Smarty - a ferramenta para compilar templates para PHP

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

Fernando Correa da Conceição <fernandoc@php.net>
Marcelo Perreira Fonseca da Silva <marcelo@php.net>
Taniel Franklin <taniel@ig.com.br>
Thomas Gonzalez Miranda <thomasgm@php.net>

Smarty - a ferramenta para compilar templates para PHP

by Monte Ohrt <monte at ohrt dot com> and Andrei Zmievski <andrei@php.net>
by Fernando Correa da Conceição <fernandoc@php.net>, Marcelo Perreira Fonseca da Silva <marcelo@php.net>, Taniel Franklin <taniel@ig.com.br>, and Thomas Gonzalez Miranda <thomasgm@php.net>

Publication date 2010-09-20

Table of Contents

List of Examples

Prefácio

Esta é sem dúvida uma das perguntas mais freqüentes nas listas de discussões sobre PHP: como eu faço meus scripts em PHP independentes do layout? O PHP é vendido como sendo uma "linguagem de script embutida no HTML", após escrever alguns projetos que misturam HTML e PHP naturalmente vem uma idéia de que a separação da forma e conteúdo é uma boa prática [TM]. Além disso, em muitas empresas os papéis de designer e programador são separados. Conseqüentemente, a busca por um sistema de templates continua.

Na nossa empresa por exemplo, o desenvolvimento de uma aplicação é feito da seguinte maneira: Após a documentação necessária estar pronta, o designer faz o esboço da interface e entrega ao programador. O programador implementa as regras de negócio no PHP e usa o esboço da interface para criar o esqueleto dos templates. O projeto então está nas mãos da pessoa responsável pelo layout HTML da página que então transforma o esboço em um layout realmente funcional. O projeto talvez vá e volte entre programação/designer HTML várias vezes. Porém, é importante ter um bom suporte à templates porque os programadores não querem ter que ficar mexendo com HTML e não querem que os designers estraguem seus códigos PHP. Os designers precisam de ajuda para alterar os arquivos de configuração, blocos dinâmicos e outros problemas relacionados à interface usada, mas eles não querem ocupar-se com as complexidades da linguagem de programação PHP.

Analisando muitas das soluções de templates disponíveis para PHP hoje em dia, a maioria somente disponibilizada uma forma rudimentar de substituição de variáveis dentro dos templates e trabalham de forma limitada com as funcionalidades dos blocos dinâmicos. Mas nossas necessidades necessitam de um pouco mais do que isso. Nós não queríamos que programadores mexendo com layout em HTML, mas isso é praticamente inevitável. Por exemplo, se um designer quiser que as cores de fundo se alternam em blocos dinâmicos, isso tem que ser feito pelo programador antecipadamente. Nós também precisamos que os designers possam usar seus próprios arquivos de configuração, e usar as variáveis definidas nestes arquivos em seus templates. E a lista de necessidades continua...

Nós começamos à escrever as especificações para um sistema de templates por volta de 1999. Após o término das especificações, nós começamos a escrever um sistema de template em C que esperávamos ser aceito para rodar com o PHP. Não só esbarramos em muitas barreiras técnicas, como também houve um enorme debate sobre o que exatamente um sistema de template deveria ou não fazer. À partir desta experiência, nós decidimos que o sistema de template fosse escrito para ser uma classe do PHP, para que qualquer um usa-se da forma que lhe fosse mais conveniente, então nós escrevemos um sistema que fazia exatamente, foi aí que surgiu o SmartTemplate™ (obs: esta classe nunca foi enviada ao público). Foi uma classe que fez quase tudo que nós queríamos: substituição de variáveis, suporte à inclusão de outros templates, integração com arquivos de configuração, código PHP embutido, funcionalidades 'if' limitada e blocos dinâmicos muito mais robustos que poderiam ser aninhados muitas vezes. Foi tudo feito usando expressões reguladores e códigos confusos, como diríamos, impenetrável. Era um sistema também extremamente lento em grandes aplicativos por causa de todo o trabalho que era feito pelas expressões regulares e o 'parsing'(interpretação) em cada chamada ao aplicativo. O maior problema do ponto de vista de um programador foi o espantoso trabalho que era necessário para configurar e processar os blocos dinâmicos dos templates. Como faríamos esse sistema ser simples de usar?

Foi então que veio a visão do que hoje é conhecido como Smarty. Nós sabemos o quão rápido é um código PHP sem o sobrecarregamento de um sistema de templates. Nós também sabemos quão meticuloso e assustador é a linguagem PHP aos olhos de um designer atual, e isso tudo poderia ser mascarado usando uma sintaxe simples nos templates. Então o que acontece se nós combinarmos essas duas forças? Assim, nasceu o Smarty...

Part I. Iniciando

Table of Contents

Chapter 1. O que é o Smarty?

O Smarty é um sistema de templates para PHP. Mais especificamente, ele fornece uma maneira fácil de controlar a separação da aplicação lógica e o conteúdo de sua apresentação. Isto é melhor descrito em uma situação onde o programador da aplicação e o designer do template executam diferentes funções, ou na maioria dos casos não são a mesma pessoa.

Por exemplo, digamos que você está criando uma página para web para mostrar um artigo de um jornal. O autor, a manchete, a conclusão e o corpo do artigo são elementos de conteúdo, eles não contém informação alguma sobre como eles devem ser mostrados. Ele são enviados ao Smarty pela aplicação, então o designer do template edita o template e usa uma combinação de tags HTML e tags de templates para formatar a apresentação destes elementos (tabelas HTML, cores de fundo, tamanhos de fontes, folhas de estilos, etc.). Se algum dia o programador precisar alterar a maneira como o conteúdo do artigo é tratado (uma mudança na lógica da aplicação). Esta mudança não afeta o design do template, o conteúdo será enviado ao template exatamente da mesma forma. De modo semelhante, se o designer do template quiser redesenhar completamente os templates, não é necessária nenhuma alteração na lógica da aplicação. Sendo assim, o programador pode fazer mudanças na lógica da aplicação sem a necessidade de reestruturar os templates, e o designer do template pode fazer mudanças nos templates sem alterar a lógica da aplicação.

Um objetivo do projeto Smarty é a separação da lógica do negócio e da lógica da apresentação. Isto significa que os templates podem certamente conter a lógica sob a circunstância que é somente para apresentação. Alguns exemplos são: a inclusão de outros templates, alternação de cores nas linhas das tabelas, colocar o texto de uma variável em maiúsculo, percorrer uma matriz de dados e mostrá-la, etc. são todos exemplos de apresentação lógica. Isto não significa que o Smarty força a separação da lógica de negócios e da lógica de apresentação. O Smarty não tem conhecimento do que é o que em sua aplicação, portanto colocar sua a lógica de negócio no template é problema seu. Caso você deseje que não haja *nenhuma* lógica em seus templates você pode certamente fazer isso trocando o conteúdo para textos e variáveis somente.

Um dos aspectos únicos do Smarty é seu sistema de compilação de templates. O Smarty lê os arquivos de templates e cria scripts PHP à partir deles. Uma vez criados, eles são executados sem ser necessário uma outra compilação do template novamente. Com isso, os arquivos de template não são 'parseados'(analisados) toda vez que um template é solicitado, e cada template tem a total vantagem de soluções de cache do compilador PHP, tais como: Zend Accelerator (http://www.zend.com/) ou PHP Accelerator (http://www.php-accelerator.co.uk).

Algumas das características do Smarty:

- Ele é extremamente rápido.
- Ele é eficiente visto que o interpretador do PHP faz o trabalho mais pesado.
- Sem elevadas interpretações de template, apenas compila uma vez.
- Ele está atento para só recompilar os arquivos de template que foram mudados.
- Você pode fazer funções próprias e seus próprios modificadores de variáveis, assim a linguagem de templates é extremamente extensível.
- Delimitadores de tag configuráveis, sendo assim você pode usar {}, {{}}, <!--{}-->, etc.
- Os construtores if/elseif/else/endif são passados para o interpretador de PHP, assim a sintaxe de expressão {if ...} pode ser tanto simples quanto complexa da forma que você queira.
- Aninhamento ilimitado de sections, ifs, etc. permitidos.

- É possível embutir o código PHP diretamente em seus arquivos de template, apesar de que isto pode não ser necessário (não recomendado) visto que a ferramenta é tão customizável.
- Suporte de caching embutido.
- Fontes de template arbitrários.
- Funções de manipulação de cache customizadas.
- Arquitetura de Plugin.

Chapter 2. Instalação

Requisitos

Smarty requer um servidor web rodando o PHP 4.0.6 superior.

Instalação Básica

Instale os arquivos da biblioteca do Smarty que estão no subdiretório /libs/ da distribuição. Estes são os arquivos PHP que você NÃO PRECISA editar. Eles são comuns a todas as aplicações e eles só são atualizados quando você atualiza para uma nova versão do Smarty.

Example 2.1. Arquivos da biblioteca do Smarty necessários

```
Smarty.class.php
Smarty_Compiler.class.php
Config_File.class.php
debug.tpl
/internals/*.php (all of them)
/plugins/*.php (todos eles para ser seguro, talvés a sua pagina precise de apenas
```

O Smarty utiliza uma constante [http://php.net/define] do PHP chamada SMARTY_DIR que é o **caminho completo** para o diretório 'libs/' do Smarty. Basicamente, se sua aplicação puder encontrar o arquivo `Smarty.class.php`, você não precisa definir SMARTY_DIR, o Smarty irá encontrar por si só. Entretanto, se `Smarty.class.php` não estiver em seu include_path, ou você não indicar um caminho absoluto para ele em sua aplicação, então você deverá definir SMARTY_DIR manualmente. SMARTY_DIR **deve incluir uma barra ao final**.

Aqui está um exemplo de como você cria uma instância do Smarty em seus scripts PHP:

Example 2.2. Cria uma instância do Smarty

```
NOTE: Smarty has a capital 'S'
require_once('Smarty.class.php');
$smarty = new Smarty();
```

Tente rodar o script acima. Se você obtiver um erro dizendo que o arquivo `Smarty.class.php` não pôde ser encontrado, você tem que fazer uma das coisas a seguir:

Example 2.3. Definir a constante SMARTY_DIR manualmente

```
// *nix style (note capital 'S')
define('SMARTY_DIR', '/usr/local/lib/php/Smarty-v.e.r/libs/');

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

// hack version example that works on both *nix and windows
// Smarty is assumend to be in 'includes/' dir under current script
define('SMARTY_DIR',str_replace("\\","/",getcwd()).'/includes/Smarty-v.e.r/libs/')

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

Example 2.4. Adicionar o diretório da biblioteca para o include_path do PHP

```
// Edite o seu arquivo php.ini, adicione o diretório da biblioteca do Smarty
// para o include_path e reinicie o servidor web.
// Então o código a seguir funcionaria:
require('Smarty.class.php');
$smarty = new Smarty;
```

Example 2.5. Defina a constante SMARTY_DIR manualmente

```
define('SMARTY_DIR','/usr/local/lib/php/Smarty/');
require(SMARTY_DIR.'Smarty.class.php');
$smarty = new Smarty;
```

Agora que os arquivos da biblioteca estão no lugar, é hora de configurar os diretórios do Smarty para a sua aplicação.

O Smarty necessita de quatro diretórios, que são chamados por padrão `'templates/'`, `'templates_c/'`, `'configs/'` e `'cache/'`.

Cada um deles pode ser definido pelas propriedades da classe Smarty `$template_dir`, `$compile_dir`, `$config_dir`, e `$cache_dir` repectivamente. É altamente recomendado que você configure um conjunto diferente destes diretórios para cada aplicação que for usar o Smarty.

Certifique-se que você sabe a localização do 'document root' do seu servidor web. Em nosso exemplo, o 'document root' é `"/web/www.mydomain.com/docs/"`. Os diretórios do Smarty só são acessados pela biblioteca do Smarty e nunca acessados diretamente pelo navegador. Então para evitar qualquer preocupação com segurança, é recomendado colocar estes diretórios *fora* do document root.

Para o nosso exemplo de instalação, nós estaremos configurando o ambiente do Smarty para uma aplicação de livro de visitas. Nós escolhemos uma aplicação só para o propósito de uma convenção de nomeação de diretório. Você pode usar o mesmo ambiente para qualquer aplicação, apenas substitua "guestbook" com o nome de sua aplicação. Nós colocaremos nossos diretórios do Smarty dentro de `"/web/www.mydomain.com/smarty/guestbook/"`.

Você precisará pelo menos de um arquivo dentro de seu 'document root', e que seja acessado pelo navegador. Nós chamamos nosso script de *"index.php"*, e o colocamos em um subdiretório dentro do 'document root' chamado `"/guestbook/"`.

Nota Técnica

É conveniente configurar o servidor web para que 'index.php' possa ser idendificado como o índice padrão do diretório, asssim se você acessar http://www.example.com/guestbook/, o script 'index.php' será executado sem adicionar 'index.php' na URL. No Apache você pode configurar isto adicioanando "index.php" ao final da sua configuração *DirectoryIndex* (separe cada item com um espaço.) como no exemplo de httpd.conf

DirectoryIndex index.htm index.html index.php index.php3 default.html index.cgi

Vamos dar uma olhada na estrutura de arquivos até agora:

Example 2.6. Exemplo de estrutura de arquivo

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

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

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

O Smarty irá precisar de **acesso de escrita** (usuários de windows por favor ignorem) em *$compile_dir* e *$cache_dir*, então tenha certesa que o usuário do servidor web possa escrever. Este é geralmente o usuário "nobody" e o grupo "nobody" (ninguém). Para SO com X usuários, o usuário padrão é "www" e o grupo "www". Se você está usando Apache, você pode olhar em seu arquivo httpd.conf (normalmente em "/usr/local/apache/conf/") para ver qual o usuário e grupo estão sendo usados.

Example 2.7. Configurando permissões de arquivos

```
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/
```

Nota Técnica

chmod 770 será a segurança correta suficientemente restrita, só permite ao usuário "nobody" e o grupo "nobody" acesso de leitura/escrita aos diretórios. Se você gostaria de abrir o acesso de leitura para qualquer um (na maioria das vezes para sua própria conveniência de querer ver estes arquivos), você pode usar o 775 ao invés do 770.

Nós precisamos criar o arquivo "index.tpl" que o Smarty vai ler. Ele estará localizado em seu $template_dir.

Example 2.8. Editando /web/www.example.com/smarty/guestbook/templates/index.tpl

```
{* Smarty *}

Ola! {$name}, bem vindo ao Smarty!
```

Nota Técnica

{* Smarty *} é um comentário de template. Ele não é exigido, mas é uma prática boa iniciar todos os seus arquivos de template com este com este comentário. Isto faz com que o arquivo seja reconhecido sem levar em consideração a sua extensão. Por exemplo, editores de texto poderiam reconhecer o arquivo e habilitar coloração de sintaxe especial.

Agora vamos editar 'index.php'. Nós vamos criar uma instancia do Smarty, assign(definir) uma variável do template e display (mostrar) o arquivo 'index.tpl'.

Example 2.9. Editando /web/www.example.com/docs/guestbook/index.php

```
<?php

// load Smarty library
require_once(SMARTY_DIR . 'Smarty.class.php');

$smarty = new Smarty();

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

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

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

Nota Técnica

No nosso exemplo, nós estamos definindo caminhos absolutor para todos os diretórios do Smarty. Se `/web/www.example.com/smarty/guestbook/` estiver dentro do seu include_path do PHP, então estas definições não são necessárias. Entretando, é mais eficinte e (com experiência) causa menos erros definir como caminhos absolutos. Isto faz ter certeza que o Smarty esta lendo os arquivos dos diretórios que você quer.

Agora carregue o arquivo `index.php` em seu navegador. Você veria "Olá, Thomas! bem vindo ao Smarty"

Você completou a configuração básica para o Smarty!

Estendendo a configuração

Esta é uma continuação da instalação básica, por favor leia a instalação básica primeiro!

Uma forma um pouco mais flexível de configurar o Smarty é estender a classe e inicializar seu ambiente de Smarty. Então, ao invés de configurar caminhos de diretórios repetidamente, preencher as mesmas variáveis, etc., nós podemos fazer isso para facilitar. Vamos criar um novo diretório "/php/includes/guestbook/" e criar um novo arquivo chamado `"setup.php"`. Em nosso ambiente de exemplo, "/php/includes" está em nosso include_path. Certifique-se de que você também definiu isto, ou use caminhos de arquivos absolutos.

Example 2.10. Editando /php/includes/guestbook/setup.php

```
<?php
// Carrega a biblioteca Smarty
require('Smarty.class.php');

// O arquivo setup.php é um bom lugar para carregar
// arquivos necessarios para a aplicação e você
// pode faze-lo aqui mesmo. Um exemplo:
// require('guestbook/guestbook.lib.php');

class Smarty_GuestBook extends Smarty {

    function Smarty_GuestBook()
    {

        // Construtor da classe.
        // Este é chamado a cada nova instância.

        $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');
    }

}
?>
```

Agora vamos alterar o arquivo index.php para usar o setup.php:

Example 2.11. Editando /web/www.example.com/docs/guestbook/index.php

```
require('guestbook/setup.php');

$smarty = new Smarty_GuestBook;

$smarty->assign('nome','Thomas');

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

Agora você pode ver que é extremamente simples criar uma instância do Smarty, apenas use Smarty_GuestBook que automaticamente inicializa tudo para a nossa aplicação.

Part II. Smarty para Designers de Template

Table of Contents

Chapter 3. Sintaxe Básica

Todas as tags de template do Smarty contém delimitadores. Por padrão, estes delimitadores são { e }, mas eles podem ser alterados.

Para os exemplos à seguir, nós assumiremos que você está usando os delimitadores padrão. Para o Smarty, todo o conteúdo fora dos delimitadores é mostrado como conteúdo estático, ou inalterável. Quando o Smarty encontra tags de template, ele tenta interpretá-las, e então mostra a saída apropriada em seu lugar.

Comentários

Os comentários do template ficam entre asteriscos dentro de delimitadores, exemplo: {* este é um comentário *}. Comentários do Smarty não são exibidos no resultado final do template. Eles são usados para fazer anotações internas nos templates.

Example 3.1. Comentários

```
{* Smarty *}

{* inclua o arquivo de cabeçalho aqui *}
{include file="cabecalho.tpl"}

{include file=$arquivoInclude}

{include file=#arquivoInclude#}

{* mostra lista dropdown *}
<select name="empresa">
{html_options values=$vals selected=$selected output=$output}
</select>
```

Funções

Cada tag Smarty mostra uma variável ou utiliza algum tipo de função. Funções são processadas e exibidas colocando-se a função e seus atributos entre delimitadores, exemplo: {funcname attr1="val" attr2="val"}.

Example 3.2. Sintaxe de funções

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

{include file="cabecalho.tpl"}

{if $enfase_nome}
    Seja bem-vindo, <font color="{#corFonte#}">{$nome}!</font>
{else}
    Seja bem-vindo, {$nome}!
{/if}

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

Ambas as funções internas e as funções personalizadas tem a mesma sintaxe nos templates. Funções internas são o funcionamento do Smarty, tais como **if**, **section** e **strip**. Elas não podem ser modificadas. Funções personalizadas são funções adicionais implementadas por modo de plugins. Elas podem ser modificadas como você quiser, ou você pode adionar novas. **html_options** e **html_select_date** são exemplos de funções personalizadas.

Atributos

A maioria das funções contém atributos que especificam ou modificam o seu comportamento. Atributos para funções do Smarty são muito parecidos com atributos da HTML. Valores estáticos não precisam ficar entre aspas, mas recomenda-se usar aspas para strings literais. Variáveis também podem ser usadas, e não precisam estar entre aspas.

Alguns atributos exigem valores booleanos (verdadeiro ou falso). Estes valores podem ser especificados sem aspas `true`, `on`, e `yes`, ou `false`, `off`, e `no`.

Example 3.3. Sintaxe de atributos de funções

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

{include file=$arquivoInclude}

{include file=#arquivoInclude#}

{html_select_date display_days=yes}

<select name="empresa">
{html_options values=$vals selected=$selected output=$output}
</select>
```

Colocando Variáveis em Aspas Duplas

Smarty irá reconhecer variáveis definidas entre asplas duplas enquanto as variáveis conterem apenas números, letras, sublinhados e conchetes []. Com qualquer outro caractere (pontos, referência à objetos, etc.) a variável deve estar entre apóstrofos.

Example 3.4. Sintaxe entre aspas

```
EXEMPLOS DE SINTAXE:
{func var="teste $foo teste"}       <-- mostra $foo
{func var="teste $foo_bar teste"}   <-- mostra $foo_bar
{func var="teste $foo[0] teste"}    <-- mostra $foo[0]
{func var="teste $foo[bar] teste"}  <-- mostra $foo[bar]
{func var="teste $foo.bar teste"}   <-- mostra $foo (e não $foo.bar)
{func var="teste `$foo.bar` teste"} <-- mostra $foo.bar

EXEMPLOS PRÁTICOS:
{include file="subdir/$tpl_name.tpl"} <-- substitui $tpl_name pelo seu valor
{cycle values="one,two,`$smarty.config.myval`"} <-- deve conter apóstrofos
```

Matemática

Matemática pode ser aplicada diretamente aos valores de variáveis.

Example 3.5. Exemplos de matemática

```
{$foo+1}

{$foo*$bar}

{* alguns exemplos mais complicados *}

{$foo->bar-$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`"}
```

Escapando da interpretação do Smarty

Algumas vezes é desejável ou mesmo necessário fazer o Smarty ignorar sessões que em outro caso ele interpretaria. Um exemplo classico é embutindo Javascript ou código CSS no template. O problema aparece porque estas linguagens usam os caracteres { e } que são os delimitadores padrão para o Smarty.

A coisa mais simples é evitar a situação em sí separando o seu código Javascript e CSS nos seus próprios arquivos e então usar os métodos padrões do HTML para acessa-los.

Incluir conteúdo literal é possível usando blocos {literal} .. {/literal}. De modo similar ao uso de entidades HTML, você pode usar {ldelim},{rdelim} ou {$smarty.ldelim} para mostrar os delimitadores atuais.

As vezes é conveniente simplesmente mudar $left_delimiter e $right_delimiter.

Example 3.6. Exemplo de modificar os delimitadores

```
<?php

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

?>
```

Aonde example.tpl é:

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

Veja também escape modifier

Chapter 4. Variáveis

No Smarty há vários tipos diferentes de variáveis. O tipo da variável depende do prefixo que ela usa (ou do símbolo pelo qual ela está contida).

