
        return '';

    if (strlen($string) > $length) {
        $length -= strlen($etc);
        $fragment = substr($string, 0, $length+1);
        if ($break_words)
            $fragment = substr($fragment, 0, -1);
        else
            $fragment = preg_replace('/\s+(\S+)?$/', '', $fragment);
        return $fragment.$etc;
    } else
        return $string;
}
?>
```

Блоковые функции

void **smarty_block_*name*()**(*$params*, *$content*, *&$smarty*, *&$repeat*);
array *$params*;
mixed *$content*;
object *&$smarty*;
boolean *&$repeat*;

Блоковые функции выглядят следующим образом: {func} .. {/func}. Другими словами, они заключены в определенном блоке шаблона и оперируют содержимым этого блока. Блоковые функции имеют приоритет перед пользовательскими функциями, имеющими то же имя, поэтому, вы не сможете использовать одновременно свои функции вида {func} и блоковые функции {func} .. {/func}.

Smarty вызывает ваши функции дважды: первый раз при открытии тэга и второй раз при закрытии тэга.

Только открывающий тэг блоковой функции может иметь атрибуты. Все атрибуты, переданные в функцию из шаблона сохраняются в *$params* в виде ассоциативного массива. Вы можете получить прямой доступ к их значениям: $params['start'] или использовать extract($params) для импорта. Атрибуты, переданные в откры-

вающем тэге доступны для вашей функции до обработки закрывающего тэга включительно.

Значение переменной *$content* зависит от того, вызывается ли ваша функция для открывающего тэга или вызов происходит при закрытии тэга. В случае с открывающим тэгом, это значение будет равно null, а в случае закрывающего тэга, значение будет равно содержимому блока в шаблоне. Заметьте, что этот блок шаблона уже будет обработан Smarty и на выводе вы получите результат обработки, а не исходный код шаблона.

Параметр *&$repeat* передается по ссылке в функцию и дает ей возможность контролировать количество отображений блока. По умолчанию *$repeat* равен true во время первого вызова блоковой функции (открывающий тэг блока) и false при всех последующих вызовах блоковой функции (закрывающий тэг блока). Каждый раз, когда ваша функция возвращает параметр *&$repeat* равный true, содержимое между {func} .. {/func} обрабатывается и ваша функция вновь вызывается, причем новое содержимое блока передается в параметре *$content*.

Если вы используете вложенные блоковые функции, есть возможность определять родительские блоковые функции. Достаточно получить значение переменной $smarty->_tag_stack. Затем останется только применить var_dump() для нее и структура будет видна.

Смотрите также: register_block(), unregister_block().

Пример 16.5. Блоковая функция

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     block.translate.php
 * Тип:      block
 * Имя:      translate
 * Назначение:  перевести блок (кусок) текста
 * -------------------------------------------------------------
 */
function smarty_block_translate($params, $content, &amp;amp;$smarty)
{
    if ($content) {
        $lang = $params['lang'];
        // здесь выполнить интеллектуальный перевод строки $content
        echo $translation;
    }
}
?>
```

Функции компилятора

Функции компилятора, как вы наверное догадались, вызываются только в процессе компиляции шаблона. Они могут быть полезными для вставки кода PHP или чувствительного ко времени статического контента в шаблон. Если одновременно зарегестрированы две одноименные функции - пользовательская и компилятора, то приоритет будет у функции компилятора.
mixed **smarty_compiler_*name*()**(*$tag_arg*, *&$smarty*);

string *$tag_arg*;
object *&$smarty*;

Функция компилятора имеет два параметра: строку аргументов тэга - чаще всего это все, что следует от наименования функции до правого разделителя, и объект Smarty. Функция должна возвращать PHP-код для вствки в скомпилированный шаблон.

Смотрите также register_compiler_function(), unregister_compiler_function().

