Smarty 教 程

目　　录

一、 模板设计

1. 基本语法

Comments [注释]

模板注释被*号包围,例如｛* this is a comment *｝。

Functions [函数]

每一个 smarty 标签输出一个变量或者调用某种函数.

在定界符内函数和变量将被处理和输出.

例如：

```
{funcname attr1="val" attr2="val"}.
{$array}
```

Attributes [属性]

大多数函数都带有自己的属性以便于明确说明或者修改他们的行为.。

smarty 函数的属性很像 HTML 中的属性.。

静态数值不需要加引号，但是字符串建议使用引号。

如果用变量作属性，它们也不能加引号。

一些属性用到了布尔值(真或假)，它们不需要加引号，可以是 true，on，yes 或者 false，off，no。

例如：

```
{include file="header.tpl"}
{include file=$includeFile}
{include file=#includeFile#}
{html_select_date display_days=yes}
```

Embedding Vars in Double Quotes [双引号里值的嵌入]

Smarty 可以识别嵌入在双引号中的只包含数字、字母、下划线和中括号[]的变量。对于其他的符号（句号、对象相关的，等等）必须用两个`(反引号，此符号和‘ ~ '在同一个键上，一般在 ESC 键下面一个键上)包住。

例如：

```
{func var="test $foo test"}          识别为 $foo
{func var="test $foo_bar test"}      识别为 $foo_bar
{func var="test $foo[0] test"}       识别为 $foo[0]
{func var="test $foo[bar] test"}     识别为 $foo[bar]
{func var="test $foo.bar test"}      识别为 $foo (而不是 $foo.bar)
{func var="test `$foo.bar` test"}    识别为 $foo.bar （用反引号"`"标记）
```

Math[数学运算]

数学运算可以直接应用于模版标记中的变量。

例如：

```
{$foo+1}
{$foo*$bar}
{$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`"}
```

2. 变量

Smarty 有几种不同类型的变量，变量的类型取决于被什么符号所修饰。

Smarty 的变量可以直接被输出或者作为函数属性和变量调节器（modifiers）的参数,或者用于内部的条件表达式等等。

如果要输出一个变量,只要用定界符（"{}"）将它括起来就可以。

Variables assigned from PHP [从 PHP 分配的变量]

调用从 PHP 分配的变量需在前加"$"符号.（同 php 一样）

调用模板内的 assign 函数分配的变量也是这样.（也是用$加变量名来调用）

例如：

index.php:

```
$smarty = new Smarty;$smarty->assign('firstname', 'Doug');
$smarty->assign('lastLoginDate', 'January 11th, 2001');
$smarty->display('index.tpl');
```

index.tpl:

```
Hello {$firstname}, glad to see you could make it.
<p>
{assign var="name" value="Bob"}
Your last login was on {$name}.
```

输出结果：

```
Hello Doug, glad to see you could make it.
```

```
<p>
Your last login was on Bob.

Associative arrays[关联数组]
```

Associative arrays [关联数组]

可以通过'.'记号来引用由 PHP 分配的关联数组变量

例如：

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>
```

输出结果：

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

Array indexes[数组下标]

可以通过数组下标来引用数组，就像 PHP 中的语法一样。

例如：

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>
```

输出结果：

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

Objects[对象]

可以通过'->'标记来引用由 PHP 分配的对象的属性。

例如：

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

输出结果：

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

Variables loaded from config files [从配置文件读取的变量]

配置文件中的变量需要通过用两个"#"（嵌套在 smarty 界定符"{}"中）或者是 smarty 的保留变量"$smarty.config."来调用，第二种语法在变量作为属性值并被引号括住的时候非常有用。

{include file="#includefile#"} 这样#includefile#将被当作字符处理，而不表示配置文件变量，但可以这样表示 {include file="`$smarty.config.includefile`"}不要忘了加（`）反引号，或者用 smarty 界定符（"{}"）。

例如：

foo.conf:

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

index.tpl:

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

index.tpl: (改变语法)

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

输出结果： (两种语法有同样的输出结果)

```
<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>
```

{$smarty} reserved variable [{$smarty}保留变量]

Request variables[页面请求变量]

就是 get,post,server,session 等变量，可以访问页面请求的变量，例如 get, post, cookies, server, environment, session .....

例如：

```
{* 显示来自 URL（GET）page 的变量的值  URL (GET) http://www.domain.com/index.php?page=foo *}
{$smarty.get.page}

{* 显示来自一个表单（POST）的变量"page" *}
{$smarty.post.page}

{* 显示 cookie "username"的值  *}
{$smarty.cookies.username}

{* 显示 server 变量"SERVER_NAME" *}
{$smarty.server.SERVER_NAME}

{* 显示系统环境（env）的变量"PATH" *}
{$smarty.env.PATH}

{* 显示 PHP session 变量"id" *}
{$smarty.session.id}

{* 显示来自 get/post/cookies/server/env 的变量"username" *}
{$smarty.request.username}
```

{$smarty.now}

可以通过{$smarty.now}来访问当前的时间戳（timestamp），可以通过 date_format 变量调节器来为特定的输出作处理。

例如：

```
{$smarty.now|date_format:"%Y-%m-%d %H:%M:%S"}
```

{$smarty.const}

可以通过{$smarty.const}来直接访问 PHP 常量

例如：

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

{$smarty.capture}

输出获取通过 {capture}..{/capture}结构可以访问用于{$smarty}的变量.....？？？不明白。。。

The output captured via {capture}..{/capture} construct can be accessed using {$smarty} variable. See section on capture for an example.

{$smarty.config}

{$smarty}变量可以用于引用装入的 config 变量，{$smarty.config.foo}等价于{#foo#}，参考 config_load 部分。

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

{$smarty}变量可以用于引用 'section' 和 'foreach'循环的变量，参考 section 和 foreach 文档。

{$smarty.template}

该变量包含当前正在被处理的模版的名字（文件名）。

3. Variable Modifiers [变量调节器]

变量调节器用于变量，自定义函数和字符串。使用"|"符号分割被处理的变量，自定义函数或字符串和变量调节器。变量调节器由赋予的参数值决定其行为。参数由":"符号分开。

语法：

{被处理的变量，自定义函数或字符串 | 变量调节器名:变量调节器参数 1:变量调节器参数 2:.....}

如果给数组变量应用单值变量的调节，结果是数组的每个值都被调节。如果只想要调节器用一个值调节整个数组，必须在调节器名字前加上@符号。

例如：

{$articleTitle|@count}（这将会在 $articleTitle 数组里输出元素的数目）

capitalize [所有单词首字符大写]

将变量里的所有单词首字大写。

例如：

index.php:

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

index.tpl:

{$articleTitle}
{$articleTitle|capitalize}

输出结果：

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

count_characters[字符计数]

参数位置	类型	是否必须	默认值	变量描述
1	boolean	No	false	决定是否计算空格字符。

```
例如：

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}

OUTPUT 输出:

Cold Wave Linked to Temperatures.
29
33
```

cat[连接字符串]

参数位置	类型	是否必须	默认值	变量描述
1	string	No	empty	将参数值连接到给定的变量后面.

例如：

index.php:

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

index.tpl:

```
{$articleTitle|cat:" yesterday."}
```

输出结果：

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

count_paragraphs[计算段数]

计算变量里的段落数量。

例如：

index.php:

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

index.tpl:

{$articleTitle}
{$articleTitle|count_paragraphs}

输出结果：

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[计算句数]

计算变量里句子的数量（统计变量中". "的数量）。

例如：

index.php:

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

index.tpl:

{$articleTitle}
{$articleTitle|count_sentences}

输出结果：

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

count_words[计算词数]

计算变量里的词数。

例如：

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}
```

输出结果：

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

date_format[格式化日期]

参数位置	类型	是否必须	默认值	变量描述
1	string	No	%b %e, %Y	输出日期的格式。
2	string	No	n/a	输入为空时的默认时间格式。

格式化从 php 函数 strftime()获得的时间和日期。

Unix 或者 mysql 等的时间戳记(parsable by strtotime)都可以传递到 smarty。

设计者可以使用 date_format 完全控制日期格式。

如果传给 date_format 的数据是空的，将使用第二个参数作为时间格式。 ？？？？？？ 不明白。。。

date_format 本质上是 php 的 strftime()函数的一个包装。

当 php 被编译的时候你可以或多或少的依靠系统的 strftime()转换有效的区分符。

可以查看系统手册的有效区分符的全表.

例如：

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"}
```

输出结果：

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

default[默认值]

参数位置	类型	是否必须	默认值	变量描述
1	string	No	empty	变量为空的时候的默认输出。

为空变量设置一个默认值，当变量为空或者未分配的时候,将由给定的默认值替代输出。

例如：

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"}

输出结果：

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

escape[编码]

参数位置	类型	是否必须	可能的值	默认值	变量描述
1	string	No	html,htmlall,url,quotes, hex,hexentity,javascript	html	使用何种编码格式。

用于 html 转码,url 转码,在没有转码的变量上转换单引号,十六进制转码,十六进制美化,或者 javascript 转码。默认是 html 转码。

```
例如：

index.php:

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

index.tpl:
```

```
{$articleTitle}
{$articleTitle|escape}
{$articleTitle|escape:"html"}        {* 转换 & " ' < > 标记 *}
{$articleTitle|escape:"htmlall"}     {* 转换全部 HTML 实体标记 *}
{$articleTitle|escape:"url"}         {* 转换为 url 格式 *}
{$articleTitle|escape:"quotes"}          {* 转换为引用格式 *}
<a href="mailto:{$EmailAddress|escape:"hex"}">{$EmailAddress|escape:"hexentity"}</a>

输出结果：

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

indent[缩进]

参数位置	类型	是否必须	默认值	变量描述
1	integer	No	4	决定缩进多少个字符。
2	string	No	一个空格	使用什么字符来代替缩进。

在每行缩进字符串，默认是 4 个字符。作为可选参数，可以指定缩进字符数。作为第二个可选参数，你可以指定缩进用什么字符代替。使用缩进时如果是在 HTML 中，则需要使用 (空格) 来代替缩进，否则没有效果。

```
例如：

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"}

输出结果：

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 [小写]

将变量字符串小写。

```
例如：

index.php:

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

index.tpl:

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

输出结果：

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

nl2br [换行符替换成
]

所有的换行符将被替换成
.功能同 PHP 中的 nl2br()函数一样。

例如：

index.php:

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

index.tpl:

```
{$articleTitle|nl2br}
```

输出结果：

Sun or rain expected
today, dark tonight

regex_replace [正则替换]

参数位置	类型	是否必须	默认值	变量描述
1	string	Yes	n/a	替换正则表达式
2	string	Yes	n/a	用来替换的文本字符串

寻找和替换正则表达式，欲使用其语法，参考 Php 手册中的 preg_replace()函数。

例如：

index.php:

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

index.tpl:

```
{* 使用空格替换每个回车,tab,和换行符 *}
{$articleTitle}
{$articleTitle|regex_replace:"/[\r\t\n]/":" "}
```

输出结果：

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

replace [替换]

参数位置	类型	是否必须	默认值	变量描述
1	string	Yes	n/a	被替换的文本字符串
2	string	Yes	n/a	用来替换的文本字符串

简单的搜索和替换字符串。

例如：

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"}        {*  将"Garden"替换为"Vineyard" *}
{$articleTitle|replace:" ":" "}                    {*  将" "替换为" " *}
```

输出结果：

```
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 [插空]

参数位置	类型	是否必须	默认值	变量描述
1	string	No	一个空格	在两个字符之间插入的字符（串）

在字符串的每个字符之间插入空格或者其他的字符(串)。

例如：

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:"^^"}
```

输出结果：

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

S^^o^^m^^e^^t^^h^^i^^n^^g^^ ^^W^^e^^n^^t^^ ^^W^^r^^o^^n^^g^^ ^^i^^n^^ ^^J^^e^^t^^ ^^C^^r^^a^^s^^h^^,^^ ^^E^^x^^p^^e^^r^^t^^s^^ ^^S^^a^^y^^.

string_format [字符串格式化]

参数位置	类型	是否必须	默认值	变量描述
1	string	Yes	n/a	使用的格式化方式

是一种格式化字符串的方法，例如格式化为十进制数等等，使用 sprintf 语法格式化。

例如：

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"}
```

输出结果：

```
23.5787446
23.58
24
```

strip [去除(多余空格)]

参数位置	类型	是否必须	默认值	变量描述

1	string	No	一个空格	替换空格的字符串

用一个空格或一个给定字符替换所有重复空格,换行和制表符。

如果要去除模板文本中的区块，使用 strip 函数。

例如：

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:" "}
```

输出结果：

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

strip_tags [去除 html 标签]

去除<和>标签，包括在<和>之间的任何内容。

例如：

index.php:

```
$smarty = new Smarty;
$smarty->assign('articleTitle', "Blind Woman Gets <font face=\"helvetica\">New Kidney</font> from Dad she
```

```
Hasn't Seen in <b>years</b>.");
$smarty->display('index.tpl');

index.tpl:

{$articleTitle}
{$articleTitle|strip_tags}

输出结果：

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

truncate [截取]

参数位置	类型	是否必须	默认值	变量描述
1	integer	No	80	截取字符的数量
2	string	No	...	截取后追加在截取词后面的字符串
3	boolean	No	false	是截取到词的边界（假）还是精确到字符（真）

从字符串开始处截取某长度的字符，默认是 80 个。

也可以指定第二个参数作为追加在截取字符串后面的文本字串，该追加字串被计算在截取长度中。

默认情况下，smarty 会截取到一个词的末尾（不截断摸个单词）。

如果要精确的截取多少个字符，可把第三个参数改为"true"。

```
例如：

index.php:

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

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

index.tpl:

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

输出结果：

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 [大写]

将变量中全部的英文字符转换为大写，如果变量中包含中文字符，则忽略。

```
例如：

index.php:

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

index.tpl:

{$articleTitle}
{$articleTitle|upper}

输出结果：
```

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

wordwrap [行宽约束]

参数位置	类型	是否必须	默认值	变量描述
1	integer	No	80	指定段落(句子)的宽度
2	string	No	\n	使用什么字符约束
3	boolean	No	false	是约束到词的边界（假）还是精确到字符（真）

可以指定段落的宽度(也就是多少个字符一行,超过这个字符数换行)，默认 80。

第二个参数可选，可以指定在约束点使用什么字符(默认是换行符\n)。

默认情况下 smarty 将截取到词尾，如果想精确到设定长度的字符，将第三个参数设为 ture（中文会出现乱码！!)。

例如：

index.php:

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

index.tpl:

```
{$articleTitle}

{$articleTitle|wordwrap:30}

{$articleTitle|wordwrap:20}
```

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

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

输出结果：

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.
```

4. Combining Modifiers [组合使用变量调节器]

对于同一个变量，你可以使用多个变量调节器。它们将按照从左到右的顺序依次组合使用。使用时必须要用"|"字符作为它们之间的分隔符。

```
例如：

index.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:". . ."}


输出结果：

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 F I C I E N C Y .
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. . .
```

5. Built-in Functions [内建函数]

Smarty 自带一些内建函数，内建函数是模板语言的一部分，用户不能创建名称和内建函数一样的自定义函数,也不能修改内建函数。

capture [获取页面输出]

属性名	类型	是否必须	默认值	描述
name	string	no	default	数据采集区域名称
assign	string	no	n/a	数据采集区域在哪分配给变量

capture 函数的作用是捕获模板输出的数据并将其存储到一个变量里,而不是把它们输出到页面.

任何在 {capture name="foo"}和{/capture}之间的数据将被存储到变量$foo 中 ,该变量由 name 属性指定.

在模板中通过 $smarty.capture.foo 访问该变量.

如果没有指定 name 属性,函数默认将使用 "default" 作为参数.

{capture}必须成对出现，即以{/capture}作为结尾,该函数不能嵌套使用.

警告：当希望捕获包含{insert}命令的数据时要特别注意，如果打开了缓存并希望将{insert}命令输出到缓存中，不要捕获该区域的数据。

例如：