Variáveis no Smarty podem tanto serem exibidas diretamente ou usadas como argumentos para atributos de funções e modificadores, dentro de expressões condicionais, etc. Para que uma variável seja exibida o nome dela deve estar dentro dos delimitadores e não pode conter nenhum outro caracter. Veja os exemplos abaixo:

```
{$Nome}

{$Contatos[row].Telefone}

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

Variáveis definidas do PHP

Variáveis que são definidas do PHP são referenciadas precedendo elas com um sinal de sifrão $. Variáveis definidas dentro do template com a função assign também são mostradas desta maneira.

Example 4.1. Variáveis definidas

```
Hello {$firstname}, glad to see you could make it.
<p>
Your last login was on {$lastLoginDate}.

MOSTRA:

Hello Doug, glad to see you could make it.
<p>
Your last login was on January 11th, 2001.
```

Associative arrays

Você também pode referenciar matrizes associativas que são definidas no PHP especificando a chave depois do símbolo '.' (ponto).

Example 4.2. Acessando variáveis de matriz associativa

```
index.php:

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

index.tpl:

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

MOSTRA:

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

Índices de Matrizes

Você pode referencia matrizes pelo seu índice, muito parecido com a sintaxe nativa do PHP.

Example 4.3. Acesando matrizes por seus índices

```
index.php:

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

index.tpl:

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

MOSTRA:

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

Objetos

Propriedades de objetos definidos do PHP podem ser referenciados especificando-se o nome da propriedade depois do símbolo '->'.

Example 4.4. Acessando propriedades de objetos

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

MOSTRA:

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

Variáveis carregadas de arquivos de configuração

Variáveis que são carregadas de arquivos de configuração são referenciadas colocando-se elas entre cancelas (#), ou com a variável smarty $smarty.config. A segunda sintaxe é útil para coloca-las entre aspas em um atributo.

foo.conf:

Example 4.5. Variáveis de configuração

```
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: (sintaxe alternativa)

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


SAÍDA: (mesma para ambos exemplos)

<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>
```

Variáveis de um arquivo de configuração não podem ser usadas até que sejam carregadas de um arquivo de configuração. Este procedimento é explicado posteriormente neste documento em **config_load**.

A variável reservada {$smarty}

A variável reservada {$smarty} pode ser utilizada para acessar variáveis especiais do template. Segue uma lista completa.

Variáveis Request

Variáveis request como get, post, cookies, server, environment, e session podem ser acessadas como mostrado nos exemplos abaixo:

Example 4.6. Mostrando váriáveis request

```
{* mostra o valor de page da URL (GET) http://www.domain.com/index.php?page=foo *}
{$smarty.get.page}

{* mostra a variável "page" de um formulário (POST) *}
{$smarty.post.page}

{* mostra o valor do cookie "username" *}
{$smarty.cookies.username}

{* mostra a variável do servidor "SERVER_NAME" *}
{$smarty.server.SERVER_NAME}

{* mostra a variável de ambiente do sistema "PATH" *}
{$smarty.env.PATH}

{* mostra a variável de session do php "id" *}
{$smarty.session.id}

{* mostra a variável "username" da união de get/post/cookies/server/env *}
{$smarty.request.username}
```

{$smarty.now}

O timestamp atual pode ser acessado com {$smarty.now}. O número reflete o número de segundos passados desde o assim chamado Epoch (1 de Janeiro de 1970) e pode ser passado diretamente para o modificador date_format para mostrar a data.

Example 4.7. Usando {$smarty.now}

```
{* usa o modificador date_format para mostrar a data e hora atuais *}
{$smarty.now|date_format:"%Y-%m-%d %H:%M:%S"}
```

{$smarty.const}

Você pode acessar o valor de constantes PHP diretamente.

Example 4.8. Usando {$smarty.const}

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

{$smarty.capture}

A saída capturada via {capture}..{/capture} pode ser acessada usando a variável {$smarty}. Veja a a seção sobre capture para um exemplo.

{$smarty.config}

A variável {$smarty} pode ser usada para referir variáveis de configuração carregadas. {$smarty.config.foo} é um sinonimo para {#foo#}. Veja a seção sobre config_load para um exemplo.

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

A variável {$smarty} pode ser usada para se referir a propriedades 'section' e 'foreach' de loop. Veja a documentação sobre section e foreach.

{$smarty.template}

Esta variável contém o nome do template atual que esta sendo processado.

{$smarty.ldelim}

This variable is used for printing the left-delimiter value literally. See also {ldelim},{rdelim}.

{$smarty.rdelim}

This variable is used for printing the right-delimiter value literally. See also {rdelim},{rdelim}.

Chapter 5. Modificadores de variáveis

Modificadores de variáveis podem ser aplicados a variáveis, funções personalizadas ou strings. Para aplicar um modificador, especifique o valor seguido por `|`(pipe) e o nome do modificador. Um modificador aceita parâmetros adicionais que afetam o seu comportamento. Estes parâmetros vem após o nome do modificador e são separados por `:` (dois pontos).

Example 5.1. Exemplo de modificador

```
{* Faz o título ficar com letras maiúsculas *}
<h2>{$titulo|upper}</h2>

{* Faz com que $topico use somente 40 caracteres, e coloca ... no fim da frase *}

Tópico: {$topico|truncate:40:"..."}

{* transforma a data em um formato legível *}
{"agora"|date_format:"%Y/%m/%d"}

{* aplica um modificador à uma função personalizada *}
{mailto|upper address="eu@dominio.dom"}
```

Se você aplicar um modificador à uma matriz ao invés de aplicar ao valor de uma variável, o modificador vai ser aplicado à cada valor da matriz especificada. Se você quer que o modificador use a matriz inteira como um valor, você deve colocar o símbolo `@` antes do nome do modificador, como a seguir: `{$tituloArtigo|@count}` (isto irá mostrar o número de elementos na matriz $tituloArtigo).

Modificadores podem ser carregados automaticamente à partir do seu $plugins_dir (veja: Nomes sugeridos) ou podem ser registrados explicitamente (veja: register_modifier). Adicionalmente, todas as funções php podem ser utiliadas como modificadores implicitamente. (O exemplo do `@count` acima usa a função count do php e não um modificador do Smarty). Usar funções do php como modificadores tem dois pequenos problemas: Primeiro: às vezes a ordem dos parâmetros da função não é a desejada (`{"%2.f"|sprintf:$float}` atualmente funciona, mas o melhor seria algo mais intuitivo. Por exemplo: `{$float|string_format:"%2.f"}` que é disponibilizado na distribuição do Smarty). Segundo: com a variável $security ativada em todas as funções do php que são usadas como modificadores precisam ser declaradas como confiáveis (trusted) na matriz $security_settings['MODIFIER_FUNCS'].

capitalize

Isto é usado para converter para maiúsculas a primeira letra de todas as palavras em uma variável.

Example 5.2. capitalize

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Police begin campaign to rundown jaywalkers.');
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|capitalize}

SAÍDA:

Police begin campaign to rundown jaywalkers.
Police Begin Campaign To Rundown Jaywalkers.
```

count_characters

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	boolean	Não	false	Isto determina quando incluir ou não os espaços em branco na contagem.

Isto é usado para contar o número de caracteres em uma variável.

Example 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}
{$articleTitle|count_characters:true}
MOSTRA:

Cold Wave Linked to Temperatures.
29
32
```

cat

Posição do Parâmetro	Tipo	Requerido	cat	Descrição
1	string	Não	*empty*	Este é o valor para concatenar com a variável dada.

Este valor é concatenado com a variável dada.

Example 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."}

MOSTRA:

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

count_paragraphs

Isto é usado para contar o número de paragrafos em uma variável.

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

index.tpl:

{$articleTitle}
{$articleTitle|count_paragraphs}

MOSTRA:

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

Isto é usado para contar o número de sentenças em uma variável.

Example 5.6. count_sentences

```
index.php:

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

index.tpl:

{$articleTitle}
{$articleTitle|count_sentences}

MOSTRA:

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

count_words

Isto é usado para contar o número de palavras em uma variável.

Example 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}

MOSTRA:

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

date_format

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	string	Não	%b %e, %Y	Este é o formato para a data mostrada.
2	string	Não	n/a	Esta é a data padrão se a entrada estiver vazia.

Isto formata a data e hora no formato strftime() indicado. Datas podem ser passadas para o Smarty como timestamps unix, timestamps mysql, ou qualquer string composta de mês dia ano(interpretavel por strtotime). Designers podem então usar date_format para ter um controle completo da formatação da data. Se a data passada para date_format estiver vazia e um segundo parâmetro for passado, este será usado como a data para formatar.

Example 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"}

MOSTRA:

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

Example 5.9. date_format conversion specifiers

```
%h - o mesmo que %b

%H - hora como um número decimal usando um relógio de 24 horas (intervalo de 00 a

%I - hora como um número decimal usando um relógio de 12 horas (intervalo de 01 a
%j - dia do ano como um número decimal (intervalo de 001 a 366)

%k - hora (relógio de 24 horas) digítos únicos são precedidos por um espaço em bra

%l - hora como um número decimal usando um relógio de 12 horas, digítos unicos são
por um espaço em branco (intervalo de 1 a 12)

%m - mês como número decimal (intervalo de 01 a 12)

%M - minuto como um número decimal

%n - caractere de nova linha

%p - ou `am' ou `pm' de acordo com o valor de hora dado, ou as strings corresponde

%r - hora na notação a.m. e p.m.

%R - hora na notação de 24 horas

%S - segundo como número decimal

%t - caractere tab

%T - hora atual, igual a %H:%M:%S

%u - dia da semana como um número decimal [1,7], com 1 representando segunda-feira

%U - número da semana do ano atual como um número decimal, começando com o primeir

%V - número da semana do ano atual como um número decimal de acordo com The ISO 86
intervalo de 01 a 53, aonde a semana 1 é a primeira semana que tenha pelo menos qu

%w - dia da semana como decimal, domingo sendo 0

%W - número da semana do ano atual como número decimal, começando com a primeira s

%x - representação preferencial da data para o local atualsem a hora

%X - representação preferencial da hora para o local atual sem a data

%y - ano como número decimal sem o século (intervalo de 00 a 99)

%Y - ano como número decimal incluindo o século

%Z - zona horária ou nome ou abreviação

%% - um caractere `%'


NOTA PARA PROGRAMADORES: date_format é essencialmente um wrapper para a função str
Você deverá ter mais ou menos especificadores de conversão disponíveis de acordo c
função strftime() do sistema operacional aonde o PHP foi compilado. De uma olhada
na página de manual do seu sistema para uma lista completa dos especificadores vál
```

default

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	string	Não	*vazio*	Este é o valor padrão para mostrar se a variável estiver vazia.

Isto é usado para definir um valor padrão para uma variável. Se a variável estiver vazia ou não for definida, o valor padrão dado é mostrado. Default usa um argumento.

Example 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"}

MOSTRA:

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

escape

Posição do Parâmetro	Tipo	Requerido	Valores Possíveis	Padrão	Descrição
1	string	Não	html,htmlall,url,quotes,hex,hexentity,javascript	html	Este é o formato de escape para usar.

Este é usado para escapar html, url, aspas simples em uma variável que já não esteja escapada, escapar hex, hexentity ou javascript. Por padrão, é escapado o html da variável.

Example 5.11. escape

```
index.php:

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

index.tpl:

{$articleTitle}
{$articleTitle|escape}
{$articleTitle|escape:"html"}    {* escapa  & " ' < > *}
{$articleTitle|escape:"htmlall"} {* escapa todas as entidades html *}
{$articleTitle|escape:"url"}
{$articleTitle|escape:"quotes"}
<a href="mailto:{$EmailAddress|escape:"hex"}">{$EmailAddress|escape:"hexentity"}</

MOSTRA:

'Stiff Opposition Expected to Casketless Funeral Plan'
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;
%27Stiff+Opposition+Expected+to+Casketless+Funeral+Plan%27
\'Stiff Opposition Expected to Casketless Funeral Plan\'
<a href="mailto:%62%6f%62%40%6d%65%2e%6e%65%74">&#x62;&#x6f;&#x62;&#x40;&#x6d;&#x6
```

indent

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	integer	Não	4	Isto define com quantos caracteres endentar.
2	string	Não	(um espaço)	Isto define qual caractere usado para endentar.

Isto endenta uma string em cada linha, o padrão é 4. Como parâmetro opcional, você pode especificar o número de caracteres para endentar. Como segundo parâmetro opcional, você pode especificar o caractere usado para endentar. (Use "\t" para tabs.)

Example 5.12. indent

```
index.php:

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

index.tpl:

{$articleTitle}

{$articleTitle|indent}

{$articleTitle|indent:10}

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

MOSTRA:

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

Isto é usado para converter para minúsculas uma variável.

Example 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}

MOSTRA:

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

nl2br

Todas as quebras de linha serão convertidas para
 na variável data. Isto é equivalente a função nl2br() do PHP.

Example 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}

MOSTRA:

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

regex_replace

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	string	Sim	*n/a*	Esta é a expressão regular a ser substituída.
2	string	Sim	*n/a*	Esta é a string que irá substituir a expressão regular.

Uma expressão regular para localizar e substituir na variável. Use a sintaxe para preg_replace() do manual do PHP.

Example 5.15. regex_replace

```
index.php:

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

index.tpl:

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

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

MOSTRA:

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

replace

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	string	Sim	*n/a*	Esta é a string a ser substituida.
2	string	Sim	*n/a*	Esta é a string que irá substituir.

Um simples localizar e substituir.

Example 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

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	string	Não	*um espaço*	O que é inserido entre cada caractere da variável.

Insere um espaço entre cada caractere de uma variável. Você pode opcionalmente passar um caractere (ou uma string) diferente para inserir.

Example 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.
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^^o^^m^^e^^t^^h^^i^^n^^g^^ ^^W^^e^^n^^t^^ ^^W^^r^^o^^n^^g^^ ^^i^^n^^ ^^J^^e^^t^^
```

string_format

Posição do parâmetro	Tipo	Requerido	Padrão	Descrição
1	string	Sim	*n/a*	Este é o formato para ser usado. (sprintf)

Este é um meio para formatar strings, como números decimais e outros. Use a sintaxe para sprintf para a formatação.

Example 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"}

MOSTRA:

23.5787446
23.58
24
```

strip

Isto substitui todos os espaços repetidos, novas linhas e tabs por um único espaço ou a string indicada.

Nota

Se você quer substituir blocos de texto do template, use a função strip.

Example 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:" "}

MOSTRA:

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

strip_tags

Isto retira as tags de marcação, basicamente tudo entre < e >.

Example 5.20. strip_tags

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', "Blind Woman Gets <font face=\"helvetica\">New Kid
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|strip_tags}

MOSTRA:

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

truncate

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	integer	Não	80	Este determina para quantos caracteres truncar.

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
2	string	Não	...	Este é o texto para adicionar se truncar.
3	boolean	Não	false	Isto determina quando truncar ou não ao final de uma palavra(false), ou no caractere exato (true).

Isto trunca a variável para uma quantidade de caracteres, o padrão é 80. Como segundo parâmetro opcional, você pode especificar uma string para mostrar ao final se a variável foi truncada. Os caracteres da string são incluídos no tamanho original para a truncagem. por padrão, truncate irá tentar cortar ao final de uma palavra. Se você quizer cortar na quantidade exata de caracteres, passe o terceiro parâmetro, que é opcional, como true.

Example 5.21. truncate

```
index.php:

$smarty = new Smarty;
$smarty->assign('articleTitle', 'Two Sisters Reunite after Eighteen Years at Check
$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}

MOSTRA:

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

Isto é usado para converter para maiúsculas uma variável.

Example 5.22. upper

```
index.php:

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

index.tpl:

{$articleTitle}
{$articleTitle|upper}

MOSTRA:

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

wordwrap

Posição do Parâmetro	Tipo	Requerido	Padrão	Descrição
1	integer	Não	80	Isto determina em quantas colunas quebrar.
2	string	Não	\n	Esta é a string usada para quebrar.
3	boolean	Não	false	Isto determina quando quebrar ou não ao final de uma palavra (false), ou no caractere exato (true).

Isto quebra uma string para uma largura de coluna, o padrão é 80. Como segundo parâmetro opcional, você pode especificar a string que será usada para quebrar o texto para a próxima linha (o padrão é um retorno de carro \n). Por padrão, wordwrap irá tentar quebrar ao final de uma palavra. Se você quiser quebrar no tamanho exato de caracteres, passe o terceiro parâmetro, que é opcional, como true.

Example 5.23. wordwrap

```
index.php:

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

index.tpl:

{$articleTitle}

{$articleTitle|wordwrap:30}

{$articleTitle|wordwrap:20}

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

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

MOSTRA:

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

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

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

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

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

Chapter 6. Combinando Modificadores

Você pode aplicar a quantidade de moficadores que quiser à uma variável. Eles serão aplicados na ordem em que foram combinados, da esquerda para direita. Eles devem ser separados com o caracter | (pipe).

Example 6.1. combinando modificadores

```
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:". . ."}
```

O texto acima mostrará:

```
Smokers are Productive, but Death Cuts Efficiency.
S M O K E R S  A R E  P R O D U C T I V E ,  B U T  D E A T H  C U T S  E F
s m o k e r s  a r e  p r o d u c t i v e ,  b u t  d e a t h  c u t s...
s m o k e r s  a r e  p r o d u c t i v e ,  b u t . . .
s m o k e r s  a r e  p. . .
```

Chapter 7. Funções internas

O Smarty contém várias funções internas. Funções internas são parte integral da linguagem de template. Você não pode criar funções personalizadas com o mesmo nome de uma função interna, e também não pode modificar funções internas.

capture

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
name	string	Não	*default*	O nome do bloco capturado
assign	string	Não	*n/a*	O nome da variável para dar o valor da saída capturada

capture é usado para coletar toda a saída do template em uma variável ao invés de mostra-lo. Qualquer conteúdo entre {capture name="foo"} e {/capture} coletado na variável especificada no atributo name. O conteúdo capturado pode ser usado no template a partir da variável especial $smarty.capture.foo aonde foo é o valor passado para o atributo name. Se você não passar um atributo name, então será usado "default". Todos os comandos {capture} devem ter o seu {/capture}. Você pode aninhar(colocar um dentro de outro) comandos capture.

Nota Tecnica

Smarty 1.4.0 - 1.4.4 coloca o conteúdo capturado dentro da variável chamada $return. A partir do 1.4.5, este funcionamento foi mudado para usar o atributo name, então atualize os seus templates de acordo.

Caution

Tenha cuidado quando capturar a saída do comando **insert**. Se você tiver o cache em on e você tiver comandos **insert** que você espera que funcione com conteúdo do cache, não capture este conteúdo.

Example 7.1. capturando conteúdo do template

```
{* nós não queremos mostrar uma linha de tabela à não ser que haja conteúdo para e
{capture name=banner}
{include file="pegar_banner.tpl"}
{/capture}
{if $smarty.capture.banner ne ""}
 <tr>
  <td>
   {$smarty.capture.banner}
  </td>
 </tr>
{/if}
```

config_load

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
file	string	Sim	*n/d*	O nome do arquivo de configuração para incluir
section	string	Não	*n/d*	O nome da seção a carregar
scope	string	Não	*local*	Como o escopo das variáveis carregadas é tratado, o qual deve ser um entre local, parent ou global. local indica que as variáveis são carregadas no contexto do template local apenas. parent indica que as variáveis são carregadas no contexto atual e no template que o chamou. global indica que as variáveis estão disponíveis para todos os templates.
global	boolean	No	*No*	Quando ou não as variáveis são visiveis para o template superior(aquele que chamou este), o mesmo que scope=parent. NOTA: este atributo esta obsoleto devido ao atributo scope, mas ainda é suportado. Se scope for indicado, este valor é ignorado.

Esta função é usada para carregar as variáveis de um arquivo de configuração dentro de um template. Veja Arquivos de Configuração para mais informações.

Example 7.2. Função config_load

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

<html>
<title>{#tituloPagina#}</title>
<body bgcolor="{#cordeFundo}">
<table border="{#tamanhoBordaTabela}" bgcolor="{#cordeFundotabela#}">
 <tr bgcolor="{#cordeFundoLinha#}">
  <td>First</td>
  <td>Last</td>
  <td>Address</td>
 </tr>
</table>
</body>
</html>
```

Arquivos de configuração podem conter seções também. Você pode carregar variáveis de uma seção adicionando o atributo *section*.

NOTA: *Config file sections* e a função embutida de template *section* não tem nada a ver um com o outro, eles apenas tem uma mesma convenção de nomes.

Example 7.3. Função config_load com seções

```
{config_load file="cores.conf" section="Consumidor"}

<html>
<title>{#tituloPagina#}</title>
<body bgcolor="{#cordeFundo}">
<table border="{#tamanhoBordaTabela}" bgcolor="{#cordeFundotabela#}">
 <tr bgcolor="{#cordeFundoLinha#}">
  <td>First</td>
  <td>Last</td>
  <td>Address</td>
 </tr>
</table>
</body>
</html>
```

foreach,foreachelse

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
from	string	Sim	*n/d*	O nome da matriz que você estará pegando os elementos

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
item	string	Yes	*n/d*	O nome da variável que é o elemento atual
key	string	Não	*n/d*	O nome da variável que é a chave atual
name	string	Não	*n/d*	O nome do loop foreach para acessar as propriedades foreach

Loops *foreach* são uma alternativa para loops *section*. *foreach* é usado para pegar cada elemento de uma matriz associativa simples. A sintaxe para *foreach* é muito mais simples do que *section*, mas tem a desvantagem de poder ser usada apenas para uma única matriz. Tags *foreach* devem ter seu par */foreach*. Os parâmetros requeridos são *from* e *item*. O nome do loop foreach pode ser qualquer coisa que você queira, feito de letras, números e sublinhados. Loops *foreach* podem ser aninhados, e o nome dos loops aninhados devem ser diferentes um dos outros. A variável *from* (normalmente uma matriz de valores) determina o número de vezes do loop *foreach*. *foreachelse* é executado quando não houverem mais valores na variável *from*.

Example 7.4. foreach

```
{* este exemplo irá mostrar todos os valores da matriz $custid *}
{foreach from=$custid item=curr_id}
 id: {$curr_id}<br>
{/foreach}
```

MOSTRA:

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

Example 7.5. foreach key

```
{* A key contém a chave para cada valor do loop

A definição se parece com isso:

$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}
```

MOSTRA:

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

Loop foreach também tem as suas próprias variáveis para manipilar as propriedades foreach. Estas são indicadas assim: {$smarty.foreach.foreachname.varname} com foreachname sendo o nome especificado no atributo *name* do foreach.

iteration

iteration é usado para mostrar a interação atual do loop.

