

Руководство по Smarty

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

Руководство по Smarty

Опубликовано 31-03-2005
Copyright © 2001-2005 New Digital Group, Inc.

Содержание

Предисловие	vii
I. Приступая к работе	1
1. Что такое Smarty?	2
2. Установка	4
Требования	4
Базовая установка	4
Расширенная установка	8
II. Smarty для дизайнеров шаблонов	10
3. Базовый синтаксис	12
Комментарии	12
Переменные	12
Функции	13
Параметры	14
Внедренные переменные в двойных кавычках	14
Арифметические операции	15
Предотвращение обработки Smarty	15
4. Переменные	17
Переменные, установленные в PHP	17
Переменные файлов конфигурации	19
Зарезервированная переменная {Smarty}	20
5. Модификаторы переменных	23
capitalize	24
count_characters	25
cat	25
count_paragraphs	26
count_sentences	26
count_words	27
date_format	27
default	30
escape	30
indent	31
lower	32
nl2br	33
regex_replace	33
replace	34
spacify	34
string_format	35
strip	36
strip_tags	36
truncate	37
upper	38
wordwrap	38
6. Комбинирование модификаторов	40
7. Встроенные функции	41
capture	41

config_load	42
foreach,foreachelse	43
include	45
include_php	47
insert	48
if,elseif,else	49
ldelim,rdelim	51
literal	51
php	52
section,sectionelse	52
strip	62
8. Пользовательские Функции	63
assign	63
counter	64
cycle	65
debug	66
eval	66
fetch	67
html_checkboxes	68
html_image	69
html_options	71
html_radios	72
html_select_date	74
html_select_time	77
html_table	81
math	84
mailto	85
popup_init	87
popup	87
textformat	91
9. Конфигурационные файлы	96
10. Отладочная консоль	98
III. Smarty для программистов	99
11. Константы	101
SMARTY_DIR	101
SMARTY_CORE_DIR	101
12. Переменные	102
\$template_dir	102
\$compile_dir	103
\$config_dir	103
\$plugins_dir	103
\$debugging	103
\$debug_tpl	103
\$debugging_ctrl	104
\$autoload_filters	104
\$compile_check	104
\$force_compile	104
\$caching	104
\$cache_dir	105
\$cache_lifetime	105

\$cache_handler_func	105
\$cache_modified_check	105
\$config_overwrite	106
\$config_booleanize	106
\$config_read_hidden	106
\$config_fix_newlines	106
\$default_template_handler_func	106
\$php_handling	106
\$security	107
\$secure_dir	107
\$security_settings	107
\$trusted_dir	108
\$left_delimiter	108
\$right_delimiter	108
\$compiler_class	108
\$request_vars_order	108
\$request_use_auto_globals	108
\$error_reporting	108
\$compile_id	108
\$use_sub_dirs	108
\$default_modifiers	109
\$default_resource_type	109
13. Методы	110
14. Кэширование	152
Настройка кэширования	152
Множественное кэширование страниц	154
Групповое кэширование	156
Управление кэшированием результатов работы плагинов	157
15. Расширенные возможности	159
Объекты	159
Префильтры	160
Постфильтры	161
Фильтры вывода	161
Управление кэшированием	162
Ресурсы	164
16. Плагины - расширение функционала Smarty	168
Как работают плагины	168
Как работают плагины	169
Соглашение об именах	169
Написание плагинов	170
Функции шаблона	170
Модификаторы	172
Блочные функции	173
Функции компилятора	174
Префильтры/Постфильтры	175
Фильтры вывода	177
Ресурсы	177
Вставки	179
IV. Приложения	181
17. Решение проблем	182

Ошибки Smarty/PHP	182
18. Советы	183
Обработка пустых переменных	183
Обработка переменных по умолчанию	183
Присвоение переменной заголовка (title) заголовку шаблона.	184
Даты	185
WAP/WML	186
Составные шаблоны	187
Скрытие E-mail адреса	188
19. Источники	189
20. Ошибки	190

Предисловие

Несомненно, один из наиболее часто задаваемых вопросов в списках рассылки РНР - "Как мне сделать свои РНР-скрипты независимыми от дизайна?". Хотя РНР называют "скриптовым языком, встраиваемым в HTML", после написания нескольких проектов, в которых РНР и HTML свободно перемешиваются, многие понимают, что отделение формы от содержания - это Хорошая Вещь [ТМ]. 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 РНР 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 РНР-коде. Дизайнерам нужна поддержка конфигурационных файлов, динамических блоков и прочих интерфейсных нюансов, но они не хотят иметь дела со сложностями языка программирования РНР.

Глядя на множество обработчиков шаблонов, доступных сегодня для РНР, 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 РНР. 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 РНР 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 РНР-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 РНР-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 РНР-код, если его не перегружать обработкой шаблонов. Мы также знали, как всеобъемлюще и непонятно может выглядеть язык РНР 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

1. Что такое Smarty?	2
2. Установка	4
Требования	4
Базовая установка	4
Расширенная установка	8

Глава 1. Что такое Smarty?

Smarty - это компилирующий обработчик шаблонов для PHP и на PHP. Говоря более четко, он предоставляет один из инструментов, которые позволяют 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` конструкции 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	4
Базовая установка	4
Расширенная установка	8

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 директории `/libs/` дистрибутива. Редактировать эти файлы НЕ СЛЕДУЕТ. Они должны использоваться всеми приложениями и изменяться только при обновлении Smarty до новой версии.

Пример 2.1. Файлы библиотеки Smarty

```
Smarty.class.php
Smarty_Compiler.class.php
Config_File.class.php
debug.tpl
/plugins/*.php (все файлы)
/internals/*.php (все файлы)
```

Smarty использует константу PHP `SMARTY_DIR`, которая указывает путь к директории библиотеки Smarty. Обычно, если приложение может найти файл `Smarty.class.php`, то нет необходимости устанавливать `SMARTY_DIR`. 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 объекта Smarty

```
<?php
require('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. Укажите абсолютный путь к директории Smarty

```
<?php
require('/usr/local/lib/php/Smarty/Smarty.class.php');
$smarty = new Smarty;
?>
```

Пример 2.4. Добавьте директорию Smarty к `include_path`

```
<?php
// Отредактируйте ваш файл php.ini, добавьте путь к Smarty
// в include_path и перезапустите веб сервер.
// Следующий код должен работать без ошибок:
require('Smarty.class.php');
$smarty = new Smarty;
?>
```

Пример 2.5. Установите значение константы `SMARTY_DIR` вручную

```
<?php
define('SMARTY_DIR','/usr/local/lib/php/Smarty/');
require(SMARTY_DIR.'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. Настоячиво 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 директории 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).

Давайте взглянем на текущую файловую структуру:

Пример 2.6. Файловая структура примера

```
/usr/local/lib/php/Smarty/Smarty.class.php
/usr/local/lib/php/Smarty/Smarty_Compiler.class.php
/usr/local/lib/php/Smarty/Config_File.class.php
/usr/local/lib/php/Smarty/debug.tpl
/usr/local/lib/php/Smarty/internals/*.php
/usr/local/lib/php/Smarty/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 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`, который будет загружаться Smarty. Он будет расположен в вашей `$template_dir`.

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

```
{* Smarty *}
Hello, {$name}!
```

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 файл и включать особенную подсветку синтаксиса.

Теперь давайте отредактируем `index.php`. Мы создадим экземпляр Smarty, присвоим значение переменной шаблона и отобразим файл `index.tpl`. В условиях нашего примера, мы поместили `"/usr/local/lib/php/Smarty"` в `include_path`. Убедитесь, что вы сделали то же самое или используете абсолютные пути.

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

```
<?php
// загружаем библиотеку Smarty
require('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','Ned');

$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 загрузите файл `index.php` из вашего веб-браузера. Вы должны увидеть надпись "Hello, Ned!"

