Manuale di Smarty

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

Smarty - il motore di template PHP con compilatore

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

Publication date 2010-09-20

Table of Contents

List of Examples

Prefazione

Indubbiamente è una delle domande più frequenti sulle mailing list del PHP: perché devo rendere i miei script PHP indipendenti dal layout? Se è vero che PHP è conosciuto come "linguaggio di scripting incorporato in HTML", dopo aver realizzato un paio di progetti che mescolano liberamente PHP e HTML nasce l'idea che separare forma e contenuti sia una buona cosa. Inoltre, in molte aziende i ruoli dei grafici (progettisti del layout) e dei programmatori sono separati. La ricerca di una soluzione con i template è quindi una conseguenza naturale.

Ad esempio, nella nostra azienda lo sviluppo di un applicazione procede così: dopo che sono stati redatti i documenti con le specifiche richieste, i progettisti delle interfacce creano dei modelli di interfaccia e li danno ai programmatori. Questi implementano la logica di business in PHP e usano i modelli di interfaccia per creare scheletri di template. A questo punto il progetto passa al progettista HTML/creatore di layout per le pagine web, che porta i template al loro massimo splendore. Il progetto potrebbe ancora andare avanti e indietro un paio di volte fra programmazione e HTML. Quindi è importante avere un buon supporto per i template, perché i programmatori non vogliono avere a che fare con l'HTML e non vogliono che i progettisti HTML facciano danni col codice PHP. I grafici hanno bisogno di supporto per i file di configurazione, i blocchi dinamici e altri elementi di interfaccia, ma non vogliono dover avere a che fare con le complicazioni del linguaggio di programmazione.

Dando un'occhiata alle diverse soluzioni di template attualmente disponibili per PHP, vediamo che la maggior parte di esse fornisce solo un modo rudimentale per sostituire variabili nei template e hanno delle forme limitate di funzionalità relative ai blocchi dinamici. Ma le nostre necessità erano un po' maggiori di queste. Noi volevamo che i programmatori evitassero DEL TUTTO di avere a che fare con l'HTML, ma questo era quasi inevitabile. Ad esempio, se un grafico voleva alternare i colori di sfondo su un blocco dinamico, questo doveva essere ottenuto preventivamente dal programmatore. Volevamo anche che i grafici potessero usare i propri file di configurazione, ed importare da questi le variabili nei template. La lista potrebbe continuare ancora.

Iniziammo così a scrivere una specifica per un motore di template verso la fine del 1999. Dopo avere finito le specifiche, iniziammo a lavorare su un motore scritto in C che, speravamo, avrebbe potuto essere incluso in PHP. Non solo però ci scontrammo con molti complicati ostacoli tecnici, ma c'era anche un dibattito molto acceso su cosa esattamente un motore di template avrebbe dovuto fare e cosa no. Da questa esperienza decidemmo che il motore sarebbe stato scritto in PHP come classe, in modo che ognuno potesse usarlo come gli pareva. Così scrivemmo un motore che faceva proprio quello e SmartTemplate™ venne alla luce (nota: questa classe non è mai stata pubblicata). Era una classe che faceva quasi tutto quello che volevamo: sostituzione delle variabili, supporto per l'inclusione di altri template, integrazione con i file di configurazione, incorporazione del codice PHP, limitate funzionalità con istruzioni 'if' e molti altri robusti blocchi dinamici che potevano essere nidificati ripetutamente. Tutto questo veniva fatto con le espressioni regolari e il codice che ne venne fuori era, per così dire, impenetrabile. Era anche notevolmente lento nelle grosse applicazioni, per via di tutta l'analisi (parsing) ed il lavoro sulle espressioni regolari che doveva fare ad ogni invocazione. Il problema più grosso dal punto di vista di un programmatore era tutto il lavoro necessario nello script PHP per creare ed elaborare i template ed i blocchi dinamici. Come rendere tutto questo più semplice?

Così nacque la visione di quello che poi è diventato Smarty. Sappiamo quanto è veloce PHP senza il sovraccarico dell'analisi dei template. Sappiamo anche quanto il linguaggio possa apparire meticoloso ed estremamente noioso per il grafico medio, e questo può essere mascherato con una sintassi di template molto più semplice. Allora, perché non combinare i due punti di forza? Così nacque Smarty...

Part I. Introduzione

Table of Contents

Chapter 1. Cos'è Smarty?

Smarty è un motore di template per PHP. Più specificatamente, fornisce un modo semplice di separare la logica e il contenuto dell'applicazione dalla sua presentazione. Questo concetto si può comprendere meglio in una situazione in cui il programmatore ed il progettista dei template hanno ruoli diversi, o nella maggior parte dei casi non sono la stessa persona.

Per esempio, diciamo che dovete creare una pagina web che mostra un articolo di giornale. Il titolo, il sommario, l'autore e il corpo dell'articolo sono gli elementi del contenuto: non contengono informazioni su come saranno presentati. Vengono passati a Smarty dall'applicazione, dopodiché il grafico modifica i template e usa una combinazione di tag HTML e tag di template per formattare la presentazione di questi elementi (tabelle HTML, colori di sfondo, dimensione dei caratteri, fogli di stile ecc.). Un giorno il programmatore ha bisogno di cambiare il sistema in cui viene ottenuto il contenuto dell'articolo (si tratta di una modifica alla logica dell'applicazione). Questa modifica non influisce sul lavoro del grafico, infatti il contenuto arriverà al template esattamente uguale a prima. Allo stesso modo, se il grafico vuole ridisegnare completamente il template, questo non richiederà modifica alla logica applicativa. Quindi, il programmatore può fare modifice alla logica senza bisogno di ristrutturare i template, e il grafico può modificare i template senza rovinare la logica dell'applicazione.