Iteration sempre começa em 1 e é incrementado um a um em cada interação.

first

first é definido como true se a interação atual do foreach for a primeira.

last

last é definido como true se a interação atual do foreach for a última.

show

show é usado como parâmetro para o foreach. *show* é um valor booleano, true ou false. Se false, o foreach não será mostrado. Se tiver um foreachelse presente, este será alternativamente mostrado.

total

total é usado para mostrar o número de interações do foreach. Isto pode ser usado dentro ou depois do foreach.

include

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
file	string	Sim	*n/d*	O nome do arquivo de template a incluir
assign	string	Não	*n/d*	O nome de uma variável que irá conter toda a saída do template
[var ...]	[var type]	Não	*n/d*	Variável para passar localmente para o template

Tags include são usadas para incluir outros templates no template atual. Quaisquer variáveis disponíveis no template atual também estarão disponíveis junto com template incluído. A tag include deve ter o atributo "file", o qual contém o caminho do arquivo a incluir.

Você pode opcionalmente passar o atributo *assign*, o qual irá especificar o nome de uma variável de template para a qual conterá todo o conteúdo do *include* ao invés de mostrá-lo.

Example 7.6. function include

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

{* O conteúdo do template vem aqui *}

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

Você pode também passar variáveis para o template incluído como atributos. Quaisquer variáveis passadas para um template incluído como atributos estão disponíveis somente dentro do escopo do template incluído. As variáveis passadas como atributos sobrescrevem as variáveis de template atuais, no caso de ambas terem o mesmo nome.

Example 7.7. Função include passando variáveis

```
{include file="cabecalho.tpl" title="Menu Principal" table_bgcolor="#c0c0c0"}

{* O conteúdo de template vem aqui *}

{include file="rodape.tpl" logo="http://meu.dominio.com/logo.gif"}
```

Use a sintaxe de template resources para incluir arquivos fora do diretório $template_dir.

Example 7.8. Exemplos de recursos para a função include

```
{* caminho absoluto *}
{include file="/usr/local/include/templates/cabecalho.tpl"}

{* caminho absoluto (idem) *}
{include file="file:/usr/local/include/templates/cabecalho.tpl"}

{* caminho absoluto do windows (DEVE usar o prefixo "file:") *}
{include file="file:C:/www/pub/templates/cabecalho.tpl"}

{* incluir a partir do recurso de template chamado "db" *}
{include file="db:header.tpl"}
```

include_php

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
file	string	Sim	*n/a*	O nome do arquivo php a incluir
once	boolean	Não	*true*	Quando incluir ou não o arquivo php mais de uma vez se incluído várias vezes
assign	string	Não	*n/a*	O nome da variável que receberá a saída do arquivo php

Nota Técnica

include_php está quase sendo retirado do Smarty, você pode obter a mesma funcionalidade usando uma função customizada em um template. A única razão para usar o include_php é se você realmente precisar deixar função php fora do diretório de plugin ou código da sua aplicação. Veja a seção templates componentizados para mais detalhes.

Tags include_php são usadas para incluir um script php no seu template. Se a segurança estiver ativada, então o script php deve estar localizado no diretório especificado na variável $trusted_dir. A tag include_php deve ter o atributo "file", o qual contém o caminho para o arquivo php incluído, pode ser um camiho tanto absoluto ou relativo a $trusted_dir.

include_php é um bom meio de manipular templates componentizados, e manter o código PHP separado dos arquivos de template. Digamos que você tenha um template que mostre a navegação do seu site, o qual é preenchido automaticamente a partir de um banco de dados. Você pode manter a sua lógica PHP que obtém os dados em um diretório separado, e inclui-la no topo do template. Agora você pode incluir este template em qualquer lugar sem se preocupar se a informação do banco de dados foi obtida antes de usar.

Por padrão, os arquivos php são incluídos apenas uma vez mesmo se incluídos várias vezes no template. Você pode especificar que ele seja incluído todas as vezes com o atributo *once*. Definindo once para false irá incluir o script php a cada vez que ele seja incluído no template.

Você pode opcionalmente passar o atributo *assign*, o qual irá especificar uma variável de template a qual irá conter toda a saída de *include_php* em vez de mostra-la.

O objeto smarty esta disponível como $this dentro do script php que você incluiu.

Example 7.9. Função include_php

```
load_nav.php
-------------

<?php

 // carrega variáveis de um banco de dados mysql e define elas para o template
 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
---------

{* caminho absoluto ou relativo a $trusted_dir *}
{include_php file="/caminho/para/load_nav.php"}

{foreach item="curr_section" from=$sections}
 <a href="{$curr_section.url}">{$curr_section.name}</a><br>
{/foreach}
```

insert

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
name	string	Sim	*n/d*	O nome da função insert (insert_name)
assign	string	Não	*n/d*	O nome da variável que irá receber a saída
script	string	Não	*n/d*	O nome de um script php que será incluido antes que a função insert seja chamada
[var ...]	[var type]	Não	*n/d*	Variável para passar para a função insert

Tags insert funcionam parecido com as tags include, exceto que as tags insert não vão para o cache quando caching esta ativado. Ela será executada a cada invocação do template.

Digamos que você tenha um template com um banner no topo da página. O banner pode conter uma mistura de html, imagens, flash, etc. Assim nós não podemos usar um link estatico aqui, e nós não queremos que este conteúdo fique no cache junto com a página. E aí que entra a tag insert: o template conhece os valores #banner_location_id# e #site_id# (obtidos de um arquivo de configuração), e precisa chamar uma função para obter o conteúdo do banner.

Example 7.10. função insert

```
{* exemplo de como obter um banner *}
{insert name="getBanner" lid=#banner_location_id# sid=#site_id#}
```

Neste exemplo, nós estamos usando o nome "getBanner" e passando os parâmetros #banner_location_id# e #site_id#. O Smarty irá procurar por uma função chamada insert_getBanner() na sua aplicação PHP, passando os valores de #banner_location_id# e #site_id# como primeiro argumento em uma matriz associativa. Todos os nomes de função insert em sua aplicação devem ser precedidas por "insert_" para prevenir possíveis problemas com nomes de funções repetidos. Sua função insert_getBanner() deve fazer alguma coisa com os valores passados e retornar os resultados. Estes resultados são mostrados no template no lugar da tag insert. Neste exemplo, o Smarty irá chamar esta função: insert_getBanner(array("lid" => "12345","sid" => "67890")); e mostrar o resultado retornado no lugar da tag insert.

Se você passar o atributo "assign", a saída da tag insert será dada para esta variável ao invés de ser mostrada no template.

Nota

definir a saída para uma variável não é útil quando o cache esta ativo.

Se você passar o atributo "script", este script php será incluido (apenas uma vez) antes da execução da função insert. Este é o caso onde a função insert não existe ainda, e um script php deve ser incluído antes para faze-la funcionar. O caminho pode ser absoluto ou relativo à variável $trusted_dir. Quando a segurança esta ativada, o script deve estar no local definido na variável $trusted_dir.

O objeto Smarty é passado como segundo argumento. Deste modo você pode refenciar o objeto Smarty de dentro da função.

Nota Tecnica

É possível ter partes do template fora do cache. se você tiver caching ativado, tags insert não estarão no cache. Ela será executada dinamicamente a cada vez que a página seja criada, mesmo com páginas em cache. Isto funciona bem para coisas como banners, pesquisa, previsões do tempo, resultados de pesquisa, áreas de opnião do usuário, etc.

if,elseif,else

Comandos {if} no Smarty tem muito da mesma flexibilidade do php, com algumas características à mais para o sistema de template. Todo *if* deve ter o seu */if*. *else* e *elseif* também são permitidos. Todos os condicionais do PHP são reconhecidos, tais como ||, or, &&, and, etc.

A seguir está uma lsita dos qualificadores, que devem estar separados dos elementos por espaço. Note que itens listado entre [conchetes] são opcionais. Os equivalentes em PHP são mostrados quando aplicáveis.

Qualificador	Alternativas	Exemplo de sintaxe	Significado	Equivalente no PHP
==	eq	$a eq $b	iguais	==
!=	ne, neq	$a neq $b	não iguais	!=
>	gt	$a gt $b	maior que	>
<	lt	$a lt $b	menor que	<
>=	gte, ge	$a ge $b	maior ou igual à	>=
<=	lte, le	$a le $b	menor ou igual à	<=
!	not	not $a	negação (unary)	!
%	mod	$a mod $b	módulo	%
is [not] div by		$a is not div by 4	divisível por	$a % $b == 0
is [not] even		$a is not even	[not] an even number (unary)	$a % 2 == 0
is [not] even by		$a is not even by $b	grouping level [not] even	($a / $b) % 2 == 0
is [not] odd		$a is not odd	[not] an odd number (unary)	$a % 2 != 0
is [not] odd by		$a is not odd by $b	[not] an odd grouping	($a / $b) % 2 != 0

Example 7.11. comandos if

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

{* um exemplo com "or" *}
{if $name eq "Fred" or $name eq "Wilma"}
 ...
{/if}

{* o mesmo que acima *}
{if $name == "Fred" || $name == "Wilma"}
 ...
{/if}

{* a seguinte sintaxe não irá funcionar, qualificadores de condição
   devem estar separados dos elementos em torno por espaços *}
{if $name=="Fred" || $name=="Wilma"}
 ...
{/if}


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

{* você pode também colocar funções php *}
{if count($var) gt 0}
 ...
{/if}

{* testa se o valor é par ou impar *}
{if $var is even}
 ...
{/if}
{if $var is odd}
 ...
{/if}
{if $var is not odd}
 ...
{/if}

{* verifica se a variável é divisível por 4 *}
{if $var is div by 4}
 ...
{/if}

{* test if var is even, grouped by two. i.e.,
0=even, 1=even, 2=odd, 3=odd, 4=even, 5=even, etc. *}
{if $var is even by 2}
 ...
{/if}

{* 0=even, 1=even, 2=even, 3=odd, 4=odd, 5=odd, etc. *}
{if $var is even by 3}
 ...
{/if}
```

ldelim,rdelim

ldelim e rdelim são usados para mostrar os delimitadores de templates literalmente, no nosso caso "{" ou "}". Ou você pode usar {literal}{/literal} para interpretar blocos de texto literalmente. Veja também {$smarty.ldelim} e {$smarty.rdelim}

Example 7.12. ldelim, rdelim

```
{* isto fará com que os delimitadores de template sejam tratados literalmente *}

{ldelim}funcname{rdelim} é como a função aparecer no Smarty!
```

O exemplo acima exibirá:

```
{funcname} é como a função aparecer no Smarty
```

literal

Tags literal permitem que um bloco de dados seja tratado literalmente, ou seja, não é interpretado pelo Smarty. Isto é tipicamente usado com blocos de código javascript ou folhas de estilo (stylesheet), que às vezes contém chaves que podem entrar em conflito com o delimitador de sintaxe. Qualquer coisa entre {literal}{/literal} não é interpretado, mas é mostrado. Se você precisa que tags de templates sejam embutidas em um bloco literal, use {ldelim}{rdelim}.

Example 7.13. Tags literal

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

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

 </script>
{/literal}
```

php

Tags php permitem que códigos php sejam embutidos diretamente nos templates. Eles não serão interpretados, não importando a definição de $php_handling. Esta opção é somente para usuários avançados e normalmente não é necessária.

Example 7.14. Tags php

```
{php}
  // incluindo um script php
  // diretamente no template.
  include("/caminho/para/condicoes_do_tempo.php");
{/php}
```

section,sectionelse

Nome do atributo	Tipo	Obrigatório	Padrão	Descrição
name	string	Sim	*n/d*	O nome da seção
loop	[$variable_name]	Sim	*n/d*	O nome da variável para determinar o número de interações
start	integer	Não	*0*	A posição do índice que a seção vai começar. Se o valor é negativo, a posição de inicio é calculada a partir do final da matriz. Por exemplo, se houverem sete valores na matriz e 'start' for -2, o índice inicial é 5. Valores inválidos (valores fora do tamanho da matriz) são automaticamente corrigidos para o valor válido mais próximo.
step	integer	Não	*1*	O valor do passo que será usado para percorrer a matriz. Por exemplo, step=2 irá percorrer os índices

Nome do atributo	Tipo	Obrigatório	Padrão	Descrição
				0,2,4, etc. Se step for negativo, ele irá caminhar pela matriz de trás para frente.
max	integer	Não	*1*	Define o número máximo de loops para a section.
show	boolean	Não	*true*	Determina quando mostrar ou não esta section

Os 'sections' de template são usados para percorrer os dados de uma matriz. Todas as tags *section* devem ser finalizadas com */section*. Os parâmetros obrigatórios são *name* e *loop*. O nome da 'section' pode ser o que você quiser, contendo letras, números e sublinhados. As 'sections' podem ser aninhadas, e os nomes das sections devem ser únicos. A variável 'loop' (normalmente uma matriz de valores) determina o número de vezes que a section será percorrida. Quando estiver exibindo uma variável dentro de uma section, o nome da section deve estar ao lado da variável dentro de conchetes []. *sectionelse* é executado quando não houver valores na variável 'loop'.

Example 7.15. section

```
{* este exemplo irá mostrar todos os valores da matriz $custid *}
{section name=consumidor loop=$custid}
 id: {$custid[consumidor]}<br>
{/section}
```

MOSTRA:

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

Example 7.16. loop de variável section

```
{* a variável 'loop' somente determina o número de vezes que irá percorrer a matri
   Você pode acessar qualquer variável do template dentro da section.
   Este exemplo assume que $custid, $nome e $endereco são todas
   matrizes contendo o mesmo número de valores *}

{section name=consumidor loop=$custid}
 id: {$custid[consumidor]}<br>
 nome: {$nome[consumidor]}<br>
 endereço: {$endereco[customer]}<br>
 <p>
{/section}
```

MOSTRA:

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

Example 7.17. Nomes de section

```
{* o nome da seção pode ser o que você qusier,
   e é usado para referenciar os dados contido na seção *}

{section name=meusdados loop=$custid}
 id: {$custid[meusdados]}<br>
 nome: {$nome[meusdados]}<br>
 endereço: {$endereco[meusdados]}<br>
 <p>
{/section}
```

Example 7.18. sections aninhadas

```
{* sections podem ser aninhadas até o nível que você quiser. Com sections aninhada
   você pode acessar complexas estruturas de dados, tais como matrizes multi-dimen
   Neste exemplo, $contact_type[customer] é uma matriz contendo os tipos de contat
   do consumidor atualmente selecionado. *}

{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}
```

MOSTRA:

```
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@mydomain.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@mydomain.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@mydomain.com<br>
<p>
```

Example 7.19. sections e matrizes associativas

```
{* Este é um exemplo de exibição de uma matriz associativa
   dentro de uma seção *}

{section name=consumidor loop=$contatos}
 nome: {$contatos[consumidor].nome}<br>
 telefone: {$contatos[consumidor].telefone}<br>
 celular: {$contatos[consumidor].celular}<br>
 e-mail: {$contatos[consumidor].email}<p>
{/section}
```

MOSTRA:

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

Example 7.20. sectionelse

```
{* sectionelse irá executar se não houverem mais valores em $custid *}

{section name=consumidor loop=$custid}
 id: {$custid[consumidor]}<br>
{sectionelse}
 não há valores em $custid.
{/section}
```

Sections também tem as suas próprias variáveis que manipulam as propriedades da section. Estas são indicadas assim: {$smarty.section.nomesection.nomevariavel}

Nota

A partir do Smarty 1.5.0, a sintaxe para as variáveis de propriedades da section mudou de {%nomesecao.nomevariavel%} para {$smarty.section.nomesection.nomevariavel}. A sintaxe antiga ainda é suportada, mas você verá referências somente à nova sintaxe no manual.

index

index é usado para mostrar o índice atual do loop, começando em zero (ou pelo atributo start caso tenha sido definido), e incrementado por um (ou pelo atributo step caso tenha sido definido).

Nota Técnica:

Se as propriedades 'start' e 'step' da section não foram modificadas, elas irão funcionar da mesma maneira que a propriedade 'interation' da section funcionam, exceto que ela começa do 0 ao invés de 1.

Example 7.21. propriedade index da section

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

MOSTRA:

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

index_prev

index_prev é usado para mostrar o índice anterior do loop. No primeiro loop, o valor dele é -1.

Example 7.22. propriedade index_prev da section

```
{section name=consumidor loop=$custid}
 {$smarty.section.consumidor.index} id: {$custid[consumidor]}<br>
 {* Para sua informação, $custid[consumidor.index] e $custid[consumidor] tem o mes
 {if $custid[consumidor.index_prev] ne $custid[consumidor.index]}
  O id do consumidor irá mudar<br>
 {/if}
{/section}
```

MOSTRA:

```
0 id: 1000<br>
    O id do consumidor irá mudar<br>
1 id: 1001<br>
    O id do consumidor irá mudar<br>
2 id: 1002<br>
    O id do consumidor irá mudar<br>
```

index_next

index_next é usado para mostrar o próximo indice do loop. No último loop, isto ainda é um mais o índice atual(respeitando a definição do atributo step, caso tenha sido definido.)

Example 7.23. propriedade index_next section

```
{section name=consumidor loop=$custid}
 {$smarty.section.consumidor.index} id: {$custid[consumidor]}<br>
 {* Para sua informação, $custid[consumidor.index] e $custid[consumidor] tem o mes
 {if $custid[consumidor.index_next] ne $custid[consumidor.index]}
     O id do consumidor irá mudar<br>
 {/if}
{/section}
```

MOSTRA:

```
0 id: 1000<br>
    O id do consumidor irá mudar<br>
1 id: 1001<br>
    O id do consumidor irá mudar<br>
2 id: 1002<br>
    O id do consumidor irá mudar<br>
```

iteration

iteration é usado para mostrar a interação atual do loop.

Nota:

'interation' não é afetado pelas propriedades start, step e max da section, diferentemente da propriedade index. Interation diferente de 'index' começa com 1 ao invés de 0. 'rownum' é um sinônimo de 'interation', eles exercem a mesma função.

Example 7.24. propriedade interation da section

```
{section name=consumidor loop=$custid start=5 step=2}
 interação atual do loop: {$smarty.section.consumidor.iteration}<br>
 {$smarty.section.consumidor.index} id: {$custid[consumidor]}<br>
 {* Para sua informação, $custid[consumidor.index] e $custid[consumidor] tem o mes
 {if $custid[consumidor.index_next] ne $custid[consumidor.index]}
     O id do consumidor irá mudar<br>
 {/if}
{/section}
```

MOSTRA:

```
interação atual do loop: 1
5 id: 1000<br>
    O id do consumidor irá mudar<br>
interação atual do loop: 2
7 id: 1001<br>
    O id do consumidor irá mudar<br>
interação atual do loop: 3
9 id: 1002<br>
    O id do consumidor irá mudar<br>
```

first

first é definido como true se a interação atual da section é a primeira.

Example 7.25. propriedade first da section

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

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

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

MOSTRA:

```
<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 é definido como true se a interação atual da section é a última.

Example 7.26. propriedade last da section

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

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

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

MOSTRA:

```
<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 é usado para mostrar a interação atual do loop, começando em um. É um sinônimo de iteration, eles exercem a mesma função.

Example 7.27. propriedade rownum da section

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

MOSTRA:

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

loop

loop é usado para exibir o número do último índice que a section percorreu. Ele pode ser usado dentro ou após o término da section.

Example 7.28. propridade index da section

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

 Foram mostrados {$smarty.section.customer.loop} consumidores acima.
```

MOSTRA:

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

Foram mostrados 3 consumidores acima.
```

show

show é usado como um parâmetro da section. *show* é um valor booleano, verdadeiro ou falso. Caso seja falso, a section não será mostrada. Se existir uma sectionelse presente, ela será exibida.

Example 7.29. atributo show da section

```
{* $mostrar_info_consumidor talvez tenha que ser enviada pela
   aplicação PHP, para decidir quando mostrar ou não mostrar esta section *}

{section name=consumidor loop=$custid show=$mostrar_info_consumidor}
 {$smarty.section.consumidor.rownum} id: {$custid[consumidor]}<br>
{/section}

{if $smarty.section.consumidor.show}
 a section foi mostrada.
{else}
 a section não foi mostrada.
{/if}
```

MOSTRA:

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

a section foi mostrada.
```

total

total é usado para exibir o número de interações que esta section irá percorrer. Ela pode ser usada dentro ou após a section.

Example 7.30. propriedade total da section

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

 Foram mostrados {$smarty.section.customer.loop} consumidores acima.
```

MOSTRA:

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

Foram mostrados 3 consumidores acima.
```

strip

Muitas vezes web designers tem problemas com espaços em branco e caracteres especiais (carriage returns) afetam a exibição do HTML ("características" do navegador), assim você é obrigado à colocar todas as suas tags juntas para obter os resultados esperados. Isso geralmente acaba tornando o template ilegível ou não manipulável.

Tudo entre as tags {strip}{/strip} no Smarty tem seus espaços extras ou caracteres especiais (carriage returns) removidos no início e fim das linhas antes de elas serem exibidas. Deste modo você pode manter seu template legível, e não se preocupar com espaços extras causando problemas.

Nota Técnica

{strip}{/strip} não afeta o conteúdo das variáveis de template. Veja modificador strip.

Example 7.31. strip tags

```
{* o código abaixo será convertido em uma linha na hora da exibição *}
{strip}
<table border=0>
 <tr>
  <td>
   <A HREF="{$url}">
   <font color="red">Isto é um teste</font>
   </A>
  </td>
 </tr>
</table>
{/strip}
```

MOSTRARÁ:

```
<table border=0><tr><td><A HREF="http://meu.dominio.com"><font color="red">Isto é
```

Observe que no exemplo acima, todas as linhas começam e terminam com tags HTML. Esteja ciente para que todas as linhas fiquem juntas. Se você tiver texto simples no início ou final de uma linha, ele será juntado na hora da conversão e pode causar resultados não desejados.

Chapter 8. Funções Personalizadas

O Smarty contém várias funções personalizadas que você pode usar em seus templates.

assign

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
var	string	Sim	*n/a*	O nome da variável que está sendo definida
value	string	Yes	*n/a*	O valor que está sendo definido

assign é usado para definir o valor de uma variável de template durante a execução do template.

Example 8.1. assign

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

O valor de $nome é {$nome}.

MOSTRA:

O valor de $nome é Bob.
```

counter

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
name	string	Não	*default*	O nome do contador
start	number	Não	*1*	O número no qual a contagem se inicia
skip	number	Não	*1*	O intervalo entre as contagens
direction	string	Não	*up*	A direção para contar (up/down)
print	boolean	Não	*true*	Quando mostrar ou não o valor
assign	string	Não	*n/a*	A variável de template que vai receber a saída

counter é usada para mostrar uma contagem. counter irá se lembrar de count em cada interação. Você pode ajustar o número, o intervalo e a direção da contagem, assim como detrminar quando mostrar ou não a

contagem. Você pode ter vários contadores ao mesmo tempo, dando um nome único para cada um. Se você não der um nome, o nome 'default' será usado.