```
{* 该例在捕获到内容后输出一行包含数据的表格，如果没有捕获到就什么也不输出 *}
{capture name="banner"}
{include file="get_banner.tpl"}
{/capture}
{if $smarty.capture.banner ne ""}        {* 如果$smarty.capture.banner 不等于（ne）空（""） *}
    <tr>
        <td>
            {$smarty.capture.banner}
        </td>
    </tr>
{/if}
```

config_load [配置加载]

属性	类型	是否必须	缺省值	描述
file	string	Yes	*n/a*	待包含的配置文件的名称
section	string	No	*n/a*	配置文件中待加载部分的名称
scope	string	no	*local*	加载数据的作用域，

属性	类型	是否必须	缺省值	描述
				取值必须为 local, parent 或 global. local 说明该变量的作用域为当前模板. parent 说明该变量的作用域为当前模板和当前模板的父模板(调用当前模板的模板). global 说明该变量的作用域为所有模板.
global	boolean	No	*No*	说明加载的变量是否全局可见，等同于 scope=parent. 注意: 当指定了 scope 属性时，可以设置该属性，但模板忽略该属性值而以 scope 属性为准。

该函数用于从配置文件中加载变量. 更多信息请查看 [配置文件]。

例如：

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

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

配置文件有可能包含多个部分，此时可以使用附加属性 section 指定从哪一部分中取得变量。

注意：配置文件中的 section 和模板内建函数 section 只是命名相同，毫不相干。

foreach,foreachelse [循环处理数组]

属性	类型	是否必须	缺省值	描述
from	string	Yes	*n/a*	待循环数组的名称
item	string	Yes	*n/a*	当前处理元素的变量名称
key	string	No	*n/a*	当前处理元素的键名
name	string	No	*n/a*	该循环的名称，用于

属性	类型	是否必须	缺省值	描述
				访问该循环

foreach 是除 section 之外处理循环的另一种方案(根据不同需要选择不同的方案)。

foreach 用于处理简单数组(数组中的元素的类型一致)，它的格式比 section 简单许多，缺点是只能处理简单数组。

foreach 必须和 /foreach 成对使用，且必须指定 from 和 item 属性。

将 from 属性指定的数组中的数据遍历处理到 item 属性指定的变量中。

参考 foreach (array_expression as $key => $value)

from <=> array_expression；item <=> $value；key <=> $key。

name 属性可以任意指定(字母、数字和下划线的组合)。

foreach 可以嵌套，但必须保证嵌套中的 foreach 名称唯一。

from 属性(通常是数组)决定循环的次数。

foreachelse 语句在 from 属性没有值的时候被执行。(from 属性所指定的值为空时，可用 foreachelse 语句指定——否则-干什么)

foreach 循环有自己的变量名，使用该变量名可以访问该循环. 使用方法为 {$smarty.foreach.foreachname.varname} ,其中 foreachname 即在 foreach 中指定的 name 属性。

例如：

foreach 演示

```
{* 该例将输出数组 $custid 中的所有元素的值 *}
{foreach from=$custid item=curr_id}
    id: {$curr_id}<br>
{/foreach}
```

```
输出结果：

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

foreach 键的演示

{*
数组定义如下:
$smarty->assign("contacts", array(array("phone" => "1", "fax" => "2", "cell" => "3"),
  array("phone" => "555-4444", "fax" => "555-3333", "cell" => "760-1234")));
*}
{* 键就是数组的下标，请参看关于数组的解释 *}

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

输出结果：

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

.index

index 包含当前数组索引，从"0"开始。

```
例如：

<table>
{foreach from=$items key=myId item=i name=foo}
   {if $smarty.foreach.foo.index % 5 == 0}              {* $smarty.foreach.foo.index 对 5 求余 *}
        <tr><th>Title</th></tr>
   {/if}
```

```
   <tr><td>{$i.label}</td></tr>
{/foreach}
</table>
```

.iteration

iteration 包含当前循环的执行次数，总是从 1 开始，每执行一次自加 1。

```
例如：

{* 输出 0|1, 1|2, 2|3, ... 等等 *}
{foreach from=$myArray item=i name=foo}
{$smarty.foreach.foo.index}|{$smarty.foreach.foo.iteration},
{/foreach}
```

.first

当前 foreach 循环第一次执行时 first 被设置成 true。

```
例如：

{* 当循环第一次执行时显示 LATEST , o 否则显示 id *}
<table>
{foreach from=$items key=myId item=i name=foo}
<tr>
   <td>{if $smarty.foreach.foo.first}LATEST{else}{$myId}{/if}</td>
   <td>{$i.label}</td>
</tr>
{/foreach}
</table>
```

.last

当前 foreach 循环执行到最后一遍时 last 被设置成 true.

例如：

```
{* 在列表最后添加水平线 *}
{foreach from=$items key=part_id item=prod name=products}
   <a href="#{$part_id}">{$prod}</a>{if $smarty.foreach.products.last}<hr>{else},{/if}
{foreachelse}
   ... content ...
{/foreach}
```

.show

show 是 foreach 的一个参数. 取值为布尔值 true 或 false. 如果指定为 false 该循环不显示，如果循环指定了 foreachelse 子句，该子句显示与否也取决于 show 的取值.

如何使用？？？

.total

total 用于显示循环执行的次数，可以在循环中或循环执行后调用.

例如：

```
{* 在最后显示行数 *}
{foreach from=$items key=part_id item=prod name=foo}
{$prod.label}<hr/>
{if $smarty.foreach.foo.last}
   <div id="total">{$smarty.foreach.foo.total} items</div>
{/if}
{foreachelse}
 ... something else ...
```

{/foreach}

include [包含文件]

Include 标签用于在当前模板中包含其它模板. 当前模板中的变量在被包含的模板中可用。

属性	类型	是否必须	缺省值	描述
file	string	Yes	*n/a*	待包含的模板文件名
assign	string	No	*n/a*	该属性指定一个变量保存待包含模板的输出
[var ...]	[var type]	No	*n/a*	传递给待包含模板的本地参数，只在待包含模板中有效

必须指定 file 属性，该属性指明模板资源的位置。

如果设置了 assign 属性，该属性对应的变量名用于保存待包含模板的输出，这样待包含模板的输出就不会直接显示了。

例如：

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

{* 模版 body *}

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

include_php [包含 PHP 脚本]

inluce_php 函数用于在模板中包含 php 脚本，如果设置了安全模式，被包含的脚本必须位于 $trusted_dir 路径下。 include_php 函数必须设置 file 属性，该属性指明被包含 php 文件的路径，可以是 $trusted_dir 的相对路径，也可以是绝对路径。

include_php 是解决模板部件化的好方法，它使得 php 代码从模板文件中被分离出来. 举个例子：假设有一个从数据库中动态取出数据用于显示站点导航的模板，你可以将取得数据内容的 php 逻辑部分分离出来保存在一个单独的文件夹下，并在模板开始的位置包含该 php 脚本. 那么就可以在任何地方包含此模板而不用担心之前数据库信息是否已被程序取出.

属性	类型	是否必须	缺省值	描述
file	string	Yes	*n/a*	待包含 php 文件的名称
once	boolean	No	*true*	如果待包含 php 文件已被包含是否仍然包含(类似 php 中的 include_once 函数)
assign	string	No	*n/a*	该属性指定一个变量保存待包含 php 文件的输出

即使是在模板中多次地调用 php 文件，默认情况下它们只被包含一次. 你可以设置 once 属性从而指明每次调用都重新包含该文件. 如果将 once 属性设置为 false，每次调用该文件都将被重新包含.

如果设置了 assign 属性，该属性对应的变量名用于保存待包含 php 的输出，这样待包含 php 文件的输出就不会直接显示了。

在待包含 php 文件中可以通过 $this 访问 smarty 对象.

例如：

load_nav.php

```
<?php

    // 从 mysql 数据库中取得数据，将数据赋给模板变量    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

```
{* 绝对路径或 $trusted_dir 的相对路径 *}
{include_php file="/path/to/load_nav.php"}

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

insert [插入函数]

Insert 函数类似于 inluce 函数，不同之处是 insert 所包含的内容不会被缓存，每次调用该模板都会重新执行该函数.

例如你在页面上端使用一个带有广告条位置的模板，广告条可以包含任何 HTML、图象、FLASH 等混合信息. 因此这里不能使用一个静态的链接 ,同时我们也不希望该广告条被缓存. 这就需要在 insert 函数指定：#banner_location_id# 和 #site_id# 值(从配置文件中取)，同时需要一个函数取广告条的内容信息.

属性	类型	是否必须	缺省值	描述
name	string	Yes	*n/a*	插入函数的名称
assign	string	No	*n/a*	该属性指定一个变量保存待插入函数输出
script	string	No	*n/a*	插入函数前需要先包含的 php 脚本名称
[var ...]	[var type]	No	*n/a*	传递给待插入函数的本地参数

例如：

```
{* 取得 banner 的例子 *}
{insert name="getBanner" lid=#banner_location_id# sid=#site_id#}
```

在此例中 ,我们使用了 getBanner 作为 name 属性 ,同时传递了 #banner_location_id# 和 #site_id# 两个参数. 接下来 Smarty 在你的 php 程序中搜索名为 insert_getBanner() 的函数 , #banner_location_id# 和 #site_id# 的值被组合成一个数组作为函数的第一个参数传递给该函数. 为了避免函数命名混乱 ,所有的 insert 函数都必须以 insert_ 开头. 你的 insert_getBanner() 函数根据传递的参数执行并返回执行的结果. 这些结果就显示在模板中调用该函数的位置. 在此例中 Smarty 调用该函数类似 insert_getBanner(array("lid"=>"12345","sid"=>67890"));并将返回的结果显示在调用的位置.

如果设置了 assign 属性，该属性对应的变量名用于保存待包含函数的输出，这样待包含函数的输出就不会直接显示了.注意：赋给模板变量的输出信息在缓存的时候同样无效.

如果指定了 script 属性，在调用函数并执行前将先包含(只包含一次)script 指定的 php 脚本. 这是为了防止被调用的函数不存在，先调用包含该函数的 php 脚本将避免该情况.

Smarty 对象作为函数的第二个参数被传递，在待包含函数中可以通过 $this 访问并修改 smarty 对象信息.

技术要点: 使模板的一部分不被缓存。如果打开了缓存，即使是在缓存页面中， insert 函数也不会被缓存，每次调用页面它们都会被动态加载， 该特性可以广泛应用于广告条、投票、实时天气预报、搜索结果、反馈信息等区域。

if,elseif,else

Smarty 中的 if 语句和 php 中的 if 语句一样灵活易用，并增加了几个特性以适宜模板引擎. if 必须于 /if 成对出现. 可以使用 else 和 elseif 子句. 可以使用以下条件修饰词：eq、ne、neq、gt、lt、lte、le、gte、ge、is even、is odd、is not even、is not odd、not、mod、div by、even by、odd by、==、!=、>、<、<=、>=. 使用这些修饰词时必须和变量或常量用空格格开.

例如：

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

```
{/if}

{* 使用"或（or）"逻辑 *}
{if $name eq "Fred" or $name eq "Wilma"}
      ...
{/if}

{* 同上 *}
{if $name == "Fred" || $name == "Wilma"}
      ...
{/if}

{* 下面的语法不会工作，元素之间必须用空格隔开 *}
{if $name=="Fred" || $name=="Wilma"}
      ...
{/if}



{* 可使用括号 *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}
      ...
{/if}

{* 可以嵌入 php 函数调用 *}
{if count($var) gt 0}
      ...
{/if}

{* 测试值是偶数（even）还是奇数（odd） *}
{if $var is even}
      ...
{/if}
{if $var is odd}
      ...
{/if}
{if $var is not odd}
      ...
{/if}

{* 测试值是否可被 4 整除 *}
{if $var is div by 4}
      ...
{/if}

{* 测试值是否是 2 的偶数。例如：
0=even, 1=even, 2=odd, 3=odd, 4=even, 5=even, etc. *}
```

```
{if $var is even by 2}
    ...
{/if}

{* 测试值是否是 3 的偶数。例如：
0=even, 1=even, 2=even, 3=odd, 4=odd, 5=odd, etc. *}
{if $var is even by 3}
    ...
{/if}
```

ldelim,rdelim [输出分隔符]

ldelim 和 rdelim 用于输出分隔符，也就是大括号 "{" 和 "}". 模板引擎总是尝试解释大括号内的内容，因此如果需要输出大括号，请使用此方法.

```
例如：

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

输出结果：

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

literal [文本处理]

Literal 标签区域内的数据将被当作文本处理，此时模板将忽略其内部的所有字符信息. 该特性用于显示有可能包含大括号等字符信息的 javascript 脚本. 当这些信息处于 {literal}{/literal} 标签中时，模板引擎将不分析它们，而直接显示.

```
例如：

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

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

    </script>
{/literal}
```

php [嵌入 php 脚本]

php 标签允许在模板中直接嵌入 php 脚本. 是否处理这些语句取决于$php_handling 的设置. 该语句通常不需要使用，当然如果你非常了解此特性或认为必须要用，也可以使用.

```
例如：

{php}
        // including a php script directly
        // from the template.
        include("/path/to/display_weather.php");
{/php}
```

section,sectionelse [遍历数组]

属性目录

index

index_prev

index_next

iteration

first

last

rownum

loop

show

total

属性	类型	是否必须	缺省值	描述
name	string	Yes	*n/a*	该循环的名称
loop	[$variable_name]	Yes	*n/a*	决定循环次数的变量名称
start	integer	No	*0*	循环执行的初始位置. 如果该值为负数，开始位置从数组的尾部算起. 例如 如果数组中有 7 个元素，指定 start 为-2，那么指向当前数组的索引为 5. 非法值(超过了循环

属性	类型	是否必须	缺省值	描述
				数组的下限)将被自动调整为最接近的合法值.
step	integer	No	*1*	该值决定循环的步长. 例如指定 step=2 将只遍历下标为 0、2、4 等的元素. 如果 step 为负值，那么遍历数组的时候从后向前遍历.
max	integer	No	*1*	设定循环最大执行次数.
show	boolean	No	*true*	决定是否显示该循环.

模板的 section 用于遍历数组中的数据. section 标签必须成对出现. 必须设置 name 和 loop 属性。名称可以是包含字母、数字和下划线的任意组合. 可以嵌套但必须保证嵌套的 name 唯一. 变量 loop (通常是数组)决定循环执行的次数. 当需要在 section 循环内输出变量时，必须在变量后加上中括号包含着的 name 变量。 sectionelse 当 loop 变量无值时被执行。

例如：

section 函数演示

```
{* 下面的例子会输出数组变量 $custid 中的所有的值 *}

{section name=customer loop=$custid}        {* 使用数组变量作为设定循环次数 loop 的值 *}
      id: {$custid[customer]}<br>     {* 使用"被遍历数组变量名[section 名(name)]"作为数组遍历的输出 *}
{/section}

输出结果：

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

loop 变量演示

{* loop 变量只决定循环的次数。
    可以访问任何来自 section 内部的变量。
    这个例子假设 $custid, $name 和 $address 数组包含相同数量的值。
    *}
{section name=customer loop=$custid}
      id: {$custid[customer]}<br>
      name: {$name[customer]}<br>
      address: {$address[customer]}<br>
      <p>
{/section}


输出结果：

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

section 名称演示

{* section 的名称可以随便取，这个名称用于 section 内部数据的引用 *}

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

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

嵌套 section 演示

{* 嵌套可以随便嵌入很深，在嵌套的 section 中，可以访问复杂的数据结构，例如多维数组。
   在这个例子中，$contact_type[customer] 是与当前 customer 相关类型的数组。 *}
{section name=customer loop=$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}

输出结果：

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>

section 遍历多维数组演示
```

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

输出结果：

```
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>
```

sectionelse 演示

```
{* 如果 $custid 没有值（为空），sectionelse 将执行 *}
{section name=customer loop=$custid}
      id: {$custid[customer]}<br>
{sectionelse}
      there are no values in $custid.
{/section}
```

Section 循环也有可供调用的变量名. 通过如下方式调用：

```
  {$smarty.section.sectionname.varname}.
```

index

index 用于显示当前循环的索引，从 0 开始(如果指定了 start 属性，那么由该值开始)，每次加 1(如果指定了 step 属性，那么由该值决定).

如果没有指定 step 和 start 属性，此值的作用和 iteration 类似，只不过从 0 开始而已.

```
section 的 index 属性演示

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

输出结果：

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

index_prev

index_prev 用于显示上一个循环索引值. 循环开始时，此值为-1.

```
section 的 index_prev 属性演示

{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{* 供参考：$custid[customer.index] 和 $custid[customer] 是同样的意思 *}
{if $custid[customer.index_prev] ne $custid[customer.index]}
The customer id changed<br>
{/if}
{/section}

输出结果：

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

index_next

index_next 用于显示下一个循环索引值. 循环执行到最后一次时，此值仍然比当前索引值大 1(如果指定了 step，取决于此值).

```
index_next 属性演示

{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{if $custid[customer.index_next] ne $custid[customer.index]}
The customer id will change<br>
{/if}
{/section}

输出结果：

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

iteration

iteration 用于显示循环的次数.

注意：iteration 不像 index 属性受 start、step 和 max 属性的影响，该值总是从 1 开始(index 是从 0 开始的).

rownum 是 iteration 的别名，两者等同.

```
section 的 iteration 属性演示

{section name=customer loop=$custid start=5 step=2}
```

```
current loop iteration: {$smarty.section.customer.iteration}<br>
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{if $custid[customer.index_next] ne $custid[customer.index]}
The customer id will change<br>
{/if}
{/section}
```

输出结果：

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

first

如果当前循环第一次执行，first 被设置为 true.

```
section 的 first 属性演示

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

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

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

输出结果：

```
<table>
```

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

last

如果当前循环执行到最后一次，last 被设置为 true.

```
section 的 last 属性演示

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

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

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

输出结果：

<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 用于显示循环的次数. 该属性是 iteration 的别名，两者等同.

section 的 rownum 属性演示

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

输出结果：

1 id: 1000
2 id: 1001
3 id: 1002

loop

loop 用于显示该循环上一次循环时的索引值. 该值可以用于循环内部或循环结束后.

section 的 loop 属性演示

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

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

输出结果：

0 id: 1000
1 id: 1001
2 id: 1002

There were 3 customers shown above.

show

show 是 section 的参数. show 取值为布尔值 true 或 false. 如果设置为 false，该循环将不显示. 如果指定了 sectionelse 子句，该语句是否显示也取决于该值.

section 的 show 属性演示

```
{* $show_customer_info 来自 PHP 应用的传递，用于管理是否显示 section *}
{section name=customer loop=$custid show=$show_customer_info}
{$smarty.section.customer.rownum} id: {$custid[customer]}<br>
{/section}

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

输出结果：

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

the section was shown.
```

total

total 用于显示循环执行总的次数. 可以在循环中或执行结束后调用此属性.

section 的 total 属性演示

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

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

输出结果：

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

```
There were 3 customers shown above.
```

strip [去处首尾空格和回车]

Web 开发者多次遇到空格和回车影响 HTML 输出的情形(浏览器的"特性")，为了得到特定的结果，因此你不得不在模板里运行所有的标签. 通常在难以理解或难以处理的模板中遇到此问题.