Пример 16.6. Простой пример функции компилятора

```
<?php
/*
* Smarty plugin
* -------------------------------------------------------------
* Файл:     compiler.tplheader.php
* Тип:      compiler
* Имя:      tplheader
* Назначение:  вывести заголовок, содержащий имя исходного файла и
*              время, когда он был скомпилирован.
* -------------------------------------------------------------
*/
function smarty_compiler_tplheader($tag_arg, &$smarty)
{
return "\necho '" . $smarty->_current_file . " compiled at " . date('Y-m-d H:M'). "';";
}
?>
```

Эта функция может быть вызвана из шаблона следующим образом:

```
{* Функция выполняется только при компиляции шаблона *}
{tplheader}
```

Результирующий код PHP в скомпилированном шаблоне будет выглядеть примерно так:

```
<?php
echo 'index.tpl compiled at 2002-02-20 20:02';
?>
```

Префильтры/Постфильтры

Концепция плагинов префильтров и постфильтров очень проста; они отличаются местом исполнения, или, точнее, временем их исполнения.
string **smarty_prefilter_*name*()**(*$source*, *&$smarty*);
string *$source*;
object *&$smarty*;

Префильтры используются для обработки исходного кода шаблона непосредственно перед компиляцией. Первый параметр функции префильтра - это исходный код шаблона, который, возможно, уже изменен дру-

гими префильтрами. Такой плагин возвращает модифицированый исходный код. Заметьте, что этот исходный код нигде не сохраняется, он используется только для компиляции.
string **smarty_postfilter_*name*()**(*$compiled*, *&$smarty*);
string *$compiled*;
object *&$smarty*;

Постфильтры используются для обработки скомпилированного вывода шаблона (по сути - PHP-кода) сразу по завершению компиляции, но перед сохранением откомпилированного шаблона в файловой системе. Первым параметром функции постфильтра является скомпилированный код шаблона, возможно уже модифицированый другими постфильтрами. Плагин возвращает модифицированную версию этого кода.

Пример 16.7. Плагин префильтра

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     prefilter.pre01.php
 * Тип:      prefilter
 * Имя:      pre01
 * Назначение:  Перевести все тэги html в нижний регистр.
 * -------------------------------------------------------------
 */
 function smarty_prefilter_pre01($source, &$smarty)
 {
    return preg_replace('!<(\w+)[^>;]+&gt;!e', 'strtolower("$1")', $source);
 }
 ?>
```

Пример 16.8. Плагин постфильтра

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     postfilter.post01.php
 * Тип:      postfilter
 * Имя:      post01
 * Назначение:  Вывести код, перечисляющий все текущие
 * переменные шаблона.
 * -------------------------------------------------------------
 */
 function smarty_postfilter_post01($compiled, &$smarty)
 {
     $compiled = "<pre>\n<?php print_r(\$this->get_template_vars()); ?>\n</pre>" . $compiled;
     return $compiled;
 }
?>
```

Фильтры вывода

Плагины фильтров вывода оперирут выходным кодом шаблона после того, как шаблон был загружен и обработан, но перед его выводом в браузер.
string **smarty_outputfilter_*name*()**(*$template_output*, *&$smarty*);
string *$template_output*;
object *&$smarty*;

Первый параметр функции фильтра вывода - это выходной код шаблона который должен быть обработан, а второй параметр - это экземпляр объекта Smarty, вызвавший этот плагин. Плагин предназначен для обработки и возврата результата.