Uno degli obiettivi progettuali di Smarty è la separazione della logica di business dalla logica di presentazione. Questo significa che i template possono contenere logica, a condizione che tale logica sia esclusivamente relativa alla presentazione. Cose come includere un altro template, alternare i colori delle righe di tabella, mostrare un dato in maiuscolo, ciclare su un array di dati per visualizzarli, ecc., sono tutti esempi di logica di presentazione. Questo non significa che Smarty forza una separazione fra la logica di business e quella di presentazione. Smarty non può sapere che cosa è una cosa e cosa è l'altra, per cui se mettete logica di business nel template sono affari vostri. Inoltre, se *non volete* alcuna logica nei template, potete sicuramente ottenere ciò riducendo il contenuto a solo testo e variabili.

Uno degli aspetti caratteristici di Smarty è la compilazione dei template. Questo significa che Smarty legge i file dei template e crea script PHP a partire da questi. Una volta creati, questi script vengono eseguiti da quel momento in poi: di conseguenza si evita una costosa analisi dei template ad ogni richiesta, e ogni template può avvantaggiarsi pienamente di strumenti per velocizzare l'esecuzione come Zend Accelerator (http://www.zend.com/) o PHP Accelerator (http://www.php-accelerator.co.uk).

Ecco alcune delle funzionalità di Smarty:

- E' estremamente veloce.
- E' efficiente, perché è l'analizzatore di PHP a fare il "lavoro sporco".
- Non c'è sovraccarico per l'analisi del template, che viene compilato una sola volta.
- E' abbastanza furbo da saper ricompilare solo i template che sono stati modificati.
- Potete creare funzioni personalizzate e modificatori di variabili personalizzati, il che rende il linguaggio dei template estremamente estensibile.
- La sintassi dei tag di delimitazione dei template è configurabile: potete usare {}, {{}}, <!--{}-->, ecc.
- I costrutti if/elseif/else/endif vengono passati al PHP, quindi la sintassi delle espressioni condizionali può essere semplice o complicata a vostro piacimento.
- E' consentito nidificare in maniera illimitata sezioni, test, ecc.
- E' possibile incorporare direttamente codice PHP nei file di template, sebbene non dovrebbe essercene bisogno (e nemmeno è raccomandato), essendo il motore così personalizzabile.

- Supporto nativo al caching
- Scelta arbitraria dei sorgenti dei template
- Funzioni personalizzate di gestione della cache
- Architettura a plugin

Chapter 2. Installazione

Requisiti

Smarty necessita di un web server su cui gira PHP 4.0.6 o successivo.

Installazione di base

Installate i file delle librerie di Smarty che si trovano nella directory /libs/ della distribuzione. Questi sono i file PHP che NON DOVETE modificare. Sono condivisi da tutte le applicazioni e vengono modificati solo quando passate ad una nuova versione di Smarty.

Example 2.1. File delle librerie di Smarty

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

Smarty usa una costante PHP chiamata SMARTY_DIR che contiene il path di sistema della directory delle librerie di Smarty. Fondamentalmente, se la vostra applicazione è in grado di trovare il file `Smarty.class.php`, non avete bisogno di impostare SMARTY_DIR, in quanto Smarty la troverà da solo. Tuttavia, se `Smarty.class.php` non si trova nel vostro include_path, o se non fornite alla vostra applicazione un percorso assoluto per questo file, allora dovete definire manualmente SMARTY_DIR. La costante SMARTY_DIR *deve* contenere uno slash (/) finale.

Ecco come creerete un'istanza di Smarty nei vostri script PHP:

Example 2.2. Creazione di un'istanza di Smarty

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

Provate a lanciare lo script qui sopra. Se ricevete un errore che dice che il file `Smarty.class.php` non si trova, dovete fare una delle cose seguenti:

Example 2.3. Fornire un percorso assoluto al file delle librerie

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

Example 2.4. Aggiungere la directory della libreria all'include_path di PHP

```
<?php
// Modificate il file php.ini, aggiungete la directory delle
// librerie di Smarty all'include_path e riavviate il server web.
// A questo punto il codice seguente dovrebbe funzionare:
require('Smarty.class.php');
$smarty = new Smarty;
?>
```

Example 2.5. Impostare manualmente la costante SMARTY_DIR

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

Ora che i file delle librerie sono al loro posto, è ora di impostare le directory di Smarty per la vostra applicazione. Smarty necessita di quattro directory chiamate (per default) `templates`, `templates_c`, `configs` e `cache`. Ciascuna di queste è definibile dalle proprietà della classe Smarty `$template_dir`, `$compile_dir`, `$config_dir`, e `$cache_dir` rispettivamente. E' altamente raccomandato impostare un insieme separato di queste directory per ogni applicazione che userà Smarty.

Assicuratevi di conoscere il percorso della document root del vostro web server. Nel nostro esempio, la document root è `/web/www.mydomain.com/docs/`. Le directory di Smarty vengono accedute solo dalle librerie di Smarty e mai direttamente dal browser. Tuttavia, per evitare problemi di sicurezza, si raccomanda di mettere queste directory *al di fuori* della document root.

Per la nostra installazione di esempio, imposteremo l'ambiente di Smarty per una applicazione di guest book. Abbiamo scelto un'applicazione al solo scopo di avere una convenzione per il nome delle directory. Potete usare lo stesso ambiente per qualsiasi applicazione, soltanto sostituendo "guestbook" con il nome della vostra applicazione. Metteremo le nostre directory di Smarty sotto `/web/www.mydomain.com/smarty/guestbook/`.

Avrete bisogno di almeno un file sotto la document root, e quello sarà lo script a cui può accedere ilbrowser. Lo chiameremo `index.php`, e lo metteremo in una sottodirectory della document root chiamata `/guestbook/`.

Nota tecnica

Conviene impostare il web server in modo che "index.php" possa essere identificato come indice di default della directory, così se provate a richiedere "http://www.example.com/guestbook/", lo script index.php verrà eseguito senza "index.php" nell'URL. In Apache questo può essere impostato aggiungendo "index.php" alla fine dell'impostazione DirectoryIndex (le voci vanno separate con uno spazio l'una dall'altra).