Smarty 在显示前将除区任何位于 {strip}{/strip} 标记中数据的首尾空格和回车. 这样可以保证模板容易理解且不用担心多余的空格导致问题.

技术要点：{strip}{/strip} 不会影响模版变量的内容，参见 strip 变量调节器。

```
strip 标签演示

{* 下面的会被输出为一行 *}
{strip}
<table border=0>
    <tr>
        <td>
            <A HREF="{$url}">
            <font color="red">This is a test</font>
            </A>
        </td>
    </tr>
</table>
{/strip}


输出结果：

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

请注意上例 ,所有行都以 HTML 标签开头结尾. 所有行被组织到一起运行. 如果在行首和行尾有文本的话，它们也会被组织到一起，就有可能得到你不想得到的结果.

6. Custom Functions[自定义函数]

用户可以使用 Smarty 自带的一组自定义函数。

assign [为模板变量赋值]

assign 用于在模板被执行时为模板变量赋值.

属性	类型	是否必须	缺省值	描述
var	string	Yes	*n/a*	被赋值的变量名
value	string	Yes	*n/a*	赋给变量的值

assign 函数演示

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

The value of $name is {$name}.

输出结果：

The value of $name is Bob.

counter [计数]

counter 用于输出一个记数过程. counter 保存了每次记数时的当前记数值. 用户可以通过调节 interval

和 direction 调节该值. 也可以决定是否输出该值. 如果需要同时运行多个计数器，必须为它们指定不同的名称. 如果没有指定名称，模板引擎使用 "default" 作为缺省值.

如果指定了 "assign" 这个特殊属性，该计数器的输出值将被赋给由 assign 指定的模板变量，而不是直接输出.

属性	类型	是否必须	缺省值	描述
name	string	No	*default*	计数器的名称
start	number	No	*1*	记数器初始值
skip	number	No	*1*	记数器间隔、步长
direction	string	No	*up*	记数器方向，(增/减)
print	boolean	No	*true*	是否输出值
assign	string	No	*n/a*	输出值将被赋给模板变量的名称

counter 函数演示

```
{* initialize the count *}
{counter start=0 skip=2 print=false}

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

输出结果：

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

cycle [轮转使用值]

Cycle 用于轮转使用一组值. 该特性使得在表格中交替输出颜色或轮转使用数组中的值变得很容易.

如果需要在模板中使用多个轮转，需要给出唯一的 name 属性.

用户可以设置 print 属性为 false 强制不输出当前值. 该特性可以很方便地略过某个值.

advance 属性用于重复使用某个值. 当该属性设置为 false 时，下次调用该轮转时将输出同样的值.

如果指定了 "assign" 这个特殊属性，该轮转的输出值将被赋给由 assign 指定的模板变量，而不是直接输出.

属性	类型	是否必须	缺省值	描述
name	string	No	*default*	轮转的名称
values	mixed	Yes	*N/A*	待轮转的值，可以是用逗号分隔的列表(请查看 delimiter 属性)或一个包含多值的数组.
print	boolean	No	*true*	是否输出值
advance	boolean	No	*true*	是否使用下一个值(为 false 时使用当前值)
delimiter	string	No	,	指出 values 属性中

属性	类型	是否必须	缺省值	描述
				使用的分隔符，默认是逗号.
assign	string	No	*n/a*	输出值将被赋给模板变量的名称

```
cycle  函数演示

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

输出结果：

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

debug [调试输出]

{debug} 将调式信息输出到页面上. 该函数是否可用取决于 Smarty 的 debug 设置.

该函数在运行时取得数据，因此不能显示使用中的模板，只能显示运行时被赋值的变量.

不过还是可以看到所有模板内当前可用的变量.

属性	类型	是否必须	缺省值	描述
output	string	No	*html*	输出类型，html 或 javascript

eval

eval 按处理模板的方式计算取得变量的值. 该特性可用于在配置文件中的标签/变量中嵌入其它模板标签/变量.

如果指定了 "assign" 这个特殊属性，该函数的输出值将被赋给由 assign 指定的模板变量，而不是直接输出.

技术要点: 待求值处理的变量被当作模板来处理. 它们和模板一样遵循同样的结构和安全特性.

技术要点: 待求值处理的变量每次调用时被重编译，不保存编译版本! 但当打开缓冲设置时，该输出会被其它模板缓冲.

属性	类型	是否必须	缺省值	描述
var	mixed	Yes	*n/a*	待求值的变量(或字符串)
assign	string	No	*n/a*	输出值将被赋给模板变量的名称

eval 函数演示

```
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}

输出结果：

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 [取文件、HTTP、FTP]

fetch 用于从本地文件系统、HTTP 或 FTP 上取得文件并显示文件的内容. 如果文件名称以"http://"开头，将取得该网站页面并显示. 如果文件名称以"ftp://"开头，将从 ftp 服务器取得该文件并显示.

如果指定了 "assign" 这个特殊属性，该函数的输出值将被赋给由 assign 指定的模板变量，而不是直接输出.(Smarty 版本 1.5.0 新特性)

技术要点：该函数不支持 HTTP 重定向，如果要取得 web 默认页，比如想取得 www.domain.com 的主页资料，但是不知道主页的具体名称，可能是 index.php 或 index.htm 或 default.php 等等，可以直接

使用该站点的 url，记得在 url 结尾处加上反斜线.

技术要点：如果模板的安全设置打开了，当取本地文件时只能取位于定义为安全文件夹下的资料.

($secure_dir)

属性	类型	是否必须	缺省值	描述
file	string	Yes	*n/a*	待请求的文件，http 或 ftp 方式.
assign	string	No	*n/a*	输出值将被赋给模板变量的名称

```
fetch  函数演示

{*  在模版中包括 javascript *}
{fetch file="/export/httpd/www.domain.com/docs/navbar.js"}

{*  在模版中嵌入来自其它 web 站点的天气文本  *}
{fetch file="http://www.myweather.com/68502/"}

{*  通过 ftp 取得新闻新闻标题文件  *}
{fetch file="ftp://user:password@ftp.domain.com/path/to/currentheadlines.txt"}

{*  分配已取得的内容到模版变量  *}
{fetch file="http://www.myweather.com/68502/" assign="weather"}
{if $weather ne ""}
      <b>{$weather}</b>
{/if}
```

html_checkboxes [html 复选框]

自定义函数 html_checkboxes 根据给定的数据创建复选按钮组. 该函数可以指定哪些元素被选定. 要么

必须指定 values 和 ouput 属性，要么指定 options 替代. 所有的输出与 XHTML 兼容.

上表未提到的其它参数在 <input> 标签中以"名称/属性"对的方式显示.

属性	类型	是否必须	缺省值	描述
name	string	No	*checkbox*	复选按钮组的名称
values	array	Yes, 或指定 options 属性	*n/a*	包含复选按钮组值的数组
output	array	Yes, 或指定 options 属性	*n/a*	包含复选按钮组显示值的数组
selected	string/array	No	*empty*	已选定的元素或元素数组
options	associative array	Yes,或指定 values 属性	*n/a*	包含值和显示的关联数组
separator	string	No	*empty*	分隔每个复选按钮的字符串
labels	boolean	No	*true*	是否为每个复选按钮添加 <label> 标签

html_checkboxes 函数演示

index.php:

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

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

index.tpl:

{html_checkboxes values=$cust_ids checked=$customer_id output=$cust_names separator="<br />"}

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="<br />"}

输出结果：

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

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

html_image [html 图片]

自定义函数 html_image 产生一个图象的 HTML 标签. 如果没有提供高度和宽度值 ,将根据图象的实际大小自动取得.

basedir 是相对图象路径的基路径. 如果没有给出该属性，将依据 WEB 服务器的根路径(环境变量 DOCUMENT_ROOT)为准. 如果模板的安全设置打开了，图象的位置必须位于为安全文件夹下.

href 是图象链接指向的位置. 如果设置了该属性，图象两侧将被加上超级链接标签，形成一个图象链接.

技术要点: html_image 需要访问磁盘以获取图象的尺寸. 如果不使用缓冲，为了优化性能，一般情况下建议使用静态图象标签而避免使用 html_iamge.

属性	类型	是否必须	缺省值	描述
file	string	Yes	*n/a*	图象文件的名称或路径
border	string	No	*0*	图象边框大小
height	string	No	*actual image height*	显示图象高度
width	string	No	*actual image width*	显示图象宽度
basedir	string	no	*web server doc root*	图象文件位置的相对路径
alt	string	no	*""*	可选图象描述(鼠标指向图象或图象文件不存在时显示的字符串信息)
href	string	no	*n/a*	图象链接到的地址

html_options [html 下拉列表]

自定义函数 html_options 根据给定的数据创建选项组. 该函数可以指定哪些元素被选定. 要么必须指定 values 和 ouput 属性，要么指定 options 替代.

如果给定值是数组，将作为 OPTGROUP 处理，且支持递归. 所有的输出与 XHTML 兼容.

如果指定了可选属性 name ,该选项列表将将被置于<select name="groupname"></select>标签对中. 如果没有指定，那么只产生选项列表.

上表未提到的其它参数在 <select> 标签中以"名称/属性"对的方式显示. 如果没有指定可选属性 name 这些参数将被忽略.

属性	类型	是否必须	缺省值	描述
values	array	Yes, unless using options attribute	*n/a*	包含下拉列表各元素值的数组
output	array	Yes, unless using options attribute	*n/a*	包含下拉列表各元素显示值的数组
selected	string/array	No	*empty*	已选定的元素或元素数组
options	associative array	Yes, unless using values and output	*n/a*	包含值和显示的关联数组
name	string	No	*empty*	下拉菜单的名称

html_options 函数演示

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

index.tpl:

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

输出结果：  (both examples)

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

html_radios [html 单选框]

自定义函数 html_radios 根据给定的数据创建单选按钮组. 该函数可以指定哪个元素被选定. 要么必须指定 values 和 ouput 属性, 要么指定 options 替代. 所有的输出与 XHTML 兼容.

上表未提到的其它参数在 <input> 标签中以"名称/属性"对的方式显示.

属性	类型	是否必须	缺省值	描述
name	string	No	*radio*	单选按钮列表的名称
values	array	Yes, 或指定 options 属性	*n/a*	包含单选按钮值的数组
output	array	Yes, 或指定 options 属性	*n/a*	包含单选按钮显示值的数组
checked	string	No	*empty*	已选定的元素
options	associative array	Yes, 或指定 values 属性	*n/a*	包含值和显示的关联数组
separator	string	No	*empty*	分隔每个单选按钮的字符串

html_radios 函数演示

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="<br />"}
```

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 />"}
```

输出结果： (both examples)

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

html_select_date [html 日期下拉列表]

自定义函数 html_select_date 用于创建日期下拉菜单. 它可以显示任意年月日.

属性	类型	是否必须	缺省值	描述
prefix	string	No	Date_	变量名称前缀
time	timestamp/YYYY-MM-DD	No	UNIX 时间戳或 年-月-日	使用时间类型 (data/time)

属性	类型	是否必须	缺省值	描述
start_year	string	No	年份或与当前年份的相对值	下拉列表中第一个年份 ,或与当前年份的相对值(正/负 几年)
end_year	string	No	同 start_year	下拉列表中最后一个年份 ,或与当前年份的相对值(正/负 几年)
display_days	boolean	No	true	是否显示天
display_months	boolean	No	true	是否显示月
display_years	boolean	No	true	是否显示年
month_format	string	No	%B	月份的表示方法(strftime)
day_format	string	No	%02d	天显示的格式(sprintf)
day_value_format	string	No	%d	天的表示方法(sprintf)
year_as_text	boolean	No	false	是否以文本方式显示年份
reverse_years	boolean	No	false	逆序显示年份
field_array	string	No	null	如果指定了名称 ,选定的区域将以

属性	类型	是否必须	缺省值	描述
				[Day],[Year],[Month] 的形式返回给 PHP(待考)
day_size	string	No	null	如果给定 ,为标签添加大小属性
month_size	string	No	null	如果给定 ,为标签添加大小属性
year_size	string	No	null	如果给定 ,为标签添加大小属性
all_extra	string	No	null	如果给定 ,为所有标签添加附加属性
day_extra	string	No	null	如果给定 ,为标签添加附加属性
month_extra	string	No	null	如果给定 ,为标签添加附加属性
year_extra	string	No	null	如果给定 ,为标签添加附加属性
field_order	string	No	MDY	显示区域的顺序
field_separator	string	No	\n	各区域间输出的分隔字符串

属性	类型	是否必须	缺省值	描述
month_value_format	string	No	%m	月份值的 strftime 表示方法，默认为 %m

html_select_date 函数演示

{html_select_date}

输出结果：

```
<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>

{* 开始和结束年份与当前年份相关 *}
{html_select_date prefix="StartDate" time=$time start_year="-5" end_year="+1" display_days=false}

输出结果： (current year is 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 [html 时间下拉列表]

自定义函数 html_select_time 用于创建时间下拉菜单. 它可以显示任意时分秒.

属性	类型	是否必须	缺省值	描述
prefix	string	No	Time_	变量名称前缀
time	timestamp	No	UNIX 时间戳或年-月-日	使用时间类型(data/time)
display_hours	boolean	No	true	是否显示小时
display_minutes	boolean	No	true	是否显示分钟
display_seconds	boolean	No	true	是否显示秒
display_meridian	boolean	No	true	是否显示正午界(上午/下午)
use_24_hours	boolean	No	true	是否使用 24 小时制
minute_interval	integer	No	1	分钟下拉列表的间隔
second_interval	integer	No	1	秒钟下拉列表的间隔
field_array	string	No	n/a	输出值到该值指定的

属性	类型	是否必须	缺省值	描述
				数组
all_extra	string	No	null	如果给定，为标签添加附加属性
hour_extra	string	No	null	如果给定，为标签添加附加属性
minute_extra	string	No	null	如果给定，为标签添加附加属性
second_extra	string	No	null	如果给定，为标签添加附加属性
meridian_extra	string	No	null	如果给定，为标签添加附加属性

html_select_time 函数演示

{html_select_time use_24_hours=true}

输出结果：

```
<select name="Time_Hour">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09" selected>09</option>
```

```
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
</select>
<select name="Time_Minute">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20" selected>20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
```

```
<option value="30">30</option>
<option value="31">31</option>
<option value="32">32</option>
<option value="33">33</option>
<option value="34">34</option>
<option value="35">35</option>
<option value="36">36</option>
<option value="37">37</option>
<option value="38">38</option>
<option value="39">39</option>
<option value="40">40</option>
<option value="41">41</option>
<option value="42">42</option>
<option value="43">43</option>
<option value="44">44</option>
<option value="45">45</option>
<option value="46">46</option>
<option value="47">47</option>
<option value="48">48</option>
<option value="49">49</option>
<option value="50">50</option>
<option value="51">51</option>
<option value="52">52</option>
<option value="53">53</option>
<option value="54">54</option>
<option value="55">55</option>
<option value="56">56</option>
<option value="57">57</option>
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Second">
<option value="00">00</option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
```

```
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23" selected>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 [html 制表]

自定义函数 html_table 将数组中的数据填充到 HTML 表格中. cols 属性决定表格有多少列. table_attr , tr_attr 和 td_attr 属性决定表格中 tr 和 td 标签的一些附加属性. 如果 tr_attr 和 td_attr 属性值为数组，将轮流使用其中的值. 如果指定了 trailpad 属性，将在表尾最后一行附加一些数据.