Se você indicar o atributo especial "assign", a saída da função counter será passada para essa variável de template ao invés de ser mostrada no template.

Example 8.2. counter

```
{* inicia a contagem *}
{counter start=0 skip=2 print=false}

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

MOSTRA:

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

cycle

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
name	string	Não	*default*	O nome do ciclo
values	mixed	Sim	*n/d*	Os valores do ciclo, ou uma lista delimitada por vírgula (veja o atributo delimiter), ou uma matriz de valores.
print	boolean	Não	*true*	Quando mostrar ou não o valor
advance	boolean	Não	*true*	Quando avançar ou não para o próximo valor
delimiter	string	Não	,	O delimitador para usar no atributo 'values'.
assign	string	Não	*n/d*	A variável de template que receberá a saída

Cycle é usado para fazer um clico através de um conjunto de valores. Isto torna fácil alternar entre duas ou mais cores em uma tabela, ou entre uma matriz de valores.

Você pode usar o cycle em mais de um conjunto de valores no seu template. Dê a cada conjunto de valores um nome único.

Você pode fazer com que o valor atual não seja mostrado definindo o atributo print para false. Isto é útil para pular um valor.

O atributo advance é usado para repetir um valor. Quando definido para false, a próxima chamada para cycle irá mostrar o mesmo valor.

Se você indicar o atributo especial "assign", a saída da função cycle será passada para uma variável de template ao invés de ser mostrado diretamente no template.

Example 8.3. cycle

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

MOSTRA:

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

debug

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
output	string	Não	*html*	Tipo de saída, html ou javascript

{debug} mostra o console de debug na página. Ele funciona independente da definição de debug. Já que ele é executado em tempo de execução, ele é capaz apenas de mostrar as variáveis definidas, e não os templates que estão em uso. Mas você pode ver todas as variáveis disponíveis no escopo do template.

eval

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
var	mixed	Sim	*n/a*	Variável (ou string) para avaliar

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
assign	string	Não	*n/a*	A variável de template que receberá a saída

eval é usado para avaliar uma variável como template. Isto pode ser usado para coisas como embutir tags/variáveis de template dentro de variáveis ou tags/variáveis dentro de variáveis em um arquivo de configuração.

Se você indicar o atributo especial "assign", a saída da função eval irá para esta variável de template ao invés de aparecer no template.

Nota Técnica

Variáveis avaliadas são tratadas igual a templates. Elas seguem o mesmo funcionamento para escapar e para segurança como se fossem templates.

Nota Técnica

Variáveis avaliadas são compiladas a cada invocação, as versões compiladas não são salvas. Entretando, se você tiver o cache ativado, a saída vai ficar no cache junto com o resto do template.

Example 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}

MOSTRA:

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

fetch

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
file	string	Sim	*n/a*	O arquivo, site http ou ftp para obter
assign	string	Não	*n/a*	A variável de template que vai receber a saída

fetch é usado para obter arquivos do sistema de arquivos local, http ou ftp, e mostrar o seu conteúdo. Se o nome do arquivo começar com "http://", a página do web site será obtida e mostrada. Se o nome do arquivo começar com "ftp://", o arquivo será obtido do servidor ftp e mostrado. Para arquivos locais, o caminho completo do sistema de arquivos deve ser dado, ou um caminho relativo ao script php executado.

Se você indicar o atributo especial "assign", a saída da função fetch será passada para uma variável de template ao invés de ser mostrado no template. (novo no Smarty 1.5.0)

Nota Técnica

fetch não suporta redirecionamento http, tenha certeza de incluir a barra no final aonde necessário.

Nota Técnica

Se a segurança do template esta ativada e você estiver obtendo um arquivo do sistema de arquivos locais, fetch irá funcionar apenas em arquivos de um dos diretórios definidos como seguros. ($secure_dir)

Example 8.5. fetch

```
{* inclui algum javascript no seu template *}
{fetch file="/export/httpd/www.domain.com/docs/navbar.js"}

{* embute algum texto sobre o tempo de outro web site *}
{fetch file="http://www.myweather.com/68502/"}

{* obtém um arquivo de notícias via ftp *}
{fetch file="ftp://user:password@ftp.domain.com/path/to/currentheadlines.txt"}

{* coloca o conteúdo obtido para uma varável de template *}
{fetch file="http://www.myweather.com/68502/" assign="weather"}
{if $weather ne ""}
 <b>{$weather}</b>
{/if}
```

html_checkboxes

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
name	string	Não	*checkbox*	O nome da lista checkbox

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
values	array	Sim, a menos que estaja usando o atributo options	*n/a*	Uma matriz de valores para os botões checkbox
output	array	Sim, a menos que estaja usando o atributo options	*n/a*	uma matriz de saída para os botões checkbox
selected	string/array	Não	*empty*	O(s) elemento(s) checkbox marcado(s)
options	matriz	Sim, a menos que esteja usando values e output	*n/a*	Uma matriz associativa de valores e saída
separator	string	Não	*empty*	string de texto para separar cada checkbox
labels	boolean	Não	*true*	Adicionar tags <label> para na saída

html_checkboxes é uma função personalizada que cria um grupo de checkbox com os dados fornecidos. Ela cuida de qual(is) item(s) estão selecionado(s) por padrão. Os atributos obrigatórios são values e output, a menos que você use options. Toda a saída é compatível com XHTML.

Todos os parâmetro que não estejam na lista acima são mostrados como pares nome/valor dentro de cada tag <input> criada.

Example 8.6. html_checkboxes

```
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','Char
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');


index.tpl:

{html_checkboxes values=$cust_ids checked=$customer_id output=$cust_names separato


index.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');

index.tpl:

{html_checkboxes name="id" options=$cust_checkboxes checked=$customer_id separator


MOSTRA: (ambos os exemplos)

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

html_image

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
file	string	Sim	*n/a*	nome/caminho para a imagem
border	string	Não	*0*	tamanho da borda de contorno da imagem

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
height	string	Não	*altura atual da imagem*	altura com a qual a imagem deve ser mostrada
width	string	Não	*largura atual da imagem*	largura com a qual a imagem deve ser mostrada
basedir	string	Não	*doc root do servidor*	diretório de base a caminhos relativos
alt	string	Não	*""*	descrição alternativa da imagem
href	string	Não	*n/a*	valor href para aonde a imagem será linkada

html_image é uma função customizada que gera uma tag HTML para uma imagem. A altura e a largura são automaticamente calculadas a partir do arquivo de imagem se nenhum valor é fornecido.

basedir é o diretório base do qual caminhos relativos de imagens estão baseados. Se não fornecido, o document root do servidor (variável de ambiente DOCUMENT_ROOT) é usada como o diretório base. Se a segurança está habilitada, o caminho para a imagem deve estar dentro de um diretório seguro.

`href` é o valor href para onde a imagem será linkada. Se um link é fornecido, uma tag <a href="LINKVALUE"><a> é posta em volta da tag da imagem.

Nota Técnica

html_image requer um acesso ao disco para ler a imagem e calcular a altura e a largura. Se você não usa caching de template, normalmente é melhor evitar html_image e deixar as tags de imagem estáticas para performance otimizada.

Example 8.7. html_image

```
index.php:

require('Smarty.class.php');
$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"}

MOSTRA: (possível)

<img src="pumpkin.jpg" alt="" border="0" width="44" height="68" />
<img src="/path/from/docroot/pumpkin.jpg" alt="" border="0" width="44" height="68"
<img src="../path/relative/to/currdir/pumpkin.jpg" alt="" border="0" width="44" he
```

html_options

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
values	array	Sim, a menos que usando atributos de options	*n/a*	uma matriz de valores para o menu dropdown
output	array	Sim, a menos que usando atributos de options	*n/a*	uma matriz de saída para o menu dropdown
selected	string/array	Não	*empty*	o elemento do options selecionado
options	associative array	Sim, a menos que usando values e output	*n/a*	uma matriz associativa de output e output
name	string	Não	*empty*	nome do grupo selecionado

html_options é uma função personalizada que cria um grupo html option com os dados fornecidos. Ela está atenta de quais itens estão selecionados por padrão. Atributos obrigatórios são 'values' e 'output', a menos que você use options no lugar.

Se um valor dado é um array, ele será tratado como um OPTGROUP html, e mostrará os grupos. Recursividade é suportada pelo OPTGROUP. Todas as saídas são compatíveis com XHTML.

Se o atributo opcional *name* é dado, as tags <select name="groupname"></select> irão incluir a lista de opções dentro dela. Caso contrário apenas a lista de opções é gerada.

Todos os parâmetros que não estão na lista acima são exibidos como nome/valor dentro de <select>-tag. Eles são ignorados se o opcional *name* não é fornecido.

Example 8.8. html_options

```
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','Carlie 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>


index.php:

require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_options', array(
   1001 => 'Taniel Franklin',
   1002 => 'Fernando Correa',
   1003 => 'Marcelo Pereira',
   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>


OUTPUT: (both examples)

<select name=customer_id>
 <option value="1000">Taniel Franklin</option>
 <option value="1001" selected="selected">Fernando Correa</option>
 <option value="1002">Marcelo Pereira</option>
 <option value="1003">Charlie Brown</option>
</select>
```

html_radios

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
name	string	Não	*radio*	nome da radio list
values	array	Sim, a menos que utilizando atributo de options	*n/a*	uma matriz de valores para radio buttons
output	array	Sim, a menos que utilizando atributo de options	*n/a*	uma matriz de saída pra radio buttons
checked	string	Não	*empty*	O elemento do radio marcado
options	associative array	Sim, a menos que utilizando values e output	*n/a*	uma matriz associativa de values e output
separator	string	Não	*empty*	string de texto para separar cada item de radio

html_radios é uma função personalizada que cria grupo de botões de radio html com os dados fornecidos. Ele está atento para qual item está selecionado por padrão. Atributos obrigatórios são 'values' e 'output', a menos que você use 'options' no lugar disso. Toda saída é compatível com XHTML.

Todos os parâmetros que não estão na lista acima são impressos como nome/valor de dentro de cada tag <input> criada.

Example 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','Carlie Brown'));
$smarty->assign('customer_id', 1001);
$smarty->display('index.tpl');
```

index.tpl:

```
{html_radios values=$cust_ids checked=$customer_id output=$cust_names separator="<
```

index.php:

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

index.tpl:

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

OUTPUT: (both examples)

```
<input type="radio" name="id[]" value="1000">Taniel Fraklin<br />
<input type="radio" name="id[]" value="1001" checked="checked"><br />
<input type="radio" name="id[]" value="1002">Marcelo Pereira<br />
<input type="radio" name="id[]" value="1003">Charlie Brown<br />
```

html_select_date

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
prefix	string	Não	Date_	Com o que prefixar o nome da variável
time	timestamp/ YYYY-MM-DD	Não	tempo atual no formato timestamp	qual date/time usar

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
			do unix ou YYYY-MM-DD	
start_year	string	Não	ano atual	o primeiro ano no menu dropdown, ou o número do ano, ou relativo ao ano atual (+/- N)
end_year	string	Não	da mesma forma que start_year	o último ano no menu dropdown, ou o número do ano, ou relativo ao ano atual (+/- N)
display_days	boolean	Não	true	se mostra os dias ou não
display_months	boolean	No	true	se mostra os meses ou não
display_years	boolean	Não	true	se mostra os anos ou não
month_format	string	Não	%B	qual o formato do mês (strftime)
day_format	string	Não	%02d	a saída do dia seria em qual formato (sprintf)
day_value_format	string	No	%d	o valor do dia seria em qual formato (sprintf)
year_as_text	booleano	Não	false	se mostra ou não o ano como texto
reverse_years	booleano	Não	false	mostra os anos na ordem reversa
field_array	string	Não	null	se um nome é dado, as caixas de seleção serão exibidos assim que os resultados forem devolvidos ao PHP na forma de name[Day], name[Year], name[Month].
day_size	string	No	null	adiciona o atributo de tamanho para a tag select se for dada
month_size	string	Não	null	adiciona o atributo de tamanho para a

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
				tag de select se for dada
year_size	string	Não	null	adiciona o atributo de tamanho para a tag de select se for dada
all_extra	string	No	null	adiciona atributos extras para todas as tags select/input se forem dadas
day_extra	string	Não	null	adiciona atributos extras para todas as tags select/input se forem dadas
month_extra	string	Não	null	adiciona atributos extras para todas as tags select/input se forem dadas
year_extra	string	Não	null	adiciona atributos extras para todas as tags select/input se forem dadas
field_order	string	Não	MDY	a ordem para se mostrar os campos
field_separator	string	Não	\n	string exibida entre os diferentes campos
month_value_format	string	Não	%m	formato strftime dos valores do mês, o padrão é %m para número de mês.
year_empty	string	No	null	Se for fornecido então o primeiro eleemento do select-box 'anos' terá este nome e o valor "". Isto é útil para fazer o select-box ler "Por favor selecione um ano" por exemplo. Note que você pode usar valores como "-MM-DD" como atributos de tempo para indicar um ano não selecionado.

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
month_empty	string	No	null	Caso fornecido então o primeiro elemento do select-box 'meses' terá este nome e o valor "". Note que você pode suar valores como "YYYY--DD" como atributos de tempo para indicar meses não selecionados.
day_empty	string	No	null	Caso fornecido então o primeiro elemento do select-box 'dias' terá este nome e o valor "". Note que você pode usar valores como "YYYY-MM-" como atributos de tempo para indicar dias não selecionados.

html_select_date é uma função personalizada que cria menus dropdowns de data para você. Ele pode mostrar qualquer um/ou todos os anos, meses e dias.

```
{html_select_date}
```

Example 8.10. html_select_date

MOSTRA

```
<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>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>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>2001</option>
</select>
```

Example 8.11. html_select_date

```
{* ano de começo e fim pode ser relativo ao ano atual *}
{html_select_date prefix="StartDate" time=$time start_year="-5" end_year="+1" disp

MOSTRA: (o ano atual é 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>December</option>
</select>
<select name="StartDateYear">
<option value="1999">1995</option>
<option value="1999">1996</option>
<option value="1999">1997</option>
<option value="1999">1998</option>
<option value="1999">1999</option>
<option value="2000" selected>2000</option>
<option value="2001">2001</option>
</select>
```

html_select_time

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
prefix	string	Não	Time_	com o que prefixar o nome da variável
time	timestamp	Não	tempo atual	qual date/time para usar
display_hours	booleano	Não	true	Exibir ou não as horas
display_minutes	booleano	Não	true	Exibir ou não os minutos
display_seconds	booleano	Não	true	Exibir ou não os segundos
display_meridian	booleano	Não	true	Exibir ou não no formato (am/pm)

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
use_24_hours	booleano	Não	true	Usar ou não relógio de 24 horas
minute_interval	inteiro	Não	1	intervalo dos números dos minutos do menu dropdown
second_interval	integer	Não	1	intervalo dos números dos segundos do menu dropdown
field_array	string	Não	n/a	exibe valores para o array deste nome
all_extra	string	Não	null	adiciona atributos extras para tags select/input se fornecidas
hour_extra	string	Não	null	adiciona atributos extras para tags select/input se fornecidas
minute_extra	string	Não	null	adiciona atributos extras para tags select/input tags se fornecidas
second_extra	string	No	null	adiciona atributos extras para tags select/input se fornecidas
meridian_extra	string	Não	null	adiciona atributos extras para tags select/input se fornecidas

html_select_time é uma função personalizada que cria menus dropdowns de hora para você. Ela pode mostrar alguns valores, ou tudo de hora, minuto, segundo e ainda no formato am/pm.

Example 8.12. html_select_time

```
<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>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>AM</option>
<option value="pm">PM</option>
</select>
```

html_table

Nome do atributo	Tipo	Obrigatório	Padrão	Descrição
loop	array	Sim	*n/d*	array de dados para ser feito o loop
cols	inteiro	Não	*3*	número de colunas na tabela
table_attr	string	Não	*border="1"*	atributos para a tag table
tr_attr	string	Não	*empty*	atributos para a tag tr (arrays estão em ciclo)
td_attr	string	Não	*empty*	atributos para a tag (arrays estão em ciclo)
trailpad	string	Não	* *	values to pad the trailing cells on last row with (se algum)
hdir	string	Não	*right*	direçao de uma linha para ser representada. Possíveis valores: *left*/*right*
vdir	string	Não	*down*	direção das colunas para serem representadas. Possíveis valores: *up*/*down*

html_table é uma função personalizada que transforma um array de dados em uma tabela HTML. O atributo *cols* determina a quantidade de colunas que a tabela terá. Os valores *table_attr*, *tr_attr* e *td_attr* determinam os atributos dados para a tabela, tags tr e td. Se *tr_attr* ou *td_attr* são arrays, eles entrarão em ciclo. *trailpad* é o valor colocado dentro do trailing cells na última linha da tabela se há alguma presente.

Example 8.13. html_table

index.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');
```

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}
```

MOSTRA:

```
<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>&amp;nbsp;</td><td>&amp;nbsp;</td><td>&amp;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>&amp;nbsp;</td><td>&amp;nbsp;</td><td>&amp;nbs
</table>
```

math

Nome do atributo	Tipo	Obrigatório	Padrão	Descrição
equation	string	Sim	*n/a*	a equação à ser executar
format	string	Não	*n/a*	o formato do resultado (sprintf)
var	numérico	Sim	*n/a*	valor da variável da equação
assign	string	Não	*n/a*	variável de template cuja saída será atribuida
[var ...]	numérica	Sim	*n/a*	valor da variável da equação

math permite o desenhista de template fazer equações matemáticas no template. Qualquer variável de template numérica pode ser usada nas equações, e o resultado é exibido no lugar da tag. As variáveis usadas na equação são passadas como parâmetros, que podem ser variáveis de template ou valores estáticos. +, -, /, *, abs, ceil, cos, exp, floor, log, log10, max, min, pi, pow, rand, round, sin, sqrt, srans and tan são todos os operadores válidos. Verifique a documentação do PHP para mais informações acerca destas funções matemáticas.

Se você fornece o atributo especial "assign", a saída da função matemática será atribuído para esta variável de template ao invés de ser exibida para o template.

Nota Técnica

math é uma função de performance cara devido ao uso da função do php eval(). Fazendo a matemática no PHP é muito mais eficiente, então sempre é possível fazer os cálculos matemáticos no PHP e lançar os resultados para o template. Definitivamente evite chamadas de funções de matemáticas repetitivamente, como dentro de loops de section.

Example 8.14. math

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

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

MOSTRA:

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#}

MOSTRA:

100


{* Você pode usar parenteses *}

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

MOSTRA:

6


{* você pode fornecer um parâmetro de formato em sprintf *}

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

MOSTRA:

9.44
```

mailto

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
address	string	Sim	*n/d*	O endereço de email
text	string	Não	*n/d*	O texto à ser exibido, o padrão é o endereço de email

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
encode	string	Não	*none*	Como codificar o e-mail. Pode ser none, hex ou javascript.
cc	string	Não	*n/d*	Endereço de e-mail para mandar uma cópia carbono(cc). Separe os endereços por vírgula.
bcc	string	Não	*n/d*	Endereço de e-mail para mandar uma cópia carbono cega(bcc). Separe os endereços por vírgula.
subject	string	Não	*n/d*	Assunto do e-mail.
newsgroups	string	Não	*n/d*	newsgroup para postar. Separe os endereços por vírgula.
followupto	string	Não	*n/d*	Endereço para acompanhar. Separe os endereços por vírgula.
extra	string	Não	*n/d*	Qualquer outra informação que você queira passar para o link, como classes de planilhas de estilo

mailto automatiza o processo de criação de links de e-mail e opcionalmente codifica eles. Codificar o e-mail torna mais difícil para web spiders pegarem endereços no seu site.

Nota Técnica

javascript é provavelmente o meio de codificação mais utilizado, entretanto você pode usar codificação hexadecimal também.