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 используйте 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

3. Базовый синтаксис	12
Комментарии	12
Переменные	12
Функции	13
Параметры	14
Внедренные переменные в двойных кавычках	14
Арифметические операции	15
Предотвращение обработки Smarty	15
4. Переменные	17
Переменные, установленные в PHP	17
Переменные файлов конфигурации	19
Зарезервированная переменная { <code>\$smarty</code> }	20
5. Модификаторы переменных	23
<code>capitalize</code>	24
<code>count_characters</code>	25
<code>cat</code>	25
<code>count_paragraphs</code>	26
<code>count_sentences</code>	26
<code>count_words</code>	27
<code>date_format</code>	27
<code>default</code>	30
<code>escape</code>	30
<code>indent</code>	31
<code>lower</code>	32
<code>nl2br</code>	33
<code>regex_replace</code>	33
<code>replace</code>	34
<code>spacify</code>	34
<code>string_format</code>	35
<code>strip</code>	36
<code>strip_tags</code>	36
<code>truncate</code>	37
<code>upper</code>	38
<code>wordwrap</code>	38
6. Комбинирование модификаторов	40
7. Встроенные функции	41
<code>capture</code>	41
<code>config_load</code>	42

foreach,foreachelse	43
include	45
include_php	47
insert	48
if,elseif,else	49
ldelim,rdelim	51
literal	51
php	52
section,sectionelse	52
strip	62
8. Пользовательские Функции	63
assign	63
counter	64
cycle	65
debug	66
eval	66
fetch	67
html_checkboxes	68
html_image	69
html_options	71
html_radios	72
html_select_date	74
html_select_time	77
html_table	81
math	84
mailto	85
popup_init	87
popup	87
textformat	91
9. Конфигурационные файлы	96
10. Отладочная консоль	98

Глава 3. Базовый синтаксис

Содержание

Комментарии	12
Переменные	12
Функции	13
Параметры	14
Внедренные переменные в двойных кавычках	14
Арифметические операции	15
Предотвращение обработки Smarty	15

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 статическое содержание, без изменений. Когда Smarty встречает тэги, 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. Они используются для внутренних примечаний в шаблонах.

Пример 3.1. Комментарии

```
{* шаблон Smarty *}

{* подключаем шапку шаблона *}
{include file="header.tpl"}

{include file=$includeFile}

{include file=#includeFile#}

{* выпадающий список *}
<SELECT name=company>
{html_options values=$vals selected=$selected output=$output}
</SELECT>
```

Переменные

Переменные шаблона начинаются со знака доллара. Они могут состоять из цифр, букв, знаков подчёркивания - как и обычные переменные PHP. 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 долларовой 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}
```

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
```

Функции

Каждый тэг Smarty либо выводит значение переменной, либо вызывает некоторую функцию. Для вызова функции надо название функции и ее параметры заключить в разделители, 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 реализованы внутри Smarty и не могут быть изменены или переопределены. Это такие функции, как **if**, **section**, **strip**. Пользовательские функции реализуются через плагины. Они могут быть изменены по вашему желанию, также вы можете написать новые. Примеры пользовательских функций: **html_options**, **html_select_date**.

Параметры

Большинство функций принимает параметры, которые уточняют или изменяют ее поведение. Параметры в Smarty очень похожи на параметры в HTML. Не обязательно заключать статические значения в кавычки, хотя текст рекомендуется заключать в кавычки. Переменные также могут быть использованы в качестве параметров, и не должны заключаться в кавычки.

Некоторые параметры принимают логические значения (правда или ложь). 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=$includeFile}

{include file=#includeFile#}

{html_select_date display_days=yes}

<SELECT name=company>
{html_options values=$vals selected=$selected output=$output}
</SELECT>
```

Внедренные переменные в двойных кавычках

Smarty распознает переменные, если они встречаются в строках, заключенных в двойные кавычки. Распознаются переменные, состоящие из чисел, букв, символов подчеркивания и квадратных кавычек. Если надо использовать другие символы для обозначения переменной (точка, указатель объекта и т.д.), то 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
```

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" }
```

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'a 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},{smarty.rdelim} 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>
```

Глава 4. Переменные

Содержание

Переменные, установленные в РНР	17
Переменные файлов конфигурации	19
Зарезервированная переменная {Smarty}	20

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

Переменные в Smarty могут быть отображены или использованы как аргументы функций и модификаторов, внутри выражений условных операторов и т.д. Для вывода значения переменной надо просто указать между разделителями имя переменной.

```
[
{$Name}

{$Contacts[row].Phone}

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

Переменные, установленные в РНР

Переменные, установленные в РНР, 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:

```
{$_smarty_tpl->get_variable('Contacts.fax')}<br>
{$_smarty_tpl->get_variable('Contacts.email')}<br>
{* you can print arrays of arrays as well *}
{$_smarty_tpl->get_variable('Contacts.phone.home')}<br>
{$_smarty_tpl->get_variable('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 его индексу. Синтаксис аналогичен PHP.

Пример 4.3. доступ к элементу 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:

```
{$_smarty_tpl->get_variable('Contacts[0]')}<br>
{$_smarty_tpl->get_variable('Contacts[1]')}<br>
{* you can print arrays of arrays as well *}
{$_smarty_tpl->get_variable('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 объектов, 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 конфигурации, необходимо заключить их имя между знаками # или через переменную \$smarty.config. Для употребления их в качестве внедренных 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 `{$_smarty}` может быть использована для получения значений переменных из файлов конфигурации. `{$_smarty.config.foo}` является синонимом для `{#foo#}`. Смотрите раздел `config_load` для примера.

`{$_smarty.section}`, `{$_smarty.foreach}`

Переменная `{$_smarty}` может быть использована для использования 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

capitalize	24
count_characters	25
cat	25
count_paragraphs	26
count_sentences	26
count_words	27
date_format	27
default	30
escape	30
indent	31
lower	32
nl2br	33
regex_replace	33
replace	34
spacify	34
string_format	35
strip	36
strip_tags	36
truncate	37
upper	38
wordwrap	38

Модификаторы переменных могут быть применены 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 элементов в массиве \$articleTitle.

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.
```

count_characters

Возвращает количество символов в строке.

Пример 5.3. 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
```

cat

Позиция параметра	Тип	Обязателен	cat	Описание
1	string	Нет	<i>пусто</i>	Данная строка добавляется к модифицируемому значению.

Данная строка добавляется к модифицируемому значению.

Пример 5.4. 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_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

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.');
```

index.tpl:

```
{ $articleTitle }  
{ $articleTitle | count_sentences }
```

OUTPUT:

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

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

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

index.php:

```
$smarty = new Smarty;
```

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

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"}
```

OUTPUT:

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

Пример 5.9. date_format conversion specifiers

%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 воскресенье 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 хотя бы 4 дня находятся в данном году. Понедельник считается первым днем недели.

`%w` - номер дня недели, где 0 - воскресенье

`%W` - номер недели в году, считая первый понедельник первым днем первой недели.

`%x` - preferred date representation for the current locale without the time

`%X` - preferred time representation for the current locale without the date

`%y` - year as a decimal number without a century (range 00 to 99)

`%Y` - year as a decimal number including the century

`%Z` - time zone or name or abbreviation

`%%` - a literal '%' character

ЗАМЕЧАНИЕ ДЛЯ ПРОГРАММИСТОВ: `date_format` - это обертка для функции `strftime()`, поэтому полный список спецификаторов, соответствующий вашей системе, вы сможете найти в соответствующем системном руководстве.

default

Позиция параметра	Тип	Обязателен	По умолчанию	Описание
1	string	Нет	<i>пусто</i>	Значение по умолчанию для пустой переменной.

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

Пример 5.10. 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 шестнадцатеричный вид (hex), конвертации каждого символа в шестнадцатеричное html представление (hexentity), защита специальных символов javascript.

Пример 5.11. escape

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

index.tpl:

{$articleTitle}
{$articleTitle|escape}
{$articleTitle|escape:"html"}  {* защищает & ' ' < > *}
{$articleTitle|escape:"htmlall"} {* защищает все BCE html объекты *}
{$articleTitle|escape:"url"}
{$articleTitle|escape:"quotes"}
<a
href="mailto:{$EmailAddress|escape:"hex"}">{$EmailAddress|escape:"hexentity"}</a>

OUTPUT:

'Stiff Opposition Expected to Casketless Funeral Plan'
'Stiff%20Opposition%20Expected%20to%20Casketless%20Funeral%20Plan'
'Stiff%20Opposition%20Expected%20to%20Casketless%20Funeral%20Plan'
'Stiff%20Opposition%20Expected%20to%20Casketless%20Funeral%20Plan'
'Stiff+Opposition+Expected+to+Casketless+Funeral+Plan'
\'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>
```

indent

Позиция тра	параме-	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 5.12. indent

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

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 5.13. lower

index.php:

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

index.tpl:

```
{ $articleTitle }
```

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

OUTPUT:

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

nl2br

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

Пример 5.14. 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 тра	параме-	Тип	Обязателен	По умолчанию	Описание
1		string	Да	<i>n/a</i>	Регулярное выражение для замены.
2		string	Да	<i>n/a</i>	Строка для замены.