属性	类型	是否必须	缺省值	描述
loop	array	Yes	*n/a*	待遍历的数组
cols	integer	No	*3*	表格的列数目
table_attr	string	No	*border="1"*	表格的属性
tr_attr	string	No	*empty*	行标签属性(或轮转数组)
td_attr	string	No	*empty*	列标签属性(或轮转数组)
trailpad	string	No	* *	最后一行附加的数据(如果有的话)
hdir	string	No	*right*	行的对齐方式，可能的值为 *left* 或

属性	类型	是否必须	缺省值	描述
				right
vdir	string	No	*down*	列的对齐方式，可能的值为 *up* 或 *down*

html_table 函数演示

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}
```

输出结果：

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

math [数学运算]

math 允许模板设计者在模板中进行数学表达式运算. 在表达式中可以使用任何数值类型的变量，结果在 math 标签的位置输出. 表达式中使用的变量被当做参数传递给函数，可以是模板变量或静态值. 目前可以使用的运算符有:+, -, /, *, abs, ceil, cos, exp, floor, log, log10, max, min, pi, pow, rand, round, sin, sqrt, srans 和 tan .关于数学函数的详细信息，请查看 PHP 文档.

如果指定了 "assign" 这个特殊属性，该函数的输出值将被赋给由 assign 指定的模板变量，而不是直接输出.

技术要点: 由于使用了 php 的 eval() 函数，math 函数的执行效率不高. 在 PHP 中做数学运算效率会更高一些，因此要尽可能在 PHP 中做数学运算，将结果赋给模板变量. 类似在 section 循环，应明确避免反复调用 math 函数.

属性	类型	是否必须	缺省值	描述
equation	string	Yes	*n/a*	待执行的表达式
format	string	No	*n/a*	结果的格式(遵从 sprintf 函数)
var	numeric	Yes	*n/a*	表达式变量值
assign	string	No	*n/a*	输出值将被赋给模板变量的名称
[var ...]	numeric	Yes	*n/a*	表达式变量值

```
math 函数演示

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

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

输出结果：

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#}

输出结果：

100

{* 可以使用圆括号 *}

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

输出结果：

6

{* 可以为输出格式提供格式参数 *}

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

输出结果：

9.44
```

mailto

mailto 自动生成电子邮件链接 ,并根据选项决定是否对地址信息编码. 电子邮件地址编码使得网络嗅探程序难以收集到电子邮件地址信息.

技术要点: 尽管可以使用 hex 进制编码，但 javascript 已经算得上是很彻底的编码形式了.

属性	类型	是否必须	缺省值	描述
address	string	Yes	*n/a*	电子邮件地址
text	string	No	*n/a*	邮件链接上显示的文本，默认为电子邮件地址
encode	string	No	*none*	编码方式，可选值为 none，hex 或 javascript
cc	string	No	*n/a*	邮件抄送地址，多条地址信息以逗号分隔
bcc	string	No	*n/a*	邮件暗送地址，多条地址信息以逗号分隔
subject	string	No	*n/a*	邮件主题
newsgroups	string	No	*n/a*	发送到新闻组的地址，多条地址信息以逗号分隔
followupto	string	No	*n/a*	追踪地址信息，多条信息以逗号分隔
extra	string	No	*n/a*	其它需要传递给链接

属性	类型	是否必须	缺省值	描述
				的信息，如 css 样式

mailto 函数演示

{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"'}

输出结果：

me@domain.com
send me some mail
<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>
me@d&
#x6f;main.com
me@domain.com
me@domain.com
me@domain.com

popup_init

popup 函数整合了 overLib(用于弹出窗口的函数库) 库. 这些函数用于上下文敏感信息如帮助窗口或工具提示. 如果准备使用 popup 函数，在页首必须先调用 popup_init 函数. overLib 由 Erik Bosrup 开发，其主页位于 http://www.bosrup.com/web/overlib/.

在 Smarty 2.1.2 版中没有带 overLib 库. 必须下载该库，将 overlib.js 文件置于文档根目录下，调用

popup_init 的时候将该文件的相对路径作为参数"src"传递.

```
popup_init 函数演示

{* popup_init 必须在页面顶部被调用一次 *}
{popup_init src="/javascripts/overlib.js"}
```

popup [创建 javascript 弹出窗口]

popup 用于创建 javascript 弹出窗口.

属性	类型	是否必须	缺省值	描述
text	string	Yes	*n/a*	弹出窗口中要显示的内容，文本或超文本
trigger	string	No	*onMouseOver*	触发弹出窗口的条件，可选择 onMouseOver(鼠标经过)或 onClick(鼠标单击)
sticky	boolean	No	*false*	弹出窗口始终显示，直到关闭
caption	string	No	*n/a*	标题文本内容
fgcolor	string	No	*n/a*	弹出窗口内部颜色
bgcolor	string	No	*n/a*	弹出窗口边框颜色

属性	类型	是否必须	缺省值	描述
textcolor	string	No	*n/a*	弹出窗口内部文字颜色
capcolor	string	No	*n/a*	弹出窗口标题颜色
closecolor	string	No	*n/a*	设置"关闭"文本信息的颜色
textfont	string	No	*n/a*	设置内容文本使用的字体类型
captionfont	string	No	*n/a*	设置标题文本的字体
closefont	string	No	*n/a*	设置"关闭"文本信息的字体类型
textsize	string	No	*n/a*	设置内容文本使用的字体大小
captionsize	string	No	*n/a*	设置标题文本使用的字体大小
closesize	string	No	*n/a*	设置"关闭"文本信息的字体大小
width	integer	No	*n/a*	sets the width of the box 设置宽度
height	integer	No	*n/a*	sets the height of the box 设置高度

属性	类型	是否必须	缺省值	描述
left	boolean	No	*false*	使弹出窗口位于鼠标左侧
right	boolean	No	*false*	使弹出窗口位于鼠标右侧
center	boolean	No	*false*	使弹出窗口的中间位置和鼠标位置重合
above	boolean	No	*false*	使弹出窗口位于鼠标上侧. 注: 仅在设置了 height 属性时有效
below	boolean	No	*false*	使弹出窗口位于鼠标下侧
border	integer	No	*n/a*	决定弹出窗口的边框厚度
offsetx	integer	No	*n/a*	横向显示位置偏移量
offsety	integer	No	*n/a*	纵向显示位置偏移量
fgbackground	url to image	No	*n/a*	使用背景图片代替背景颜色
bgbackground	url to image	No	*n/a*	使用背景图片代替边

属性	类型	是否必须	缺省值	描述
				框颜色. 注 1: 必须设置 bgcolor 为""，边框颜色才不会显示. 注 2: 当有关闭链接时 ,Netscape 会重新渲染表格，看起来可能会有点问题.
closetext	string	No	*n/a*	自定义关闭链接显示文本
noclose	boolean	No	*n/a*	不显示关闭链接
status	string	No	*n/a*	设置状态栏显示的文本
autostatus	boolean	No	*n/a*	设置弹出窗口状态栏显示的文本为当前窗口状态栏显示的文本. 注: 该设置将覆盖 status 的设置
autostatuscap	string	No	*n/a*	设置状态栏显示的文本为标题栏显示的信息. 注: 该设置将覆盖 status 和

属性	类型	是否必须	缺省值	描述
				autostatus 设置
inarray	integer	No	*n/a*	该属性告诉 overLib 在 ol_text 数组中读该属性指定的索引的元素到 text 中. 该属性可以代替 text 属性
caparray	integer	No	*n/a*	该属性告诉 overLib 在 ol_caps 数组中读该属性指定的索引的元素到 caption 中.
capicon	url	No	*n/a*	弹出前先显示该图象
snapx	integer	No	*n/a*	横向将弹出窗口附着于指定位置
snapy	integer	No	*n/a*	纵向将弹出窗口附着于指定位置
fixx	integer	No	*n/a*	锁定弹出窗口的横向位置. 注: 该设置将覆盖其它横向设置
fixy	integer	No	*n/a*	锁定弹出窗口的纵向位置. 注: 该设置将

属性	类型	是否必须	缺省值	描述
				覆盖其它纵向设置
background	url	No	*n/a*	设置图象作为输出表格区块的背景
padx	integer,integer	No	*n/a*	在背景图象后附加横向空白. 注: 该属性有两个参数
pady	integer,integer	No	*n/a*	在背景图象后附加纵向空白. 注: 该属性有两个参数
fullhtml	boolean	No	*n/a*	允许用户完全控制背景图片上的 HTML. HTML 代码位于"text"属性中
frame	string	No	*n/a*	控制弹出窗口在不同的框架中. 关于此函数更多详细信息，请查阅 overlib 文档.
timeout	string	No	*n/a*	调用特定 javascript 脚本函数，将返回值显示在弹出窗口中.

属性	类型	是否必须	缺省值	描述
delay	integer	No	*n/a*	使得弹出窗口像一个工具提示，窗口将显示到该属性指定的时间(毫秒)
hauto	boolean	No	*n/a*	自动决定弹出窗口位于鼠标的左侧或右侧.
vauto	boolean	No	*n/a*	自动决定弹出窗口位于鼠标的上侧或下侧.

```
popup  函数演示

{* popup_init  必须在页面顶部被调用一次  *}
{popup_init src="/javascripts/overlib.js"}

{*  当移动鼠标经过时，用弹出窗口创建一个链接  *}
<A href="mypage.html" {popup text="This link takes you to my page!"}>mypage</A>

{*  可以在弹出窗口文本中使用 html、链接等  *}
<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 [文本格式化]

textformat 用于格式化文本. 该函数主要清理空格和特殊字符 ,对段落按单词边界换行和行缩进等段落格式化处理.

属性	类型	是否必须	缺省值	描述
style	string	No	*n/a*	预处理风格
indent	number	No	*0*	单行缩进的字符数目
indent_first	number	No	*0*	首行缩进的字符数目
indent_char	string	No	*(single space)*	填充缩进区域的字符(或字符串)
wrap	number	No	*80*	单行长度，超过该长度自动折行
wrap_char	string	No	*\n*	折行使用的字符(或字符串)，被附加在行尾
wrap_cut	boolean	No	*false*	如果设置为真，换行时不考虑换行点所在位置是否为完整单词 ,直接换行. 反之将在单词的边界处换行.
assign	string	No	*n/a*	输出值将被赋给模板变量的名称

textformat 函数演示

{textformat wrap=40}

This is foo.

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

This is bar.

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

{/textformat}

输出结果：

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

This is bar.

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

{textformat wrap=40 indent=4}

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

This is bar.

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

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

{/textformat}

输出结果：

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

  This is bar.

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

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

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

This is bar.

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

{/textformat}

输出结果：

```
  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}
```

输出结果：

```
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.
```

7. Config Files [配置文件]

配置文件有利于设计者管理文件中的模板全局变量。最简单的例子就是模板色彩变量。一般情况下你如果想改变一个程序的外观色彩，你就必须通过去更改每一个文件的颜色变量。如果有这个配置文件的话，色彩变量就可以保存在一个地方，只要改变这个配置文件就可以实现你色彩的更新。

配置文件语法演示：

```
# global variables
pageTitle = "Main Menu"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

[Customer]
pageTitle = "Customer Info"

[Login]
pageTitle = "Login"
focus = "username"
Intro = """This is a value that spans more
  than one line. you must enclose
                    it in triple quotes."""

# hidden section
[.Database]
host=my.domain.com
db=ADDRESSBOOK
user=php-user
pass=foobar
```

配置文件变量值能够在引号中使用，但是没有必要。你可以用单引号或者双引号。如果你有一个不只在一个区域内使用的变量值，你可以使用三引号(""")将它完整的封状起来，可以把它们放进配置文件，只要没有语法错误。建议在程序行前使用“#”加一些注释信息来标示。

上面关于配置文件的例子共有两个部分。每部分的名称都是用一个“[]”给括起来。每部分的名称命名规则就是任意的字符串，只要不包括有符号“[”或者“]”。例子开头的四个变量都是全局变量，也就是说不仅仅是可以在一个区域内使用。这些变量总是从配置文件中载入。如果某个特定的局部变量已经载入，这样全局

变量和局部变量都还可以载入。如果当某个变量名既是全局变量又是局部变量时，局部变量将被优先赋予值来使用。如果在一个局部中两个变量名相同的话，最后一个将被赋值使用。

配置文件是通过内建函数载入到模板｛config_load｝

可以在某个段时期通过预先想好的变量名或者局部名隐藏变量或者完整的一个节。当你的应用程序读取配置文件和取得有用数据而不用读取模板时这个非常有用，如果你有第三方来做模板编辑的话，可以肯定的说它们不能通过载入配置文件到模板而读取到任何有用的数据。

8. Debugging Console [调试控制台]

SMARTY 里面包括有一个调式控制台。它可以告诉你模板里面包含的所有内容，同时也可以为当前使用模板中的变量和配置文件变量赋值。一个叫 debug.tpl 的模板包含了很多控制调式控制台格式化的 SMARTY 类，在 SMARTY 中把变量 $debugging 设置为 true 如果需要的话设置变量 $debug_tpl 为模板源文件路径（在 SMARTY_DIR 用已经自定义）。当你载入页面时，有一个 JAVASCRIPT 控制台窗口将弹出且告诉你所有模板中包含的名称和当前页已经赋值的变量。如果要了解某个模板的详细变量，可以去看 {debug} 模板定义函数章节。如果要关闭掉调试控制台，设置变量 $debugging 为 false 就可以了。如果你开启了 $debugging ctrl 选项，也可以通过放置 SMARTY_DEBUG 在 URL 来临时打开调试控制台。

技术提示：当你使用函数 fetch() API 时调试控制台不能用，在使用 display() 时才可以使用。它将自动地把 javascript 添加到已经应用模板的每一个按扭中。如果你不喜欢 javascript ，你可以编辑文件 debug.tpl 模板，格式化输出为你自己喜欢的格式。调试数据是没有被缓存的，并且 debug.tpl 信息也没有包含在调试控制台的输出文件中。

注意：每个模板和配置文件的载入都是以秒来计算的，甚至是以几分之一秒。

二、 Smarty For Programmers [程序员篇]

9. Constants [常量]

SMARTY_DIR [Smarty 目录]

定位 Smarty 类文件的完整系统路径，如果没有定义 Smarty 目录，Smarty 将会试着自动创建合适的值。

如果定义了,路径必须要以斜线结束。

```
SMARTY_DIR 演示

// 设置到 smarty 的路径
define("SMARTY_DIR","/usr/local/lib/php/Smarty/");
require_once(SMARTY_DIR."Smarty.class.php");
```

10. Variables [变量]

$template_dir [模板目录变量]

该变量定义默认模板目录的名字。当包含文件时，如果不提供一个源类型（即源地址），那么将会到模板目录中寻找。默认情况下，目录是 ".templates", 也就是说他将会在和 php 执行脚本相同的目录下寻找模板目录。

技巧注意：不推荐把模板目录放在 web 服务器根目录下。

$compile_dir [编译目录变量]

该变量定位编译模板的目录名字。默认情况下，目录是 :"./templates_c", 也就是说他将会在和 php 执行脚本相同的目录下寻找编译目录。

技巧：该设置必须是一个相对或绝对路径。包含路径不用于写文件。

技巧：不推荐把编译目录放在 web 服务器根目录下。

$config_dir [配置目录变量]

该变量定义用于存放模板配置文件的目录，默认情况下，目录是 :"./configs", 也就是说他将会在和 php 执行脚本相同的目录下寻找配置目录。

技巧：不推荐把编译目录放在 web 服务器根目录下。

$plugins_dir [插件目录变量]

该变量定义 Smarty 寻找所需插件的目录。默认是在 SMARTY 目录下的"plugins"目录。如果提供了一个相对路径，Smarty 将首先在 SMARTY 目录下寻找，然后到当前工作目录下寻找，继而到 php 包含路径中的每个路径中寻找。

技巧：为了获取最好性能，不要将插件目录安装成必须使用 php 包含路径的境地。最好用一个绝对路径名字，或者是一个相对 SMARTY 目录或当前工作目录的路径。

$debugging [调试变量]

它能启动调试控制台。该控制台是一个 javascript 窗口，该窗口告诉你被包含的模板和应用于当前模板页

面的变量。

$debug_tpl [调试模板变量]

该变量定义用于调试控制台的模板文件名字。默认情况下，其名字为：debug.tpl，位于 Smarty 目录中。

$debugging_ctrl [调试控制变量]

该变量作用是允许以交替的方式启动调试设置。NONE 表示不允许交替方式。URL 表示当关键词 SMARTY_DEBUG 出现在 QUERY_STRING（查询字符串）中时，针对此脚本的调用，调试被启动。如果 $debugging 变量是真，则忽略其调试控制变量值。

$global_assign [全局配置变量]

定义一组总是隐式地作用于模板引擎的变量列表。这对于使全局变量或服务器变量可用于所有模板是很方便的，因为无需手工配置它们。在全局配置中的每一元素应当要么是一全局变量的名字，要么是一键/值对（键是全局数组名，值是对应该全局数组的变量数组）。$SCRIPT_NAME 变量默认属于 $HTTP_SERVER_VARS 数组。

技巧：服务器变量可以通过$smarty 变量获得，如{$smarty.server.SCRIPT_NAME}。参看$smarty 变量一节。

$undefined [未定义变量]

该变量为 Smarty 设定没有得到定义的变量的值,默认为空.当前仅用来设定全局配置中的未定义变量为一个默认值.

$autoload_filters [自动加载过滤器变量]

如果希望在每次模板调用过程中加载过滤器,你可以指定他们使用此变量,Smarty 将自动为你加载它们.该变量是一个联合数组,在数组中键是过滤器类型,值是过滤器名字所组成的数组.

例如:

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

$compile_check [编译检查变量]

每次调用 PHP 应用程序,Smarty 会试着查看自上次编译时间以来,当前模板是否被修改过.如果修改过了,她会重新编译那个模板.如果模板还没有被编译过,她将编译模板而不管编译检查如何设置.默认情况下编译检查这个变量设置为 true.一旦一个应用程序投入产品中(模板将不会修改了),就不再需要编译检查这一步了.为了最大性能,确定将$compile_check 设为"false".注意:如果设为了"false",虽然模板文件被修改,但你不会看到修改结果,因为模板没有得到重新编译.如果启动了缓存和编译检查,一旦有关模板文件或配置文件被更新,缓存文件将会重建.

$force_compile [强迫编译变量]

强迫 Smarty 每次调用(重新)编译模板时.这项设置不受$compile_check 的限制.默认情况下,它是无效的.

它对于开发和调试很方便.但它决不能使用于产品环境下.如果启动了缓存,每次将会重新生成缓存文件.

$caching [缓存变量]

告诉 Smarty 是否缓存模板的输出.默认情况下,她设为 0,或无效.如果模板产生冗余内容,建议打开缓存.这样有利于获得良好的性能增益.你也可以为同一模板设有多个缓存.当值为 1 或 2 时启动缓存.1 告诉 Smarty 使用当前的$cache_lifetime 变量判断缓存是否过期.2 告诉 Smarty 使用生成缓存时的 cache_lifetime 值.用这种方式你正好可以在获取模板之前设置缓存生存时间，以便较精确地控制缓存何时失效.