Example 8.15. mailto

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

MOSTRA:

<a href="mailto:me@domain.com" >me@domain.com</a>
<a href="mailto:me@domain.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@%64%6f%6d%61%69%6e.%63%6f%6d" >&amp;#x6d;&amp;#x65;&amp;#x4
#x6f;&amp;#x6d;&amp;#x61;&amp;#x69;&amp;#x6e;&amp;#x2e;&amp;#x63;&amp;#x6f;&amp;#x
<a href="mailto:me@domain.com?subject=Hello%20to%20you%21" >me@domain.com</a>
<a href="mailto:me@domain.com?cc=you@domain.com%2Cthey@domain.com" >me@domain.com<
<a href="mailto:me@domain.com" class="email">me@domain.com</a>
```

popup_init

popup é uma integração com overLib, uma biblioteca usada para janelas popup. Esta é usada para informações sensíveis ao contexto, como janelas de ajuda ou dicas. popup_init deve ser usada uma vez ao topo de cada página que você planeje usar a função popup. overLib foi escrita por Erik Bosrup, e a página esta localizada em http://www.bosrup.com/web/overlib/.

A partir da versão 2.1.2 do Smarty, overLib NÃO vem com a distribuição. Baixe o overLib, coloque o arquivo overlib.js dentro da sua arvore de documentos e indique o caminho relativo para o parâmetro "src" de popup_init.

Example 8.16. popup_init

```
{* popup_init deve ser utilizada uma vez no topo da página *}
{popup_init src="/javascripts/overlib.js"}
```

popup

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
text	string	Sim	*n/d*	O text/html para mostrar na janela popup

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
trigger	string	Não	*onMouseOver*	O que é usado para fazer a janela aparecer. Pode ser onMouseOver ou onClick
sticky	boolean	Não	*false*	Faz a janela colar até que seja fechada
caption	string	Não	*n/d*	Define o texto para o título
fgcolor	string	Não	*n/d*	A cor usada dentro da caixa popup
bgcolor	string	Não	*n/d*	A cor da borda da caixa popup
textcolor	string	Não	*n/d*	Define a cor do texto dentro da caixa popup
capcolor	string	Não	*n/d*	Define a cor do título da caixa
closecolor	string	Não	*n/d*	Define a cor do texto para fechar
textfont	string	Não	*n/d*	Define a cor do texto para ser usado no texto principal
captionfont	string	Não	*n/d*	Define a fonte para ser usada no Título
closefont	string	Não	*n/d*	Define a fonte para o texto "Close"
textsize	string	Não	*n/d*	Define a fonte do texto principa
captionsize	string	Não	*n/d*	Define o tamanho da fonte do título
closesize	string	Não	*n/d*	Define o tamanho da fonte do texto "Close"
width	integer	Não	*n/d*	Define a largura da caixa
height	integer	Não	*n/d*	Define a altura da caixa
left	boolean	Não	*false*	Faz os popups irem para a esquerda do mouse
right	boolean	Não	*false*	Faz os popups ir para a diresita do mouse

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
center	boolean	Não	*false*	Faz os popups ir para o centro do mouse
above	boolean	Não	*false*	Faz os popups irem para acima do mouse. NOTA: somente possível se height foi definido
below	boolean	Não	*false*	Faz os popups irem abaixo do mouse
border	integer	Não	*n/d*	Torna as bordas dos popups grossas ou finas
offsetx	integer	Não	*n/d*	A que distancia do mouse o popup irá aparecer, horizontalmente
offsety	integer	Não	*n/d*	A que distancia do mouse o popup irá aparecer, verticalmente
fgbackground	url para imagem	Não	*n/d*	Define uma imagem para usar ao invés de uma cor dentro do popup.
bgbackground	url to image	Não	*n/d*	define uma imagem para ser usada como borda ao invés de uma cor para o popup. Nota: você deve definir bgcolor como "" ou a cor irá aparecer também. NOTA: quando tiver um link "Close", o Netscape irá redesenhar as células da tabela, fazendo as coisas aparecerem incorretamente
closetext	string	Não	*n/d*	Define o texto "Close" para qualquer outra coisa
noclose	boolean	Não	*n/d*	Não mostra o texto "Close" em coladas com um título

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
status	string	Não	*n/d*	Define o texto na barra de status do browser
autostatus	boolean	Não	*n/d*	Define o texto da barra de status para o texto do popup. NOTA: sobrescreve a definição de status
autostatuscap	string	Não	*n/d*	define o texto da barra de status como o texto do título NOTA: sobrescreve o status e autostatus
inarray	integer	Não	*n/d*	Indica ao overLib para ler o texto deste índice na matriz ol_text array, localizada em overlib.js. Este parâmetro pode ser usado ao invés do texto
caparray	integer	Não	*n/d*	diz para overLib ler o título a partir deste índice na matriz ol_caps
capicon	url	Não	*n/d*	Mostra a imagem antes do título
snapx	integer	Não	*n/d*	snaps the popup to an even position in a horizontal grid
snapy	integer	Não	*n/d*	snaps the popup to an even position in a vertical grid
fixx	integer	No	*n/d*	locks the popups horizontal position Note: overrides all other horizontal placement
fixy	integer	No	*n/d*	locks the popups vertical position Note: overrides all other vertical placement
background	url	Não	*n/d*	Define uma imagem para ser

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
				usada como fundo ao invés da tabela
padx	integer,integer	Não	*n/d*	Prenche a imagem de fundo com espaços em branco horizontal para colocação do texto. Nota: este é um comando de dois parâmetros
pady	integer,integer	Não	*n/d*	Prenche a imagem de fundo com espaços em branco vertical para colocação do texto. Nota: este é um comando de dois parâmetros
fullhtml	boolean	Não	*n/d*	Permite a você controlar o html sobre a figura de fundo completamente. O código HTML é esperado no atributo "text"
frame	string	Não	*n/d*	Controla popups em frames diferentes. Veja a página da overlib para maiores informações sobre esta função
timeout	string	Não	*n/d*	Utiliza uma função e pega o valor de retorno como texto que deva ser mostrado na janela popup
delay	integer	Não	*n/d*	Faz com que o popup funcione como um tooltip. Irá aparecer apenas após um certo atraso em milésimos de segundo
hauto	boolean	Não	*n/d*	Determina automaticamente se

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
				o popup deve aparecer a esquerda ou direita do mouse.
vauto	boolean	Não	*n/d*	Determina automaticamente se o popup deve aparecer abaixo ou acima do mouse.

popup é usado para criar janelas popup com javascript.

Example 8.17. popup

```
{* popup_init deve ser utilizada uma vez no topo da página *}
{popup_init src="/javascripts/overlib.js"}

{* cria um link com uma janela popup que aparece quando se passa o mouse sobre ele
<A href="mypage.html" {popup text="This link takes you to my page!"}>mypage</A>

{* você pode usar html, links, etc no texto do popup *}
<A href="mypage.html" {popup sticky=true caption="mypage contents"
text="<UL><LI>links<LI>pages<LI>images</UL>" snapx=10 snapy=10}>mypage</A>
```

textformat

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
style	string	Não	*n/d*	estilo pré-definido
indent	number	Não	*0*	O número de caracteres para endentar cada linha.
indent_first	number	Não	*0*	O número de caracteres para endentar a primeira linha
indent_char	string	Não	*(single space)*	O caractere (ou string de caracteres) para indenta
wrap	number	Não	*80*	Quantidade de caracteres antes de quebrar cada linha
wrap_char	string	Não	*\n*	O caractere (ou string de caracteres) para usar para quebrar cada linha
wrap_cut	boolean	Não	*false*	Se true, wrap irá quebrar a linha no caractere exato em

Nome do Atributo	Tipo	Obrigatório	Padrão	Descrição
				vez de quebrar ao final da palavra
assign	string	No	*n/d*	A variável de template que irá receber a saída

textformat é uma função de bloco usada para formatar texto. Basicamente ela remove espaços e caracteres especiais, e formata os parágrafos quebrando o texto ao final de palavras e identando linhas.

Você pode definir os parâmetros explicitamente, ou usar um estilo pré-definido. Atualmente o único estilo disponível é "email".

```
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}
```

Example 8.18. textformat

```
MOSTRA:

        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}

MOSTRA:

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.
```

Chapter 9. Arquivos de Configuração

Arquivos de configuração são úteis para designers que gerenciam variáveis globais para os templates à partir de um arquivo. Um exemplo são as cores do template. Normalmente se você quisesse mudar o tema de cores de uma aplicação, você teria que abrir cada arquivo de template e alterar as cores. Com arquivos de configurações, as cores podem ser armazenadas em um lugar, e apenas um arquivo precisaria ser alterado.

Example 9.1. Exemplo de sintaxe de um arquivo de configuração

```
# variáveis globais
tituloPagina = "Menu Principal"
corfundoPagina = #000000
corfundoTabela = #000000
corlinhaTabela = #00ff00

[Consumidor]
tituloPagina = "Informações do Consumidor"

[Login]
tituloPagina = "Login"
focus = "nomeusuario"
Intro = """Este é um valor que ultrapassa uma
           linha. Você deve colocá-lo
           dentre três aspas."""

# seção invisível
[.BancoDeDados]
host=meu.dominio.com
bd=LIVRODEVISITAS
usuario=usuario-php
senha=foobar
```

Valores de variáveis de arquivos de configuração pode estar entre aspas, mas não é necessário. Você pode usar tanto aspas simples como duplas. Se você tiver um valor que ocupe mais de uma linha, coloque-o dentre três aspas (""""). Você pode colocar comentários em arquivos de configuração com qualquer sintaxe que não é válida para um arquivo de configuração. Nós recomendamos usar um # (cancela) no início de cada linha que contém o comentário.

Este arquivo de configuração tem duas seções. Nomes de seções devem estar entre conchetes []. Nomes de seção podem ser string arbritraria que não contenham os símbolos `[` ou `]`. As quatro variáveis no topo são variáveis globais, ou variáveis que não pertencem à uma seção. Estas variáveis sempre são carregadas do arquivo de configuração. Se uma seção em particular é carregada, então as variáveis globais e as variáveis desta seção também são carregadas. Se uma variável de seção e global já existirem, a variável de seção será utilizada. Se você tiver duas variáveis na mesma seção com o mesmo nome, a última será utilizada.

Arquivos de configuração são carregados no template usando a função embutida **config_load**.

Você pode esconder as variáveis ou uma seção inteira colocando um ponto antes do nome da seção ou variável. Isso é útil em casos no qual sua aplicação lê arquivos de configuração e obtém dados sensíveis que não são necessários para o sistema de templates. Se a edição de seus templates é terceirizada, você terá certeza que eles não irão ler os dados sensíveis do arquivo de configuração que é carregado no template.

Chapter 10. Debugging Console

Há um console para debug incluso no Smarty. O console informa à você todos os templates incluídos, variáveis definidas e variáveis de arquivos de configuração do template atual. Um template chamado "debug.tpl" está incluso com a distribuição do Smarty o qual controla a formtação do console. Defina a variável $debugging para true no Smarty, e se necessário defina $debug_tpl com o caminho do diretório onde está o arquivo debug.tpl (o diretório padrão é o da constante SMARTY_DIR). Quando você carrega uma página, um javascript abre uma janela pop-up e fornece à você o nome de todos os templates incluídos e variáveis definidas ara a página atual. Para ver as variáveis disponíveis para um template específico, veja a função {debug}. Para desabilitar o console de debug, defina a variável $debugging para false. Você também pode ativar temporariamente o console de debug colocando na URL, caso você tenha ativado esta opção na variável $debugging_ctrl.

Nota Técnica

O console de debug não funciona quando você usa a API fetch(), somente quando você estiver usando display(). Isto é um conjunto de comandos em javascript adicionados ao final do template gerado. Se você não gosta de javascript, você pode editar o template debug.tpl para exibir saída no formato que você quiser. Dados do debug não são armazenados em cache e os dados do debug.tpl não são inclusos no console de debug.

Note

O tempo de carregamento de cada template e arquivo de configuração são exibidos em segundos, ou então frações de segundo.

Part III. Smarty para Programadores

Table of Contents

Chapter 11. Constantes

SMARTY_DIR

Isso deve ser o caminho completo do path para a localização dos arquivos de classe da Smarty. Se isso não for definido, então a Smarty irá tentar determinar o valor apropriado automaticamente. Se definido, o path deve finalizar com uma barra.

Example 11.1. SMARTY_DIR

```
// set path to Smarty directory
define("SMARTY_DIR","/usr/local/lib/php/Smarty/");

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

Chapter 12. Variáveis

$template_dir

Este é o nome padrão do diretório de template. Se você não fornecer um tipo de recurso quando incluir arquivos, então ele irá ser encontrado aqui. Por padrão isso é "./templates", significando que isso irá olhar para o diretório de templates no mesmo diretório que está executando o script PHP.

Notas Técnicas

Não é recomendado colocar este diretório sob um diretório document root do seu webserver.

$compile_dir

Esse é o nome do diretório onde os template compilados estão localizados Por padrão isso é "./templates_c", significando que isso irá olhar para o diretório de templates no mesmo diretório que está executando o script PHP.

Notas Técnicas

Essa configuração deve ser um path relativo ou um path absoluto. include_path não é usado para escrever em arquivos.

Notas Técnicas

Não é recomendado colocar este diretório sob um diretório document root do seu webserver.

$config_dir

Este é o diretório usado para armazenar arquivos de configuração usados nos templates. O padrão é "./configs", significando que isso irá olhar para o diretório de templates no mesmo diretório que está executando o script PHP.

Notas Técnicas

Não é recomendado colocar este diretório sob um diretório document root do seu webserver.

$plugins_dir

Esse é o diretório onde Smarty irá procurar por plugins que são necessários. O Padrão é "plugins" sob o SMARTY_DIR. Se vocêes especificar um path relativo, Smarty irá primeiro procurar sob o SMARTY_DIR, então relativo para o cwd (current working directory), então relativo para cada entrada no seu PHP include path.

Notas técnicas

Para uma melhor performance, não configure seu plugins_dir para ter que usar o PHP include path. Use um path absoluto, ou um path relativo para SMARTY_DIR ou o cwd.

$debugging

Isso habilita o debugging console. O console é uma janela de javascript que informa à você sobre os arquivos de template incluídos e variáveis destinadas para a página de template atual.

$debug_tpl

Este é o nome do arquivo de template usado para o console de debug. Por padrão, é nomeado como debug.tpl e está localizado no SMARTY_DIR.

$debugging_ctrl

Isso permite caminhos alternativos de habilitar o debug. NONE não significa que métodos alternativos são permitidos. URL significa quando a palavra SMARTY_DEBUG foi encontrado na QUERY_STRING, que o debug está habilitado para a chamada do script. Se $debugging é true, esse valor é ignorado.

$autoload_filters

Se há algum filtro que você deseja carregar em cada chamada de template, você pode especificar-lhes usando essa variável e a Smarty irá automaticamente carregá-los para você. A variável é um array associativo onde as chaves são tipos de filtro e os valores são arrays de nomes de filtros. Por exemplo:

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

$compile_check

Em cima de cada requisição da aplicação PHP , Smarty testa para ver se o template atual foi alterado (diferentes time stamp) desde a última compilação. Se isso foi alterado, ele irá recompilar o template. Se o template não foi compilado, ele irá compilar de qualquer maneira dessa configuração. Por padrão esta variável é setada como true. Uma vez que a aplicação está em produção (templates não serão alterados), o passo compile_check não é necessário. Tenha certeza de setar $compile_check para "false" para maior performance. Note que se você alterar isso para "false" e o arquivo de template está alterado, você *não* irá ver a alteração desde que o template seja recompilado. Se caching está habilitado e compile_check está habilitado, então os arquivos de cache não serão regerados se um complexo arquivo de ou um arquivo de configuração foi atualizado. Veja $force_compile ou clear_compiled_tpl.

$force_compile

Isso força Smarty para (re)compilar templates a cada requisição. Essa configuração sobreescreve $compile_check. Por padrão isso está desabilitado. Isso é útil para desenvolvimento e debug. Isso nunca deve ser usado em ambiente de produção. Se caching está habilitado, os arquivo(s) de cache serão regerados à todo momento.

$caching

Isto diz à Smarty se há ou não saída de cache para o template. Por padrão isso está setado para 0, ou desabilitado. Se seu template gerar conteúdo redundante, é necessário ligar o caching. Isso irá resultar num ganho significativo de performance. Você pode também ter múltiplos caches para o mesmo template.

Um valor de 1 ou 2 caching habilitados. 1 diz à Smarty para usar a variável atual $cache_lifetime para determinar se o cache expirou. Um valor 2 diz à Smarty para usar o valor cache_lifetime então para quando o cache foi gerado. Desta maneira você pode setar o cache_lifetime imediatamente antes de buscar o template para ter controle sobre quando este cache em particular expira. Veja também is_cached.

Se $compile_check está habilitado, o conteúdo do cache irá ser regerado se algum dos templates ou arquivos de configuração que são parte deste cache estiverem alterados. Se $force_compile está habilitado, o conteúdo do cache irá sempre ser regerado.

$cache_dir

Isso é o nome do diretório onde os caches do template são armazenados. Por padrão isso é "./cache", significando que isso irá olhar para o diretório de cache no mesmo diretório que executar scripts PHP. Você pode tambe usar sua própria função customizada de manuseamento de cache para manipular arquivos de cache, que irão ignorar esta configuração.

Notas Técnicas

Essa configuração deve ser ou um relativo ou absoluto path. include_path não é usado para escrever em arquivos.

Notas Técnicas

Não é recomendado colocar este diretório sob um diretório document root do seu webserver.

$cache_lifetime

Isso é o comprimento de tempo em segundos que um cache de template é válido. Uma vez que este tempo está expirado, o cache irá ser regerado. $caching deve ser configurado para "true" para $cache_lifetime para ter algum propósito. Um valor de -1 irá forçar o cache a nunca expirar. Um valor de 0 irá fazer com que o cache seja sempre regerado (bom somente para testes, o método mais eficiente de desabilitar caching é setá-lo para $caching = false.)

Se $force_compile está habilitado, os arquivos de cache serão regerados todo o tempo, eficazmente desativando caching. Você pode limpar todos os arquivos de cache com a função clear_all_cache(), ou arquivos individuais de cache (ou grupos) com a função clear_cache().

Notas Técnicas

Se você quiser dar para certos templates seu próprio tempo de vida de um cache, você poderia fazer isso configurando $caching = 2, então configure $cache_lifetime para um único valor somente antes de chamar display() ou fetch().

$cache_handler_func

Você pode fornecer uma função padrão para manipular arquivos de cache ao invés de usar o método built-in usando o $cache_dir. Veja a seção cache handler function section para obter detalhes.

$cache_modified_check

Se configurado para true, Smarty irá respeitar o If-Modified-Since header enviado para o cliente. Se o timestamp do arquivo de cache não foi alterado desde a última visita, então um header "304 Not Modified" irá ser enviado ao invés do conteúdo. Isso funciona somente em arquivos de cache sem tags **insert**.

$config_overwrite

Se configurado para true, variáveis lidas no arquivo de configurações irão sobrescrever uma a outra. Do contrário, as variáveis serão guardadas em um array. Isso é útil se você quer armazenar arrays de dados em arquivos de configuração, somente lista tempos de cada elemento múltiplo. true por padrão.

$config_booleanize

Se setado para true, os valores do arquivo de configuração de on/true/yes e off/false/no ficará convertido para valores booleanos automaticamente. Desta forma você pode usar os valores em um template como: {if #foobar#} ... {/if}. Se foobar estiver on, true ou yes, a condição {if} irá executar. true por padrão.

$config_read_hidden

Se configurado para true, esconde seções (nomes de seções começados com um período) no arquivo de configuração podem ser lidos do template. Tipicamente você deixaria isto como false, desta forma você pode armazenar dados sensitivos no arquivo de configuração como um parâmetro de banco de dados e sem preocupar-se sobre o template carregá-los. false é o padrão.

$config_fix_newlines

Se setado para true, mac e dos newlines (\r e \r\n) no arquivo de configuração serão convertidos para \n quando eles forem interpretados. true é o padrão.

$default_template_handler_func

Essa função é chamada quando um template não pode ser obtido de seu recurso.

$php_handling

Isso diz à Smarty como manipular códigos PHP contido nos templates. Há quatro possíveis configurações, padrão sendo SMARTY_PHP_PASSTHRU. Note que isso NÃO fará efeito com códigos php dentro de tags {php}{/php} no template.

- SMARTY_PHP_PASSTHRU - Smarty echos tags as-is.
- SMARTY_PHP_QUOTE - Smarty quotes the tags as html entities.
- SMARTY_PHP_REMOVE - Smarty irá remover as tags do template.
- SMARTY_PHP_ALLOW - Smarty irá executar as tags como códigos PHP.

NOTE: Usando códigos PHP code dentro de templates é altamente desencorajado. Use custom functions ou modifiers ao invés disso.

$security

$security true/false, o padrão é false. Security é bom para situações quando você tem partes inconfiáveis editando o template (via ftp por exemplo) e você quer reduzir os riscos de comprometimento da segurança

do sistema através da linguagem de template. Habilitando-o faz-se cumprir as regras da linguagem de template, a menos que especificamente cancelada com $security_settings:

- Se $php_handling está setado para SMARTY_PHP_ALLOW, isso é implicitamente alterado para SMARTY_PHP_PASSTHRU
- Funçõs PHP não são permitidas em blocos IF, exceto estes especificados no $security_settings
- templates podem ser somente incluidos no diretório listado em $secure_dir array
- Arquivos locais podem ser somente trazidos do diretório listado em $secure_dir usando no array {fetch}
- Estas tags {php}{/php} não são permitidas
- Funções PHP não são permitidas como modificadores, exceto estes especificados no $security_settings

$secure_dir

Isso é um array de todos os diretórios locais que são considerados seguros. {include} e {fetch} usam estes (diretórios) quando security está habilitado.

$security_settings

Essas configurações são usadas para cancelar ou especificar configurações de segurança quando security está habilitado. Estas possuem as seguintes configurações possíveis:

- PHP_HANDLING - true/false. Se setado para true, a configuração de $php_handling não é checada para security.
- IF_FUNCS - Isso é um array de nomes de funções PHP permitidas nos blocos IF.
- INCLUDE_ANY - true/false. Se setado para true, algum template pode ser incluído para um arquivo do sistema, apesar de toda a lista de $secure_dir.
- PHP_TAGS - true/false. Se setado para true, as tags {php}{/php} são permitidas nos templates.
- MODIFIER_FUNCS - Isso é um array de nomes de funções PHP permitidas usadas como modificadores de variável.

$trusted_dir

$trusted_dir somente usado quando $security está habilitado. Isso é um array de todos os diretórios que são considerados confiáveis. Diretórios confiáveis são onde você irá deixar seus scripts php que são executados diretamente para o template com {include_php}.

$left_delimiter

Este é o delimitador esquerdo usado para a linguagem de template. O padrão é "{".

$right_delimiter

Este é o delimitador direito usado para a linguagem de template. O padrão é "}".

$compiler_class

Especifica o nome do compilador de classes que Smarty irá usar para compilar templates. O padrão é 'Smarty_Compiler'. Para usuários avançados somente.

$request_vars_order

A ordem na qual as variáveis requeridas serão registradas, similar ao variables_order no php.ini

$request_use_auto_globals

Especifica se a Smarty deve usar variáveis globais do php $HTTP_*_VARS[] ($request_use_auto_globals=false que é o valor padrão) ou $_*[] ($request_use_auto_globals=true). Isso afeta templates que fazem uso do {$smarty.request.*}, {$smarty.get.*} etc. . Atenção: Se você setar $request_use_auto_globals para true, variable.request.vars.order não terão efeito mas valores de configurações do php `gpc_order` são usados.

$error_reporting

Quando este valor é definido para um valor não nulo, o seu valor é usado como o nível de error_reporting [http://php.net/error_reporting] do php dentro de display() e fetch(). Quando debugging esta ativado este valor é ignorado e o nível de erro é mantido intocado.

Veja também trigger_error(), debugging e Troubleshooting.

$compile_id

Identificador de compilação persistente. Como uma alternativa para passar o mesmo compile_id para cada chamada de função, você pode setar este compile_id e isso irá ser usado implicitamente após isso.

$use_sub_dirs

Configure isso para false se seu ambiente de PHP não permite a criação de subdiretórios pela Smarty. Subdiretórios são muito eficientes, então use-os se você conseguir.

$default_modifiers

Isso é um array de modificadores implicitamente aplicados par cada variável no template. Por Exemplo, para cada variável HTML-escape por padrão, use o array('escape:"htmlall"'); Para fazer a variável isenta para modificadores padrão, passe o modificador especial "smarty" com um valor de parâmetro "nodefaults" modificando isso, como {$var|smarty:nodefaults}.

$default_resource_type

Isso diz à Smarty qual tipo de recurso usar implicitamente. O valor padrão é 'file', significando que $smarty->display('index.tpl'); e $smarty->display('file:index.tpl'); são idênticos no significado. Veja o capítulo resource para detalhes.