Diamo un'occhiata alla struttura dei file fino ad ora:

Example 2.6. Esempio di struttura dei file

```
/usr/local/lib/php/Smarty/Smarty.class.php
/usr/local/lib/php/Smarty/Smarty_Compiler.class.php
/usr/local/lib/php/Smarty/Config_File.class.php
/usr/local/lib/php/Smarty/debug.tpl
/usr/local/lib/php/Smarty/internals/*.php
/usr/local/lib/php/Smarty/plugins/*.php

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

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

Smarty necessita del diritto di scrittura su *$compile_dir* e su *$cache_dir*, quindi assicuratevi che l'utente del web server possa scriverci sopra. Di solito si tratta dell'utente "nobody" e gruppo "nobody". Per utenti di OS X, il default è utente "www" e gruppo "www". Se usate Apache, potete guardare nel file httpd.conf (di solito in "/usr/local/apache/conf/") per vedere quale utente e gruppo vengono usati.

Example 2.7. Impostazione dei permessi sui file

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

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

Nota tecnica

chmod 770 vi garantisce una notevole sicurezza, in quanto consente solo all'utente e al gruppo "nobody" l'accesso in lettura/scrittura alle directory. Se volete consentire la lettura a chiunque (soprattutto per vostra comodità, se volete guardare questi file), potete impostare invece 775.

Ora dobbiamo creare il file index.tpl che Smarty caricherà. Si troverà nella directory $template_dir.

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

```
{* Smarty *}

Hello, {$name}!
```

Nota tecnica

{* Smarty *} è un commento del template. Non è obbligatorio, ma è buona pratica iniziare tutti i file di template con questo commento. Rende semplice riconoscere il file, indipendentemente dalla sua estensione. Ad esempio, un editor di testo potrebbe riconoscere il file ed attivare una particolare evidenziazione della sintassi.

Ora editiamo index.php. Creeremo un'istanza di Smarty, valorizzeremo una variabile del template e faremo il display del file index.tpl. Nel nostro ambiente di esempio, "/usr/local/lib/php/Smarty" si trova nell'include_path. Assicuratevi che sia così anche per voi, oppure usate percorsi assoluti.

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

```
<?php

// caricamento delle librerie di Smarty
require('Smarty.class.php');

$smarty = new Smarty;

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

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

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

Nota tecnica

Nell'esempio stiamo usando percorsi assoluti per tutte le directory di Smarty. Se `/web/www.example.com/smarty/guestbook/` fa parte dell'include_path di PHP, questo non è necessario. Comunque, è più efficiente e (per esperienza) meno soggetto ad errori usare percorsi assoluti. Questo vi garantisce che Smarty prenda i file dalle directory giuste.

Ora richiamate il file `index.php` dal browser. Dovreste vedere "Hello, Ned!"

Avete completato l'installazione base di Smarty!

Installazione avanzata

Questo è il seguito della installazione di base, siete pregati di leggerla prima!

Un modo leggermente più flessibile di installare Smarty è di estendere la classe e inizializzare il vostro ambiente di Smarty. Così, invece di impostare ripetutamente i percorsi delle directory, riassegnare le stesse variabili ecc., possiamo farlo in un unico punto. Creiamo una nuova directory "/php/includes/guestbook/" e un file chiamato `setup.php`. Nel nostro ambiente di esempio, "/php/includes" fa parte dell'include_path. Assicuratevi che sia così anche per voi, oppure usate percorsi assoluti.

Example 2.10. Edit di /php/includes/guestbook/setup.php

```
<?php

// caricamento delle librerie di Smarty
require('Smarty.class.php');

// Il file setup.php è un buon punto dal quale caricare
// le librerie necessarie all'applicazione, quindi
// potete farlo qui. Ad esempio:
// require('guestbook/guestbook.lib.php');

class Smarty_GuestBook extends Smarty {

   function Smarty_GuestBook()
   {

        // Costruttore della Classe. Questi dati vengono automaticamente impostati
        // per ogni nuova istanza.

        $this->Smarty();

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

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

}
?>
```

Ora modifichiamo il file index.php per usare setup.php:

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

```
<?php

require('guestbook/setup.php');

$smarty = new Smarty_GuestBook;

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

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

Come potete vedere, è molto semplice creare un'istanza di Smarty, basta usare Smarty_GuestBook che inizializza automaticamente tutto ciò che serve alla nostra applicazione.

Part II. Smarty Per Progettisti di Template

Table of Contents

Chapter 3. Sintassi di base

Tutti i tag dei template di Smarty sono racchiusi fra delimitatori. Per default i delimitatori sono { e }, ma possono essere cambiati.

Per questi esempi supporremo di usare i delimitatori di default. In Smarty, tutto il contenuto al di fuori dei delimitatori viene mostrato come contenuto statico, senza modifiche. Quando Smarty incontra i tag dei template, cerca di interpretarli, e visualizza al loro posto l'output relativo.

Commenti

I commenti nei template sono preceduti e seguiti da asterischi, i quali sono a loro volta compresi dai tag delimitatori: {* questo è un commento *} I commenti di Smarty non vengono visualizzati nell'output del template. Sono usati per note interne al template.

Example 3.1. Commenti