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

Пример 5.15. 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 тра	параме-	Тип	Обязателен	По умолчанию	Описание
1		string	Да	<i>n/a</i>	Строка для поиска.
2		string	Да	<i>n/a</i>	Строка для замены.

Выполняется простой поиск и замена строки.

Пример 5.16. 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 тра	параме-	Тип	Обязателен	По умолчанию	Описание
1		string	Нет	<i>один пробел</i>	Строка вставляется между каждым символом переменной.

spacify позволяет вставить пробел между каждым символом переменной. Можно также указать другой сим-

вол (или строку) для вставки.

Пример 5.17. `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.

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^

string_format

Позиция тра	параме-	Тип	Обязателен	По умолчанию	Описание
1		string	Да	<i>n/a</i>	Формат. (sprintf)

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

Пример 5.18. `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 5.19. 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:"&nbsp;" }
```

OUTPUT:

```
Grandmother of
eight makes      hole in one.
Grandmother of eight makes hole in one.
Grandmother&nbsp;of&nbsp;eight&nbsp;makes&nbsp;hole&nbsp;in&nbsp;one.
```

strip_tags

Вырезает HTML теги, обычно все между `<` и `>`.

Пример 5.20. 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 New Kidney from Dad she Hasn't Seen in years.
Blind Woman Gets New Kidney from Dad she Hasn't Seen in years.

truncate

Позиция тра	параме-	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 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 5.21. 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 5.22. upper

index.php:

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

index.tpl:

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

OUTPUT:

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

wordwrap

Позиция тра	параме-	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 вставляться на месте переноса (по умолчанию символ новой строки). По умолчанию `wordwrap` пытается переносить по словам, но если указать третьим параметром `true`, то переноситься будет по конкретному символу.

Пример 5.23. `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

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

```
index.php:
<?php
$smarty = new Smarty;
$smarty->assign('articleTitle', 'Smokers are Productive, but Death Cuts Efficiency.');
```

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

```
?>
```

```
index.tpl:
```

```
{ $articleTitle }
```

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

```
{ $articleTitle|lower|spacify|truncate }
```

```
{ $articleTitle|lower|truncate:30|spacify }
```

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

Результат выполнения данного примера:

```
Smokers are Productive, but Death Cuts Efficiency.
```

```
SMOKERS ARE PRODUCTIVE, BUT DEATH CUTS EFFICIENCY.
```

```
smokers are productive, but death cuts...
```

```
smokers are productive, but...
```

```
smokers are p...
```

Глава 7. Встроенные функции

Содержание

capture	41
config_load	42
foreach,foreachelse	43
include	45
include_php	47
insert	48
if,elseif,else	49
ldelim,rdelim	51
literal	51
php	52
section,sectionelse	52
strip	62

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 присвоить вывод шаблона какой-либо переменной вместо его вывода на экран. Любое содержание между {capture name="foo"} и {/capture} сохраняется в переменную, указанную в атрибуте name. Затем его можно использовать в шаблоне с помощью специальной переменной `$smarty.capture.foo`, где `foo` - значение, переданное атрибуту name. Если атрибут name не указан, то используется default. Каждая команда {capture} должна иметь пару {/capture}. capture поддерживает вложение.

Техническое замечание: Smarty 1.4.0 - 1.4.4 помещало захваченный вывод в переменную `$return`. С версии 1.4.5 поведение было изменено на использование атрибута name, так что обновите ваши шаблоны соответственно.

Предостережение

Будьте осторожны, сохраняя вывод команды **insert**. Если вы используете кэширование и встречаетесь команды **insert** в области кэширования, то не сохраняйте данный вывод.

Пример 7.1. Сохранение вывода шаблона

```
{* we don't want to print a table row unless content is displayed *}
{capture name=header}
{include file="get_banner.tpl"}
```

```

{/capture}
{if $smarty.capture.banner ne ""}
    <tr>
        <td>
            {Smarty.capture.banner}
        </td>
    </tr>
{/if}

```

config_load

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
file	string	Да	<i>n/a</i>	Имя config файла для загрузки
section	string	Нет	<i>n/a</i>	Имя секции для загрузки
scope	string	Нет	<i>local</i>	Способ обработки области видимости загруженных переменных. Должен быть одним 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 означает, что переменные доступны из любого шаблона.
global	boolean	Нет	<i>No</i>	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.

Эта функция используется для загрузки переменных в шаблон из файлов конфигурации. Смотри Файлы конфигурации для дополнительной информации.

Пример 7.2. функция config_load

```
{config_load file="colors.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>
```

Файлы конфигурации также могут содержать секции. Можно загрузить переменные из определенной секции, указав атрибут *section*.

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

Пример 7.3. функция config_load с указанием секции

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

<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>
```

foreach, foreachelse

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
from	array	Да	n/a	Массив, по которо-

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				му надо пройтись
item	string	Да	<i>n/a</i>	Имя переменной, которая будет выступать в качестве значения текущего элемента
key	string	Нет	<i>n/a</i>	Имя переменной, которая будет выступать в качестве ключа текущего элемента
name	string	Нет	<i>n/a</i>	Название цикла <code>foreach</code> для доступа к его свойствам

Циклы *foreach* являются альтернативой циклам *section*. Циклы *foreach* используются для прохождения по одному массиву. Синтаксис *foreach* намного проще, чем *section*, но его можно использовать только для одного массива. Тэг *foreach* должен иметь в пару тэг */foreach*. Обязательные параметры - *from* и *item*. Название цикла *foreach* может быть любой последовательностью букв, цифр и знаков подчеркиваний `_`. Циклы *foreach* могут быть вложенные, и имена вложенных циклов должны быть уникальные. Переменная *from* (обычно массив значений) указывает количество итераций цикла. *foreachelse* выполняется, если нету значений в переменной *from*.

Пример 7.4. foreach

```
{* выводятся все значения массива $custid *}
{foreach from=$custid item=curr_id}
    id: {Scurr_id}<br>
{/foreach}
```

OUTPUT:

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

Пример 7.5. foreach key

```
{* The key contains the key for each looped value
```

assignment looks like this:

```
$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=$contacts}
  {foreach key=key item=item from=$contact}
    {$key}: {$item}<br>
  {/foreach}
{/foreach}
```

OUTPUT:

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

Циклы `foreach` имеют свои свойства, доступ к которым реализуется через `{$smarty.foreach.foreachname.varname}`, где `foreachname` это название цикла (значение атрибута *name*), а `varname` - имя свойства.

iteration

Количество отработанных итераций.

Отсчет начинается с 1 и увеличивается на единицу на каждой итерации.

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..

include

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

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 имя ресурса шаблона.

Опциональный атрибут `assign` указывает, что вывод шаблона будет присвоен переменной вместо отображения.

Пример 7.6. функция `include`

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

{* тело шаблона *}

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

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 7.7. функция `include`: передача переменных

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

{* тело шаблона *}

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

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

Пример 7.8. функция `include`: пример использования ресурсов

```
{* абсолютный путь к файлу *}
```

```
{include file="/usr/local/include/templates/header.tpl"}

{* абсолютный путь к файлу (аналогично) *}
{include file="file:/usr/local/include/templates/header.tpl"}

{* абсолютный путь к файлу в стиле windows (НЕОБХОДИМО использовать префикс "file:") *}
{include file="file:C:/www/pub/templates/header.tpl"}

{* подключить шаблон из ресурса "db" *}
{include file="db:header.tpl"}
```

include_php

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

Тэг `include_php` используется для подключения php скрипта в шаблон. Если `security` включен, то php скрипт должен быть расположен в папке `$trusted_dir`. Тэг `include_php` должен иметь атрибут `"file"`, который указывает путь подключаемого php файла (относительный к `$trusted_dir` или абсолютный путь).