Пример 16.9. Плагин фильтра вывода

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     outputfilter.protect_email.php
 * Тип:     outputfilter
 * Имя:     protect_email
 * Назначение:  Конвертировать символ @ в адресах email в %40 для
 *           простейшей защиты от спамботов.
 * -------------------------------------------------------------
 */
function smarty_outputfilter_protect_email($output, &$smarty)
{
    return preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.([a-zA-Z]{2,3}|[0-9]{1,3}))!',
                        '$1%40$2', $output);
}
?>
```

Ресурсы

Плагины ресурсов описывают источники данных, из которых берется исходный код шаблона или компоненты PHP-скрипта для Smarty. Вот примеры ресурсов: базы данных, LDAP, разделяемая память (shared memory), сокеты, и прочее.

Необходимо 4 функции для того, чтобы зарегестрировать каждый тип ресурса. Каждая такая функция получает запрашиваемый ресурс в качестве первого параметра и объект Smarty как последний параметр. Остальные параметры зависят от функции.
bool **smarty_resource_*name*_source()**(*$rsrc_name*, *&$source*, *&$smarty*);
string *$rsrc_name*;
string *&$source*;
object *&$smarty*;
bool **smarty_resource_*name*_timestamp()**(*$rsrc_name*, *&$timestamp*, *&$smarty*);
string *$rsrc_name*;
int *&$timestamp*;

object *&$smarty*;
bool **smarty_resource_*name*_secure()**(*$rsrc_name*, *&$smarty*);
string *$rsrc_name*;
object *&$smarty*;
bool **smarty_resource_*name*_trusted()**(*$rsrc_name*, *&$smarty*);
string *$rsrc_name*;
object *&$smarty*;

Первая функция получает ресурс. Ее первый параметр, это переменная, переданная по ссылке. В нее будет сохранен результат. Функция вернет true если сможет удачно получить ресурс и false в ином случае.

Вторая функция получает время последней модификации запрошенного ресурса (в виде UNIX timestamp). Второй параметр представляет собой переменную, переданную по ссылке, в которой и будет сохранено время. Функция вернет true если timestamp будет определен в правильной форме, и false в ином случае.

Третья функция возвращает true или false в зависимости от того, является ли запрашиваемый ресурс безопасным. Эта функция используется только для ресурсов шаблона, но в любом случае должна быть определена.

Четвертая функция возвращает true или false в зависимости от того, заслуживает ли запрашиваемый ресурс доверия (is trusted) или нет. Эта функция используется только для компонентов PHP-скрипта, запрошенных тэгом **include_php** или **insert** с *src* атрибутом. Тем не менее, она должна объявляться даже для ресурсов шаблона.

Смотрите также: register_resource(), unregister_resource().

Пример 16.10. Плагин ресурса

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     resource.db.php
 * Тип:      resource
 * Имя:      db
 * Назначение:  Получает шаблон из базы данных
 * -------------------------------------------------------------
 */
 function smarty_resource_db_source($tpl_name, &$tpl_source, &$smarty)
{
    // выполняем обращение к базе данных для получения шаблона
    // и занесения полученного результата в в $tpl_source
    $sql = new SQL;
    $sql->query("select tpl_source
                   from my_table
                  where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_source = $sql->record['tpl_source'];
        return true;
    } else {
        return false;
    }
}
```

```
function smarty_resource_db_timestamp($tpl_name, &$tpl_timestamp, &$smarty)
{
    // выполняем обращение к базе данных для присвоения значения $tpl_timestamp.
    $sql = new SQL;
    $sql->query("select tpl_timestamp
                 from my_table
                where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_timestamp = $sql->record['tpl_timestamp'];
        return true;
    } else {
        return false;
    }
}

function smarty_resource_db_secure($tpl_name, &$smarty)
{
    // предполагаем, что шаблоны безопасны
    return true;
}

function smarty_resource_db_trusted($tpl_name, &$smarty)
{
    // не используется для шаблонов
}
?>
```

Вставки

Плагины вставок используются для исполнения функций, вызываемых тэгом **insert** в шаблоне.
string **smarty_insert_*name*()**(*$params*, *&$smarty*);
array *$params*;
object *&$smarty*;

Первый параметр функции представляет собой ассоциативный массив атрибутов, переданых для вставки. Доступ к этим значениям можно получить как напрямую: т.е. $params['start'] так и используя extract($params) для импорта.

Функция вставки возвращает результат, которым будет заменен тэг **insert** в шаблоне.