```
{* Smarty *}

{* includiamo il file dell'header *}
{include file="header.tpl"}

{include file=$includeFile}

{include file=#includeFile#}

{* visualizziamo una casella a discesa *}
<select name="company">
{html_options values=$vals selected=$selected output=$output}
</select>
```

Funzioni

Ogni tag di Smarty può stampare una variable o chiamare una qualche funzione. Le funzioni vengono richiamate richiudendo la funzione e i suoi attributi fra i delimitatori, così: {nomefunzione attr1="val" attr2="val"}.

Example 3.2. sintassi delle funzioni

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

{include file="header.tpl"}

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

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

Sia le funzioni incorporate che le funzioni utente hanno la stessa sintassi nel template. Le funzioni incorporate sono il cuore pulsante di Smarty, ad esempio **if**, **section** e **strip**. Non possono essere modificate. Le funzioni utente sono funzioni addizionali sviluppate attraverso i plugin. Potete modificarle a piacere, e potete crearne di nuove. **html_options** e **html_select_date** sono esempi di funzioni utente.

Attributi

La maggior parte delle funzioni accetta attributi che specificano o modificano il loro comportamento. Gli attributi delle funzioni Smarty assomigliano agli attributi HTML. I valori statici non hanno bisogno di essere racchiusi fra virgolette, ma è raccomandato farlo per le stringhe. Possono essere usate anche variabili, che non devno essere fra virgolette.

Alcuni attributi richiedono valori booleani (vero o falso). Per specificarli si possono usare i seguenti valori, senza virgolette: `true`, `on`, e `yes`, oppure `false`, `off`, e `no`.

Example 3.3. sintassi per gli attributi delle funzioni

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

{include file=$includeFile}

{include file=#includeFile#}

{html_select_date display_days=yes}

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

Incorporare variabili fra virgolette

Smarty riconosce le variabili incorporate nelle stringhe fra virgolette (") se contengono solo numeri, lettere, underscore (_) e parentesi quadre ([]). Se sono presenti altri caratteri (punti, riferimento a oggetti, ecc.) la variabile deve essere posta tra backticks (`). I backticks si possono ottenere digitando ALT+96 (sul tastierino numerico).

Example 3.4. embedded quotes syntax

```
ESEMPI DI SINTASSI:
{func var="test $foo test"}       <-- riconosce $foo
{func var="test $foo_bar test"}   <-- riconosce $foo_bar
{func var="test $foo[0] test"}    <-- riconosce $foo[0]
{func var="test $foo[bar] test"}  <-- riconosce $foo[bar]
{func var="test $foo.bar test"}   <-- riconosce $foo (non $foo.bar)
{func var="test `$foo.bar` test"} <-- riconosce $foo.bar

ESEMPI PRATICI:
{include file="subdir/$tpl_name.tpl"} <-- sostituisce $tpl_name col valore relativ
{cycle values="one,two,`$smarty.config.myval`"} <-- necessita di backticks
```

Funzioni aritmetiche

Le funzioni aritmetiche possono essere applicate direttamente ai valori delle variabili.

Example 3.5. esempi di funzioni aritmetiche

```
{$foo+1}

{$foo*$bar}

{* alcuni esempi più complessi *}

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

Evitare il parsing di Smarty

A volte è desiderabile o necessario che Smarty ignori sezioni che altrimenti verrebbero analizzate. Un esempio tipico è l'incorporazione di codice Javascript o CSS in un template. Il problema nasce dal fatto che questi linguaggi utilizzano i caratteri { e } che per Smarty sono i delimitatori di default.

La cosa più semplice sarebbe evitare queste situazioni tenendo il codice Javascript e CSS separato in appositi file e usando i collegamenti standard dell'HTML per recuperarli.

E' possibile includere contenuto letterale usando blocchi di questo tipo: {literal} .. {/literal}. Potete anche usare, in modo simile alle entità HTML, {ldelim},{rdelim} oppure {$smarty.ldelim},{$smarty.rdelim} per visualizzare i delimitatori senza che Smarty ne analizzi il contenuto.

Spesso risulta semplicemente conveniente cambiare il $left_delimiter ed il $right_delimiter di Smarty.

Example 3.6. esempio di cambio dei delimitatori

```
<?php

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

?>
```

Dove example.tpl è:

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

Chapter 4. Variabili

Smarty usa parecchi tipi diversi di variabili. Il tipo di variabile dipende da quale simbolo si usa come prefisso (o come delimitatore).

In Smarty le variabili possono essere visualizzate direttamente oppure usate come argomenti per gli attributi e i modificatori delle funzioni, oppure in espressioni condizionali, ecc. Per stampare una variabile, è sufficiente includerla fra i delimitatori in modo che sia l'unica cosa contenuta fra essi. Esempi:

```
{$Name}

{$Contacts[row].Phone}

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

Variabili valorizzate da PHP

Le variabili valorizzate da PHP sono referenziate facendole precedere da un segno di dollaro $. Anche le variabili valorizzate internamente al template con la funzione assign vengono visualizzate in questo modo.

Example 4.1. variabili valorizzate

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

Questo visualizzerà:

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

Array associativi

Potete fare riferimento ad array associativi valorizzati da PHP specificando l'indice dopo il punto '.'

Example 4.2. accesso ad array associativi

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

dove il contenuto di index.tpl è:

```
{$Contacts.fax}<br />
{$Contacts.email}<br />
{* ovviamente si possono usare anche array multidimensionali *}
{$Contacts.phone.home}<br />
{$Contacts.phone.cell}<br />
```

questo visualizzerà:

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

Array con indici numerici

Potete referenziare gli array con il loro indice, come in PHP.

Example 4.3. accesso agli array per indice numerico

```
<?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');

?>
```

dove index.tpl è:

```
{$Contacts[0]}<br />
{$Contacts[1]}<br />
{* anche qui si possono usare array multidimensionali *}
{$Contacts[2][0]}<br />
{$Contacts[2][1]}<br />
```

Questo visualizzerà:

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

Oggetti

Le proprietà di oggetti valorizzate da PHP possono essere referenziate indicando il nome della proprietà dopo il simbolo '->'

Example 4.4. accesso alle proprietà degli oggetti

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

Questo visualizzerà:

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

Variabili caricate da file di configurazione

Le variabili caricate dai file di configurazione sono referenziate racchiudendole fra due simboli cancelletto (#), oppure attraverso la variabile $smarty.config. La seconda sintassi è utile per includerle in valori di attributi indicati fra virgolette.

foo.conf:

Example 4.5. variabili di configurazione

```
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: (sintassi alternativa)

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

questo è l'output prodotto da entrambi gli esempi:

```
<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>
```

Le variabili dei file di configurazione non possono essere usate fino a dopo che sono state caricate dal file che le contiene. Questa procedura viene spiegata più avanti in questo documento, in **config_load**.

La variabile riservata {$smarty}

La variabile riservata {$smarty} può essere usate per accedere a parecchie variabili speciali del template. Quella che segue è la lista completa.