`include_php` это хороший способ для управления компонентными шаблонами, держать PHP код вне файлов шаблонов. Допустим, у вас есть шаблон, который выводит навигацию сайта, информация о которой динамически загружается из базы данных. Вы можете хранить php файл, который получает данные из базы данных, в отдельной папке и подключать его в начале шаблона. Теперь можно подключать этот файл шаблона в любом месте, не волнуясь о происхождении информации (база данных или нет).

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 7.9. функция `include_php`

load_nav.php

<?php

```
// Загрузка переменных из базы данных mysql и передача их в шаблон
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 или относительный относительно $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}
```

insert

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Да	<i>n/a</i>	Имя функции вставки (insert_name)
assign	string	Нет	<i>n/a</i>	Имя переменной, которой будет присвоен вывод
script	string	Нет	<i>n/a</i>	Имя php файла, который будет подключен перед вызовом функции вставки
[var ...]	[var type]	Нет	<i>n/a</i>	Переменные, передаваемые в функцию вставки

Тэг insert очень похож на тэг include, кроме случая, когда кэширование включено. Insert тэг не кешируется. Он будет выполнен каждый раз, при обращении к шаблону.

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 файла конфигурации) и должен вызвать функцию, чтобы получить код баннера.

Пример 7.10. функция `insert`

```
{* пример установки 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 попытается 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 существовать, и должен быть подключен php файл, чтобы определить функцию. Путь к файлу должен быть либо абсолютным, либо относительным относительно `$trusted_dir`. Когда security активирована, то php файл должен быть в папке `$trusted_dir`.

Объект 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.

if, elseif, else

Конструкция `{if}` в Smarty такая же гибкая, как и аналогичная конструкция в PHP, только с несколько расширенными возможностями. Каждый тэг `{if}` должен иметь пару `{/if}`. `{else}` и `{elseif}` так же допустимы. Допустимы все квалификаторы из PHP, такие как `//`, `or`, `&&`, `and` 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
<code>==</code>	<code>eq</code>	<code>\$a eq \$b</code>	равно
<code>!=</code>	<code>ne</code> , <code>neq</code>	<code>\$a neq \$b</code>	не равно
<code>></code>	<code>gt</code>	<code>\$a gt \$b</code>	больше

Квалификатор	Альтернативы	Пример синтаксиса	Описание
<	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 7.11. Конструкция if

```
{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}
```

ldelim, rdelim

`ldelim` и `rdelim` используются для отображения разделителей. В нашем случае это "{" и "}". Вы можете использовать блок `{literal}/{/literal}` для экранирования блоков текста. См. также `{Smarty.ldelim}` и `{Smarty.rdelim}`

Пример 7.12. 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!
```

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 7.13. тэг `literal`

```
{literal}
<script language=javascript>

    <!--
        function isblank(field) {
            if (field.value == "")
                { return false; }
            else
                {
                    document.loginform.submit();
                    return true;
                }
        }
    // -->

</script>
{/literal}
```

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 7.14. тэг `php`

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

section, sectionelse

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
name	string	Да	<i>n/a</i>	Название секции
loop	mixed	Да	<i>n/a</i>	Значение, определяющее количество

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				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 производиться в обратном направлении.
max	integer	Нет	1	Максимальное количество итераций цикла.
show	boolean	Нет	true	Указывает, показывать или нет эту секцию

Секции используются для обхода массива данных. Все тэги *section* должны иметь в пару тэг */section*. Обязательные параметры: *name* и *loop*. Имя секции может быть какой угодно последовательностью букв, цифр и знаков подчеркиваний. Секции могут быть вложенными. Имена вложенных секций должны отличаться друг от друга. Переменная цикла (обычно массив) определяет количество итераций цикла. Когда отображаем переменную цикла, имя секции должно быть указана после имени переменной в квадратных скобках []. Тэг *sectionelse* выполняется, когда переменная цикла пуста.

Пример 7.15. section

```
{* Этот пример выведет все значения массива $custid *}
{section name=customer loop=$custid}
    id: {$custid[customer]}<br>
{/section}
```

OUTPUT:

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

Пример 7.16. section loop variable

```
{* Переменная цикла определяет только количество итераций цикла.
Вы имеете доступ к любой переменной шаблона в секции.
Этот пример подразумевает, что каждый из массивов $custid, $name и $address
содержат одинаковое количество значений *}
{section name=customer loop=$custid}
    id: {$custid[customer]}<br>
    name: {$name[customer]}<br>
    address: {$address[customer]}<br>
    <p>
{/section}
```

OUTPUT:

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

Пример 7.17. имена секций

```
{* имя секции может быть любым и используется для указания
данных в секции *}
{section name=mydata loop=$custid}
```

```

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

```

Пример 7.18. вложенные секции

** секции могут быть неограниченно вложенными. С помощью вложенных секций вы можете организовать доступ к комплексным структурам информации, таким как многомерные массивы. В данном примере `$contact_type[customer]` является массивом, где содержатся типы контактов данного клиента. **

```

{section name=customer loop=$custid}
  id: {$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}
<p>
{/section}

```

OUTPUT:

```

id: 1000<br>
name: John Smith<br>
address: 253 N 45th<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: john@example.com<br>
<p>
id: 1001<br>
name: Jack Jones<br>
address: 417 Mulberry ln<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jack@example.com<br>
<p>
id: 1002<br>
name: Jane Munson<br>
address: 5605 apple st<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jane@example.com<br>
<p>

```

Пример 7.19. секции и ассоциативные массивы

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

OUTPUT:

```
name: John Smith<br>
home: 555-555-5555<br>
cell: 555-555-5555<br>
e-mail: john@example.com<p>
name: Jack Jones<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jack@example.com<p>
name: Jane Munson<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: jane@example.com<p>
```

Пример 7.20. sectionelse

```
{* sectionelse обработается, когда $custid не содержит значений *}
{section name=customer loop=$custid}
    id: {$custid[customer]}<br>
{sectionelse}
    there are no values in $custid.
{/section}
```

Секции также имеют свои собственные атрибуты, которые определяют определенные настройки секции. Они указываются примерно так: {\$smarty.section.sectionname.varname}

ЗАМЕЧАНИЕ: Начиная со Smarty версии 1.5.0, синтаксис атрибутов секций изменился с {%sectionname.varname%} на {\$smarty.section.sectionname.varname}. Старый синтаксис пока поддерживается, но вы встретите только новый стиль в примерах данного руководства.

index

index хранит текущий индекс цикла, начиная с 0 (или значения атрибута start), и увеличивается на единицу (или на значение атрибута step).

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 7.21. атрибут секции `index`

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

OUTPUT:

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

`index_prev`

`index_prev` хранит предыдущий индекс цикла. На первой итерации устанавливается в -1.

Пример 7.22. атрибут секции `index_prev`

```
{section name=customer loop=$custid}
  {Smarty.section.customer.index} id: {Smarty.section.customer.index}<br>
  {* FYI, $custid[customer.index] and $custid[customer] are identical in meaning *}
  {if $custid[customer.index_prev] ne $custid[customer.index]}
    The customer id changed<br>
  {/if}
{/section}
```

OUTPUT:

```
0 id: 1000<br>
  The customer id changed<br>
1 id: 1001<br>
  The customer id changed<br>
2 id: 1002<br>
  The customer id changed<br>
```

`index_next`

`index_next` хранит следующий индекс цикла. На последней итерации цикла будет иметь значение на единицу больше текущего индекса (или на другое значение, если указан атрибут `step`).

Пример 7.23. атрибут секции `index_next`

```
{section name=customer loop=$custid}
```

```
{Smarty.section.customer.index} id: {Scustid[customer]}<br>
{* FYI, $custid[customer.index] and $custid[customer] are identical in meaning *}
{if $custid[customer.index_next] ne $custid[customer.index]}
    The customer id will change<br>
{/if}
{/section}
```

OUTPUT:

```
0 id: 1000<br>
    The customer id will change<br>
1 id: 1001<br>
    The customer id will change<br>
2 id: 1002<br>
The customer id will change<br>
```

iteration

iteration хранит текущую итерацию цикла.