如果启动了编译检查,一旦任何的模板文件或配置文件(有关缓存部分的配置文件)被修改,缓存的内容将会重新生成.如果启动了强迫编译,缓存的内容将总会重新生成.

$cache_dir [缓存目录变量]

这是存放模板缓存的目录名.默认情况下,它是:"./cache",也就是说你可以在和 php 执行脚本相同目录下寻找缓存目录.你也可以用你自己的自定义缓存处理函数来控制缓存文件,它将会忽略这项设置.

技巧:这项设置必须是一个相对或绝对路径.包含路径不用于写文件.

技巧:不推荐将此目录放在 web 服务器根目录下.

$cache_lifetime [缓存生存时间变量]

该变量定义模板缓存有效时间段的长度(单位 秒),一旦这个时间失效,则缓存将会重新生成.如果要想实现所有效果,$caching 必须因$cache_lifetime 需要而设为"true".值为-1 时,将强迫缓存永不过期.0 值将导致缓存总是重新生成(仅有利于测试,一个更有效的使缓存无效的方法是设置$caching = false.)

如果启动了强迫编译,则缓存文件每次将会重新生成.要想有效地停止缓存,你可以利用 clear_all_cache() 函数清除所有的缓存文件,或者利用 clear_cache() 函数清除个别文件(或文件组).

技巧:如果你想给某些模板设定它们自己的缓存生存时间,你可以在调用 display()或 fetch()函数之前,通过设置$caching = 2,然后设置$cache_lifetime 为一个唯一值来实现.

$cache_handler_func [缓存处理函数变量]

可以提供一个自定义函数来处理缓存文件,而不是通过变量$cache_dir 使用内置方法.详见:cache handler function section

$cache_modified_check [缓存修正检查变量]

如果设置该变量为真,Smarty 将分析客户端发送来的 If-Modified-Since 头信息.如果缓存文件时间戳自上次访问以来没有改变,则发送一个"304 Not Modified"头,而不是缓存文件内容.这种方式仅工作于没有{insert} 标记的缓存内容.

$config_overwrite [配置覆盖变量]

如果设该变量为真,则从配置文件中读取出来的变量将会互相覆盖.否则,变量将会放到一个数组中.如果你想把配置文件中的数据存储到数组里,这种方式是很有用的,仅仅列出每个元素多次就可以了.默认情况下,设为真.

$config_booleanize [配置布尔化变量]

如果该变量设为真,配置文件中的 on/true/yes 和 off/false/no 值会自动转化为布尔值.这样的话,你就可以在模板中像{if #foobar#} ... {/if}这样使用这些值了.如果 foobar 为 on, true 或 yes,那么 {if}语句就会执行了.默认情况下,该变量值为真.

$config_read_hidden [配置读取隐藏变量]

如果设为真,在配置文件中的隐藏节块(******不会翻译啦)可以从模板中读取出来.典型的你会设为假,这样你可以在配置文件里存放敏感数据,例如数据库参数,而不用担心模板会调用他们.默认情况下,该变量设为假.

$config_fix_newlines [配置固定换行符变量]

如果该变量设为真,那么在配置文件中的 mac 和 dos 换行符(\r and \r\n)在语法解析时将会转换为\n.默认情况下,该变量为真.

$default_template_handle r_func [默认模板处理函数变量]

该函数在模板不能从它的源目录下获取时会得到调用.

$php_handling [PHP 处理变量]

该变量告诉 Smarty 怎样处理嵌入到模板中的 php 代码.有四种可能的设置,默认为

SMARTY_PHP_PASSTHRU.注意：改变量的设置不会影响模板里面{php}{/php}标记中的 php 代码.

SMARTY_PHP_PASSTHRU - 原样输出标记.

SMARTY_PHP_QUOTE - 作为 html 实体引用标记

SMARTY_PHP_REMOVE - 从模板中移出标记.

SMARTY_PHP_ALLOW - 将作为 php 代码执行标记.

极度不赞成将 php 代码嵌入到模板中.

$security [安全变量]

安全变量要么是真,要么是假.默认为假.当你不信任模板中的可编辑部分(例如通过 ftp 方式上传编辑的),并想通过模板语言减小系统非安全的风险时,安全变量设为真比较适合.设为真会将下面的规则强加于模板语言中,除非特别地用$security_settings 覆盖.

如果变量$php_handling 设为了 SMARTY_PHP_ALLOW, 则会隐式地被修改成 SMARTY_PHP_PASSTHRU

PHP 函数在 IF 语句中是不允许的,除了在$security_settings 中另行指出.

模板仅可以包含于$secure_dir 数组列出的目录中.

本地文件仅可以用{fetch}获取于$secure_dir 数组列出的目录中.

不允许有{php}{/php}标记.

PHP 函数不允许作为修正器,除了在$security_settings 中指出.

$secure_dir [安全目录变量]

这是一个与安全有关的本地目录数组变量.当启动安全变量时{include} 和 {fetch}会用到此数组变量.

$security_settings [安全配置变量]

用于当启动安全变量时覆盖或另行指定安全配置.有以下几种可能的配置:

PHP_HANDLING - 真或假.如果真,则不检查$php_handling 的配置.

IF_FUNCS - 这是一个可允许包含在 if 语句中的 php 函数名数组.

INCLUDE_ANY - 真或假.如果真,可以从文件系统中包含任何模板,而不管$secure_dir 目录设置如何.

PHP_TAGS - 真或假.如果真,{php}{/php}允许在模板中存在.

MODIFIER_FUNCS - 这是一个可用作修正器的 php 函数名数组.

$trusted_dir [信任目录变量]

信任目录变量仅用在$security 启动之后.这是一个所有建立信任的目录数组变量.你可以将 php 脚本放到这些信任目录中,这些脚本会直接在模板里以{include_php}标记得到执行.

$left_delimiter [左结束符变量]

用于模板语言中,默认是"{".

$right_delimiter [右结束符变量]

用于模板语言中,默认是"}".

$compiler_class [编译类变量]

指定 Smarty 用来编译模板的编译类名.默认为:'Smarty_Compiler'.仅适合于高级用户.

$request_vars_order [变量顺序变量]

请求变量的顺序在这里配置,类似于 php.ini 中的变量顺序.

$request_use_auto_globals [自动全局变量]

指定 Smarty 是否使用 php 的$HTTP_*_VARS[]数组变量(默认$request_use_auto_globals=false)或$_*[]数组($request_use_auto_globals=true)变量.这对使用了{$smarty.request.*}, {$smarty.get.*}等标记的模板有影响.值得注意的是:如果设置$request_use_auto_globals 为真,variable.request.vars.order 就无效了但 php 的 gpc_order 配置值还有作用.

$compile_id [编译 ID 变量]

永久的编译鉴别号.作为一个可选办法将相同的编译号传递给每个函数调用,你可以设置这个编译 id,随后此 id 将会被隐含地使用.

$use_sub_dirs [子目录变量]

如果你的 php 环境不允许 Smarty 创建子目录,则设此变量为假.子目录非常有用,所以尽可能的使用他们.

$default_modifiers [默认修正器变量]

这是一个修正器数组变量,这些修正器隐式地作用于模板中的每个变量.例如:针对 HTML,用数组('escape:"htmlall"')默认地忽略每个变量;为了使一个变量免除于默认修正器,请将参数为"nodefaults"的特殊"smarty"修正器作用于它,例如:{$var|smarty:nodefaults}.

$default_resource_type [默认源类型变量]

该变量告诉 smarty 隐式地使用什么源类型.默认值为'file',也就是说,$smarty->display('index.tpl');和 and $smarty->display('file:index.tpl');从意思上是一样的.祥见 resource 一章。

11. Methods [方法]

append [添加]

```
void append (mixed var)

void append (string varname, mixed var)

void append (string varname, mixed var, boolean merge)
```

添加指定的元素到数组中。如果是添加一个字符串，该字符串会被转换为数组格式后再进行添加。所添加的数据可以采用 名称, 数值的格式，或者是使用 “=>”连接的联合数组格式。如果第三个可选参数被指定为 TRUE，所添加的数据会和数组中现有数据进行合并，而不是直接添加。

注意：使用第三个参数“merge”时要考虑到数组的索引，所以，如果添加和被添加的数组都是以数字为索引，他们会互相覆盖，或者产生不连续的索引。这并不象 PHP 中的 array_merge()函数，后者会删除原有的数字索引，重新对索引进行编号。

添加演示：

```
// passing name/value pairs 以 名称，数值 的方式添加
$smarty->append("Name","Fred");$smarty->append("Address",$address);
// passing an associative array 以联合数组的方式添加
$smarty->append(array("city" => "Lincoln","state" => "Nebraska"));
```

append_by_ref [引用添加]

```
void append_by_ref (string varname, mixed var)


void append_by_ref (string varname, mixed var, boolean merge)
```

本函数用于采用引用的方式把变量的值添加到原有值之后。如果采用引用的方式添加了一个变量，那么当这个变量的值改变时，被添加的值也随之改变。对于对象，append_by_ref()函数也能够避免对于被添加对象的内存拷贝。关于变量引用的进一步解释可以查看 PHP 手册。如果第三个可选参数设置为 true，数值将会和现有数组合并，而不是添加在数组后面。

注意：使用第三个参数"merge"时要考虑到数组的索引，所以，如果添加和被添加的数组都是以数字为索引，他们会互相覆盖，或者产生不连续的索引。这并不象 PHP 中的 array_merge()函数，后者会删除原有的数字索引，重新对索引进行编号。

引用添加演示：

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

assign [赋值]

```
void assign (mixed var)


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

用来赋值到模板中。可以指定一对 名称/数值 ，也可以指定包含 名称/数值 的联合数组。

```
赋值演示：

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

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

assign_by_ref [引用赋值]

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

采用引用的方式赋值到模板中，而不是在模板中创建一个数值的副本。引用的概念可以查看 PHP 手册的变量引用部分。

注意：本函数用于采用引用的方式赋值到模板中。如果把一个变量采用引用的方式赋值后，变量的值又被改变了，那么在模板中将会看到变量改变后的值。对于对象，assign_by_ref()函数也避免了对象的内存拷

贝。进一步的解释可以查看 PHP 手册的变量引用部分。

引用赋值演示：

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

clear_all_assign [清除所有赋值]

```
void clear_all_assign ()
```

清除所有已赋值到模板中的值。

清除所有赋值演示：

```
// 清除所有已赋值的变量
$smarty->clear_all_assign();
```

clear_all_cache [清除所有缓存]

```
void clear_all_cache (int expire time)
```

清除所有模板缓存。作为可选参数“expire time”，你可以指定一个以秒为单位的最小时间，超过这个时间的缓存都将被清除掉。

清除所有缓存演示：

```
// 清除全部缓存
```

```
$smarty->clear_all_cache();
```

clear_assign [清除赋值]

```
void clear_assign (string var)
```

清除指定模板变量的值。可以指定单独的一个变量名称，或者是一个数组。

```
清除赋值演示：

// 清除一个变量
$smarty->clear_assign("Name");

// 清除多个变量
$smarty->clear_assign(array("Name","Address","Zip"));
```

clear_cache [清除缓存]

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

清除指定模板的缓存。如果这个模板有多个缓存，你可以用第二个参数指定要清除缓存的缓存号，还可以通过第三个参数指定编译号。你可以把模板分组，以便可以方便的清除一组缓存。更多的信息可以查看缓存部分。第四个参数是可选的，用来指定超过某一时间（以秒为单位）的缓存才会被清除。

```
清除缓存演示：

// 清除某一模板的缓存
$smarty->clear_cache("index.tpl");

// 清除某一模板的多个缓存中指定缓存号的一个
```

```
$smarty->clear_cache("index.tpl","CACHEID");
```

clear_compiled_tpl [清除已编译模板]

```
void clear_compiled_tpl (string tpl_file)
```

清除指定模板资源的编译版本，如果不指定则清除所有已编译模板。除非特殊的需要，一般情况下不需要使用该函数。

（Fwolf:这个函数并不会删除模板本身，而是删除 templates_c 目录中存放的编译后生成的文件，这些编译后的文件会在下一次页面调用的时候再一次被生成。）

```
清除已编译模板演示：

清除指定模板资源的编译文件
$smarty->clear_compiled_tpl("index.tpl");

清除所有已编译的模板文件
$smarty->clear_compiled_tpl();
```

clear_config [清除配置]

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

清除所有配置变量，如果指定了变量名称，则只清除所指定的配置变量。

```
清除配置演示：

// 清除所有配置变量
$smarty->clear_config();
```

```
清除一个配置变量
$smarty->clear_config('foobar');
```

config_load [加载配置]

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

加载配置文件，并将其中的数据传送到模板中，它的功能和 config_load 函数是一样的。

注意：到 Smarty 2.4.0 为止，已赋值的模板变量对于 fetch()和 display()方法是共享调用的，而使用 config_load()加载的变量是全局的。并且在编译时配置文件也将被编译，以加快执行速度，并且遵守 force_compile 和 compile_check 设置。

```
加载配置演示：

// 加载配置变量
$smarty->config_load('my.conf');

加载其中一节
$smarty->config_load('my.conf','foobar');
```

display [显示]

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

显示模板，需要指定一个合法的模板资源的类型和路径。还可以通过第二个可选参数指定一个缓存号，相关的信息可以查看缓存。

通过第三个可选参数，可以指定一个编译号。这在你想把一个模板编译成不同版本时使用，比如针对不同的语言编译模板。编译号的另外一个作用是，如果你有多个$template_dir 模板目录，但只有一个$compile_dir 编译后存档目录，这时可以为每一个$template_dir 模板目录指定一个编译号，以避免相同的模板文件在编译后会互相覆盖。相对于在每一次调用 display()的时候都指定编译号，也可以通过设置$compile_id 编译号属性来一次性设定。

显示演示：

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

只有在缓存不存在时才调用数据库
if(!$smarty->is_cached("index.tpl"))
{
 // 装入数据
 $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);
}

// 显示输出
$smarty->display("index.tpl");
```

通过模板资源的语法来使用不在$template_dir 模板目录下的文件。

显示模板资源演示：

```
// 绝对路径
$smarty->display("/usr/local/include/templates/header.tpl");
```

```
// 绝对路径（另外一种方式）
$smarty->display("file:/usr/local/include/templates/header.tpl");

// WINDOS 平台下的绝对路径（必须使用“file:”前缀）
$smarty->display("file:C:/www/pub/templates/header.tpl");

// 从模板资源"db"中调用
$smarty->display("db:header.tpl");
```

fetch [取得输出的内容]

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

返回一个模板输出的内容（HTML 代码），而不是直接显示出来，需要指定一个合法的模板资源的类型和路径。你还可以通过第二个可选参数指定一个缓存号，相关的信息可以查看缓存。

通过第三个可选参数，可以指定一个编译号。这在你想把一个模板编译成不同版本时使用，比如针对不同的语言编译模板。编译号的另外一个作用是，如果你有多个$template_dir 模板目录，但只有一个$compile_dir 编译后存档目录，这时可以为每一个$template_dir 模板目录指定一个编译号，以避免相同的模板文件在编译后会互相覆盖。相对于在每一次调用 display()的时候都指定编译号，也可以通过设置$compile_id 编译号属性来一次性设定。

```
取得输出的内容演示：

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

$smarty->caching = true;

只有在缓存不存在时才调用数据库
if(!$smarty->is_cached("index.tpl"))
```

```
{
 // 装入数据
 $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);

}

// 捕获输出
$output = $smarty->fetch("index.tpl");

// 对将要输出的内容进行处理
echo $output;
```

get_config_vars [取配置变量的值]

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

返回指定配置变量的值。如果不指定变量名称，则会返回一个包含所有已加载配置变量的值的数组。

```
取配置变量的值演示：

// 得到已加载配置变量“foo”的值
$foo = $smarty->get_config_vars('foo');

// 得到所有已加载配置变量的值
$config_vars = $smarty->get_config_vars();

// 输出
print_r($config_vars);
```

get_registered_object [取得已注册的对象]

```
array get_registered_object (string object_name)
```

返回一个已注册对象的引用。用于在一个自定义函数里直接调用已注册的对象。

取得已注册的对象演示：

```
function smarty_block_foo($params, &$smarty) {
    if (isset[$params['object']]) {
        // 取得已注册对象的引用/地址
        $obj_ref =& $smarty->&get_registered_object($params['object']);
        // 现在就可以使用$obj_ref 来直接调用对象了，它直接指向该对象，而不是一个拷贝
        }
    }
```

get_template_vars [取得模板变量的值]

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

返回一个指定的已赋值的模板变量的值。如果不指定参数，则返回一个包含所有已赋值模板变量的值的数组。

取得模板变量的值演示：

```
// 得到一个已赋值的模板变量的值
$foo = $smarty->get_template_vars('foo');

// 得到所有已赋值的模板变量的值
$tpl_vars = $smarty->get_template_vars();

// 查看
```

```
print_r($tpl_vars);
```

is_cached [是否已被缓存]

```
void is_cached (string template, [string cache_id])
```

在指定模板的缓存存在是返回真。只有在缓存设置为真时才可用。