Variabili della richiesta HTTP

Alle variabili get, post, cookies, server, environment e session si può accedere come mostrato negli esempi qui sotto:

Example 4.6. visualizzazione delle variabili request

```
{* visualizza il valore di "page" dall'URL (GET) http://www.example.com/index.php?
{$smarty.get.page}

{* visualizza la variabile "page" da un modulo (POST) *}
{$smarty.post.page}

{* visualizza il valore del cookie "username" *}
{$smarty.cookies.username}

{* visualizza la variabile del server "SERVER_NAME" *}
{$smarty.server.SERVER_NAME}

{* visualizza la variabile di ambiente "PATH" *}
{$smarty.env.PATH}

{* visualizza la variabile di sessione PHP "id" *}
{$smarty.session.id}

{* visualizza la variabile "username" dalla fusione di get/post/cookies/server/env
{$smarty.request.username}
```

Note

Per motivi storici si può accedere direttamente a {$SCRIPT_NAME}, sebbene {$smarty.server.SCRIPT_NAME} sia la maniera consigliata per ottenere questo valore.

{$smarty.now}

Si può accedere al timestamp corrente con {$smarty.now}. Questo numero rappresenta il numero di secondi passati dalla cosiddetta Epoch (1° gennaio 1970) e può essere passato direttamente al modificatore date_format per la visualizzazione.

Example 4.7. uso di {$smarty.now}

```
{* uso del modificatore date_format per mostrare data e ora attuali *}
{$smarty.now|date_format:"%Y-%m-%d %H:%M:%S"}
```

{$smarty.const}

Può essere usato per accedere direttamente alle costanti PHP.

Example 4.8. uso di {$smarty.const}

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

{$smarty.capture}

Si può accedere all'output catturato attraverso il costrutto {capture}..{/capture} con la variabile {$smarty}. Consultare la sezione capture per avere un esempio.

{$smarty.config}

La variabile {$smarty} può essere usata per referenziare le variabili di configurazione caricate. {$smarty.config.foo} è sinonimo di {#foo#}. Consultare la sezione config_load per avere un esempio.

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

La variabile {$smarty} può essere usata per referenziare le proprietà dei loop 'section' e 'foreach'. Vedere la documentazione di section e foreach.

{$smarty.template}

Questa variabile contiene il nome del template attualmente in fase di elaborazione.

{$smarty.version}

Questa variabile contiene la versione di Smarty con cui il template è stato compilato.

{$smarty.ldelim}

Questa variabile è usata per stampare il delimitatore sinistro di Smarty in modo letterale, cioè senza che venga interpretato come tale. Vedere anche {ldelim},{rdelim}.

{$smarty.rdelim}

Questa variabile è usata per stampare il delimitatore destro di Smarty in modo letterale, cioè senza che venga interpretato come tale. Vedere anche {ldelim},{rdelim}.

Chapter 5. Modificatori delle variabili

I modificatori delle variabili si possono applicare alle variabili, alle funzioni utente o a stringhe. Per applicare un modificatore bisogna indicare il valore seguito da | (pipe) e dal nome del modificatore. Un modificatore può accettare parametri addizionali che modificano il suo comportamento. Questi parametri seguono il nome del modificatore e sono separati da : (due punti).

Example 5.1. esempio di modificatore

```
{* Mettere il titolo in maiuscolo *}
<h2>{$title|upper}</h2>

{* Troncare il topic a 40 caratteri usando ... alla fine *}
Topic: {$topic|truncate:40:"..."}

{* Formattare una stringa indicata direttamente *}
{"now"|date_format:"%Y/%m/%d"}

{* Applicare un modificatore ad una funzione utente *}
{mailto|upper address="me@domain.dom"}
```

Se applicate un modificatore ad un array invece che ad un singolo valore, il modificatore verrà applicato ad ogni valore dell'array. Se volete che il modificatore lavori sull'intero array considerandolo un valore unico, dovete premettere al nome del modificatore un simbolo @, così: `{$articleTitle|@count}` (questo stampa il numero di elementi nell'array $articleTitle).

I modificatori possono essere autocaricati dalla $plugins_dir (vedere Convenzioni di nomenclatura) oppure possono essere registrati esplicitamente (vedere register_modifier). Inoltre tutte le funzioni php possono essere usate implicitamente come modificatori. (L'esempio `@count` visto sopra usa in realtà la funzione php count e non un modificatore di Smarty). L'uso delle funzioni php come modificatori porta con sé due piccoli trabocchetti: Primo: A volte l'ordine dei parametri delle funzioni non è quello desiderato (`{"%2.f"|sprintf:$float}` funziona, ma non è molto intuitivo. Più facile è `{$float|string_format:"%2.f"}`, che è fornito da Smarty). Secondo: con $security attivato, tutte le funzioni php che si vogliono usare come modificatori devono essere dichiarate affidabili nell'array $security_settings['MODIFIER_FUNCS'].

capitalize

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	booleano	No	false	Stabilisce se le parole contenenti cifre verranno trasformate in maiuscolo

Si usa per mettere in maiuscolo la prima lettera di tutte le parole nella variabile.

Example 5.2. capitalize

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

count_characters

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	booleano	No	false	Stabilisce se gli spazi devono essere inclusi nel conteggio.

E' usato per contare il numero di caratteri contenuti in una variabile.

Example 5.3. count_characters

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

```
Cold Wave Linked to Temperatures.
29
33
```

cat

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	stringa	No	*vuoto*	Valore che viene concatenato alla variabile.

Questo valore viene concatenato alla variabile data.

Example 5.4. cat

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

Dove index.tpl è:

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

Questo stamperà:

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

count_paragraphs

Si usa per contare il numero di paragrafi contenuti in una variabile.