ЗАМЕЧАНИЕ: Значение не зависит от атрибутов start, step и max, в отличие атрибута index. Итерации также начинаются с 1, а не с 0, как index. rownum является синонимом для iteration.

Пример 7.24. атрибут секции iteration

```
{section name=customer loop=$custid start=5 step=2}
    current loop iteration: {Smarty.section.customer.iteration}<br>
    {Smarty.section.customer.index} id: {Scustid[customer]}<br>
    {* FYI, $custid[customer.index] and $custid[customer] are identical in meaning *}
    {if $custid[customer.index_next] ne $custid[customer.index]}
        The customer id will change<br>
    {/if}
{/section}
```

OUTPUT:

```
current loop iteration: 1
5 id: 1000<br>
    The customer id will change<br>
current loop iteration: 2
7 id: 1001<br>
    The customer id will change<br>
current loop iteration: 3
9 id: 1002<br>
The customer id will change<br>
```

first

first имеет значение истина, если текущая итерация цикла - первая.

Пример 7.25. атрибут секции first

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

  <tr><td>{$smarty.section.customer.index} id:
    {$custid[customer]}</td></tr>

  {if $smarty.section.customer.last}
    </table>
  {/if}
{/section}
```

OUTPUT:

```
<table>
<tr><td>0 id: 1000</td></tr>
<tr><td>1 id: 1001</td></tr>
<tr><td>2 id: 1002</td></tr>
</table>
```

last

last имеет значение истина, если текущая итерация цикла - последняя. one.

Пример 7.26. атрибут секции last

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

  <tr><td>{$smarty.section.customer.index} id:
    {$custid[customer]}</td></tr>

  {if $smarty.section.customer.last}
    </table>
  {/if}
{/section}
```

OUTPUT:

```
<table>
<tr><td>0 id: 1000</td></tr>
<tr><td>1 id: 1001</td></tr>
<tr><td>2 id: 1002</td></tr>
</table>
```

rownum

rownum хранит текущую итерацию цикла, начиная с 1. rownum - синоним для iteration.

Пример 7.27. атрибут секции rownum

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

OUTPUT:

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

loop

loop хранит последний отработанный индекс цикла. Может быть использован как внутри секции, так и после нее.

Пример 7.28. аттрбуг секции index

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

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

OUTPUT:

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

There were 3 customers shown above.

show

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

Пример 7.29. атрибут секции *show*

```
{* $show_customer_info может быть передана из PHP программы
    указывая, показывать или нет эту секцию *}
{section name=customer loop=$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}
```

OUTPUT:

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

the section was shown.

total

`total` хранит количество всех итераций цикла. Может быть использован как в секции, так и после нее.

Пример 7.30. атрибут секции *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.

OUTPUT:

```
0 id: 1000<br>
2 id: 1001<br>
4 id: 1002<br>
```

There were 3 customers shown above.

strip

Часто вебдизайнеры сталкиваются с проблемой, что пробелы и переносы строк влияют на отображение HTML в браузере ("фишки" браузера), то есть может понадобится склеить все теги в шаблоне вместе, чтобы получить желаемый результат. Но в результате получается нечитаемый или трудноредактируемый 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 7.31. тэг 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}
```

OUTPUT:

```
<table border=0><tr><td><A HREF="http://my.example.com"><font color="red">This is a test</font></A></td></tr></table>
```

Заметьте, что в данном примере все строки начинаются и заканчиваются HTML тэгами. Учтите, что все строки склеиваются вместе. Если есть обычный текст в начале строки, то может не получиться желаемый результат.

Глава 8. Пользовательские Функции

Содержание

assign	63
counter	64
cycle	65
debug	66
eval	66
fetch	67
html_checkboxes	68
html_image	69
html_options	71
html_radios	72
html_select_date	74
html_select_time	77
html_table	81
math	84
mailto	85
popup_init	87
popup	87
textformat	91

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

assign

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
var	string	Да	<i>n/a</i>	Имя переменной, значение которой будет устанавливаться
value	string	Да	<i>n/a</i>	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	<i>default</i>	Имя счетчика
start	number	Нет	<i>1</i>	Начальное значение счетчика
skip	number	Нет	<i>1</i>	Шаг счетчика
direction	string	Нет	<i>up</i>	Направление (вверх - up/вниз - down)
print	boolean	Нет	<i>true</i>	выводить значение счетчика или нет
assign	string	Нет	<i>n/a</i>	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 ли 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	<i>default</i>	Название цикла
values	mixed	Да	<i>N/A</i>	Значения, по которым будет производиться цикл. Либо список, разделенный запятыми (либо другим указанным разделителем), либо массив значений.
print	boolean	Нет	<i>true</i>	Выводить значение, или нет
advance	boolean	Нет	<i>true</i>	Переключаться или нет на следующее значение
delimiter	string	Нет	,	Разделитель, используемый в атрибуте values.
assign	string	Нет	<i>n/a</i>	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 истину (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	<i>html</i>	Тип вывода (html или javascript)

{debug} выводит консоль отладки. Это работает независимо от значения опции debug. Так как этот тэг обрабатывается в процессе 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	<i>n/a</i>	Переменная (или строка) для обработки
assign	string	Нет	<i>n/a</i>	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 конфигурации.

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 city.

You must supply a state.

fetch

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
file	string	Да	<i>n/a</i>	файл, http или ftp сайт для отображения
assign	string	Нет	<i>n/a</i>	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 специальный 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 указанных 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	<i>checkbox</i>	название списка флажков
values	array	Да, если не указан атрибут options	<i>n/a</i>	Массив значений для флажков
output	array	Да, если не указан атрибут options	<i>n/a</i>	массив названий флажков
selected	string/array	Нет	<i>пусто</i>	выбранный флажок(флажки)
options	associative array	Да, если не указаны атрибуты values и output	<i>n/a</i>	Ассоциативный массив значений и названий
separator	string	Нет	<i>пусто</i>	строка разделяющая каждый флажок
labels	boolean	Нет	<i>true</i>	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

```
<?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');
?>
```

шаблон index.tpl:

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

```
<?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');
?>
```

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	<i>n/a</i>	название/путь к изображению

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
height	string	Нет	<i>реальная высота изображения</i>	высота изображения
width	string	Нет	<i>реальная ширина изображения</i>	ширина изображения
basedir	string	Нет	<i>корень веб сервера</i>	папка, от которой указаны относительные пути
alt	string	Нет	<i>""</i>	альтернативное описание изображения
href	string	Нет	<i>n/a</i>	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 окружения `DOCUMENT_ROOT`). Если `security` включено, то путь к изображению 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 лучше не пользоваться тэгом `html_image` и вставлять статические тэги изображений, для достижения оптимального быстрого действия.

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)

```



```

html_options

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
values	массив	Да, если не указан атрибут options	<i>n/a</i>	массив значений для выпадающего списка
output	массив	Да, если не указан атрибут options	<i>n/a</i>	массив названий для выпадающего списка
selected	string/array	Нет	<i>пусто</i>	Выбранный элемент(ы)
options	ассоциативный массив	Да, если не указаны атрибуты values и output	<i>n/a</i>	ассоциативный массив значений и названий
name	string	Нет	<i>пусто</i>	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	<i>radio</i>	название элементов выбора
values	массив	Да, если не указан атрибут options	<i>n/a</i>	массив значений элементов выбора
output	массив	Да, если не указан атрибут options	<i>n/a</i>	массив названий элементов выбора
checked	string	Нет	<i>пусто</i>	Значение выбранного элемента
options	ассоциативный массив	Да, если не указаны атрибуты values и output	<i>n/a</i>	ассоциативный массив значений и названий элементов выбора

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
separator	string	Нет	<i>пусто</i>	текст, разделяющий элементы выбора

Пользовательская функция `html_radios` генерирует HTML код группы элементов выбора (radio button group). Автоматически устанавливает выбранное значение, если оно указано. Требует наличия атрибутов `values` и `output` или атрибута `options`. Сгенерированный HTML 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)