Пример 16.11. Плагин вставки

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Файл:     insert.time.php
 * Тип:      time
 * Имя:      time
 * Назначение:  Вставка текущей даты/времени в определенном формате
```

```
 * -------------------------------------------------------------
 */
function smarty_insert_time($params, &amp;amp;$smarty)
{
    if (empty($params['format'])) {
        $smarty->trigger_error("insert time: missing 'format' parameter");
        return;
    }

    $datetime = strftime($params['format']);
    return $datetime;
}
?>
```

Часть IV. Приложения

Содержание

Глава 17. Решение проблем

Содержание

Ошибки Smarty/PHP

Smarty может ловить многие ошибки, например отсутствующие атрибуты тэгов или недопустимые имена переменных. Если это произойдет, вы увидите ошибку наподобие следующей:

Пример 17.1. Ошибка Smarty

```
Warning: Smarty: [in index.tpl line 4]: syntax error: unknown tag - '%blah'
     in /path/to/smarty/Smarty.class.php on line 1041

Fatal error: Smarty: [in index.tpl line 28]: syntax error: missing section name
     in /path/to/smarty/Smarty.class.php on line 1041
```

Smarty покажет вам имя шаблона, номер строки и ошибку. Далее сообщение об ошибке состоит из фактического номера строки в классе Smarty, где возникла ошибка.

Есть определенные ошибки, которые не может поймать Smarty, например отсутствующие закрывающие тэги. Такие ошибки обычно приводят к ошибкам разбора PHP на этапе компиляции.

Пример 17.2. Ошибки разбора PHP

```
Parse error: parse error in /path/to/smarty/templates_c/index.tpl.php on line 75
```

Когда вы встречаетесь с ошибкой разбора PHP, номер строки, в которой допущена ошибка, будет соответствовать скомпилированному PHP-скрипту, а не самому шаблону. Обычно вы можете посмотреть на шаблон и увидить синтаксическую ошибку. Типичные ошибки: отсутствующие закрывающие тэги для {if}{/if} или {section}{/section}, или синтаксис логики внутри тэга {if}. Если вы не можете найти ошибку, вам может понадобиться открыть скомпилированный PHP-файл и перейти к номеру строки чтобы выяснить, в чём заключается ошибка в шаблоне.

Пример 17.3. Other common errors

- ```
  Warning: Smarty error: unable to read resource: "index.tpl" in...
  or
  Warning: Smarty error: unable to read resource: "site.conf" in...
  ```

  - Значение $template_dir неверно, эта директория не существует или файл index.tpl находится за пределами директории templates/
  - В шаблоне присутствует функция {config_load} (либо была вызвана функция config_load()) и значение $config_dir неверно, эта директория не существует или файл site.conf находится за пределами этой директории.

- ```
  Fatal error: Smarty error: the $compile_dir 'templates_c' does not exist,
  or is not a directory...
  ```

  Переменная $compile_dir установлена неверно, эта директория не существует или templates_c является файлом, а не директорией.

- ```
  Fatal error: Smarty error: unable to write to $compile_dir '....
  ```

  У веб сервера нет прав на запись в директорию $compile_dir. Смотрите конец страницы Базовая установка для получения информации о правах доступа.

- ```
  Fatal error: Smarty error: the $cache_dir 'cache' does not exist,
  or is not a directory. in /..
  ```

  Это означает, что $caching включено и $cache_dir установлена неправильно, директория не существует или cache является файлом, а не директорией.

- ```
  Fatal error: Smarty error: unable to write to $cache_dir '/...
  ```

  Это означает, что $caching включено и у веб сервера нет прав на запись в директорию $cache_dir. Смотрите конец страницы Базовая установка для получения информации о правах доступа.

См. также debugging, $error_reporting и trigger_error().
  ```

Глава 18. Советы

Содержание

Обработка пустых переменных

Иногда, например, для того чтобы фон таблицы работал корректно, необходимо вывести вместо пустого значения переменной, значение по умолчанию " ". Многие бы использовали конструкцию {if}, но в Smatry есть более короткий путь - используя модификатор переменной *default*.