Example 5.5. count_paragraphs

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

Dove index.tpl è:

```
{$articleTitle}
{$articleTitle|count_paragraphs}
```

Questo stamperà:

```
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

E' usato per contare il numero di frasi contenute in una variabile.

Example 5.6. count_sentences

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

count_words

E' usato per contare il numero di parole contenute in una variabile.

Example 5.7. count_words

```
<?php

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

?>
```

Dove index.tpl è:

```
{$articleTitle}
{$articleTitle|count_words}
```

Questo stamperà:

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

date_format

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	stringa	No	%b %e, %Y	E' il formato per la data in output.
2	stringa	No	nessuno	E' la data di default se la variabile in input è vuota.

Questo modificatore formatta una data e un'ora nel formato dato di strftime(). Le date possono essere passate a Smarty come timestamp Unix, timestamp MySql o una qualsiasi stringa contenente mese giorno anno (riconoscibile da strtotime). I progettisti quindi possono usare date_format per avere il pieno controllo della formattazione della data. Se la data passata a date_format è vuota ed è presente un secondo parametro, verrà usato questo come data da formattare.

Example 5.8. date_format

```
<?php

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

?>
```

Dove index.tpl è:

```
{$smarty.now|date_format}
{$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"}
```

Questo stamperà:

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

Parametri di conversione di date_format:

- %a - nome abbreviato del giorno della settimana in base all'impostazione di "locale"
- %A - nome intero del giorno della settimana in base all'impostazione di "locale"
- %b - nome abbreviato del mese in base all'impostazione di "locale"
- %B - nome intero del mese in base all'impostazione di "locale"
- %c - rappresentazione preferita di ora e data in base all'impostazione di "locale"
- %C - numero del secolo (l'anno diviso per 100 e troncato ad intero, range da 00 a 99)
- %d - giorno del mese come numero decimale (range da 00 a 31)
- %D - corrisponde a %m/%d/%y

- %e - giorno del mese come numero decimale; la cifra singola è preceduta da uno spazio (range da 1 a 31)
- %g - anno in base alle settimane, su due cifre [00,99]
- %G - anno in base alle settimane, su quattro cifre [0000,9999]
- %h - corrisponde a %b
- %H - ora come numero decimale, su 24 ore (range da 00 a 23)
- %I - ora come numero decimale, su 12 ore (range da 01 a 12)
- %j - giorno dell'anno come numero decimale (range da 001 a 366)
- %k - ora (su 24 ore) con le cifre singole precedute da spazio (range da 0 a 23)
- %l - ora (su 12 ore) con le cifre singole precedute da spazio (range da 1 a 12)
- %m - mese come numero decimale (range da 01 a 12)
- %M - minuto come numero decimale
- %n - carattere di "a capo"
- %p - `am' o `pm' (antimeridiane o postmeridiane) in base all'ora, o valore corrispondente in base all'impostazione di "locale"
- %r - ora completa nella notazione con a.m. e p.m.
- %R - ora completa nella notazione su 24 ore
- %S - secondi come numero decimale
- %t - carattere di tabulazione
- %T - ora corrente, con formato equivalente a %H:%M:%S
- %u - giorno della settimana come numero decimale [1,7], in cui 1 rappresenta Lunedì
- %U - numero della settimana nell'anno come numero decimale, partendo dalla prima Domenica come primo giorno della prima settimana
- %V - Il numero della settimana ISO 8601:1988 come numero decimale, range da 01 a 53, dove la settimana 1 è la prima ad avere almeno 4 giorni nell'anno, e Lunedì è il primo giorno della settimana.
- %w - giorno della settimana come numero decimale, dove la Domenica è 0
- %W - numero della settimana nell'anno come numero decimale, partendo dal primo lunedì come primo giorno della prima settimana
- %x - rappresentazione preferita della data secondo l'impostazione di "locale", senza l'ora
- %X - rappresentazione preferita dell'ora secondo l'impostazione di "locale", senza data
- %y - anno come numero decimale su due cifre (range da 00 a 99)
- %Y - anno come numero decimale su quattro cifre

- %Z - time zone o nome o abbreviazione
- %% - il carattere `%'

Nota per i programmatori

date_format è fondamentalmente un involucro per la funzione PHP strftime(). Potete avere disponibili più o meno specificatori di conversione, in base alla funzione strftime() del sistema su cui PHP è stato compilato. Controllate le pagine di manuale del vostro sistema per una lista completa degli specificatori validi.

default

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	stringa	No	*vuoto*	E' il valore di default da stampare se la variabile è vuota.

E' usato per impostare un valore di default per una variabile. Se la variabile è vuota o non impostata, il valore di default viene stampato al suo posto. Prende un parametro.

Example 5.9. default

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

escape

Posizione del Parametro	Tipo	Obbligatorio	Valori possibili	Default	Descrizione
1	stringa	No	html,htmlall,url,quotes,hex,hexentity,javascript	html	E' il tipo di escape da utilizzare.

E' usato per fare un escape di tipo html, url, su apici per una variabile su cui non sia già stato fatto l'escape, hex (esadecimale), hexentity o javascript. Per default viene applicato un escape di tipo html.

Example 5.10. escape

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

indent

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	intero	No	4	Stabilisce di quanti caratteri deve essere l'indentazione.
2	stringa	No	(uno spazio)	Questo è il carattere usato per l'indentazione.

Questo modificatore effettua un'indentazione della stringa ad ogni riga, per default di 4 caratteri. Come parametro opzionale si può specificare di quanti caratteri deve essere l'indentazione. Si può indicare anche, come secondo parametro opzionale, quale carattere usare per l'indentazione (usare "\t" per il tabulatore).

Example 5.11. indent

```
<?php

$smarty = new Smarty;
$smarty->assign('articleTitle', 'NJ judge to rule on nude beach.
Sun or rain expected today, dark tonight.
Statistics show that teen pregnancy drops off significantly after 25.');
$smarty->display('index.tpl');

?>
```

Dove index.tpl è:

```
{$articleTitle}

{$articleTitle|indent}

{$articleTitle|indent:10}

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

Questo stamperà:

```
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

Si usa per trasformare una variabile in lettere minuscole.

Example 5.12. lower

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

nl2br

Tutti i caratteri di interruzione di linea verranno convertiti in tag
 nella variabile data. E' equivalente alla funzione PHP nl2br().

Example 5.13. nl2br

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

regex_replace

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	stringa	Sì	*nessuno*	E' l'espressione regolare da sostituire.
2	stringa	Sì	*nessuno*	E' la stringa di testo da usare per la sostituzione.

Un 'trova e sostituisci' di una espressione regolare su una variabile. Usare la sintassi per preg_replace() dal manuale PHP.

Example 5.14. regex_replace

```
<?php

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

?>
```

Dove index.tpl è:

```
{* sostituisce i carriage return, i tab e gli a capo con uno spazio *}

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

Questo stamperà:

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

replace

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	stringa	Sì	*nessuno*	E' la stringa di testo da sostituire.
2	stringa	Sì	*nessuno*	E' la stringa di testo da usare per la sostituzione.