```
<input type="radio" name="id" value="1000">Joe Schmoe<br />
<input type="radio" name="id" value="1001" checked="checked">Jack Smith<br />
<input type="radio" name="id" value="1002">Jane Johnson<br />
```

```
<input type="radio" name="id" value="1003">Charlie Brown<br />
```

html_select_date

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
prefix	string	Нет	Date_	префикс названий переменных
time	timestamp/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
field_array	string	Нет	null	название переменной (name), которая будет содержать выбранные значения в

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				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 (am/pm)
use_24_hours	boolean	Нет	true	использовать 24-часовой формат времени
minute_interval	integer	Нет	1	интервал элементов

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				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.

Атрибут `time` имеет несколько разных форматов: он может быть представлен в виде уникальной временной метки (UNIX timestamp) или строки в формате ГГТММДДЧЧММСС или любой 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	<i>n/a</i>	массив данных, по которому будет произведен обход
cols	integer	Нет	<i>3</i>	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	<i>empty</i>	количество строк в таблице. если атрибут rows не указан, но указан cols, тогда кол-во строк вычисляется исходя из кол-ва колонок и кол-ва элементов, которые необходимо отобразить.
inner	string	No	<i>cols</i>	направление последовательного отображения элементов из массива loop. <i>cols</i> означает, что элементы отображаются колонка за колонкой. <i>rows</i> означает, что элементы отображаются строка за строкой.
table_attr	string	Нет	<i>border="1"</i>	дополнительные атрибуты тэга table
tr_attr	string	Нет	<i>пусто</i>	дополнительные атрибуты тэга tr (если указан массив, то его элементы циклически повторяются)

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
td_attr	string	Нет	<i>пусто</i>	дополнительные атрибуты тэга td (если указан массив, то его элементы циклически повторяются)
trailpad	string	Нет	<i>&nbsp;</i>	значение остаточных ячеек на последней строке таблицы
hdir	string	Нет	<i>right</i>	направление отображения каждого ряда. допустимые значения: <i>left</i> (слева направо), <i>right</i> (справа налево)
vdir	string	Нет	<i>down</i>	направление отображения каждой колонки. допустимые значения: <i>down</i> (сверху вниз), <i>up</i> (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 его значения циклически 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>&nbsp;</td><td>&nbsp;</td><td>&nbsp;</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>&nbsp;</td><td>&nbsp;</td><td>&nbsp;</td></tr>
</table>

```

math

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
equation	string	Да	<i>n/a</i>	математической вы- ражение
format	string	Нет	<i>n/a</i>	формат результата (sprintf)
var	numeric	Да	<i>n/a</i>	переменная выраже- ния
assign	string	Нет	<i>n/a</i>	переменная шабло- на, которой будет присвоен вывод
[var ...]	numeric	Да	<i>n/a</i>	дополнительные пе- ременные выраже- ния

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`, `strans` и `tan`. Ознакомьтесь 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 8.14. 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

mailto

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
address	string	Да	<i>n/a</i>	адрес e-mail
text	string	Нет	<i>n/a</i>	название ссылки. По умолчанию: адрес e-mail
encode	string	Нет	<i>none</i>	Способ кодирования e-mail. Может быть одним из none, hex или javascript.
cc	string	Нет	<i>n/a</i>	адреса e-mail для

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				точной копии. Адреса разделяются запятыми.
bcc	string	Нет	<i>n/a</i>	адреса e-mail для "слепой" копии. Адреса разделяются запятыми.
subject	string	Нет	<i>n/a</i>	тема письма.
newsgroups	string	Нет	<i>n/a</i>	в какие конференции передавать. конференции разделяются запятыми.
followupto	string	Нет	<i>n/a</i>	адреса для дальнейшего перенаправления. Адреса разделяются запятыми.
extra	string	Нет	<i>n/a</i>	Дополнительный 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 также можете использовать шестнадцатичное кодирование.

Пример 8.15. 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"}
```

OUTPUT:

```
<a href="mailto:me@example.com" >me@example.com</a>
<a href="mailto:me@example.com" >send me some mail</a>
<SCRIPT 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%40%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>
```

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 8.16. popup_init

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

popup

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

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				(Заккрыть)
textfont	string	Нет	<i>n/a</i>	шрифт текста в окне
captionfont	string	Нет	<i>n/a</i>	шрифт заголовка окна
closefont	string	Нет	<i>n/a</i>	шрифт текста "Close" (Заккрыть)
textsize	string	Нет	<i>n/a</i>	размер шрифта в окне
captionsize	string	Нет	<i>n/a</i>	размер шрифта заголовка
closesize	string	Нет	<i>n/a</i>	размер шрифта текста "Close" (Заккрыть)
width	integer	Нет	<i>n/a</i>	ширина окна
height	integer	Нет	<i>n/a</i>	высота окна
left	boolean	Нет	<i>false</i>	создавать окно слева от курсора мыши
right	boolean	Нет	<i>false</i>	создавать окно справа от курсора мыши
center	boolean	Нет	<i>false</i>	создавать окно на месте мыши
above	boolean	Нет	<i>false</i>	создает окно выше курсора мыши. ЗАМЕЧАНИЕ: возможно только если указан атрибут height
below	boolean	Нет	<i>false</i>	создает окно под курсором мыши
border	integer	Нет	<i>n/a</i>	толщина границы окна
offsetx	integer	Нет	<i>n/a</i>	горизонтальное смещение окна от курсора мыши.
offsety	integer	Нет	<i>n/a</i>	вертикальное смещение окна от курсора мыши
fgbackground	url to image	Нет	<i>n/a</i>	определяет фоновое изображение текста, вместо fgcolor.
bgbackground	url to image	Нет	<i>n/a</i>	определяет фоновое изображение для границ окна, вместо bgcolor. ЗАМЕЧА-

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				НИЕ: Может понадобиться установить bgcolor в "" или будет показан цвет, а не изображение. ЗАМЕЧАНИЕ: При наличии ссылки "Close" Netscape перерисовывает ячейки таблицы, что приводит к некорректному отображению.
closetext	string	Нет	<i>n/a</i>	устанавливает текст вместо "Close"
noclose	boolean	Нет	<i>n/a</i>	не отображать текст "Close"
status	string	Нет	<i>n/a</i>	установить значение строки статуса в браузере
autostatus	boolean	Нет	<i>n/a</i>	установить значение строки статуса в браузере в текст окна. ЗАМЕЧАНИЕ: отменяет значение status
autostatuscap	string	Нет	<i>n/a</i>	установить значение строки статуса в браузере в текст заголовка. ЗАМЕЧАНИЕ: отменяет значение autostatus
inarray	integer	Нет	<i>n/a</i>	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	<i>n/a</i>	указывает, что заголовок окна следует взять из указанного элемента массива ol_caps

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
caricon	url	Нет	<i>n/a</i>	выводит изображение перед заголовком окна.
snapx	integer	Нет	<i>n/a</i>	ровняет окно к горизонтальной сетке
snapy	integer	Нет	<i>n/a</i>	ровняет окно к вертикальной сетке
fixx	integer	Нет	<i>n/a</i>	закрепляет горизонтальное положение окна. ЗАМЕЧАНИЕ: отменяет все остальные параметры горизонтального положения
fixy	integer	Нет	<i>n/a</i>	закрепляет вертикальное положение окна. ЗАМЕЧАНИЕ: отменяет все остальные параметры вертикального положения
background	url	Нет	<i>n/a</i>	указывает фоновое изображение окна
padx	integer, integer	Нет	<i>n/a</i>	дополняет фоновое изображение горизонтальными отступами к тексту. ЗАМЕЧАНИЕ: этот параметр принимает два значения
pady	integer, integer	Нет	<i>n/a</i>	дополняет фоновое изображение вертикальными отступами к тексту. ЗАМЕЧАНИЕ: этот параметр принимает два значения
fullhtml	boolean	Нет	<i>n/a</i>	Позволяет полностью контролировать HTML над фоновым изображением. HTML код ожидается в параметре text
frame	string	Нет	<i>n/a</i>	Контролировать всплывающие окна в

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
				различных фреймах. См. сайт overlib для дополнительной информации по этой функции
timeout	string	Нет	<i>n/a</i>	вызывает указанную javascript функцию и использует результат как текст окна
delay	integer	Нет	<i>n/a</i>	время жизни окна в миллисекундах. Позволяет реализовывать всплывающие подсказки.
haut	boolean	Нет	<i>n/a</i>	автоматическое определение горизонтального местоположения окна относительно мыши.
vaut	boolean	Нет	<i>n/a</i>	автоматическое определение вертикального местоположения окна относительно мыши.

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

Пример 8.17. popup

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

{* создаем ссылку с всплывающим окном, которое появляется 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 всплывающем 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>
```

textformat

Имя атрибута	Тип	Обязателен	По умолчанию	Описание
style	string	Нет	<i>n/a</i>	предустановленный стиль

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

функция `textformat` используется для форматирования текста, заключенного внутри нее. 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 цветую 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 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 синтаксис, который не является допустимым синтаксисом файлов конфигурации. 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 используется последний встретившийся вариант.