Chapter 13. Métodos

append

```
void append(var);

mixed var;

void append(varname, var);

string varname;
mixed var;

void append(varname, var, merge);

string varname;
mixed var;
boolean merge;
```

Isso é usado para adicionar um elemento para um array fixado. Se você adicionar uma string como valor, isso irá converter-se para um valor de array e então adicioná-lo. Você pode explicitamente passar pares nomes/valores, ou arrays associativos contendo o par nome/valor. Se você passar o terceiro parâmetro opcional para true, o valor unir-se ao array atual ao invés de ser adicionado.

Notas Técnicas

O parâmetro de união respeita a chave do array, então se você mesclar dois índices númericos de um array, eles devem sobrescrever-se um ao outro ou em resultados não sequências de chave. Isso é diferente da função de PHP array_merge() que apaga as chaves e as renumera.

Example 13.1. append

```
// passing name/value pairs
$smarty->append("Name","Fred");
$smarty->append("Address",$address);

// passing an associative array
$smarty->append(array("city" => "Lincoln","state" => "Nebraska"));
```

append_by_ref

```
void append_by_ref(varname, var);

string varname;
mixed var;

void append_by_ref(varname, var, merge);

string varname;
```

```
mixed var;
boolean merge;
```

Isso é usado para adicionar vlaores para o template por referência. Se você adicionar uma variável por referência e então alterar este valor o valor adicionado enxergará a alteração também. Para objetos, append_by_ref() também evita uma cópia em memória do objeto adicionado. Veja o manual do PHP em referenciando variáveis para uma melhor explanação sobre o assunto. Se você passar o terceiro parâmetro opcional para true, o valor irá ser mesclado com o array atual ao invés de adicioná-lo.

Notas Técnicas

O parâmetro de união respeita a chave do array, então se você mesclar dois índices númericos de arrays, eles devem sobrescrever-se um ao outro ou em resultados não sequências de chave. Isso é diferente da função de PHP array_merge() que apaga as chaves numéricas e as renumera.

Example 13.2. append_by_ref

```
// appending name/value pairs
$smarty->append_by_ref("Name",$myname);
$smarty->append_by_ref("Address",$address);
```

assign

```
void assign(var);

mixed var;

void assign(varname, var);

string varname;
mixed var;
```

Isso é usado para fixar valores para o template. Você pode explicitamente passar pares de nomes/valores, ou um array associativo contendo o par de nome/valor.

Example 13.3. assign

```
// passing name/value pairs
$smarty->assign("Name","Fred");
$smarty->assign("Address",$address);

// passing an associative array
$smarty->assign(array("city" => "Lincoln","state" => "Nebraska"));
```

assign_by_ref

```
void assign_by_ref(varname, var);

string varname;
mixed var;
```

Isso é usado para fixar valores para o template por referência ao invés de fazer uma cópia. Veja o manual do PHP na parte sobre referência de variáveis para uma explanação mais detalhada.

Notas Técnicas

Isso é usado para fixar valores para o template por referência. Se você fixar uma variável por referência e então alterar o valor dela, o valor fixado enxergará o valor alterado também. Para objetos, assign_by_ref() também restringe uma cópia de objetos fixados em memória. Veja o manual do php em refereciando variáveis para uma melhor explanação.

Example 13.4. assign_by_ref

```
// passing name/value pairs
$smarty->assign_by_ref("Name",$myname);
$smarty->assign_by_ref("Address",$address);
```

clear_all_assign

```
void clear_all_assign();

;
```

Isso limpa o valor de todas as variáveis fixadas.

Example 13.5. clear_all_assign

```
// clear all assigned variables
$smarty->clear_all_assign();
```

clear_all_cache

```
void clear_all_cache(expire time);

int expire time;
```

Isso limpa completamente o cache de template. Como um parâmetro opcional, você pode fornecer um ano mínimo em segundos que o arquivo de cache deve ter antes deles serem apagados.

Example 13.6. clear_all_cache

```
// clear the entire cache
$smarty->clear_all_cache();
```

clear_assign

```
void clear_assign(var);

string var;
```

Isso limpa o valor de uma variável fixada. Isso pode ser um valor simples, ou um array de valores.

Example 13.7. clear_assign

```
// clear a single variable
$smarty->clear_assign("Name");

// clear multiple variables
$smarty->clear_assign(array("Name","Address","Zip"));
```

clear_cache

```
void clear_cache(string template,
                 string cache id,
                 string compile id,
                 int expire time);
```

Isso limpa o cache de um template específico. Se você tem múltiplos caches para este arquivo, você limpa o cache específico fornecendo o cache id como o segundo parâmetro. Você pode também passar um compile id como um terceiro parâmetro. Você pode "agrupar" templates juntos e então eles podem ser removidos como um grupo. Veja o caching section para maiores informações. Como um quarto parâmetro opcional, você pode fornecer um ano mínimo em segundos que o arquivo de cache deve ter antes dele ser apagado.

Example 13.8. clear_cache

```
// clear the cache for a template
$smarty->clear_cache("index.tpl");

// clear the cache for a particular cache id in an multiple-cache template
$smarty->clear_cache("index.tpl","CACHEID");
```

clear_compiled_tpl

```
void clear_compiled_tpl(tpl_file);

string tpl_file;
```

Isso limpa a versão compilada do recurso de template especificado, ou todos os arquivos de templates compilados se nenhum for especificado. Essa função é para uso avançado somente, não normalmente necessária.

Example 13.9. clear_compiled_tpl

```
// clear a specific template resource
$smarty->clear_compiled_tpl("index.tpl");

// clear entire compile directory
$smarty->clear_compiled_tpl();
```

clear_config

```
void clear_config(string var);
```

Isso limpa todas as variáveis de configuração fixadas. Se um nome de variável é fornecido, somente esta variável é apagada.

Example 13.10. clear_config

```
// clear all assigned config variables.
$smarty->clear_config();

// clear one variable
$smarty->clear_config('foobar');
```

config_load

```
void config_load(string file,
                 string section);
```

Isso carrega o arquivo de configuração de dados e fixa-o para o template. Isso funciona idêntico a função config_load.

Notas Técnicas

À partir da Smarty 2.4.0, variáveis de template fixadas são mantidas através de fetch() e display(). Variáveis de configuração carregadas de config_load() são sempre de escopo global. Arquivos de configuração também são compilados para execução rápida, e repeita o force_compile e compile_check parâmetros de configuração.

Example 13.11. config_load

```
// load config variables and assign them
$smarty->config_load('my.conf');

// load a section
$smarty->config_load('my.conf','foobar');
```

display

```
void display(string template,
             string cache_id,
             string compile_id);
```

Isso mostra o template. Fornecendo um válido template resource tipo e path. Como um segundo parâmetro opcional, você pode passar um cache id. Veja o caching section para maiores informações.

Como um terceiro parâmetro opcional, você pode passar um compile id. Isso está no evento que você quer compilar diferentes versões do mesmo template, como ter templates compilados separadamente para

diferentes linguagens. Outro uso para compile_id é quando você usa mais do que um $template_dir mas somente um $compile_dir. Seta um compile_id em separado para cada $template_dir, de outra maneira templates com mesmo nome irão sobrescrever-se um ao outro. Você pode também setar a variável $compile_id ao invés de passar isso para cada chamada de display().

Example 13.12. display

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

// only do db calls if cache doesn't exist
if(!$smarty->is_cached("index.tpl"))
{

    // dummy up some data
    $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);

}

// display the output
$smarty->display("index.tpl");
```

Use a sintaxe para template resources para mostrar arquivos fora do $template_dir directory.

Example 13.13. Exemplos de recursos da função display

```
// absolute filepath
$smarty->display("/usr/local/include/templates/header.tpl");

// absolute filepath (same thing)
$smarty->display("file:/usr/local/include/templates/header.tpl");

// windows absolute filepath (MUST use "file:" prefix)
$smarty->display("file:C:/www/pub/templates/header.tpl");

// include from template resource named "db"
$smarty->display("db:header.tpl");
```

fetch

```
  string fetch(string template,
```

```
                string cache_id,
                string compile_id);
```

Isso retorna a saída do template ao invés de mostrá-lo. Fornecendo um tipo ou path válido template resource. Como um segundo parâmetro opcional, você pode passar o cache id. Veja o caching section para maiores informações.

Como um terceiro parâmetro opcional, você pode passar um compile id. Isso está no evento que você quer compilar diferentes versões do mesmo template, como ter templates compilados separadamente para diferentes linguagens. Outro uso para compile_id é quando você usa mais do que um $template_dir mas somente um $compile_dir. Seta um compile_id em separado para cada $template_dir, de outra maneira templates com mesmo nome irão sobrescrever-se uns aos outros. Você pode também setar a variável $compile_id ao invés de passá-la para cada chamada de fetch().

Example 13.14. fetch

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

$smarty->caching = true;

// only do db calls if cache doesn't exist
if(!$smarty->is_cached("index.tpl"))
{

    // dummy up some data
    $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);

}

// capture the output
$output = $smarty->fetch("index.tpl");

// do something with $output here

echo $output;
```

get_config_vars

```
  array get_config_vars(string varname);
```

Isso retorna o valor da variável de configuração dada. Se nenhum parâmetro é dado, um array de todas as variáveis dos arquivos de configurações é retornado.

Example 13.15. get_config_vars

```
// get loaded config template var 'foo'
$foo = $smarty->get_config_vars('foo');

// get all loaded config template vars
$config_vars = $smarty->get_config_vars();

// take a look at them
print_r($config_vars);
```

get_registered_object

```
array get_registered_object(object_name);

string object_name;
```

Isso retorna uma referência para um objeto registrado. Isso é útil para dentro de uma função customizada quando você precisa acessar diretamente um objeto registrado.

Example 13.16. get_registered_object

```
function smarty_block_foo($params, &$smarty) {
 if (isset[$params['object']]) {
  // get reference to registered object
     $obj_ref =& $smarty->&get_registered_object($params['object']);
  // use $obj_ref is now a reference to the object
 }
}
```

get_template_vars

```
  array get_template_vars(string varname);
```

Isso retorna o valor de uma variável fixada. Se nenhum parâmetro é dado, um array de todas as variáveis fixadas é retornado.

Example 13.17. get_template_vars

```
// get assigned template var 'foo'
$foo = $smarty->get_template_vars('foo');

// get all assigned template vars
$tpl_vars = $smarty->get_template_vars();

// take a look at them
print_r($tpl_vars);
```

is_cached

```
void is_cached(template, cache_id);
```

```
string template;
[string cache_id];
```

Isso retorna true se há um cache válido para esse template. Isso somente funciona se caching está setado para true.

Example 13.18. is_cached

```
$smarty->caching = true;

if(!$smarty->is_cached("index.tpl")) {
    // do database calls, assign vars here
}

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

Você pode também passar um cache id como um segundo parâmetro opcional no caso você quer múltiplos caches para o template dado.

Example 13.19. is_cached with multiple-cache template

```
$smarty->caching = true;

if(!$smarty->is_cached("index.tpl","FrontPage")) {
    // do database calls, assign vars here
}

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

load_filter

```
void load_filter(type, name);

string type;
string name;
```

Essa função pode ser usada para carregar um filtro de plugin. O primeiro argumento especifica o tipo do filtro para carregar e pode ser um dos seguintes: 'pre', 'post', ou 'output'. O segundo argumento especifica o nome do filtro de plugin, por exemplo, 'trim'.

Example 13.20. Carregando filtros de plugins

```
$smarty->load_filter('pre', 'trim'); // load prefilter named 'trim'
$smarty->load_filter('pre', 'datefooter'); // load another prefilter named 'datefo
$smarty->load_filter('output', 'compress'); // load output filter named 'compress'
```

register_block

```
void register_block(name, impl, cacheable, cache_attrs);

string name;
mixed impl;
```

```
bool cacheable;
array or null cache_attrs;
```

Use isso para registrar dinamicamente blocos de funções de plugins. Passe no bloco de nomes de função, seguido por uma chamada de função PHP que implemente isso.

A chamada de uma função-php *`impl`* pode ser (a) uma string contendo o nome da função ou (b) um array no formato `array(&$object, $method)` com `&$object` sendo uma referência para um objeto e `$method` sendo uma string contendo o nome do método ou (c) um array no formato `array(&$class, $method)` com `$class` sendo um nome de classe e `$method` sendo um método desta classe.

`$cacheable` e *`$cache_attrs`* podem ser omitidos na maior parte dos casos. Veja Controlando modos de Saída de Cache dos Plugins para obter informações apropriadas.

Example 13.21. register_block

```
/* PHP */
$smarty->register_block("translate", "do_translation");

function do_translation ($params, $content, &$smarty, &$repeat) {
    if (isset($content)) {
        $lang = $params['lang'];
        // do some translation with $content
        return $translation;
    }
}

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

register_compiler_function

```
void register_compiler_function(name, impl, cacheable);
```

```
string name;
mixed impl;
bool cacheable;
```

Use isso para registrar dinamicamente uma função de plugin compilador. Passe no nome da função compilador, seguido pela função PHP que implemente isso.

A chamada para função-php *`impl`* pode ser uma string contendo o nome da função ou (b) um array no formato `array(&$object, $method)` com `&$object` sendo uma referência para um objeto e `$method` sendo uma string contendo o nome do método ou (c) um array no formato `array(&$class, $method)` com `$class` sendo um nome de classe e `$method` sendo o método desta classe.

`$cacheable` pode ser omitido na maioria dos casos. Veja Controlando modos de Saída de Cache dos Plugins para obter informações apropriadas.

register_function

```
void register_function(name, impl, cacheable, cache_attrs);
```

```
string name;
mixed impl;
bool cacheable;
array or null cache_attrs;
```

Use isso para registrar funções de plugins dinamicamente para o template. Passe no template o nome da função, seguido pelo nome da função PHP que implemente isso.

A chamada para função-php *`impl`* pode ser (a) uma string contendo o nome da função ou (b) um array no formato `array(&$object, $method)` com `&$object` sendo uma referência para um objeto e `$method` sendo uma string contendo o nome do método ou (c) um array no formato `array(&$class, $method)` com `$class` sendo um nome de classe e `$method` sendo um método desta classe.

`$cacheable` e *`$cache_attrs`* podem ser omitidos na maioria dos casos. Veja Controlando modos de Saída Cache dos Plugins para obter informações apropriadas.

Example 13.22. register_function

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

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

// agora você pode usar isso no Smarty para mostrar a data atual: {date_now}
// ou, {date_now format="%Y/%m/%d"} para formatar isso.
```

register_modifier

```
void register_modifier(name, impl);

string name;
mixed impl;
```

Use isso para modificar dinamicamente plugins registrados. Passe no template o nome do modificador, seguido da função PHP que implemente isso.

A chamada da função-php *`impl`* pode ser (a) uma strin contendo o nome da função ou (b) um array no formato `array(&$object, $method)` com `&$object` sendo uma referência para um objeto e `$method` sendo uma string contendo o nome do método ou (c) um array no formato `array(&$class, $method)` com `$class` sendo um nome de classe e `$method` sendo um método desta classe.

Example 13.23. register_modifier

```
// let's map PHP's stripslashes function to a Smarty modifier.

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

// now you can use {$var|sslash} to strip slashes from variables
```

register_object

```
void register_object(object_name, $object, allowed methods/properties,
format, block methods);

string object_name;
object $object;
array allowed methods/properties;
boolean format;
array block methods;
```

Isso é para registrar um objeto para uso no template. Veja a seção de objetos do manual para examplos.

register_outputfilter

```
void register_outputfilter(function);

mixed function;
```

Use isso para registrar dinamicamente filtros de saída para operações na saída do template antes de mostrá-lo. Veja Filtros de Saída de Templates para maiores informações de como configurar uma função de filtro de saída.

A chamada da função-php *`function`* pode ser (a) uma string contendo um nome de função ou (b) um array no formato `array(&$object, $method)` com `&$object` sendo uma referência para um objeto e `$method` sendo uma string contendo o nome do método ou (c) um array no formato `array(&$class, $method)` com `$class` sendo um nome de classe e `$method` sendo um método desta classe.

register_postfilter

```
void register_postfilter(function);

mixed function;
```

Use isso para registrar dinamicamente pósfiltros para rodar templates após eles terem sido compilados. Veja pósfiltros de template para maiores informações de como configurar funções de pósfiltragem.

A chamada da função-php *`function`* pode ser (a) uma string contendo um nome de função ou (b) um array no formato `array(&$object, $method)` com `&$object` sendo uma referência para um objeto e `$method` sendo uma string contendo o nome do método ou (c) um array no formato `array(&$class, $method)` com `$class` sendo um nome de classe e `$method` sendo um método desta classe.

register_prefilter

```
void register_prefilter(function);

mixed function;
```

Use isso para registrar préfiltros dinamicamente para rodar templates antes deles serem compilados. Veja template prefilters para maiores informações de como configurar uma função de préfiltragem.

A chamada da função-php *`function`* pode ser (a) uma string contendo um nome de função ou (b) um array no formato `array(&$object, $method)` com `&$object` sendo uma referência para um objeto e `$method` sendo uma string contendo o nome do método ou (c) um array no formato `array(&$class, $method)` com `$class` sendo um nome de classe e `$method` sendo um método desta classe.

register_resource

```
void register_resource(name, resource_funcs);

string name;
array resource_funcs;
```

Use isso para registrar dinamicamente um recurso de plugin com a Smarty. Passe no nome o recurso e o array de funções PHP que implementam isso. Veja template resources para maiores informações de como configurar uma função para retornar templates.

Notas Técnicas

Um nome de recurso deve ter ao menos dois caracteres de comprimento. Um caracter do nome de recurso irá ser ignorado e usado como parte do path do arquivo como, $smarty->display('c:/path/to/index.tpl');

A função-php-array *`resource_funcs`* deve ter 4 ou 5 elementos. Com 4 elementos os elementos são as functions-callbacks para as respectivas funções "source", "timestamp", "secure" e "trusted" de recurso. Com 5 elementos o primeiro elemento tem que ser um objeto por referência ou um nome de classe do objeto ou uma classe implementando o recurso e os 4 elementos seguintes tem que ter os nomes de métodos implementando "source", "timestamp", "secure" e "trusted".

Example 13.24. register_resource

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

trigger_error

```
void trigger_error(error_msg, level);

string error_msg;
[int level];
```

Essa função pode ser usada para saída de uma mensagem de erro usando Smarty. O parâmetro *`level`* pode ser um dos valores usados para a função de php trigger_error(), ex.: E_USER_NOTICE, E_USER_WARNING, etc. Por padrão é E_USER_WARNING.

template_exists

```
bool template_exists(template);
```

```
string template;
```

Essa função checa se o template especificado existe. Isso pode aceitar um path para o template no filesystem ou um recurso de string especificando o template.

unregister_block

```
void unregister_block(name);

string name;
```

Use isso para desregistrar dinamicamente um bloco de funções de plugin. Passe no bloco o nome da função.

unregister_compiler_function

```
void unregister_compiler_function(name);

string name;
```

Use essa função para desregistrar uma função de compilador. Passe o nome da função de compilador.

unregister_function

```
void unregister_function(name);

string name;
```

Use isso para desregistrar dinamicamente uma função de plugin do template. Passe no template o nome da função.

Example 13.25. unregister_function

```
// nós não queremos que designers template tenham acesso aos nossos arquivos do si

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

unregister_modifier

```
void unregister_modifier(name);

string name;
```

Use isso para desregistrar dincamimente um modificador de plugin. Passe no template o nome do modificador.

Example 13.26. unregister_modifier

```
// nós não queremos que designers de template usem strip tags para os elementos

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

unregister_object

```
void unregister_object(object_name);

string object_name;
```

Use isso para desregistrar um objeto.

unregister_outputfilter

```
void unregister_outputfilter(function_name);

string function_name;
```

Use isso para desregistrar dinamicamente um filtro de saída.

unregister_postfilter

```
void unregister_postfilter(function_name);

string function_name;
```

Use isso para dinamicamente desregistrar um pósfiltro.

unregister_prefilter

```
void unregister_prefilter(function_name);

string function_name;
```

Use isso para dinamicamente desregistrar um préfiltro.

unregister_resource

```
void unregister_resource(name);

string name;
```

Use isso para dinamicamente desregistrar um recurso de plugin. Passe no parâmetro nome o nome do recurso.

Example 13.27. unregister_resource

```
$smarty->unregister_resource("db");
```

Chapter 14. Caching

Caching é usado para aumentar a velocidade de chamada para display() ou fetch() salvando isso num arquivo de saída. Se há uma versão de cache disponível para a chamada, isso é mostrado ao invés de regerar a saída de dados. Caching pode fazer coisas tremendamente rápidas, especialmente templates com longo tempo computacional. Desde a saída de dados do display() ou fetch() está em cache, um arquivo de cache poderia ser composto por diversos arquivos de templates, arquivos de configuração, etc.

Desde que templates sejam dinâmicos, é importante isso ter cuidado com o que você está fazendo cache e por quanto tempo. Por exemplo, se você está mostrando a página principal do seu website na qual as alterações de conteúdo são muito frequentes, isso funciona bem para cache dessa por uma hora ou mais. Um outro modo, se você está mostrando uma página com um mapa do tempo contendo novas informações por minuto, não faz sentido fazer cache nesta página.

Configurando Caching

A primeira coisa a fazer é habilitar o caching. Isso é feito pela configuração $caching = true (or 1.)

Example 14.1. Habilitando Caching

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

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

Com caching habilitado, a chamada para a função display('index.tpl') irá trazer o template como usual, mas também salva uma cópia disso para o arquivo de saída (uma cópia de cache) in the $cache_dir. Na próxima chamada de display('index.tpl'), a cópia em cache será usada ao invés de trazer novamente o template.

Notas Técnicas

Os arquivos no $cache_dir são nomeados com similaridade ao nome do arquivo de template. Embora eles terminem com a extensão ".php", eles não são realmente scripts executáveis de php. Não edite estes arquivos!

Cada página em cache tem um período de tempo limitado determinado por $cache_lifetime. O padrão do valor é 3600 segundos, ou 1 hora. Após o tempo expirar, o cache é regerado. É possível dar tempos individuais para caches com seu próprio tempo de expiração pela configuração $caching = 2. Veja a documentação em $cache_lifetime para detalhes.