Una semplice ricerca e sostituzione su una variabile.

Example 5.15. replace

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

```
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

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	stringa	No	*uno spazio*	E' ciò che viene inserito fra i caratteri della variabile.

spacify è un modo per inserire uno spazio fra tutti i caratteri di una variabile. E' possibile, opzionalmente, passare un diverso carattere (o stringa) da inserire.

Example 5.16. spacify

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

Dove index.tpl è:

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

Questo stamperà:

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

string_format

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	stringa	Sì	*nessuno*	E' il formato da usare. (sprintf)

Questo è un modo di formattare stringhe, ad esempio per i numeri decimali e altro. Utilizzare la sintassi della funzione PHP sprintf().

Example 5.17. string_format

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

```
23.5787446
23.58
24
```

strip

Sostituisce tutte le sequenze di spazi, a capo e tabulatori con un singolo spazio o con la stringa fornita.

Nota

Se volete fare lo strip su blocchi di testo del template, usate la funzione strip.

Example 5.18. strip

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

strip_tags

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	booleano	No	true	Stabilisce se i tag saranno sostituiti con ' ' (true) o con '' (false)

Questo elimina i tag di markup, cioè fondamentalmente qualsiasi cosa compresa fra < and >.

Example 5.19. strip_tags

```
<?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');

?>
```

Dove index.tpl è:

```
{$articleTitle}
{$articleTitle|strip_tags} {* equivale a {$articleTitle|strip_tags:true} *}
{$articleTitle|strip_tags:false}
```

Questo stamperà:

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

truncate

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	intero	No	80	Stabilisce a quanti caratteri effettuare il troncamento.
2	stringa	No	...	Testo da aggiungere in fondo quando c'è troncamento.
3	booleano	No	false	Stabilisce se troncare dopo una parola (false), o al carattere esatto (true).

Effettua il troncamento di una variabile ad un certo numero di caratteri, per default 80. Come secondo parametro opzionale potete specificare una stringa di testo da mostrare alla fine se la variabile è stata

troncata. Questi caratteri non vengono conteggiati nella lunghezza della stringa troncata. Per default, truncate cercherà di tagliare la stringa al termine di una parola. Se invece volete effettuare il troncamento alla lunghezza esatta in caratteri, passate il terzo parametro opzionale come true.

Example 5.20. truncate

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

```
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

Si usa per trasformare una variabile in maiuscolo.

Example 5.21. upper

```
<?php

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

?>
```

Dove index.tpl è:

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

Questo stamperà:

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

wordwrap

Posizione del Parametro	Tipo	Obbligatorio	Default	Descrizione
1	intero	No	80	Stabilisce la larghezza della colonna.
2	stringa	No	\n	Questa è la stringa usata per andare a capo.
3	booleano	No	false	Stabilisce se andare a capo dopo una parola intera (false), o al carattere esatto (true).

Dispone una stringa su più righe usando come riferimento una certa larghezza di colonna, per default 80. Come secondo parametro opzionale potete specificare una stringa da usare per separare le righe (il default è \n). Per default, wordwrap cercherà di andare a capo dopo una parola intera. Se volete che vada a capo all'esatta larghezza in caratteri, passate il terzo parametro opzionale come true.

Example 5.22. wordwrap

```
<?php

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

?>
```

Dove index.tpl è:

```
{$articleTitle}

{$articleTitle|wordwrap:30}

{$articleTitle|wordwrap:20}

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

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

Questo stamperà:

```
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<br />
years.

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

Chapter 6. Combinare i modificatori

Potete applicare qualsiasi numero di modificatori ad una variabile. Verranno eseguiti nell'ordine in cui li avete indicati, da sinistra a destra. Devono essere separati con un carattere | (pipe).

Example 6.1. combinare i modificatori

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

index.tpl:

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

L'esempio sopra stamperà:

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

Chapter 7. Funzioni incorporate

Smarty è dotato di numerose funzioni incorporate. Queste funzioni sono integrate nel linguaggio del template: non è possibile creare funzioni utente con gli stessi nomi, e nemmeno modificare le funzioni incorporate.

capture

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
name	stringa	no	*default*	Nome del blocco catturato
assign	stringa	no	*nessuno*	Nome della variabile cui assegnare l'output catturato

capture si usa per intercettare l'output del template assegnandolo ad una variabile invece di visualizzarlo. Qualsiasi contenuto compreso fra {capture name="foo"} e {/capture} viene aggiunto alla variabile specificata nell'attributo name. Il contenuto catturato può essere usato nel template utilizzando la variabile speciale $smarty.capture.foo dove foo è il nome passato nell'attributo name. Se non fornite un attributo name, verrà usato "default". Tutti i comandi {capture} devono essere chiusi con {/capture}. E' possibile nidificarli.

Nota tecnica

Le versioni da 1.4.0 a 1.4.4 di Smarty mettevano il contenuto catturato nella variabile $return. A partire dalla 1.4.5 si utilizza l'attributo name, quindi modificate i vostri template di conseguenza.

Caution

Fate attenzione se catturate l'output di **insert**. Se avete il caching attivato e usate comandi **insert** che vi aspettate vengano eseguiti nel contenuto in cache, non catturate questo contenuto.