```
是否已被缓存演示：

$smarty->caching = true;
if(!$smarty->is_cached("index.tpl")) {
    // 调用数据库，并对变量进行赋值
    ......
}

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

如果模板有多个缓存的话，可以通过第二个可选参数指定缓存号。

```
多缓存模板的判断演示：

$smarty->caching = true;

if(!$smarty->is_cached("index.tpl","FrontPage")) {
    // 调用数据库，并对变量进行赋值
    ......
    }

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

load_filter [加载过滤器]

void load_filter (string type, string name)

这个函数用来加载过滤器插件。第一个参数指定过滤器的类型，只能是下列之一 :“pre”,“post”, 或者是“output”。第二个参数指定过滤器插件的名称，比如“trim”。

加载过滤器插件演示：

```
// 装入名为 'trim' 的预过滤器
$smarty->load_filter('pre', 'trim');

// 装入另一个名为 'datefooter' 的预过滤器
$smarty->load_filter('pre', 'datefooter');

// 装入名为 'compress' 的输出过滤器
$smarty->load_filter('output', 'compress');
```

register_block [注册一个块]

void register_block (string name, mixed impl, bool cacheable, array or null cache_attrs)

用来动态注册/定义块函数插件。前两个参数指定块函数名称和执行函数的名称。

执行函数的格式可以是一个包含函数名称的字符串；也可以是一个 array(&$object, $method)数组形式，其中&$object 是一个对象的引用 ,而$method 是它的一个方法 ,还可以是一个 array(&$ class, $method)数组形式，其中$class 是一个类的名称，$method 是类中的一个方法。

$cacheable 和 $cache_attrs 参数在大多数情况下可以省略。它们的使用方法可以查看控制插件输出的缓存。

注册一个块演示：

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

function do_translation ($params, $content, &$smarty, &$repeat) {
        if (isset($content)) {
                $lang = $params['lang'];
                // 对 $content 进行转换
                ......
                return $translation;
                }
        }

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

register_compiler_function [注册编译函数]

```
void register_compiler_function (string name, mixed impl, bool cacheable)
```

动态注册一个编译函数插件，前两个参数是编译函数的名称、执行函数的名称。

执行函数的格式可以是一个包含函数名称的字符串；也可以是一个 array(&$object, $method)数组形式，其中&$object 是一个对象的引用，而$method 是它的一个方法，还可以是一个 array(&$ class, $method)数组形式，其中$class 是一个类的名称，$method 是类中的一个方法。

$cacheable 参数在大多数情况下可以省略。具体的使用方法可以查看控制插件输出的缓存。

register_function [注册函数]

void register_function (string name, mixed impl, bool cacheable, array or null cache_attrs)

动态注册模板函数插件，前两个参数是模板函数名称和执行函数名称。

执行函数的格式可以是一个包含函数名称的字符串；也可以是一个 array(&$object, $method)数组形式，其中&$object 是一个对象的引用，而$method 是它的一个方法，还可以是一个 array(&$ class, $method)数组形式，其中$class 是一个类的名称，$method 是类中的一个方法。

$cacheable 和 $cache_attrs 参数在大多数情况下可以省略。它们的使用方法可以查看控制插件输出的缓存。

注册函数演示：

```
$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());
        }

        // 现在你可以在模板中这样显示日期：{date_now}
        // 或者用{date_now format="%Y/%m/%d"}的格式进行格式化
```

register_modifier [注册修饰器]

void register_modifier (string name, mixed impl)

动态注册一个修饰器插件，需要制定模板修饰器的名称，和实现具体功能的函数。

执行函数的格式可以是一个包含函数名称的字符串；也可以是一个 array(&$object, $method)数组形式，其中&$object 是一个对象的引用，而$method 是它的一个方法，还可以是一个 array(&$ class, $method)数组形式，其中$class 是一个类的名称，$method 是类中的一个方法。

注册修饰器演示：

```
// 把 PHP 的 stripslashes 函数影像为 Smarty 模板的修饰器
$smarty->register_modifier("sslash","stripslashes");

// 现在可以在模板中使用{$var|sslash}的格式来去除变量中的特殊符号了
```

register_object [注册对象]

```
void register_object (string object_name, object $object, array allowed methods/properties, boolean format, array block methods)
```

注册一个在模板中使用的对象，相关的例子在手册的对象小节中。

register_outputfilter [注册输出过滤器]

```
void register_outputfilter (mixed function)
```

动态注册一个输出过滤器，以便在模板输出之前进行操作。如何设立输出过滤函数，请查看模板输出过滤。

执行函数的格式可以是一个包含函数名称的字符串；也可以是一个 array(&$object, $method)数组形式，其中&$object 是一个对象的引用 ,而$method 是它的一个方法 ,还可以是一个 array(&$ class, $method)数组形式，其中$class 是一个类的名称，$method 是类中的一个方法。

register_postfilter [注册提交过滤器]

void register_postfilter (mixed function)

动态注册一个在模板编译之后执行的提交过滤器。如何设立提交过滤函数，请查看模板提交过滤器。

执行函数的格式可以是一个包含函数名称的字符串；也可以是一个 array(&$object, $method)数组形式，其中&$object 是一个对象的引用 ,而$method 是它的一个方法 ,还可以是一个 array(&$ class, $method)数组形式，其中$class 是一个类的名称，$method 是类中的一个方法。

register_prefilter [注册预过滤器]

void register_prefilter (mixed function)

注册一个在模板编译之前执行的预过滤器。如何设立预过滤函数，请查看模板预过滤器。

执行函数的格式可以是一个包含函数名称的字符串；也可以是一个 array(&$object, $method)数组形式，其中&$object 是一个对象的引用 ,而$method 是它的一个方法 ,还可以是一个 array(&$ class, $method)

数组形式，其中$class 是一个类的名称，$method 是类中的一个方法。

register_resource [注册资源]

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

动态注册一个资源插件，前两个参数是资源的名称和执行函数的数组。关于如何建立一个函数取出模板，请看模板资源部分。

注意：资源的名称主要要有两个字符组成。一个字符的资源名称将被忽略，并被视为是文件路径的一部分，比如$smarty->display('c:/path/to/index.tpl');。

执行函数数组必须包含 4 或 5 个元素。包含 4 个元素的情况下，这些函数分别是处理“source 源文本”、“timestamp 时间戳”“secure 安全”、“trusted 可信的”的函数。5 个元素的情况下，第一个元素必须是一个对象的引用，或者是一个类名，后面的 4 个元素是实现处理“source 源文本”、“timestamp 时间戳”、“secure 安全”、“trusted 可信的”的方法。

注册资源演示：

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

trigger_error [触发错误]

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

用来使用 Smarty 内置的方法输出错误信息。参数 level 可以是 PHP 函数 trigger_error()的一个参数值，如 E_USER_NOTICE, E_USER_WARNING 等，一般默认为 E_USER_WARNING。

template_exists [模板是否存在]

```
bool template_exists (string template)
```

检查制定的模板是否存在，参数 template 既可以是模板的文件路径，也可以是指定的资源字符串。

unregister_block [注销一个块]

```
void unregister_block (string name)
```

动态注销一个块函数插件，参数是块函数的名称。

unregister_compiler_function [注销编译函数]

```
void unregister_compiler_function (string name)
```

动态注销一个编译函数，参数 name 是编译函数的名称。

unregister_function [注销函数]

```
void unregister_function (string name)
```

动态注销一个模板函数插件，参数是模板函数的名称。

```
注销函数演示：

// 让模板设计者不能操作系统文件
$smarty->unregister_function("fetch");
```

unregister_modifier [注销修饰器]

```
void unregister_modifier (string name)
```

动态注销修饰器插件，参数是模板修饰器的名称。

```
注销修饰器演示：

// 禁止模板设计者去除元素的标记
$smarty->unregister_modifier("strip_tags");
```

unregister_object [注销对象]

```
void unregister_object (string object_name)
```

注销一个对象。

unregister_outputfilter [注销输出过滤器]

```
void unregister_outputfilter (string function_name)
```

动态注销一个输出过滤器。

unregister_postfilter [注销提交过滤器]

```
void unregister_postfilter (string function_name)
```

动态注销提交过滤器（在模板编译完成后执行）。

unregister_prefilter [注销预过滤器]

```
void unregister_prefilter (string function_name)
```

动态注销一个预过滤器（模板编译前执行）。

unregister_resource [注销资源]

```
void unregister_resource (string name)
```

动态注销一个资源插件，参数是资源的名称。

注销资源演示：

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

12. Caching [缓存]

缓存被用来保存一个文档的输出从而加速 display()或 fetch()函数的执行。如果一个函数被加进缓存，那么实际输出的内容将用缓存来代替。缓存可让事物非常快速的执行，特别是带有长计算时间的模板。一旦 display()或 fetch()用缓存输出，那么一个缓存文档将非常容易用几个模板文档或是配置文档等来组成〔功力不小〕。

一旦模板是动态〔应该不难理解〕的，哪些文档你加了缓存，缓存时间多长都是很重要的。举个例子，比如你站点的首页内容不是经常更改，那么对首页缓存一个小时或是更长都可得到很好效果。相反，几分钟就要更新一下信息的天气地图页面，用缓存就不好了。

Setting Up Caching [建立缓存]

首先要做的就是让缓存可用。这就要设置$caching = true(或 1.)

使缓存可用演示：

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

$smarty->caching = true;

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

建立缓存后，display('index.tpl')函数会把模板返回原来状态〔没缓存〕，也会把输出保存 copy〖n.名词〗到$cache_dir.下次调用 display('index.tpl')，保存的缓存 copy〖n.〗会被再用来代替原来的模板。

技术提示 :在$chche_dir 目录里的文档命名跟模板一致。尽管是用.php 作为扩展名 ,但并不会被当作 php 代码来解析。所以不要去修改它。

每个缓存页都有一个用$cache_lifetime 来控制的会话期。初始值是 3600 秒，就是一小时。会话期结束，缓存就会重建。你可以通过设置$caching=2 来控制单个缓存文件各自的的过期时间。祥细内容察看 $cache_lifetime 里面的列表。

设置单个缓存会话期〔时间〕演示：

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

$smarty->caching = 2; // $smarty->caching 为 2 时，可精确设置每个缓存的生存期

// 将 index.tpl 的 cache_lifetime（缓存生存期）设置为 5 分钟
$smarty->cache_lifetime = 300;          //单位为秒
$smarty->display('index.tpl');

// set the cache_lifetime for home.tpl to 1 hour
将 home.tpl 的 cache_lifetime 设置为 1 小时
$smarty->cache_lifetime = 3600;
$smarty->display('home.tpl');
```

```
//提示：在$chching=2 后面的$chche_lifetime 不会起作用。  ？？？？？？
// home.tpl 的缓存会话期设为 1 小时后，不会再按$cache_lifetime 的值改变。
// home.tpl 的缓存会在 1 小时后结束。

$smarty->cache_lifetime = 30;            // 缓存 30 秒
$smarty->display('home.tpl');
```

如果$compile_check 可用 ,每个跟缓存文档相关的模板文档和配置文档都会被检查来确定是否需要修改。

在缓存产生后，改动任何文档，缓存也跟着更新改动。设置$compile_check 为 false，是实现最佳性能的

最小改动。

```
$compile_check 演示：

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

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

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

一旦$force_compile 可用，缓存文档会一直重建。这将有效的关闭缓存。$force_compile 只是用来调试，

更有效关闭缓存的方法是让$caching = false(或 0.)

is_cached()函数可用来测试一个模板是否有有效的缓存。如果一个缓存模板需要从数据库中获取数据，可

以用这个函数来跳过这个过程。

```
使用 is_cached() 演示：

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

$smarty->caching = true;
```

```
if(!$smarty->is_cached('index.tpl')) {
    // 没有缓存可用，在这里执行变量分配
    $contents = get_database_contents();
    $smarty->assign($contents);
}

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

可以插入模板函数 insert 来使部分页面动态化。除了在右下方显示的标语外整个页面都可以缓存。在缓存内容里面可以插入函数来使标语也动态化。查看相关文档关于 insert 的细节和例子。

可以用 clear_all_cache()来清除所有缓存，或用 clear_cache()来清除单个缓存文档。

清除缓存演示：

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

$smarty->caching = true;

// 清除所有缓存文件
$smarty->clear_all_cache();

// 只清除模板 index.tpl 的缓存文件
$smarty->clear_cache('index.tpl');

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

Multiple Caches Per Page [每页多个缓存]

可以用单个函数 display()或 fetch()来输出多个缓存文档。display('index.tpl')在多种条件下会有不同的输出内容，要单独的把缓存分开。可以通过函数 display() 的第二参数 cache_id 来达到效果。

传给 display()一个 cache_id 演示：

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

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

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

上面，我们通过变量$my_cache_id 作为 cache_id 来 display()。在 index.tpl 里$my_cache_id 的每个唯一值，会建立单独的缓存。在这个例子里，"article_id"在 URL 传送，并用作 cache_id。

技术提示：要注意从客户端(web 浏览器)传值到 Smarty(或任何 PHP 应用程序)的过程。尽管上面的例子用 article_id 从 URL 传值看起来很方便，却可能有糟糕的后果[安全问题]。cache_id 被用来在文件系统里创建目录，如果用户想为 article_id 赋一个很大的值，或写一些代码来快速发送随机的 article_ids，就有可能会使服务器出现问题。确定在使用它之前清空已存在的数据。在这个例子，可能你知道 article_id 的长度是 10 字符，并只由字符-数字组成，并且必须在数据库里是个可用的 article_id。要注意检查这个问题！

确定传给 is_cached()和 clear_cache()的第二参数是同一个 cache_id。

传给 is_cached()一个 cache_id 演示：

```
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)) {
    // 没有缓存可用，在这里执行变量分配
```

```
    $contents = get_database_contents();
    $smarty->assign($contents);
}

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

可以通过把 clear_cache()的第一参数设为 null 来为特定的 cache_id 清除所有缓存。

```
为特定的 cache_id 清除所有缓存演示：

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

$smarty->caching = true;

// 清除所有 cache_id 的值为 "sports" 的缓存
$smarty->clear_cache(null,"sports");

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

通过这种方式，可以用相同的 cache_id 来把你的缓存集合起来。

Cache Groups [缓存集合]

可以通过建立 cache_id 集合做更祥细的集合体。在 cache_id 的值里用竖线"|"来分开子集合。可以尽可能多的包含子集合。

```
cache_id 集合演示：

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

$smarty->caching = true;

// 清除所有以 "sports|basketball" 作为 cache_id 开头的缓存组
```

```
// （清除 "sports/basketball" 目录下的缓存？？）
$smarty->clear_cache(null,"sports|basketball");

// 清除所有以 "sports" 作为 cache_id 开头的缓存组
// 包括 "sports|basketball"，或 "sports|(anything)|(anything)|(anything)|..."
// 清除 "sports" 目录下的缓存？？
$smarty->clear_cache(null,"sports");

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

技术提示:缓存集合并不像 cache_id 一样对模板使用路径。比如 如果你 display('themes/blue/index.tpl'), 那么在"themes/blue"目录下你并不能清除缓存。想要清除缓存，必须先用 cache_id 把缓存集合，像这样 display('themes/blue/index.tpl','themes|blue')；然后就可以用 clear_cache(null,'themes|blue')清除 blue theme 下的缓存。

Controlling Cacheability of Plugins' Output [控制插件输出的缓冲能力]

自从 Smarty-2.6.0 插件以来，如果注册它们，则插件的缓存能力能够被重新声明。register_block，register_compiler_function 和 register_function 的第 3 个参数就是$cacheable，并且它的值默认为 true。当然，在 2.6.0 版本之前它的默认值也是这样的。

当用$cacheable=false 来这册一个插件，则每次这个页面被输出的时候，这个插件就会被使用，即使这个页面来自缓存。这个插件函数的行为有点像这个函数 insert。

和{insert}相反，插件的属性默认是不缓存的。通过使用第四个参数 $cache_attrs ，它们能够被重新声明为缓存。 $cache_attrs 是一个属性名字的数组，可以被缓存，所以每次当它被从缓存中取出的时候，这个插件函数获得值-----因为这个页面会被写入用来缓存。

阻止插件从缓存中输出演示：

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')) {
  // 从 db 和 assign... 中取得 $obj
  $smarty->assign_by_ref('obj', $obj);
}

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

index.tpl:

```
Time Remaining: {remaining endtime=$obj->endtime}
```

直到$obj 运行结束，时间的秒数在每一个页面输出的时候会改变，即使这个页面被缓存。只要结束时间属性被缓存，当页面被写入到缓存但是没有对这个页面接着的请求的时候对象只是不得不重新从数据库里取出而已。

阻止一个模板文件的整篇被缓存演示：

index.php:

```
require('Smarty.class.php');
```

```
$smarty = new Smarty;
$smarty->caching = true;

function smarty_block_dynamic($param, $content, &$smarty) {
  return $content;
}

// 阻止被"{dynamic}{/dynamic}"围绕的片断被缓存
$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}
```

当重新加载这个页面，你将会注意到这两个日期不同。一个是“动态”，一个是“静态”。你能够在{dynamic}...{/dynamic}之间作任何事情，并且保证它将不会像剩下的页面一样被缓存。

13. Advanced Features [高级特征]

Objects [对象]

SMARTY 允许通过模板访问 PHP 对象。有两种方式来访问它们。一种是注册对象到模板，然后通过类似于用户自定义函数的形式来访问它。另一种方法给模板分配对象,然后通过访问其它赋值变量类似的方法进行访问。第一种方法有一个很好的模板语法，同时它作为一个注册对象被限制为几个固定的方法和目标，

这样是比较安全的。然而一个注册对象不能够在相对自身数组里面循环使用和赋值。总之，你根据你自己的需求来觉得选用那种方法，但是使用第一种方法的话，可以让你的模伴语法达到最小。

如果安全选项激活后，就没有私有成员或者函数能够被访问（以"_"开头）。如果有一个同名成员或者对象存在，那么方法将被使用。

你可以限制成员和函数，但是外解可以通过列举它们为一个第三注册变量的数组的方式来访问它。

一般情况下，参数通过模板传递给对象的方法和自定义函数获得参数的方法是一样的。一个混合数组作为第一个参数传递，而 SMARTY 对象作为第二个。如果你想像传统的对象参数一样一次传递一个参数，设置第四个参数为 FALSE 即可。

使用已注册的或已分配的对象演示：

index.php

```
// the object

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

$myobj = new My_Object;
// 注册将会被引用的对象
$smarty->register_object("foobar",$myobj);

// 如果想要限制对某个方法或属性的访问，将它们列出来
$smarty->register_object("foobar",$myobj,array('meth1','meth2','prop1'));

// 如果想要使用传统的对象参数格式，传递一个 false 的布尔值
$smarty->register_object("foobar",$myobj,null,false);
```

```
// 同样可以分配对象，可以时，通过 ref 分配
$smarty->assign_by_ref("myobj", $myobj);

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


index.tpl

{* 访问已注册的对象 *}
{foobar->meth1 p1="foo" p2=$bar}

{* 同样可以分配输出 *}
{foobar->meth1 p1="foo" p2=$bar assign="output"}
the output was {$output)

{* 访问已分配的对象 *}
{$myobj->meth1("foo",$bar)}
```

Prefilters [预过滤器]

模板预过滤器是一些 PHP 函数，模板就是在那些函数编译后才运行。这样有利于预先处理你的模板，删除不需要的内容，监视对模板进行的操作，例如：预过滤器同样能够通过 load filter() 函数和设置 $autoload filters 变量来注册或者从工具目录里载入。SMARTY 将传递模板输出作为第一个参数，通过自定义函数返回处理结果。

使用模板预过滤器演示：

index.php

```
// 将这些加入应用
function remove_dw_comments($tpl_source, &$smarty)
{
  return preg_replace("/<!--#.*-->/U","",$tpl_source);
}

// 注册预过滤器
$smarty->register_prefilter("remove_dw_comments");
$smarty->display("index.tpl");
```

index.tpl

```
<!--# this line will get removed by the prefilter -->
```

Postfilters [后过滤器]

模板后过滤器是一些 PHP 函数，模板就是在那些函数编译后才运行。后过滤器同样能够通过 load filter() 函数和设置 $autoload filters 变量来注册或者从工具目录里载入。SMARTY 将传递模板输出作为第一个参数，通过自定义函数返回处理结果。

使用模板后过滤器演示：

index.php

```
// 将这些加入应用
function add_header_comment($tpl_source, &$smarty)
{
  return "<?php echo \"<!-- Created by Smarty! -->\n\" ?>\n".$tpl_source;
}

// 注册后过滤器
$smarty->register_postfilter("add_header_comment");
$smarty->display("index.tpl");

{* 已编译的 Smarty 模板 index.tpl *}
<!-- Created by Smarty! -->
{* 其余的模板内容... *}
```

Output Filters [输出过滤器]

当模板通过函数 display() 或者 fetch()被调用时，它的输出能够通过一个或者多个过滤器而发出。

它与预过滤器的不同之处就是预过滤器编译模板是在模板保存到磁盘之前，输出过滤器是在它执行的时候

才操作模板输出的。

输出过滤器同样能够通过 load filter() 函数和设置 $autoload filters 变量来注册或者从工具目录里载入。

SMARTY 将传递模板输出作为第一个参数，通过自定义函数返回处理结果。

使用模板输出过滤器演示：

```
<?php
// 将这些加入应用
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;
}

// 注册输出过滤器
$smarty->register_outputfilter("protect_email");
$smarty->display("index.tpl");

// 现在在模板输出中出现任何一个 email 地址，将会针对 spambots（垃圾邮件？）有一个简单的保护
?>
```

Cache Handler Function [缓冲处理函数]

作为一个可选择使用的默认基于文本的缓冲机制，可以定制或者自定义缓冲处理函数来进行读、写、清除缓冲文件。

如果要在应用程序下建立一个函数，SMARTY 将会使用一个缓冲处理。设置这个函数名为 $cache handler func 中类变量名。Smarty 将使用这个函数来处理缓冲数据。第一个参数不管是‘读’,‘写’,‘清除’,

都是动态的。第二个参数是 Smarty 对象。第三个参数是缓冲的内容。在进行写操作之前，Smarty 传递这些缓冲内容给这些参量。在进行读操作之前，Smarty 预处理那些缓冲数据相关或者封装好了的函数。在进行清除操作之前，从他没有使用起，就传递一个虚拟变量。第四个参数是模板文件名（需要读和写的文件），第五个函数是缓冲 ID（额外选项），然后第六个参数是编译的 ID（额外选项）。

注意：最新的参数名 （$exp_time）在 Smarty-2.6.0 中才加入的。

使用 MySQL 作为缓存源演示：

```
<?php
/*
example usage:

include('Smarty.class.php');
include('mysql_cache_handler.php');

$smarty = new Smarty;
$smarty->cache_handler_func = 'mysql_cache_handler';

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

mysql database is expected in this format:

create database SMARTY_CACHE;

create table CACHE_PAGES(
CacheID char(32) PRIMARY KEY,
CacheContents MEDIUMTEXT NOT NULL
);
*/

function mysql_cache_handler($action, &$smarty_obj, &$cache_content, $tpl_file=null, $cache_id=null, $compile_id=null, $exp_time=null)
{
    // set db host, user and pass here
    $db_host = 'localhost';
    $db_user = 'myuser';
    $db_pass = 'mypass';
    $db_name = 'SMARTY_CACHE';
```

```
    $use_gzip = false;

    // create unique cache id
    $CacheID = md5($tpl_file.$cache_id.$compile_id);

    if(! $link = mysql_pconnect($db_host, $db_user, $db_pass)) {
        $smarty_obj->_trigger_error_msg("cache_handler: could not connect to database");
        return false;
    }
    mysql_select_db($db_name);

    switch ($action) {
        case 'read':
            // save cache to database
            $results = mysql_query("select CacheContents from CACHE_PAGES where
CacheID='$CacheID'");
            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;

}

?>
```

Resources [资源]

模板可以来自各种各样的资源。当你显示或者取得一个模板，或者当你在一个模板里面包含另外一个模板时，只要提供一个类型的资源，紧跟正确的路径和模板名。

如果一个模板资源没有明确的给出 $default_resource_type 变量值，就认为这个资源为假定值。

Templates from $template_dir [来自 $template_dir 的模板]

来自 $template_dir 的模板不需要模版资源，尽管可以使用 file: resource 的组合。

只需提供所使用模版与 template_dir 的相对路径。

使用来自 $template_dir 的模板演示：

```
// 来自 PHP 脚本
$smarty->display("index.tpl");
$smarty->display("admin/menu.tpl");
$smarty->display("file:admin/menu.tpl"); // 同上

{* 来自 Smarty 模版内部 *}
{include file="index.tpl"}
{include file="file:index.tpl"} {* 同上 *}
```

Templates from any directory [来自任意路径的模版]

模版在 $template_dir 之外，需要 file: template 资源类型，跟着是绝对路径和模版名。

使用来自任意路径的模版演示：

```
// 来自 PHP 脚本
$smarty->display("file:/export/templates/index.tpl");
$smarty->display("file:/path/to/my/templates/menu.tpl");

{* 来自 Smarty 模版内部 *}
{include file="file:/usr/local/share/templates/navigation.tpl"}
```

Windows Filepaths [Windows 文件路径]

如果在 Windows 环境下使用，文件路径通常在路径名的开头包括驱动器盘符(C:)，确定在路径中使用 "file:" 来避免命名空间冲突和得到不期望的结果。

使用来自 windows 的路径演示：

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

{* 来自 Smarty 模版内部 *}
{include file="file:D:/usr/local/share/templates/navigation.tpl"}
```

Templates from other sources [来自其它资源的模版]

无论 PHP 是否可以访问的资源，都可以作为模版使用：databases, sockets, LDAP 等等。通过写资源插件函数和通过 Smarty 注册它们来使用。

更多信息参照 resource plugins 部分。

注意：不能覆盖内置的文件资源，但是可以通过注册为另一个资源名的方式提供来自文件系统的取自模版的资源。

使用定制资源演示：

```
// 来自 PHP 脚本
// 将这些函数放入应用的某处

function db_get_template ($tpl_name, &$tpl_source, &$smarty_obj)
{
 // 调用数据库取得模版
 // 组装到 $tpl_source 中
 $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)
{
  // 调用数据库来组装 $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)
{
  // 假设所有的模版都是安全的
  return true;
}

function db_get_trusted($tpl_name, &$smarty_obj)
{
  // 不用于模版
}

// 将资源名注册为 "db"
$smarty->register_resource("db", array("db_get_template",
  "db_get_timestamp",
  "db_get_secure",
  "db_get_trusted"));

// 使用来自 PHP 脚本的资源
$smarty->display("db:index.tpl");

{* 使用来自 Smarty 模版内部的资源 *}
{include file="db:/extras/navigation.tpl"}
```

Default template handler function [默认模版处理函数]

You can specify a function that is used to retrieve template contents in the event the template cannot be retrieved from its resource. One use of this is to create templates that do not exist on-the-fly.

可以指定用于取得模版内容的函数，其中一个应用是创建不存在于传输过程中的模版。

使用默认模版处理函数演示：

```
<?php
// 把这个函数放入应用的某处

function make_template ($resource_type, $resource_name, &$template_source, &$template_timestamp, &$smarty_obj)
{
    if( $resource_type == 'file' ) {
        if ( ! is_readable ( $resource_name )) {
            // 创建模版文件，返回内容
            $template_source = "This is a new template.";
            $template_timestamp = time();
            $smarty_obj->_write_file($resource_name,$template_source);
            return true;
        }
  } else {
        // 不是文件
        return false;
  }
}

// 设置默认处理
$smarty->default_template_handler_func = 'make_template';
?>
```

14. Extending Smarty With Plugins [利用插件扩展 Smarty]

2.0 版本引入了被广泛应用于自定义 Smarty 功能的插件机制。它包括如下类型：

functions 函数插件

modifiers 修饰插件

block functions 区块函数插件

compiler functions 编译函数插件

prefilters 预滤器插件

postfilters 补滤器插件

outputfilters 输出过滤插件

resources 资源插件

inserts 嵌入插件

为了与旧有方式保持向后兼容，除资源插件外，保留了通过 register_* API 方式装载函数的处理方法。如果不是使用 API 方式而是使用直接修改类变量 $custom_funcs, $custom_mods 等的方法，那么就需要修改程序。或者使用 API 的方法，或者将自定义功能转换成插件。

How Plugins Work [插件如何工作]

插件总在需要的时候被装载。只有在模板脚本里调用的特定修饰、函数、资源插件等会被装载。此外，即便在同一个请求中有几个不同的 Smarty 实体运行，每个插件也只被装载一次。

预/补过滤器插件和输出过滤器插件的装载方式有些不同。由于在模板中未被提及，它们必须在模板被处理前通过 API 函数明确地装入系统。同类型的多个过滤器插件依据被装载的次序先后不同分别先后执行。

插件目录是包含一条路径信息的字符串或包含多条路径信息的字符串数组。安装插件的时候，将插件置于

其中一个目录下，Smarty 会自动识别使用。

Naming Conventions [命名约定]

插件文件和函数必须遵循特定的命名约定以便 Smarty 识别。

插件文件必须命名如下：

type . name .php

其中 type 是如下插件中的一种：

function

modifier

block

compiler

prefilter

postfilter

outputfilter

resource

insert

name 为仅包含字母、数字和下划线的合法标志符。

例如: function.html_select_date.php, resource.db.php, modifier.spacify.php.

插件内的函数应遵循如下命名约定：

smarty_ type _ name ()

type 和 name 的意义如前。

如果指定的插件文件不存在或命名不合规范，Smarty 会输出相应的错误信息。

Writing Plugins [编写插件]

Smarty可自动从文件系统装载插件 ,或者运行时通过register_* API函数装载。可以通过unregister_* API函数卸载已经装载的插件。

只在运行时装载的插件的函数名称不需要遵守命名约定。

如果某个插件依赖其它插件内的某些功能（例如某些插件功能捆绑于 Smarty 内）,那么可以通过如下方法装载必须的插件：

```
require_once $smarty->_get_plugin_filepath('function', 'html_options');
```

Smarty 对象通常作为传递给插件的最后一个参数（有两个例外：1、修饰插件根本不接受传递过来的

Smarty 对象。2、为了向上兼容老版本的 Smarty ,区块插件将 &$repeat 作为最后一个参数 ,因此 Smarty 对象是倒数第二个参数。)

Template Functions [模板函数插件]

```
void smarty_function_ name (array $params, object &$smarty)
```

模板传递给模板函数的所有的属性都包含在参数数组 $params 中 ,既可以通过如： $params['start'] 的方式直接处理其中的值，也可以使用 extract($params) 的方式将所有值导入符号表中。

函数输出（返回值）的内容将取代模板中函数名称出现的位置（例如：fetch()函数）。同时函数也可能只是执行些后台任务，并无任何输出

如果函数需要向模板中增加变量或者使用 Smarty 提供的某些功能，可以通过 $smarty 对象实现。

相关内容参看: register_function(), unregister_function().

有输出插件函数演示：

```
<?php
/*
 * Smarty 插件
 * -------------------------------------------------------------
 * File: function.eightball.php
 * Type: function
 * Name: eightball
 * 目的: 输出一个随机的 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];
}
?>
```

在模板中调用方法如下：

```
Question: Will we ever have time travel?
Answer: {eightball}.
```

无输出插件函数演示：

```
<?php
/*
  * Smarty  插件
  * -------------------------------------------------------------
  * File: function.assign.php
  * Type: function
  * Name: assign
  *  目的:  为模版变量分配一个值
  * -------------------------------------------------------------
  */
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 [修正器插件]

修正器是一些短小的函数，这些函数被应用于模版中的一个变量，然后变量再显示或用于其他的一些文档。

我们可以把修正器链接起来。

```
mixed smarty_modifier_ name (mixed $value, [mixed $param1, ...])
```

修正器插件的第一个参数是不可缺少的。剩余的参数是可选的，它们的有无取决于期望执行哪一种操作。

修正器必须有返回值。

也可参考 register_modifier(), unregister_modifier().

简单修正器插件演示：

这个插件主要目的是用另一个名字替换一个内置 PHP 函数的名字。他没有任何多余的参数。

```
<?php
/*
 * Smarty  插件
 * -------------------------------------------------------------
 * File: modifier.capitalize.php
 * Type: modifier
 * Name: capitalize
 *  目的:  将字符串转换成大写字母
 * -------------------------------------------------------------
 */
function smarty_modifier_capitalize($string)
{
```

```
 return ucwords($string);
}
?>
```

更加复杂的修正器插件演示：

```
<?php
/*
 * Smarty 插件
 * -------------------------------------------------------------
 * File: modifier.truncate.php
 * Type: modifier
 * Name: truncate
 * 目的: 截取字符串为某一个的长度，如果有需要，将词从中间分开，并添加到字符串 $etc
 * -------------------------------------------------------------
 */
function smarty_modifier_truncate($string, $length = 80, $etc = '...',
 $break_words = false)
{
 if ($length == 0)
 return '';

 if (strlen($string) > $length) {
 $length -= strlen($etc);
 $fragment = substr($string, 0, $length+1);
 if ($break_words)
 $fragment = substr($fragment, 0, -1);
 else
 $fragment = preg_replace('/\s+(\S+)?$/', '', $fragment);
 return $fragment.$etc;
 } else
 return $string;
}
?>
```

Block Functions [区块函数插件]

```
void smarty_block_ name (array $params, mixed $content, object &$smarty)
```

块函数的形式是这样的：{func} .. {/func}。换句话说，他们用标记圈起一个块，然后对这个块的内容进行操作。块函数优先于同名的传统函数，即你不能同时有同名的传统函数{func}和块函数{func} .. {/func}。

默认地你的函数执行被 Smarty 调用两次 :一次是在开始标记 ,另一次是在结束标记(参考下面的&$repeat 怎样改变这种情况)

块函数仅开始标记可以有属性。所有从模板传替给模板函数的属性被囊括与一个集合数组参数中。你可以直接获取其值，例如：$params['start']或者是用 extract($params)将它们导入符号表中。当处理结束标记时，开始标记的属性对你的函数也是可用的。

变量 $content 的值取决于是否因开始标记或结束标记调用你的函数。假如是开始标记，它会是空的，如果是结束标记，它会是模板块的内容。请注意模板块已经被 Smarty 处理,所以你接收到的结果是输出后的模板而不是原样模板。

参数 &$repeat 通过参考引用传递给函数执行过程并为其提供一个可能值来控制显示块多少遍。默认情况下在首次调用块函数（块开始标记）时变量 $repeat 是真，在随后的所有块函数调用中其始终是假。每当函数执行返回的 &$repeat 是真时，在{func} .. {/func}之间的内容再次求值,函数执行接收一个新块参数 $content 内容值被再次调用。

如果你嵌套了块函数，通过访问变量$smarty->_tag_stack 找出父块函数是可能的。仅仅对块函数运行一下 var_dump()，函数结构就会一目了然了。

参看: register_block(), unregister_block().

区块函数插件演示：