Файлы конфигурации загружаются в шаблон в помощью функции **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.

Глава 10. Отладочная консоль

В Smarty включена консоль для отладки. Консоль позволяет узнать все включенные файлы, присвоенные переменные и настройки из конфигурационных файлов. Шаблон "debug.tpl", поставляемый вместе со Smarty, управляет видом консоли. Установите опцию Smarty \$debugging в true и, если надо, укажите в \$debug_tpl путь к шаблону debug.tpl (по умолчанию это SMARTY_DIR). Когда вы загружаете страницу, должно появиться всплывающее 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.

Замечание: Консоль отладки не работает, когда используется функция API fetch(). Необходимо использовать только функцию display(). Она генерирует javascript код в начале каждой сгенерированной страницы. Если вам не нравится javascript, можно отредактировать 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.

Часть III. Smarty для программистов

Содержание

11. Константы	101
SMARTY_DIR	101
SMARTY_CORE_DIR	101
12. Переменные	102
\$template_dir	102
\$compile_dir	103
\$config_dir	103
\$plugins_dir	103
\$debugging	103
\$debug_tpl	103
\$debugging_ctrl	104
\$autoload_filters	104
\$compile_check	104
\$force_compile	104
\$caching	104
\$cache_dir	105
\$cache_lifetime	105
\$cache_handler_func	105
\$cache_modified_check	105
\$config_overwrite	106
\$config_booleanize	106
\$config_read_hidden	106
\$config_fix_newlines	106
\$default_template_handler_func	106
\$php_handling	106
\$security	107
\$secure_dir	107
\$security_settings	107
\$trusted_dir	108
\$left_delimiter	108
\$right_delimiter	108
\$compiler_class	108
\$request_vars_order	108
\$request_use_auto_globals	108
\$error_reporting	108
\$compile_id	108
\$use_sub_dirs	108
\$default_modifiers	109
\$default_resource_type	109
13. Методы	110
14. Кэширование	152

Настройка кэширования	152
Множественное кэширование страниц	154
Групповое кэширование	156
Управление кэшированием результатов работы плагинов	157
15. Расширенные возможности	159
Объекты	159
Префильтры	160
Постфильтры	161
Фильтры вывода	161
Управление кэшированием	162
Ресурсы	164
16. Плагины - расширение функционала Smarty	168
Как работают плагины	168
Как работают плагины	169
Соглашение об именах	169
Написание плагинов	170
Функции шаблона	170
Модификаторы	172
Блочные функции	173
Функции компилятора	174
Префильтры/Постфильтры	175
Фильтры вывода	177
Ресурсы	177
Вставки	179

Глава 11. Константы

Содержание

SMARTY_DIR	101
SMARTY_CORE_DIR	101

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

<code>\$template_dir</code>	102
<code>\$compile_dir</code>	103
<code>\$config_dir</code>	103
<code>\$plugins_dir</code>	103
<code>\$debugging</code>	103
<code>\$debug_tpl</code>	103
<code>\$debugging_ctrl</code>	104
<code>\$autoload_filters</code>	104
<code>\$compile_check</code>	104
<code>\$force_compile</code>	104
<code>\$caching</code>	104
<code>\$cache_dir</code>	105
<code>\$cache_lifetime</code>	105
<code>\$cache_handler_func</code>	105
<code>\$cache_modified_check</code>	105
<code>\$config_overwrite</code>	106
<code>\$config_booleanize</code>	106
<code>\$config_read_hidden</code>	106
<code>\$config_fix_newlines</code>	106
<code>\$default_template_handler_func</code>	106
<code>\$php_handling</code>	106
<code>\$security</code>	107
<code>\$secure_dir</code>	107
<code>\$security_settings</code>	107
<code>\$trusted_dir</code>	108
<code>\$left_delimiter</code>	108
<code>\$right_delimiter</code>	108
<code>\$compiler_class</code>	108
<code>\$request_vars_order</code>	108
<code>\$request_use_auto_globals</code>	108
<code>\$error_reporting</code>	108
<code>\$compile_id</code>	108
<code>\$use_sub_dirs</code>	108
<code>\$default_modifiers</code>	109
<code>\$default_resource_type</code>	109

`$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 рекомендуется размещать эту директорию в пределах корневой директории веб-сервера.

`$compile_dir`

Имя каталога, в котором хранятся скомпилированные шаблоны. По умолчанию установлено в `./templates_c`, т.е. поиск каталога с скомпилированными 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 относительно 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`

Задаёт 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` задаёт 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 любом случае, переменные 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 шаблона.
- SMARTY_PHP_ALLOW - Smarty будет выполнять теги как PHP-код.

Замечание: Встраивать PHP-код в шаблоны весьма не рекомендуется. 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 заслуживающие 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 директорий, которые считаются безопасными. {include} и {fetch} используют этот параметр при включенном безопасном режиме.

\$security_settings

Это используется для изменения или указания настроек безопасности когда безопасность (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 "{".

\$right_delimiter

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

\$compiler_class

Задаёт 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 значение, 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 использоваться именно это значение.

\$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	111
append_by_ref	112
assign	113
assign_by_ref	114
clear_all_assign	115
clear_all_cache	116
clear_assign	117
clear_cache	118
clear_compiled_tpl	119
clear_config	120
config_load	121
display	122
fetch	124
get_config_vars	126
get_registered_object	127
get_template_vars	128
is_cached	129
load_filter	131
register_block	132
register_compiler_function	133
register_function	134
register_modifier	135
register_object	136
register_outputfilter	137
register_postfilter	138
register_prefilter	139
register_resource	140
trigger_error	141
template_exists	142
unregister_block	143
unregister_compiler_function	144
unregister_function	145
unregister_modifier	146
unregister_object	147
unregister_outputfilter	148
unregister_postfilter	149
unregister_prefilter	150
unregister_resource	151

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()`, 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, соответственно, 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(), 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(). Конфигурационные 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 отображает 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 записываться 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");
?>
```

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 записываться 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;  
?>
```

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 зарегистрированный объект. Может быть полезно в случае необходимости получения прямого доступа к зарегистрированному 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 (а) строка, содержащая имя функции, или (б) массив вида `array(&$object, $method)`, где `&$object` является ссылкой на объект, а `$method` является строкой, содержащей имя метода, или (с) массив в форме `array($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 (а) строка, содержащая имя функции, или (б) массив вида `array(&$object, $method)`, где `&$object` является ссылкой на объект, а `$method` является строкой, содержащей имя метода, или (с) массив в форме `array($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 (а) 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 (а) строка, содержащая имя функции, или (б) массив вида `array(&$object, $method)`, где `&$object` является ссылкой на объект, а `$method` является строкой, содержащей имя метода, или (с) массив в форме `array($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("slash", "stripslashes");

// теперь можно использовать {$var|slash} чтобы вырезать слешы 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 регистрирует объект для использования в шаблоне. Обратитесь к разделу "Объекты" за примерами.

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 получения дополнительной информации.