Example 14.2. Configurando cache_lifetime por cache

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 2; // lifetime is per cache

// set the cache_lifetime for index.tpl to 5 minutes
$smarty->cache_lifetime = 300;
$smarty->display('index.tpl');

// set the cache_lifetime for home.tpl to 1 hour
$smarty->cache_lifetime = 3600;
$smarty->display('home.tpl');

// NOTE: the following $cache_lifetime setting will not work when $caching = 2.
// The cache lifetime for home.tpl has already been set
// to 1 hour, and will no longer respect the value of $cache_lifetime.
// The home.tpl cache will still expire after 1 hour.
$smarty->cache_lifetime = 30; // 30 seconds
$smarty->display('home.tpl');
```

Se $compile_check está habilitado, cada arquivo de template e arquivo de configuração que está envolvido com o arquivo em cache é checado por modificações. Se algum destes arquivos foi modificado desde que o último cache foi gerado, o cache é imediatamente regerado. Isso é ligeiramente uma forma de optimização de performance de overhead, deixe $compile_check setado para false.

Example 14.3. Habilitando $compile_check

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;
$smarty->compile_check = true;

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

Se $force_compile está habilitado, os arquivos de cache irão sempre ser regerados. Isso é efetivamente desativar caching. $force_compile é usualmente para propósitos de debug somente, um caminho mais eficiente de desativar caching é setar o $caching = false (ou 0.)

A função is_cached() pode ser usada para testar se um template tem um cache válido ou não. Se você tem um template com cache que requer alguma coisa como um retorno do banco de dados, você pode usar isso para pular este processo.

Example 14.4. Usando is_cached()

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

if(!$smarty->is_cached('index.tpl')) {
 // No cache available, do variable assignments here.
 $contents = get_database_contents();
 $smarty->assign($contents);
}

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

Você pode deixar partes da sua página dinâmica com a função de template insert. Vamos dizer que sua página inteira pode ter cache exceto para um banner que é mostrado abaixo do lado direito da sua página. Usando uma função insert para o banner, você pode deixar esse elemento dinâmico dentro do conteúdo de cache. Veja a documentação em insert para detalhes e exemplos.

Você pode limpar todos os arquivos de cache com a função clear_all_cache(), ou arquivos de cache individuais (ou grupos) com a função clear_cache().

Example 14.5. Limpando o cache

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// clear out all cache files
$smarty->clear_all_cache();

// clear only cache for index.tpl
$smarty->clear_cache('index.tpl');

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

Multiple Caches Per Page

Você pode ter múltiplos arquivos de cache para uma simples chamada de display() ou fetch(). Vamos dizer que uma chamada para display('index.tpl') deve ter vários conteúdo de saída diferentes dependendo de alguma condição, e você quer separar os caches para cada um. Você pode fazer isso passando um cache_id como um segundo parâmetro para a chamada da função.

Example 14.6. Passando um cache_id para display()

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

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

Acima, nós estamos passando a variável $my_cache_id para display() com o cache_id. Para cada valor único de $my_cache_id, um cache em separado irá ser gerado para index.tpl. Nesse exemplo, "article_id" foi passado em URL e é usado como o cache_id.

Notas Técnicas

Tenha muito cuidado quando passar valores do cliente (web brownser) dentro da Smarty (ou alguma aplicação PHP.) Embora o exemplo acima usando o article_id vindo de uma URL pareça fácil, isso poderia ter consequências ruins. O cache_id é usado para criar um diretório no sistema de arquivos, então se o usuário decidir passar um valor extremamente largo para article_id, ou escrever um script que envia article_ids randômicos em um ritmo rápido, isso poderia possivelmente causar problemas em nível de servidor. Tenha certeza de limpar algum dado passado antes de usar isso. Nessa instância, talvez você saiba que o article_id tem um comprimento de 10 caracteres e isso é constituído somente de alfa-numéricos, e deve ser um article_id válido no database. Verifique isso!

Tenha certeza de passar o mesmo cache_id como o segundo parâmetro para is_cached() e clear_cache().

Example 14.7. Passando um cache_id para 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)) {
 // No cache available, do variable assignments here.
 $contents = get_database_contents();
 $smarty->assign($contents);
}

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

Você pode limpar todos os caches para um cache_id em particular passando o primeiro parâmetro null para clear_cache().

Example 14.8. Limpando todos os caches para um cache_id em particular

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// clear all caches with "sports" as the cache_id
$smarty->clear_cache(null,"sports");

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

Desta maneira, você pode "agrupar" seus caches juntos dando-lhes o mesmo cache_id.

Grupos de Cache

Você pode fazer agrupamentos mais elaborados configurando grupos de cache_id. Isso é realizado pela separação de cada sub-grupo com uma barra vertical "|" no valor do cache_id. Você pode ter muitos sub-grupos com você desejar.

Example 14.9. Grupos de cache_id

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// clear all caches with "sports|basketball" as the first two cache_id groups
$smarty->clear_cache(null,"sports|basketball");

// clear all caches with "sports" as the first cache_id group. This would
// include "sports|basketball", or "sports|(anything)|(anything)|(anything)|..."
$smarty->clear_cache(null,"sports");

$smarty->display('index.tpl',"sports|basketball");
```

Notas Técnicas

O agrupamento de cache id NÃO use o path do template como alguma parte do cache_id. Por exemplo, se você tem display('themes/blue/index.tpl'), você não pode limpar o cache para tudo que estiver sob o diretório "themes/blue". Se você quiser fazer isso, você deve agrupá-los no cache_id, como display('themes/blue/index.tpl','themes|blue'); Então você pode limpar os caches para o tema azul com with clear_cache(null,'themes|blue');

Controlling Cacheability of Plugins' Output

Desde Smarty-2.6.0 os caches de plugins pode ser declarados ao registrá-los. O terceiro parâmetro para register_block, register_compiler_function e register_function é chamado `$cacheable` e o padrão para true que é também o comportamento de plugins na versão da Smarty antecessores à 2.6.0

Quando registrando um plugin com $cacheable=false o plugin é chamado todo o tempo na página que está sendo mostrada, sempre se a página vier do cache. A função de plugin tem um comportamento levemente como uma função insert.

Em contraste para {insert} o atributo para o plugin não está em cache por padrão. Eles podem ser declarados para serem cacheados com o quarto parâmetro *`$cache_attrs`*. *`$cache_attrs`* é um array de nomes de atributos que devem ser cacheados, então a função de plugin pega o valor como isso sendo o tempo que a página foi escrita para o cache todo o tempo isso é buscado do cache.

Example 14.10. Prevenindo uma saída de plugin de ser cacheada

```
index.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')) {
    // fetch $obj from db and assign...
    $smarty->assign_by_ref('obj', $obj);
}

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


index.tpl:

Tempo restante: {remain endtime=$obj->endtime}
```

O número de segundos até que o endtime de $obj alcança alterações em cada display de página, mesmo que a página esteja em cache. Desde o atributo endtime esteja em cache o objeto somente tem que ser puxado do banco de dados quando a página está escrita para o cache mas não em requisições subsequentes da página.

Example 14.11. Prevenindo uma passagem inteira do template para o cache

```
index.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');


index.tpl:

Page created: {"0"|date_format:"%D %H:%M:%S"}

{dynamic}

Now is: {"0"|date_format:"%D %H:%M:%S"}

... do other stuff ...

{/dynamic}
```

Quando recarregado a página que você irá notar que ambas as datas diferem. Uma é "dinâmica" e uma é "estática". Você pode fazer qualquer coisa entre as tags {dynamic}...{/dynamic} e ter certeza que isso não irá ficar em cache como o restante da página.

Chapter 15. Advanced Features

Objetos

O Smarty permite acesso a objetos do PHP através de seus templates. Há duas formas de acessá-los. Uma forma é registrar objetos para o template, então os acessa via sintaxe similar a funções customizáveis. A outra forma é atribuir objetos para os templates e acessá-los como se fossem uma variável atribuída. O primeiro método tem uma sintaxe de template muito mais legal. E também mais segura, à medida que um objeto registrado pode ser restrito a certos métodos e propriedades. Entretanto, um objeto registrado não pode ser posto em loop ou ser atribuido em arrays de objetos, etc. O método que você escolher será determinado pelas suas necessidades, mas use o primeiro método se possível para manter um mínimo de sintaxe no template.

Se a segurança está habilitada, nenhum dos métodos privados ou funções podem acessados (começando com "_"). Se um método e propriedade de um mesmo nome existir, o método será usado.

Você pode restringir os métodos e propriedades que podem ser acessados listando os em um array como o terceiro parâmetro de registração.

Por definição, parâmetros passados para objetos através dos templates são passados da mesma forma que funções customizáveis os obtém. Um array associativo é passado como o primeiro parâmetro, e o objeto smarty como o segundo. Se você quer que os parâmetros passados um de cada vez por cada argumento como passagem de parâmetro de objeto tradicional, defina o quarto parâmetro de registração para falso.

O quinto parâmetro opcional só tem efeito com *`format`* sendo `true` e contém uma lista de métods de ob que seriam tratados como blocos. Isso significa que estes métodos tem uma tag de fechamento no template (`{foobar->meth2}...{/foobar->meth2}`) e os parâmetros para os métodos tem a mesma sinopse como os parâmetros para block-function-plugins: Eles pegam 4 parâmetros *`$params`*, *`$content`*, *`&$smarty`* e *`&$repeat`* e eles também comportam-se como block-function-plugins.

Example 15.1. usando um objeto registrado ou atribuído

```
<?php
// O objeto

class My_Object {
 function meth1($params, &$smarty_obj) {
  return "this is my meth1";
 }
}

$myobj = new My_Object;
// registrando o objeto (será por referência)
$smarty->register_object("foobar",$myobj);
// Se você quer restringie acesso a certos métodos ou propriedades, liste-os
$smarty->register_object("foobar",$myobj,array('meth1','meth2','prop1'));
// Se você quer usar o formato de parâmetro de objeto tradicional, passe um boolea
$smarty->register_object("foobar",$myobj,null,false);

// Você pode também atribuir objetos. Atribua por referência quando possível.
$smarty->assign_by_ref("myobj", $myobj);

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

TEMPLATE:

{* accessa nosso objeto registrado *}
{foobar->meth1 p1="foo" p2=$bar}

{* você pode também atribuir a saída *}
{foobar->meth1 p1="foo" p2=$bar assign="output"}
the output was {$output}

{* acessa nosso objeto atribuído *}
{$myobj->meth1("foo",$bar)}
```

Prefilters

Os prefilters de Template são funções de PHP nas quais seus templates são rodados antes de serem compilados. Isto é bom para preprocessamento de seus templates para remover comentários indesejados, mantendo o olho no que as pessoas estão colocando nos seus templates, etc. Prefilters podem ser ou registrado ou carregado do diretório de plugins usando a função load_filter() ou pela configuração da variável $autoload_filters. O Smarty passará o código fonte do template como o primeiro argumeto, e espera a função retornar o código fonte do template resultante.

Example 15.2. Usando um prefilter de template

```
<?php
// Ponha isto em sua aplicação
function remove_dw_comments($tpl_source, &$smarty)
{
    return preg_replace("/<!--#.*-->/U","",$tpl_source);
}

// registrar o prefilter
$smarty->register_prefilter("remove_dw_comments");
$smarty->display("index.tpl");
?>

{* Smarty template index.tpl *}
<!--# esta linha será removida pelo prefilter -->
```

Postfilters

Os postfilters de template são funções de PHP nas quais seus templates são rodados imediatamente depois de serem compilados. Os postfilters podem ser ou registradocarrgados do diretório de plugins usando a função load_filter() ou pela variável de configuração $autoload_filters. O Smarty passará o código fonte do template compilado como o primeiro argumento, e espera a função retornar o resultado do processamento.

Example 15.3. usando um postfilter de template

```
<?php
// ponha isto em sua aplicação
function add_header_comment($tpl_source, &$smarty)
{
    return "<?php echo \"<!-- Created by Smarty! -->\n\" ?>\n".$tpl_source;
}

// registra o postfilter
$smarty->register_postfilter("add_header_comment");
$smarty->display("index.tpl");
?>

{* compiled Smarty template index.tpl *}
<!-- Created by Smarty! -->
{* rest of template content... *}
```

Output Filters (Filtros de Saída)

Quando o template é invocado via display() ou fetch(), sua saída pode ser enviada através de um ou mais filtros de saída. Estes diferem dos postfilters porque postfilters operam em templates compilados antes de serem salvos para o disco, e os filtros de saída operam na saída do template quando ele é executado.

Filtros de Saída podem ser ou registrado ou carregado do diretório de plugins usando a função load_filter() ou configurando a variável $autoload_filters. O Smarty passará a saída como o primeiro argumento, e espera a função retornar o resultado do processamento.

Example 15.4. usando um filtro de saída de template

```
<?php
// ponha isto em sua aplicação
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;
}

// registra o outputfilter
$smarty->register_outputfilter("protect_email");
$smarty->display("index.tpl");

// agora qualquer ocorrência de um endereço de email na saída do template terá uma
// simples proteção contra spambots
?>
```

Função Manipuladora de Cache

Como uma alternativa ao uso do mecanismo de caching padrão baseado em arquivo, você pode especificar uma função de manipulação de cache customizada que será usada para ler, escrever e limpar arquivos de cache.

Crie uma função em sua aplicação que o Smarty usará como um manipulador de cache. Defina o nome dela na variável de classe $cache_handler_func. O Smarty agora usará esta para manipular dados no cache. O primeiro argumento é a ação, que é um desses 'read', 'write' e 'clear'. O segundo parâmetro é o objeto do Smarty. O terceiro parâmetro é o conteúdo que está no cache. No write, o Smarty passa o conteúdo em cache nestes parâmetros. No 'read', o Smarty espera sua função aceitar este parâmetro por referência e preenche ele com os dados em cache. No 'clear', passa uma variável simulacra aqui visto que ela não é usada. O quarto parâmetro é o nome do arquivo de template (necessário para ler/escrever), o quinto parâmetro é a cache_id (opcional), e o sexto é a compile_id (opcional).

Note que: O último parâmetro ($exp_time)foi adicionado no Smarty-2.6.0.

Example 15.5. exemplo usando MySQL como uma fonte de cache

```
 switch ($action) {
  case 'read':
   // save cache to database
   $results = mysql_query("select CacheContents from CACHE_PAGES where CacheID='$C
   if(!$results) {
    $smarty_obj->_trigger_error_msg("cache_handler: query failed.");
   }
   $row = mysql_fetch_array($results,MYSQL_ASSOC);

   if($use_gzip && function_exists("gzuncompress")) {
    $cache_contents = gzuncompress($row["CacheContents"]);
   } else {
    $cache_contents = $row["CacheContents"];
   }
   $return = $results;
   break;
  case 'write':
   // save cache to database

   if($use_gzip && function_exists("gzcompress")) {
    // compress the contents for storage efficiency
    $contents = gzcompress($cache_content);
   } else {
    $contents = $cache_content;
   }
   $results = mysql_query("replace into CACHE_PAGES values(
       '$CacheID',
       '".addslashes($contents)."')
      ");
   if(!$results) {
    $smarty_obj->_trigger_error_msg("cache_handler: query failed.");
   }
   $return = $results;
   break;
  case 'clear':
   // clear cache info
   if(empty($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("cache_handler: query failed.");
   }
   $return = $results;
   break;
  default:
   // error, unknown action
   $smarty_obj->_trigger_error_msg("cache_handler: unknown action \"$action\"");
   $return = false;
   break;
 }
 mysql_close($link);
 return $return;

}

?>
```

Recursos (Resources)

Os templates podem vir de uma variedade de fontes. Quando você exibe (display) ou busca (fetch) um template, ou inclui um template de dentro de outro template, você fornece um tipo de recurso, seguido pelo caminho e nome do template apropriado. Se um recurso não é dado explicitamente o valor de $default_resource_type é assumido.

Templates partindo do $template_dir

Templates a partir do $template_dir não exigem um recurso de template, apesar de você usar o arquivo: resource for consistancy. Apenas forneça o caminho para o template que você quer usar em relação ao diretório root $template_dir.

Example 15.6. Usando templates partindo do $template_dir

```
// from PHP script
$smarty->display("index.tpl");
$smarty->display("admin/menu.tpl");
$smarty->display("file:admin/menu.tpl"); // Igual ao de cima

{* de dentro do template do Smarty *}
{include file="index.tpl"}
{include file="file:index.tpl"} {* igual ao de cima *}
```

Templates partindo de qualquer diretório

Os Templates de fora do $template_dir exigem o arquivo: tipo de recurso do template, seguido pelo caminho absoluto e nome do template.

Example 15.7. usando templates partindo de qualquer diretório

```
// de dentro do script PHP
$smarty->display("file:/export/templates/index.tpl");
$smarty->display("file:/path/to/my/templates/menu.tpl");

{* de dentro do template do Smarty *}
{include file="file:/usr/local/share/templates/navigation.tpl"}
```

Caminhos de arquivos do Windows

Se você está usando uma máquina windows, caminhos de arquivos normalmente incluem uma letra do drive (C:) no começo do nome do caminho. Esteja certo de usar "file:" no caminho para evitar conflitos de nome e obter os resultados desejados.

Example 15.8. usando templates com caminhos de arquivo do windows

```
// de dentro do script PHP
$smarty->display("file:C:/export/templates/index.tpl");
$smarty->display("file:F:/path/to/my/templates/menu.tpl");

{* de dentro do template do Smarty *}
{include file="file:D:/usr/local/share/templates/navigation.tpl"}
```

Templates partindo de outras fontes

Você pode resgatar templates usando qualquer fonte possível de você acessar com PHP: banco de dados, sockets, LDAP, e assim por diante. Você faz isto escrevendo as funções de plugin de recurso e registrando elas com o Smarty.

Veja a seção plugins de recurso para mais informação sobre as funções que você deve fornecer.

Note

Note que você pode ativar manualmente o recurso de `arquivo` embutido, mas não pode fornecer um recurso que busca templates a partir do sistema de arquivos de alguma outra forma registrando sob um outro nome de recurso. `file` resource, but you can provide a resource that fetches templates from the file system in some other way by registering under another resource name.

// no script PHP

// ponha estas funções em algum lugar de sua aplicação

Example 15.3. usando recursos customizáveis

```
function db_get_template ($tpl_name, &$tpl_source, &$smarty_obj)
{
    // faça o banco de dados chamar aqui para buscar o seu template,
    // preenchendo o $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)
{
    // faça o banco de dados chamar daqui para preencher a $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)
{
    // assume-se que todos os templates são seguros
    return true;
}

function db_get_trusted($tpl_name, &$smarty_obj)
{
    // não usado para templates
}

// registrar o nome de recurso "db"
$smarty->register_resource("db", array("db_get_template",
                                       "db_get_timestamp",
                                       "db_get_secure",
                                       "db_get_trusted"));

// usando o recurso a partir do script PHP
$smarty->display("db:index.tpl");

{* usando o recurso de dentro do template do Smarty *}
{include file="db:/extras/navigation.tpl"}
```

Função Manipuladora de Template Padrão

Você pode especificar a função que é usada para devolver o conteúdo do template no evento em que o template não pode ser devolvido de seu recurso. Um uso disto é para criar templates que não existem "on-the-fly" (templates cujo conteúdo flutua muito, bastante variável).

Example 15.10. usando a função manipuladora de template padrão

```
<?php
// ponha esta função em algum lugar de sua aplicação

function make_template ($resource_type, $resource_name, &$template_source, &$templ
{
 if( $resource_type == 'file' ) {
  if ( ! is_readable ( $resource_name )) {
   // cria um arquivo de template, retorna o conteúdo.
   $template_source = "This is a new template.";
   $template_timestamp = time();
   $smarty_obj->_write_file($resource_name,$template_source);
   return true;
  }
    } else {
  // não é arquivo
  return false;
    }
}

// defina a manipuladora padrão
$smarty->default_template_handler_func = 'make_template';
?>
```

Chapter 16. Extendendo a Smarty com Plugins

A Versão 2.0 introduziu a arquitetura de plugin que é usada para quase todas as funcionalidades customizáveis da Smarty. Isto inclui:

- funções
- modificadores
- funções de bloco
- funções de compilador
- prefiltros
- posfiltros
- filtros de saída
- recursos
- inserir

Com a exceção de recursos, a compatibilidade com a forma antiga de funções de manipulador de registro via register_* API é preservada. Se você não usou o API mas no lugar disso modificou as variáveis de classe `$custom_funcs`, `$custom_mods`, e outras diretamente, então você vai precisar ajustar seus scripts para ou usar API ou converter suas funcionalidade customizadas em plugins.

Como os Plugins Funcionam

Plugins são sempre lidos quando requisitados. Apenas os modificadores específicos, funções, recursos, etc convocados em scripts de template serão lidos. Além disso, cada plugin é lido apenas uma vez, mesmo se você tem várias instâncias diferentes da Smarty rodando na mesma requisição.

Pre/posfiltros e filtros de saída são uma parte de um caso especial. Visto que eles não são mencionados nos templates, eles devem ser registrados ou lidos explicitamente via funções de API antes do template ser processado. A ordem em que multiplos filtros do mesmo tipo são executados dependem da ordem em que eles são registrados ou lidos.

O diretório de plugins pode ser uma string contendo um caminho ou um array contendo multiplos caminhos. Para instalar um plugin, simplesmente coloque-o em um dos diretórios e a Smarty irá usá-lo automaticamente.

Convenções de Aparência

Arquivos e funções de Plugin devem seguir uma convenção de aparência muito específica a fim de ser localizada pela Smarty.

Os arquivos de plugin devem ser nomeados da sequinte forma:

```
tipo.nome.php
```

Onde `tipo` é um dos seguintes tipos de plugin:

- function

- modifier
- block
- compiler
- prefilter
- postfilter
- outputfilter
- resource
- insert

E `nome` seria um identificador válido (letras, números, e underscores apenas).

Alguns exemplos: `function.html_select_date.php`, `resource.db.php`, `modifier.spacify.php`.

As funções de plugin dentro dos arquivos do plugin devem ser nomeadas da seguinte forma:

```
smarty_tipo, _nome
```

O significado de `tipo` e `nome` são os mesmos de antes.

A Smarty mostrará mensagens de erro apropriadas se o arquivo de plugins que é necessário não é encontrado, ou se o arquivo ou a função de plugin estão nomeadas inadequadamente.

Escrevendo Plugins

Os Plugins podem ser ou lidos pela Smarty automaticamente do sistema de arquivos ou eles podem ser registrados no tempo de execução via uma das funções de API register_* . Eles podem também ser com o uso da função API unregister_* .

Para os plugins que são registrados no tempo de execução, o nome da(s) função(ões) de plugin não têm que seguir a convenção de aparência.

Se um plugin depende de alguma funcionalidade fornecida por um outro plugin (como é o caso com alguns plugins embutidos com a Smarty), então a forma apropriada para ler o plugin necessário é esta:

```
require_once $smarty->_get_plugin_filepath('function', 'html_options');
```

Como uma regra geral, o objeto da Smarty é sempre passado para os plugins como o último parâmetro (com duas exceções: modificadores não passam o objeto da Smarty em tudo e blocks passam *`&$repeat`* depois do objeto da Smarty para manter compatibilidade a antigas versões da Smarty).