Пример 18.1. Вывод , если переменная пуста

```
{* длинный путь *}

{if $title eq ""}
  
{else}
        {$title}
{/if}


{* короткий путь *}

{$title|default:" "}
```

См. также default и Обработка переменных по умолчанию.

Обработка переменных по умолчанию

Если переменная встречается часто, то использование модификатора default каждый раз можно избежать, используя функцию {assign}.

Пример 18.2. Назначение переменной шаблона значения по умолчанию

```
{* где-то в начале шаблона *}
{assign var="title" value=$title|default:"no title"}

{* если переменная $title была пустой, то сейчас она содержит "no title" *}
{$title}
```

См. также default и Обработка пустых переменных.

Присвоение переменной заголовка (title) шаблону-шапке

Если большинство ваших шаблонов имеют похожие верхние и нижние части, то имеет смысл вынести их в отдельные файлы. Но как быть, если шапка должна иметь различные заголовки на различных страницах? Вы можете передавать текст заголовка шапке в момент её включения.

Пример 18.3. Присвоение переменной заголовка (title) шаблону-шапке

mainpage.tpl

```
{include file="header.tpl" title="Main Page"}
{* тут находится тело шаблона *}
{include file="footer.tpl"}
```

archives.tpl

```
{config_load file="archive_page.conf"}
{include file="header.tpl" title=#archivePageTitle#}
{* тут находится тело шаблона *}
{include file="footer.tpl"}
```

header.tpl

```
<html>
 <head>
  <title>{$title|default:"BC News"}</title>
 </head>
 <body>
```

footer.tpl

```
</body>
</html>
```

При отображении главной страниц, строка "Main Page" передаётся в header.tpl и в последствии используется как заголовок. При отображении страницы архива, заголовком будет "Archives". Обратите внимание, что в примере с архивом мы используем переменную из файла archives_page.conf, вместо того, чтобы жёстко задать

её в шаблоне. Также обратите внимание, что в случае, если переменная $title не задана, при помощи модификатора переменной default заголовком будет "BC News".

Даты

Обычно даты в Smarty всегда передаются как временные метки, что позволяет проектировщикам шаблонов использовать date_format для полного контроля над форматированием даты и также делает легким сравнение дат там, где это необходимо.

> **Замечание:** Начиная с версии Smarty 1.4.0, вы можете передавать даты в Smarty в виде меток времени Unix (unix timestamps), mysql, или в любом другом виде, который понимает функция strtotime() [http://php.net/strtotime].

Пример 18.4. using date_format

```
{$startDate|date_format}
```

Результат работы:

```
Jan 4, 2001
```

```
{$startDate|date_format:"%Y/%m/%d"}
```

Результат работы:

```
2001/01/04
```

```
{if $date1 < $date2}
    ...
{/if}
```

Когда {html_select_date} используется в шаблоне, программист наверняка захочет преобразовать данные из формы назад в формат метки времени. Вот функция, которая поможет вам сделать это.

Пример 18.5. Преобразование элементов формы ввода даты назад к метке времени

```
<?php

// Предполагается, что ваши элементы формы названы
// startDate_Day, startDate_Month, startDate_Year

$startDate = makeTimeStamp($startDate_Year, $startDate_Month, $startDate_Day);

function makeTimeStamp($year='', $month='', $day='')
{
   if(empty($year)) {
       $year = strftime("%Y");
```

```
    }
    if(empty($month)) {
        $month = strftime("%m");
    }
    if(empty($day)) {
        $day = strftime("%d");
    }

    return mktime(0, 0, 0, $month, $day, $year);
}
?>
```

См. также {html_select_date}, {html_select_time}, date_format и $smarty.now

WAP/WML

WAP/WML шаблоны требуют, чтобы заголовок Content-type [http://php.net/header] был передан вместе с шаблоном. Простейший путь - написать пользовательскую функцию, которая будет выводить заголовки. Если вы используете кэширование, это не сработает, так что мы сделаем это с помощью тега {insert}; не забывайте, что теги {insert} не кэшируются! Убедитесь, что перед шаблоном в браузер ничего не выводится, иначе отправить заголовок не получится.