Техническое замечание: Имя ресурса должно состоять минимум из двух букв. Однобуквенные имена ресурсов будут игнорироваться и использоваться 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, реализующих "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 deregистрации 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 deregистрации функции компиляции. В качестве аргумента передается имя функции компиляции.

unregister_function

unregister_function

unregister_function

void **unregister_function** (string name)

Используйте функцию для динамической дерегистрации 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 deregистрации 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 deregистрации объекта.

unregister_outputfilter

unregister_outputfilter

unregister_outputfilter

void **unregister_outputfilter** (string function_name)

Используйте функцию для динамической deregистрации фильтра вывода.

unregister_postfilter

unregister_postfilter

unregister_postfilter

void **unregister_postfilter** (string function_name)

Используйте функцию для динамической deregистрации постфильтра.

unregister_prefilter

unregister_prefilter

unregister_prefilter

void **unregister_prefilter** (string function_name)

Используйте для динамической deregистрации префильтра.

unregister_resource

unregister_resource

unregister_resource

void **unregister_resource** (string name)

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

Пример 13.27. unregister_resource

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

Глава 14. Кэширование

Содержание

Настройка кэширования	152
Множественное кэширование страниц	154
Групповое кэширование	156
Управление кэшированием результатов работы плагинов	157

Кэширование используется для ускорения вызовов `display()` или `fetch()` путем сохранения их результатов в файл. Кэшированная версия файла (если таковая будет найдена) отображается сразу, без регенерации 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 (кэшированную копию) в `$cache_dir`. При следующем вызове `display('index.tpl')`, вместо повторной интерпретации шаблона, будет использована кэшированная 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 кэшированная страничка существует на протяжении определенного времени, указанного в `$cache_lifetime`. Значение по умолчанию равно 3600 секундам или 1 часу. После того, как это время истекает, кэш обновляется. Существует возможность присвоить каждой кэшированной страничке собственное время жизни, установив `$caching = 2`. Смотрите документацию `$cache_lifetime` для получения подробных сведений.

Пример 14.2. Установка собственного `cache_lifetime` для кэшированной 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 кэшированный 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 присваиваем 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 кэшированной 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	159
Префильтры	160
Постфильтры	161
Фильтры вывода	161
Управление кэшированием	162
Ресурсы	164

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";
    }
}

$smartyobj = 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 зарегистрированы 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 абсолютного 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 файловыми 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, сокет,

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 функционала Smarty

Содержание

Как работают плагины	168
Как работают плагины	169
Соглашение об именах	169
Написание плагинов	170
Функции шаблона	170
Модификаторы	172
Блочные функции	173
Функции компилятора	174
Префильтры/Постфильтры	175
Фильтры вывода	177
Ресурсы	177
Вставки	179

Архитектура версии 2.0 позволяет внедрять плагины, которыми являются практически все настраиваемые элементы функционала Smarty. Сюда входят:

- функции
- модификаторы
- блочные 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 deregистрировать, 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($params, &$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($params, &$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 присоединить 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;
}
?>
```

Блочные функции

```
void smarty_block_name()($params, $content, &$smarty, &$repeat);
array $params;
mixed $content;
object &$smarty;
boolean &$repeat;
```

Блочные функции выглядят следующим образом: {func} .. {/func}. Другими словами, они заключены в определенном блоке шаблона и оперируют содержимым этого блока. Блочные функции имеют приоритет перед пользовательскими функциями, имеющими то же имя, поэтому, вы не сможете использовать одновременно свои функции вида {func} и блочные функции {func} .. {/func}.

Smarty вызывает ваши функции дважды: первый раз при открытии тэга и второй раз при закрытии тэга.

Только открывающий тэг блочной 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 (кусоч) текста
 * -----
 */
function smarty_block_translate($params, $content, &$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 зарегистрированы 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 вставки 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 модифицированный 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 модифицированный 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;!', '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 оперируют 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 зарегистрировать 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 &$smarty;  
bool smarty_resource_name_secure($src_name, &$smarty);  
string $src_name;  
object &$smarty;  
bool smarty_resource_name_trusted($src_name, &$smarty);  
string $src_name;  
object &$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, переданных 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($params, &$smarty)  
{  
    if (empty($params['format'])) {  
        $smarty->trigger_error("insert time: missing 'format' parameter");  
        return;  
    }  
  
    $datetime = strftime($params['format']);  
    return $datetime;  
}  
?>
```

Часть IV. Приложения

Содержание

17. Решение проблем	182
Ошибки Smarty/PHP	182
18. Советы	183
Обработка пустых переменных	183
Обработка переменных по умолчанию	183
Присвоение переменной заголовка (title) заголовку шаблона.	184
Даты	185
WAP/WML	186
Составные шаблоны	187
Скрытие E-mail адреса	188
19. Источники	189
20. Ошибки	190

Глава 17. Решение проблем

Содержание

Ошибки Smarty/PHP	182
-------------------------	-----

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 увидеть 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.

Глава 18. Советы

Содержание

Обработка пустых переменных	183
Обработка переменных по умолчанию	183
Присвоение переменной заголовка (title) заголовку шаблона.	184
Даты	185
WAR/WML	186
Составные шаблоны	187
Соккрытие E-mail адреса	188

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 есть более короткий путь в Smarty используя *default* модификатор переменной.

Пример 18.1. Вывод , если переменная пуста

```
{* длинный путь *}

{if $title eq ""}
    &nbsp;
{else}
    { $title }
{/if}

{* короткий путь *}

{ $title|default:"&nbsp;" }
```

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

```
{* вверху шаблона *}
{assign var="title" value=$title|default:"no title"}
```

```
{* если переменная $title была пустой, то сейчас она содержит "no title" *}
{$title}
```

Присвоение переменной заголовка (title) заголовку шаблона.

Если большинство ваших шаблонов имеют похожие верхние и нижние части, то имеет смысл вынести их в отдельные файлы. Но если шапка должна иметь различные заголовки на различных страницах? Смотрите пример ниже.

Пример 18.3. Присвоение переменной заголовка (title) заголовку шаблона.

```
mainpage.tpl
-----

{include file="header.tpl" title="Main Page"}
{* тело шаблона *}
{include file="footer.tpl"}

archives.tpl
-----

{config_load file="archive_page.conf"}
{include file="header.tpl" title=#archivePageTitle#}
{* тело шаблона *}
{include file="footer.tpl"}

header.tpl
-----
<HTML>
<HEAD>
<TITLE>{$title|default:"BC News"}</TITLE>
</HEAD>
<BODY>

footer.tpl
-----
</BODY>
</HTML>
```

Если выводится главная страница, то заголовок будет "Main Page", если архивы, то заголовок берется из файла конфигурации. Если заголовок будет пустой, то выведется значение по умолчанию "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 принимает `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, программистам может понадобиться получить дату в виде временной метки. Вот функция, которая поможет это сделать.

Пример 18.5. Преобразование элементов формы ввода даты назад к временной метке

```
<?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)) {
        $year = strtotime("%Y");
    }
    if(empty($month)) {
        $month = strtotime("%m");
    }
}
```

```

if(empty($day)) {
    $day = strftime("%d");
}

return mktime(0, 0, 0, $month, $day, $year);
}
?>

```

WAP/WML

WAP/WML шаблоны требуют, чтобы заголовок Content-type был передан вместе с шаблоном. Простейший путь - написать пользовательскую функцию, которая будет выводить заголовки. Так как мы будем пользоваться тэгами insert, то заголовки не будут кэшироваться. Так же ничего не должно выводиться в браузер до шаблона, иначе заголовки могут быть неправильно восприняты.

Пример 18.6. Использование insert для записи WML Content-Type заголовков

```

<?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>

```

```

</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 РНР (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 собственный плагин для получения данных и присваивания их переменной шаблона.

Пример 18.7. составной шаблон

```

<?php

// поместите файл "function.load_ticker.php" в директорию плагинов

// 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);
}

?>

```

```
{* in index.tpl *}
```

```
{load_ticker symbol="YHOO" assign="ticker"}
```

Название ценной бумаги: {\${ticker.name}} Цена ценной бумаги: {\${ticker.price}}

Соккрытие 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, чтобы 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 плагина mailto.

Пример 18.8. Пример соккрытия e-mail адреса

```
{* in 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 декодирование этих значений, но это маловероятно.

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.