Funções de Template

```
void smarty_function_name($params, &$smarty);

array $params;
object &$smarty;
```

Todos os atributos passados para as funções de template a partir do template estão contidas em *`$params`* como um array associativo. Ou acessa esses valores diretamente i.e `$params['start']` ou usa `extract($params)` para importá-los para dentro da tabela símbolo.

A saída (valor de retorno) da função será substituída no lugar da tag da função no template (a função `fetch`, por exemplo). Alternativamente, a função pode simplesmente executar alguma outra tarefa sem ter alguma saída (a função `assign`).

Se a função precisa passar valores a algumas variáveis para o template ou utilizar alguma outra funcionalidade fornecida com a Smarty, ela pode usar o objeto *`$smarty`* fornecido para fazer isso.

Veja também: register_function(), unregister_function().

Example 16.1. função de plugin com saída

```
<?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('Yes',
                     'No',
                     'No way',
                     'Outlook not so good',
                     'Ask again soon',
                     'Maybe in your reality');

    $result = array_rand($answers);
    return $answers[$result];
}
?>
```

que pode ser usada no template da seguinte forma:

```
Pergunta: Nós sempre teremos tempo para viajar?
Resposta: {eightball}.
```

Example 16.2. função de plugin sem saída

```
<?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);
}
?>
```

Modifiers

Modificadores são funções que são aplicadas a uma variável no template antes dela ser mostrada ou usada em algum outro contexto. Modificadores podem ser encadeados juntos.

```
mixed smarty_modifier_name($value, $param1);

mixed $value;
[mixed $param1, ...];
```

O primeiro parâmetro para o plugin midificador é o valor em que o modificador é suposto operar. O resto dos parâmetros podem ser opcionais, dependendo de qual tipo de operação é para ser executada.

O modificador deve retornar o resultado de seu processamento.

Veja também: register_modifier(), unregister_modifier().

Example 16.3. Plugin modificador simples

Este plugin basiamente é um alias de uma função do PHP. Ele não tem nenhum parâmetro adicional.

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     modifier.capitalize.php
 * Type:     modifier
 * Name:     capitalize
 * Purpose:  capitalize words in the string
 * -------------------------------------------------------------
 */
function smarty_modifier_capitalize($string)
{
    return ucwords($string);
}
?>
```

Example 16.4. Plugin modificador mais complexo

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     modifier.truncate.php
 * Type:     modifier
 * Name:     truncate
 * Purpose:  Truncate a string to a certain length if necessary,
 *           optionally splitting in the middle of a word, and
 *           appending the $etc string.
 * -------------------------------------------------------------
 */
function smarty_modifier_truncate($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;
}
?>
```

Block Functions

```
void smarty_block_name($params, $content, &$smarty);

array $params;
mixed $content;
object &$smarty;
```

Funções de Block são funções da forma: {func} .. {/func}. Em outras palavras, ele enclausura um bloco de template e opera no conteúdo deste bloco. Funções de Block tem precedência sobre funções customizadas com mesmo nome, assim, você não pode ter ambas, função customizável {func} e função de Bloco {func} .. {/func}.

Por definição a implementação de sua função é chamada duas vezes pela Smarty: uma vez pela tag de abertura, e outra pela tag de fechamento (veja `&$repeat` abaixo para como mudar isto).

Apenas a tag de abertura da função de bloco pode ter atributos. Todos os atributos passados para as funções de template estão contidos em *`$params`* como um array associativo. Você pode ou acessar esses valores diretamente, i.e. `$params['start']` ou usar `extract($params)` para importá-los para dentro da tabela símbolo. Os atributos da tag de abertura são também acessíveis a sua função quando processando a tag de fechamento.

O valor da variável *`$content`* depende de que se sua função é chamada pela tag de fechamento ou de abertura. Caso seja a de abertura, ele será `null`, se for a de fechamento o valor será do conteúdo do bloco de template. Note que o bloco de template já terá sido processado pela Smarty, então tudo que você receberá é saída do template, não o template original.

O parâmetro *`&$repeat`* é passado por referência para a função de implementação e fornece uma possibilidade para ele controlar quantas vezes o bloco é mostrado. Por definição *`$repeat`* é `true` na primeira chamada da block-function (a tag de abertura do bloco) e `false` em todas as chamadas subsequentes à função de bloco (a tag de fechamento do bloco). Cada vez que a implementação da função retorna com o *`&$repeat`* sendo true, o conteúdo entre {func} .. {/func} é avaliado e a implementação da função é chamada novamente com o novo conteúdo do bloco no parâmetro *`$content`*.

Se você tem funções de bloco aninhadas, é possível descobrir qual é a função de bloco pai acessando a variável `$smarty->_tag_stack`. Apenas faça um var_dump() nela e a estrutura estaria visível.

See also: register_block(), unregister_block().

Example 16.5. função de bloco

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     block.translate.php
 * Type:     block
 * Name:     translate
 * Purpose:  translate a block of text
 * -------------------------------------------------------------
 */
function smarty_block_translate($params, $content, &$smarty)
{
    if (isset($content)) {
        $lang = $params['lang'];
        // do some intelligent translation thing here with $content
        return $translation;
    }
}
```

Funções Compiladoras

Funções compiladoras só são chamadas durante a compilação do template. Elas são úteis para injeção de código PHP ou conteúdo estático time-sensitive dentro do template. Se há ambos, uma função compiladora e uma função customizável registrada sob o mesmo nome, a função compiladora tem precedência.

```
mixed smarty_compiler_name($tag_arg, &$smarty);

string $tag_arg;
object &$smarty;
```

À função compiladora são passados dois parâmetros: a tag string de argumento da tag - basicamente, tudo a partir do nome da função até o delimitador de fechamento, e o objeto da Smarty. É suposto que retorna o código PHP para ser injetado dentro do template compilado.

See also register_compiler_function(), unregister_compiler_function().

Example 16.6. função compiladora simples

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     compiler.tplheader.php
 * Type:     compiler
 * Name:     tplheader
 * Purpose:  Output header containing the source file name and
 *           the time it was compiled.
 * -------------------------------------------------------------
 */
function smarty_compiler_tplheader($tag_arg, &$smarty)
{
    return "\necho '" . $smarty->_current_file . " compiled at " . date('Y-m-d H:M
}
?>
```

Esta função pode ser chamada em um template da seguinte forma:

```
{* esta função é executada somente no tempo de compilação *}
{tplheader}
```

O código PHP resultante no template compilado seria algo assim:

```
<php
echo 'index.tpl compiled at 2002-02-20 20:02';
?>
```

Prefiltros/Posfiltros

Plugins Prefilter e postfilter são muito similares em conceito; onde eles diferem é na execução -- mais precisamente no tempo de suas execuções.

```
string smarty_prefilter_name($source, &$smarty);

string $source;
object &$smarty;
```

Prefilters são usados para processar o fonte do template imediatamente antes da compilação. O primeiro parâmetro da função de prefilter é o fonte do template, possivelmente modificado por alguns outros prefilters. O Plugin é suposto retornar o fonte modificado. Note que este fonte não é salvo em lugar nenhum, ele só é usado para a compilação.

```
string smarty_postfilter_name($compiled, &$smarty);

string $compiled;
object &$smarty;
```

Postfilters são usados para processar a saída compilada do template (o código PHP) imediatamente após a compilação ser feita e antes do template compilado ser salvo no sistema de arquivo. O primeiro parâmetro para a função postfilter é o código do template compilado, possivelmente modificado por outros postfilters. O plugin é suposto retornar a versão modificada deste código.

Example 16.7. Plugin prefilter

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     prefilter.pre01.php
 * Type:     prefilter
 * Name:     pre01
 * Purpose:  Convert html tags to be lowercase.
 * -------------------------------------------------------------
 */
 function smarty_prefilter_pre01($source, &$smarty)
 {
     return preg_replace('!<(\w+)[^>]+>!e', 'strtolower("$1")', $source);
 }
?>
```

Example 16.8. Plugin postfilter

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     postfilter.post01.php
 * Type:     postfilter
 * Name:     post01
 * Purpose:  Output code that lists all current template vars.
 * -------------------------------------------------------------
 */
 function smarty_postfilter_post01($compiled, &$smarty)
 {
     $compiled = "<pre>\n<?php print_r(\$this->get_template_vars()); ?>\n</pre>" .
     return $compiled;
 }
?>
```

Filtros de saída

Filtros de saída operam na saída do template, depois que o template é lido e executado, mas antes a saída é mostrada.

```
string smarty_outputfilter_name($template_output, &$smarty);

string $template_output;
object &$smarty;
```

O primeiro parâmetro para a função do filtro de saída é a saída do template que precisa ser processada, e o segundo parâmetro é a instância da Smarty invocando o plugin. O plugin deve fazer o precessamento e retornar os resultados.

Example 16.9. output filter plugin

```
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     outputfilter.protect_email.php
 * Type:     outputfilter
 * Name:     protect_email
 * Purpose:  Converts @ sign in email addresses to %40 as
 *           a simple protection against spambots
 * -------------------------------------------------------------
 */
 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);
 }
```

Recursos (Resources)

Os plugins de Recursos são como uma forma genérica de fornecer códigos fontes de template ou componentes de script PHP para a Smarty. Alguns exemplos de recursos: banco de dados, LDAP, memória compartilhada, sockets, e assim por diante.

Há um total de 4 funções que precisam estar registradas para cada tipo de recurso. Cada função receberá o recurso requisitado como o primeiro parâmetro e o objeto da Smarty como o último parâmetro. O resto dos parâmetros dependem da função.

```
bool smarty_resource_name_source($rsrc_name, &$source, &$smarty);

string $rsrc_name;
string &$source;
object &$smarty;

bool smarty_resource_name_timestamp($rsrc_name, &$timestamp, &$smarty);

string $rsrc_name;
int &$timestamp;
object &$smarty;

bool smarty_resource_name_secure($rsrc_name, &$smarty);

string $rsrc_name;
object &$smarty;

bool smarty_resource_name_trusted($rsrc_name, &$smarty);

string $rsrc_name;
object &$smarty;
```

A primeira função deve devolver o recurso. Seu segundo parâmetro é uma variável passada por referência onde o resultado seria armazenado. A função deve retornar `true` se ela está apta a devolver com sucesso o recurso e caso contrário retorna `false`.

A segunda função deve devolver a última modificação do recurso requisitado (como um timestamp Unix). O segundo parâmetro é uma variável passada por referência onde o timestamp seria armazenado. A função deve retornar `true` se o timestamp poderia ser determinado com sucesso, e caso contrário retornaria `false`.

A terceira função deve retornar `true` ou `false`, dependendo do recurso requisitado está seguro ou não. Esta função é usada apenas para recursos de template mas ainda assim seria definida.

A quarta função deve retornar `true` ou `false`, dependendo do recurso requisitado ser confiável ou não. Esta função é usada apenas para componentes de script PHP requisitados pelas tags **include_php** ou **insert** com o atributo *`src`*. Entretanto, ela ainda assim mesmo seria definida para os recursos de template.

Veja também: register_resource(), unregister_resource().

Example 16.10. Plugin resource (recurso)

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     resource.db.php
 * Type:     resource
 * Name:     db
 * Purpose:  Fetches templates from a database
 * -------------------------------------------------------------
 */
function smarty_resource_db_source($tpl_name, &$tpl_source, &$smarty)
{
    // do database call here to fetch your template,
    // populating $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)
{
    // faz o banco de dados chamar aqui para preencher $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)
{
    // assume que todos os templates são seguros
    return true;
}

function smarty_resource_db_trusted($tpl_name, &$smarty)
{
    // não usado para templates
}
?>
```

Inserts

Plugins Insert são usados para implementar funções que são invocadas por tags **insert** no template.

```
string smarty_insert_name($params, &$smarty);

array $params;
object &$smarty;
```

O primeiro parâmetro para a função é um array associativo de atributos passados para o insert. Ou acessa esses valores diretamente, i.e. `$params['start']` ou usa `extract($params)` para importá-los para dentro da tabela símbolo.

A função insert deve retornar o resultado que será substituído no lugar da tag **insert** no template.

Example 16.11. Plugin insert

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     insert.time.php
 * Type:     time
 * Name:     time
 * Purpose:  Inserts current date/time according to format
 * -------------------------------------------------------------
 */
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;
}
?>
```

Part IV. Apêndices

Table of Contents

Chapter 17. Localização de Erros

Erros do Smarty/PHP

O Smarty pode obter muitos erros, tais como: atributos de tags perdidos ou nomes de variáveis mal formadas. Se isto acontece, você verá um erro similar ao seguir:

Example 17.1. Erros do 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
```

O Smarty te mostra o nome do template, o número da linha e o erro. Depois disso, o erro consiste do número da linha da classe Smarty em que o erro ocorreu.

Há certos erros que o Smarty não consegue detectar, tais como uma tag de fechamento errada. Estes tipos de erro geralmente acabam gerando erros em tempo de processamento do interpretador de erros do PHP.

Example 17.2. Erros de análise do PHP

```
Parse error: parse error in /path/to/smarty/templates_c/index.tpl.php on line 75
```

Quando você encontra um erro de análise do PHP, o número da linha do erro corresponderá ao script PHP compilado, não o template em si. Normalmente você pode no template localizar o erro de sintaxe. Aqui algumas coisas para você procurar: falta de fechamento de tags para {if}{/if} ou {section}{/section}, ou erro de lógica dentro de uma tag {if}. Se você não conseguir encontrar o erro, talvez seja necessário abrir o arquivo PHP compilado e ir até o número da linha exibido, para saber onde se encontra o erro correspondente no template.

Example 17.3. Other common errors

- ```
  Warning: Smarty error: unable to read resource: "index.tpl" in...
  or
  Warning: Smarty error: unable to read resource: "site.conf" in...
  ```

  - The $template_dir is incorrect, doesn't exist or the file `index.tpl` is not in the `templates/` directory
  - A {config_load} function is within a template (or config_load() has been called) and either $config_dir is incorrent , does not exist or `site.conf` is not in the directory.

- ```
  Fatal error: Smarty error: the $compile_dir 'templates_c' does not exist,
  or is not a directory...
  ```

  Either the $compile_dir is incorrectly set, the directory does not exist, or `templates_c` is a file and not a directory.

- ```
  Fatal error: Smarty error: unable to write to $compile_dir '....
  ```

  The $compile_dir is not writable by the web server. See the bottom of the installing smarty page for permissions.

- ```
  Fatal error: Smarty error: the $cache_dir 'cache' does not exist,
  or is not a directory. in /..
  ```

  This means that $caching is enabled and either; the $cache_dir is incorrectly set, the directory does not exist, or `cache` is a file and not a directory.

- ```
  Fatal error: Smarty error: unable to write to $ca
  ```

  This means that $caching is enabled and the $cache_dir is not writable by the web server. See the bottom of the installing smarty page for permissions.

See also debugging, $error_reporting and trigger_error().
  ```

Chapter 18. Dicas & Truques

Manipulação de Variável Vazia

Há momentos que você quer mostrar um valor padrão para uma variável vazia ao invés de não mostrar nada, tal como mostrar " " para que os planos de fundo de tabelas funcionem corretamente. Muitos usariam uma instrução {if} para fazer isso, mas há um macete que pode ser feito usando-se o modificador de variável *padrão* do Smarty.

Example 18.1. Imprimindo   quando uma variável está vazia

```
{* A forma mais longa *}

{if $titulo eq ""}
 &amp;nbsp;
{else}
 {$titulo}
{/if}


{* A forma mais simples *}

{$titulo|default:"&amp;nbsp;"}
```

Manipulação do valor padrão de uma Variável

Se uma variável é usada freqüentemente em seus templates, aplicar o modificador de variável *padrão* toda vez pode se tornar algo muito desagradável. Você pode evitar isto atribuindo um valor padrão para a variável usando a função assign.

Example 18.2. Atribuindo o valor padrão para uma variável de template

```
{* coloque isto em algum lugar no topo de seu template *}
{assign var="titulo" value=$titulo|default:"sem título"}

{* Se o $titulo estava vazio, ele agora contém o valor "sem titulo" quando você ex
{$titulo}
```

Passando a variável titulo para o template de cabeçalho

Quando a maioria de seus templates usam os mesmos cabeçalhos e mesmos rodapés, é comum dividi-los um em cada template e então incluí-los. Mas o que fazer se o cabeçalho precisa ter um titulo diferente, dependendo de que página ele está vindo? Você pode passar o titulo para o cabeçalho quando ele é incluído.

Example 18.3. Passando a variável titulo para o template de cabeçalho

```
paginaprincipal.tpl
------------

{include file="cabecalho.tpl" titulo="Página Principal"}
{* O conteúdo do template vem aqui *}
{include file="rodape.tpl"}


arquivos.tpl
------------

{config_load file="pagina_arquivos.conf"}
{include file="cabecalho.tpl" titulo=#tituloPaginaArquivos#}
{* O conteúdo do template vem aqui *}
{include file="rodape.tpl"}


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


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

Quando a página for extraída, o título da "Página Principal" é passado ao template 'cabecalho.tpl', e será imediatamente usado como título da página. Quando a página de arquivos é extraída, o título muda para "Arquivos". No que no exemplo de arquivos, nós estamos usando uma variável que vem do arquivo 'pagina_arquivos.conf' ao invés de uma variável definida no código. Note também que "BC News" é mostrado somente se a variável $titulo não conter valor algum, isto é feito usando-se o modificador de variáveis *padrão*.

Datas

Em geral, sempre envie datas ao Smarty no formato timestamp. Deste modo o desginer do template pode usar o modificador date_format para ter um controle total sobre a formatação da data, e também facilita a comparação de datas se necessário.

Nota: No Smarty 1.4.0, você pode enviar datas ao Smarty no formato unix timestamp, mysql timestamp, ou qualer outra data que possa ser lida pela função strtotime().

Example 18.4. usando date_format

```
{$startDate|date_format}
```

Irá mostrar:

```
Jan 4, 2001
```

```
{$startDate|date_format:"%Y/%m/%d"}
```

Irá mostrar:

```
2001/01/04
```

```
{if $data1 < $data2}
   ...
{/if}
```

Quando se está usando {html_select_date} em um template, o programador normalmente vai querer converter a saída de um formulário de volta para o formato timestamp. Abaixo está uma função que irá ajudá-lo à fazer isto.

Example 18.5. Convertendo datas de volta ao formato timestamp

```
<?php
// presume-se que os elementos de seu formulário são chamados de
// startDate_Day, startDate_Month, startDate_Year

$startDate = makeTimeStamp($startDate_Year,$startDate_Month,$startDate_Day);

function makeTimeStamp($year="",$month="",$day="")
{
 if(empty($year))
  $year = strftime("%Y");
 if(empty($month))
  $month = strftime("%m");
 if(empty($day))
  $day = strftime("%d");

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

WAP/WML

Os templates WAP/WML exigem um cabeçalho com o tipo de conteúdo (Content-Type) PHP para serem passados junto com o template. O modo mais fácil de se fazer isso seria escrever uma função personalizada que envia-se este cabeçalho. Se você está usando cache, isto não irá funcionar, então nós faremos isso usando a tag insert (lembre-se que tags de insert não são guardadas no cache!). Certifique-se de que nada é enviado ao navegador antes do template, caso contrário o cabeçalho não irá funcionar.

Example 18.6. Usando insert para escrever um cabeçalho WML Content-Type

```
<?php
// certifique-se que o apache está configurado para reconhecer extensões .wml!
// coloque esta função em algum lugar de seu aplicativo, ou no arquivo Smarty.addo

function insert_header($params)
{
   // esta função espera o argumento $content
   if (empty($params['content'])) {
       return;
   }
   header($params['content']);
   return;
}
?>
```

seu template do Smarty deve começar com a tag insert, veja o exemplo à seguir:

```
{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/w

<!-- begin new wml deck -->
<wml>
<!-- begin first card -->
<card>
<do type="accept">
<go href="#two"/>
</do>
<p>
Bem-vindo ao WAP com Smarty!
Pressione OK para continuar...
</p>
</card>
<!-- begin second card -->
<card id="two">
<p>
Bem fácil isso, não é?
</p>
</card>
</wml>
```

Templates componentizados

Tradicionalmente, programar templates para suas aplicações é feito da seguinte maneira: Primeiro, você guardar suas variáveis junto com a aplicação PHP, (talvez obtendo-as de consultas à banco de dados). Após,

você instancia seu objeto Smarty, atribui valores às variáveis e mostra o template. Digamos que nós temos um registrador de estoque em nosso template. Nós coletaríamos os dados do estoque em nossa aplicação, e então atribuíriamos valores as variáveis referentes à ele no template e depois exibiríamos o template na tela. Agora não seria legal se você pudesse adicionar este registrador de esto em qualquer aplicação simplesmente incluindo um template nela, e sem se preocupar com a busca dos dados futuramente?

Você pode fazer isto escrevendo um plugin personalizado que obteria o conteúdo e atribuiria ele à uma variável definida no template.

Example 18.7. Template componentizado

```
<?php

// coloque o arquivo "function.load_ticker.php" no diretório plugin

// configura nossa função para pegar os dados do estoque
function fetch_ticker($symbol)
{
   // coloque a lógica que obtém os dados de
   // algum recurso e guarde na variável $ticker_info
   return $ticker_info;
}

function smarty_function_load_ticker($params, &$smarty)
{
   // chama a função
   $ticker_info = fetch_ticker($params['symbol']);

   // atribuite o valor à uma variável no template
   $smarty->assign($params['assign'], $ticker_info);
}
?>




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

{* Smarty *}

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

Nome no estoque: {$ticker.name} Preço no estoque: {$ticker.price}
```

Ofuscando endereços de E-mail

Você já se espantou como seu endereço de E-mail entra em tantas listas de spam? A única forma dos spammers coletarem seu(s) endereço(s) de E-mail(s) é de páginas web. Para ajudar à combater este

problema, você pode fazer seu endereço de E-mail aparecer em javascript misturado em código HTML, e ainda assim ele irá aparecer e funcionar corretamente no navegador. Isto é feito com o plugin chamado 'mailto'.

Example 18.8. Exemplo de ofuscamento de um Endereço de E-mail

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

envia informações para
{mailto address=$EmailAddress encode="javascript" subject="Olá"}
```

Nota técnica

Este método não é 100% a prova de falha. Um spammer poderia criar um programa para coletar o e-mail e decodificar estes valores, mas é muito pouco provável.

Chapter 19. Recursos

A homepage do Smarty está localizada em http://www.smarty.net/. Você pode entrar na lista de discussão enviando um e-mail para ismarty-discussion-subscribe@googlegroups.com. O arquivo da lista de discussão pode ser visualizado em http://groups.google.com/group/smarty-discussion.

Chapter 20. BUGS

Verifique o arquivo BUGS que vem com a última distribuição do Smarty, ou verifique a seção BUGS do website.