Пример 18.6. Использование {insert} для записи заголовка Content-Type для WML

```
<?php

// убедитесь, что apache настроен на обработку расширений .wml!
// добавьте эту функцию в своё приложение или в Smarty.addons.php
function insert_header($params)
{
    // эта функция ожидает аргумент $content
    if (empty($params['content'])) {
        return;
    }
    header($params['content']);
    return;
}

?>
```

ваш шаблон Smarty *должен* начинаться с тэга insert:

```
{insert name=header content="Content-Type: text/vnd.wap.wml"}

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">

<!-- begin new wml deck -->
<wml>
 <!-- begin first card -->
```

```
<card>
 <do type="accept">
  <go href="#two"/>
 </do>
 <p>
  Welcome to WAP with Smarty!
  Press OK to continue...
 </p>
</card>
<!-- begin second card -->
<card id="two">
 <p>
  Pretty easy isn't it?
 </p>
</card>
</wml>
```

Составные шаблоны

По традиции, программирование шаблонов в вашем приложении идёт следующим путём: Сначала вы формируете переменные внутри вашего приложения PHP (возможно, используя запросы к базе данных). Затем вы создаёте экземпляр объекта Smarty, назначаете переменные и отображаете шаблон. Давайте представим себе такую ситуацию: К примеру, у нас есть котировщик ценных бумаг в нашем шаблоне. Мы собираем данные о котировках ценных бумаг в нашем приложении, затем передаём эти переменные в шаблон и отображаем его. Правда, было бы здорово, если бы этот котировщик можно было перенести в другое приложение, просто подключив к нему шаблон, не беспокоясь об источнике данных.

Вы можете сделать это, написав собственное расширение для получения данных и присваивания их переменной шаблона.

Пример 18.7. составной шаблон

function.load_ticker.php - поместите файл в директорию $plugins

```
<?php

// настраиваем нашу функцию для получения информации о ценных бумагах
function fetch_ticker($symbol)
{
   // здесь находится логика формирования $ticker_info
   // из какого-то источника
   return $ticker_info;
}

function smarty_function_load_ticker($params, &$smarty)
{
   // вызываем функцию
   $ticker_info = fetch_ticker($params['symbol']);

   // присваиваем переменную шаблона
```

```
    $smarty->assign($params['assign'], $ticker_info);
}
?>
```

index.tpl

```
{load_ticker symbol="YHOO" assign="ticker"}

Stock Name: {$ticker.name} Stock Price: {$ticker.price}
```

См. также {include_php}, {include} и {php}.

Сокрытие E-mail адреса

Вы когда-нибудь удивлялись, как ваш e-mail адрес попадает в такое количество спамерских рассылок? Один из способов сбора e-mail адресов заключается в просмотре веб-страниц. Чтобы помочь предотвратить эту проблему, вы можете сделать так, чотбы ваш e-mail адрес отображался в скрытом за javascript'ом виде в HTML-исходниках, в то же время выглядя и работая корректно в браузере. Это можно совершить при помощи расширения {mailto}.

Пример 18.8. Пример сокрытия e-mail адреса

```
{* в index.tpl *}

По вопросам обращайтесь на
{mailto address=$EmailAddress encode="javascript" subject="Hello"}
```

> **Техническое Замечание:** Этот метод не может гарантировать 100% защиты. Существует вероятность, что спамер запрограммирует свой сборщик e-mail адресов на раскодирование этих значений, но это маловероятно... будем надеяться.

См. также escape и {mailto}.

Глава 19. Источники

Домашняя страница Smarty: http://smarty.php.net/Для подписки на новости, надо послать письмо на smarty-general-subscribe@lists.php.net. Архив подписки доступен на http:/ / marc.theaimsgroup.com/ ?l=smarty-general&r=1&w=2.

Глава 20. Ошибки

Смотрите файл BUGS, который поставляется вместе с стандартной поставкой Smarty или ищите список на сайте.