```
<?php
/*
 * Smarty 插件
 * -------------------------------------------------------------
 * File: block.translate.php
 * Type: block
 * Name: translate
 * 目的: 翻译文本块
 * -------------------------------------------------------------
 */
function smarty_block_translate($params, $content, &$smarty)
{
 if (isset($content)) {
 $lang = $params['lang'];
 // 对 $content 做只能翻译工作
 return $translation;
 }
}
```

Compiler Functions [编译函数插件]

编译函数仅在模板编译过程中被调用。对于将 PHP 代码或对时间敏感的静态内容嵌入到模板中，他们是比较有用的。如果编译函数和普通函数都注册了同一个名字，则编译函数具有优先使用权。

```
mixed smarty_compiler_ name (string $tag_arg, object &$smarty)
```

此编译函数有两个参数：标记参数字符串——基本上是从函数名字直到结束位置的所有内容，另一个参数是 Smarty 对象。该函数将返回嵌入到被编译模板中的 PHP 代码。

参看 register_compiler_function(), unregister_compiler_function().

简单编译函数插件演示：

```
<?php
/*
 * Smarty  插件
 * -------------------------------------------------------------
 * File: compiler.tplheader.php
 * Type: compiler
 * Name: tplheader
 *  目的:  输出包含资源文件名和编译时间的头
 * -------------------------------------------------------------
 */
function smarty_compiler_tplheader($tag_arg, &$smarty)
{
  return "\necho '" . $smarty->_current_file . " compiled at " . date('Y-m-d H:M'). "';";
}
?>
```

这个函数在模板中以如下方式调用：

```
{*  这个函数在编译时执行  *}
{tplheader}
```

在编译之后的模板中被嵌入的 PHP 代码内容如下：

```
<php
echo 'index.tpl compiled at 2002-02-20 20:02';
?>
```

Prefilters/Postfilters [预滤器/补滤器插件]

从概念上看，预滤器和后滤器插件都很简单；不同之处就在于它们的执行，更确切地说是它们的执行时刻（非时间段，而是某一个时刻）。

string smarty_prefilter_ name (string $source, object &$smarty)

预滤器用来在编译之前直接处理模板源文件。预滤器函数的第一个参数是模板源文件，该文件可能被其他一些预滤器修正过。此预滤器插件将返回修正过的源文件。请记住此源文件仅用来编译，它不会在任何地方被保存。

string smarty_postfilter_ name (string $compiled, object &$smarty)

后滤器用来在编译之后直接处理模板的编译输出（PHP 代码），但须在编译之后的模板被保存到文件系统之前就进行操作。预滤器函数的第一个参数是编译之后的模板代码，该代码可能被其他一些后滤器修正过。此后滤器插件将返回修正过的代码文件。

预滤器插件演示：

```
<?php
/*
 * Smarty 插件
 * -------------------------------------------------------------
 * File: prefilter.pre01.php
 * Type: prefilter
 * Name: pre01
 * 目的: 将 html 标签转换为小写
 * -------------------------------------------------------------
 */
 function smarty_prefilter_pre01($source, &$smarty)
 {
 return preg_replace('!<(\w+)[^>]+>!e', 'strtolower("$1")', $source);
 }
?>
```

后滤器插件演示：

```
<?php
```

```
/*
 * Smarty 插件
 * -------------------------------------------------------------
 * File: postfilter.post01.php
 * Type: postfilter
 * Name: post01
 * 目的: 输出代码列出所有当前模版变量
 * -------------------------------------------------------------
 */
 function smarty_postfilter_post01($compiled, &$smarty)
 {
 $compiled = "<pre>\n<?php print_r(\$this->get_template_vars()); ?>\n</pre>" . $compiled;
 return $compiled;
 }
?>
```

Output Filters [输出过滤插件]

输出过滤器插件的作用是，在装载并执行完一个模板之后显示模板之前，操作该模板的输出。

```
string smarty_outputfilter_ name (string $template_output, object &$smarty)
```

输出过滤器函数第一个参数是需要处理的模板输出，第二个参数是调用这个插件的 Smarty 实例。此插件将会对参数进行处理并返回相应的结果。

输出过滤器插件演示：

```
/*
 * Smarty 插件
 * -------------------------------------------------------------
 * File: outputfilter.protect_email.php
 * Type: outputfilter
 * Name: protect_email
 * 目的: 把 email 地址中的 @ 转换为 %40，作为一个简单的反垃圾邮件保护
 * -------------------------------------------------------------
 */
```

```
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);
}
```

Resources [资源插件]

资源插件被认为是为 Smarty 提供模板源或 PHP 脚本组件的一种普通方式。一些资源例子如:数据库、LDAP、共享内存、sockets（套接字）等等。

需要为每一种类型的资源注册四个函数。每一个函数将接收被请求的资源作为第一个参数，Smarty 对象作为最后一个参数。剩余的参数取决于函数的不同。

```
bool smarty_resource_ name _source (string $rsrc_name, string &$source, object &$smarty)
```

```
bool smarty_resource_ name _timestamp (string $rsrc_name, int &$timestamp, object &$smarty)
```

```
bool smarty_resource_ name _secure (string $rsrc_name, object &$smarty)
```

```
bool smarty_resource_ name _trusted (string $rsrc_name, object &$smarty)
```

第一个函数将会检索资源。它的第二个参数是一个参考引用变量，结果值会存放到该变量里面。如果此函数能成功的检索到资源，将会返回 true，否则返回 false。

第二个函数将会检索被请求资源的最后修改时间（UNIX 时间戳）。它的第二个参数是一个参考引用变量，时间戳值会存放到该变量里面。如果此函数能成功的确定时间戳，将会返回 true，否则返回 false。

第三个函数将会返回 true 或 false，取决于被请求资源是否安全。这个函数仅用于模板资源，但仍应被定义。

第四个函数将会返回 true 或 false，取决于被请求资源是否被信任。这个函数仅用于被 {include_php} 或 {insert} 标记以 src 属性请求的 PHP 脚本组件。但仍应被定义，甚至用于模板资源也不例外。

参看 register_resource(), unregister_resource().

资源插件演示：

```
<?php
/*
 * Smarty 插件
 * -------------------------------------------------------------
 * File: resource.db.php
 * Type: resource
 * Name: db
 * 目的: 从数据库中取得模版
 * -------------------------------------------------------------
 */
function smarty_resource_db_source($tpl_name, &$tpl_source, &$smarty)
{
 // 执行数据库调用取得模版，装入 $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)
{
  // 调用数据库，装入 $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)
{
  // 假设所有模版是安全的
  return true;
}

function smarty_resource_db_trusted($tpl_name, &$smarty)
{
  // 未用于模版
}
?>
```

Inserts [嵌入插件]

插入插件用来执行在模板中被 {insert} 标记调用的函数 。

```
string smarty_insert_ name (array $params, object &$smarty)
```

函数的第一个参数是一个传递给插入动作的属性集合数组。每一种方式都可以直接获取那些属性值，例如

$params['start']，或用 extract($params)将属性导入符号表中。

插入函数将会返回某值，该值将在模板中的 {insert} 标记处被替换。

插入插件演示：

```
<?php
/*
 * Smarty  插件
 * -------------------------------------------------------------
 * File: insert.time.php
 * Type: time
 * Name: time
 *  目的:  根据格式( format )插入当前的日期/时间
 * -------------------------------------------------------------
 */
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;
}
?>
```

三、 Appendixes [附录]

15. Troubleshooting [疑难解答]

Smarty/PHP errors [错误]

Smarty 能够发现许多类似缺少标签属性或者不规范变量名这样的错误。如果发生这种错误，就会有下面的错误提示：

错误演示：

```
Warning: Smarty: [in index.tpl line 4]: syntax error: unknown tag - '%blah'
  in /path/to/smarty/Smarty.class.php on line 1041

警告：Smarty: 在 index.tpl 文件第 4 行，语法错误:'%blah'标签未知

Fatal error: Smarty: [in index.tpl line 28]: syntax error: missing section name
  in /path/to/smarty/Smarty.class.php on line 1041

严重错误：Smarty: 在 index.tpl 文件第 28 行，语法错误:缺少节段名字在 /path/to/smarty/路径中的 Smarty.class.php 文件 1041 行
```

Smarty 可以显示模板名称以及行号和错误。这些错误显示未所发生错误所属的 smarty 类所在的实际行号。

某些错误 Smarty 不能捕捉，像缺少结束标签。这些类型的错误通常会在在 php 分析语法错误的编译时间中就捕捉出来了.

PHP 解析错误演示：

```
Parse error: parse error in /path/to/smarty/templates_c/index.tpl.php on line 75
```

当遇到一个 php 解析错误时，错误行号将反应到 php 编译脚本，而不是模板本身。通常，会看到模板并发现语法错误。通常会发现：缺少 if}{/if} ，或者{section}{/section}的结束标签。或者{if}标签内的逻辑语法错误。如果你不能检查出错误，那就得在模板中打开 php 编译文件按照行号找出相应错误。

16. Tips & Tricks [使用技巧和经验]

Blank Variable Handling[空白变量处理]

也许有时想给一个没有赋值的变量输出一个默认的值来代替什么也不输出 ,比如输出' '[这是一个空格符号，HTML 里的标记]来使得表格里的背景能够正常工作。大多数人使用{if}语句来处理这些，但是用 Smatry 这里有一个捷径，那就是使用 default [默认的]值赋给变量。

```
如果一个变量是空值，输出   演示：

{* 麻烦的方式 *}

{if $title eq ""}
       
{else}
      {$title}
{/if}

{* 简单的方式 *}

{$title|default:" "}
```

Default Variable Handling[默认变量处理]

如果一个变量在整个模板里被经常使用，而每次要用到这个变量的时候都要给它赋默认的值，那将是无法忍受的。可以通过使用 assign[赋值]这个函数来给这个变量赋上它的默认值。

```
赋给一个模板变量默认的值演示：

{* 在模版顶部执行 *}
{assign var="title" value=$title|default:"no title"}
```

```
{* 如果 $title 为空，当输出它的时候就包含值 "no title" *}
{$title}
```

Passing variable title to header template[传递变量型标题给头模板]

当大多数模板文件用到了一些相同的头模板文件和脚模板文件，通常是把这些模板分开并且包含他们。但是如果头模板需要什么来使得显示一些不同的标题，取决于用到的那些页面，可以传递这些标题给头模板---当它们被包含的时候。

传递变量标题给头模板演示：

mainpage.tpl

```
{include file="header.tpl" title="Main Page"}
{* 模版 body 在这里 *}
{include file="footer.tpl"}
```

archives.tpl

```
{config_load file="archive_page.conf"}
{include file="header.tpl" title=#archivePageTitle#}
{* 模版 body 在这里 *}
{include file="footer.tpl"}
```

header.tpl

```
<HTML>
<HEAD>
<TITLE>{$title|default:"BC News"}</TITLE>
</HEAD>
<BODY>
```

footer.tpl

```
</BODY>
</HTML>
```

当这个主页[main page]被浏览的时候，'Main Page'这个标题就会传递给头模板文件[header.tpl]，并且结果会被用成为标题。当这个档案页[archives page]被浏览的时候，文件标题将会是'Archives'。注意在这个档案页例子中，我们用了一个来自这个文件[archives_page.conf]的变量来代替一个硬性的代码变量。当然，要是变量[$title]没有初始化，我们会发现'BC News'被输出----用那种使用默认值的变量的方法。

Dates[日期]

就像一条经验法则，经常是把日期当成时间戳传递给 Smarty。 这样使得模板设计人员可以使用 date_format[日期格式化]这个函数来对日期格式化进行全面控制，当然，这使得在需要比较日期的时候比较容易。

注意：从 Smarty 1.4.0 起，可以像用 unix 中的时间戳,mysql 中的时间戳,或者任何能够用这个函数[strtotime()]进行分析的时间戳那样把日期传递给 Smarty。

使用日期格式化演示：

{$startDate|date_format}

输出结果：

Jan 4, 2001

{$startDate|date_format:"%Y/%m/%d"}

输出结果：

2001/01/04

```
{if $date1 < $date2}
    ...
{/if}
```

当在一个模板文件里使用{html_select_date}的时候，程序员将可能想在一个输出文件中把这种形式转回成时间戳的形式。下面就是一个可以帮助你达到目的的函数。

```
将日期转换回时间戳演示：

// 假设表单元素命名为 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[WAP/WML]

WAP/WML 模板需要一个说明文件内容形式的文件头和模板同时传递出去。最容易的方式就是编写一个客户型的函数用来输出这个文件头。如果使用缓冲，那将不会起作用，所以我们使用一些插入标记（记住：插入标记不是缓冲！）。请确保在你的模板输出到浏览器之前没有任何东西被输出，否则文件头将失效。

```
使用插入来写一个 WML 类型的文件头演示：

// 确保 apache 配置了 .wml 扩展
// 将该函数放入应用的某处，或者放入 Smarty.addons.php
```

```
function insert_header() {
  // 该函数需要 $content 参数
  extract(func_get_arg(0));
  if(empty($content))
  return;
  header($content);
  return;
}

// Smarty 模版必须有 insert 标签

{insert name=header content="Content-Type: text/vnd.wap.wml"}

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<!-- begin new wml deck -->
<wml>
<!-- begin first card -->
<card>
<do type="accept">
<go href="#two"/>
</do>
<p>
Welcome to WAP with Smarty!
Press OK to continue...
</p>
</card>
<!-- begin second card -->
<card id="two">
<p>
Pretty easy isn't it?
</p>
</card>
</wml>
```

Componentized Templates[组合的模板]

这项技巧有点普通，但是仍然是个不错的思想，如果要使用的话，自担风险。

传统上，把模板编入应用程序中步骤如下：首先，累计一下 PHP 程序中的变量，(可能用到数据库查询。) 然后，实例化一个 Smarty 对象，给变量赋值并且输出这个模板。因此，例如在模板里有一个资源标记。可以在应用程序中收集这些资源数据，然后给模板中的这些变量赋值并输出。现在如果仅仅是通过包含这些模板就可以把资源标记加入到任何一个应用程序中不是很好吗？

可以通过使用{php}{/php}这两个标签来在模板里嵌入 PHP。这样，就能够建立包含自身的使用自己的数据结构的模板来给他们自身的变量赋值。通过像这样逻辑的嵌入，能够把模板和逻辑性结合起来。这种方法使得模板不管源于何处总是像一个构件一样组合在一起。

组合的模板演示：

```
{* Smarty *}

{php}

    // 为取得股票数据定义函数
    function fetch_ticker($symbol,&$ticker_name,&$ticker_price) {
        // 从资源中取得 $ticker_name 和 $ticker_price 的逻辑
    }

    // 调用该函数
    fetch_ticker("YHOO",$ticker_name,$ticker_price);

    // 分配模版变量
    $this->assign("ticker_name",$ticker_name);
    $this->assign("ticker_price",$ticker_price);

{/php}

Stock Name: {$ticker_name} Stock Price: {$ticker_price}
```

就像 Smarty 1.5.0，这里有一种更清晰的方法。可以在模板里使用{include_php ...}这些标签来包含 php。使用这种方法，能够保持 PHP 的逻辑性和模板的逻辑性分离。参照 include_php 这个函数来获得更多信

息。

使用 include_php 组合的模板演示：

load_ticker.php

```
<?php
	// 为取得股票数据定义函数
	function fetch_ticker($symbol,&$ticker_name,&$ticker_price) {
		// 从资源中取得 $ticker_name 和 $ticker_price 的逻辑
	}

	// 调用该函数
	fetch_ticker("YHOO",$ticker_name,$ticker_price);

	// 分配模版变量
	$this->assign("ticker_name",$ticker_name);
	$this->assign("ticker_price",$ticker_price);
?>
```

index.tpl

```
{* Smarty *}

{include_php file="load_ticker.php"}

Stock Name: {$ticker_name} Stock Price: {$ticker_price}
```

Obfuscating E-mail Addresses[拒绝电子邮件地址]

想知道电子邮件地址是怎样出现在如此众多的猎头邮件列表中吗？猎头们收集电子邮件地址的一种方式就是从网页上获取。为了对抗这种方式，可以让电子邮件在 HTML 源文件中以动态 javascript 的方式显示，这样在浏览器里就能正确的显示出来。这个是通过 maito 插件做到的。

拒绝电子邮件地址演示：

index.tpl

Send inquiries to
{mailto address=$EmailAddress encode="javascript" subject="Hello"}

技术提示：这种方法不是 100%安全的。一个猎头可能知道编写他的电子邮件收集器来解码，但不是很可能的。

17. Resources [相关资源]

Smarty 的主页位于 http://smarty.php.net/。

您能加入邮件列表，通过发送邮件至信箱 smarty-general-subscribe@lists.php.net。您也可以通过如下链接加入邮件列表 http://marc.theaimsgroup.com/?l=smarty&r=1&w=2

PHP 中文社区是一个 PHP 技术讨论社区，欢迎加入社区参与讨论。

18. BUGS [漏洞]

最新的 Smarty 分卷可查阅漏洞文件，或上网查阅。