Example 7.1. catturare il contenuto del template

```
{* vogliamo stampare la riga di tabella solo se c'è del contenuto *}
{capture name=banner}
{include file="get_banner.tpl"}
{/capture}
{if $smarty.capture.banner ne ""}
 <tr>
  <td>
   {$smarty.capture.banner}
  </td>
 </tr>
{/if}
```

config_load

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
file	stringa	sì	*nessuno*	Nome del file di configurazione da importare
section	stringa	no	*nessuno*	Nome della sezione da caricare
scope	stringa	no	*local*	Campo di applicazione delle variabili caricate, che può essere local, parent o global. local significa che le variabili vengono caricate nel contesto del template locale. parent significa che le variabili vengono caricate sia nel contesto locale che nel template genitore che lo ha chiamato. global significa che le variabili sono disponibili a tutti i template.
global	booleano	no	*false*	Se le variabili sono visibili o meno al template genitore: equivale a scope=parent. NOTA: Questo attributo è deprecato per via dell'esistenza dell'attributo scope, ma è ancora supportato. Se è presente scope, questo valore è ignorato.

Questa funzione è usata per caricare variabili nel template da un file di configurazione. Vedere Config Files per maggiori informazioni.

Example 7.2. funzione config_load

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

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

I file di configurazione possono contenere sezioni. Potete caricare variabili da una sezione con l'attributo aggiuntivo *section*.

Note

Le sezioni dei file di configurazione e la funzione incorporata dei template chiamata *section* non hanno nulla a che fare fra di loro, hanno soltanto lo stesso nome.

Example 7.3. funzione config_load con section

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

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

foreach,foreachelse

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
from	array	sì	*nessuno*	Array sul quale viene eseguito il ciclo
item	stringa	sì	*nessuno*	Nome della variabile che rappresenta l'elemento attuale
key	stringa	no	*nessuno*	Nome della variabile che rappresenta la chiave attuale
name	stringa	no	*nessuno*	Nome del ciclo foreach per l'accesso alle sue proprietà

I cicli *foreach* sono un'alternativa ai cicli *section*. *foreach* si usa per ciclare su un singolo array associativo. La sintassi di *foreach* è molto più semplice di *session*, ma in compenso può essere usata solo per un array singolo. I tag *foreach* devono essere chiusi con */foreach*. I parametri obbligatori sono *from* e *item*. Il nome del ciclo foreach può essere quello che preferite, composto di lettere, numeri e underscore. I cicli *foreach* possono essere nidificati, ma i nomi dei cicli nidificati devono essere diversi tra di loro. La variabile *from* (di solito un array di valori) determina quante volte verrà eseguito il ciclo *foreach*. *foreachelse* viene eseguito quando non ci sono valori nella variabile *from*.

Example 7.4. foreach

```
{* questo esempio stamperà tutti i valori dell'array $custid *}
{foreach from=$custid item=curr_id}
 id: {$curr_id}<br>
{/foreach}

OUTPUT:

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

Example 7.5. foreach con key

```
{* key contiene la chiave per ogni valore del ciclo

l'assegnazione può essere qualcosa del genere:

$smarty->assign("contacts", array(array("phone" => "1", "fax" => "2", "cell" => "3
      array("phone" => "555-4444", "fax" => "555-3333", "cell" => "760-1234")));

*}

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

OUTPUT:

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

I cicli foreach hanno anche le proprie variabili che gestiscono le proprietà del foreach. Queste vengono indicate così: {$smarty.foreach.foreachname.varname}, dove foreachname è il nome indicato come attributo *name* del foreach

iteration

iteration si usa per mostrare l'iterazione corrente del ciclo.

iteration comincia sempre per 1 ed è incrementata di uno ad ogni iterazione.

first

first vale true quando l'iterazione attuale è la prima del ciclo.

last

last vale true quando l'iterazione attuale è l'ultima del ciclo.

show

show si usa come parametro per il foreach. *show* è un valore booleano, true o false. Quando è false, il foreach non verrà visualizzato. Se è presente un foreachelse, verrà visualizzato al suo posto.

total

total si usa per visualizzare il numero di iterazioni che il ciclo foreach effettuerà. Può essere usato all'interno o dopo il foreach.

include

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
file	stringa	sì	*nessuno*	Nome del file di template da includere
assign	stringa	no	*nessuno*	Nome della variabile cui sarà assegnato l'output dell'include
[variabile ...]	[tipo variabile]	no	*nessuno*	Variabile da passare localmente al template

I tag include sono usati per includere altri template in quello attuale. Tutte le variabili del template corrente sono disponibili anche nel template incluso. Il tag include deve comprendere l'attributo "file", che contiene il percorso del template da includere.

Opzionalmente si può passare l'attributo *assign*, che specifica un nome di variabile del template alla quale sarà assegnato l'output dell'*include*, invece di essere visualizzato.

Example 7.6. funzione include

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

{* qui va il corpo del template *}

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

Potete anche passare variabili ai template inclusi sotto forma di attributi. Queste variabili saranno disponibili soltanto nello scope del file incluso. Le variabili attributo prevalgono su quelle del template attuale in caso di omonimia.

Example 7.7. funzione include con passaggio di variabili

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

{* qui va il corpo del template *}

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

Usate la sintassi delle risorse dei template per includere file esterni alla directory $template_dir.

Example 7.8. esempi di funzione include con le risorse dei template

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

{* percorso assoluto (come sopra) *}
{include file="file:/usr/local/include/templates/header.tpl"}

{* percorso assoluto su windows (NECESSARIO usare il prefisso "file:") *}
{include file="file:C:/www/pub/templates/header.tpl"}

{* include da una risorsa chiamata "db" *}
{include file="db:header.tpl"}
```

include_php

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
file	stringa	sì	*nessuno*	Nome del file php da includere
once	booleano	no	*true*	Se includere o no il file php più di una volta nel caso venga richiesto più volte
assign	stringa	no	*nessuno*	Nome della variabile cui sarà assegnato l'output di include_php

Nota Tecnica

include_php è deprecato da Smarty, in quanto potete ottenere la stessa funzionalità attraverso una funzione utente. L'unica ragione per usare include_php è se avete una reale necessità di tenere fuori la funzione php dalla directory dei plugin o dal vostro codice applicativo. Vedere l'esempio di template a componenti per i dettagli.

i tag include_php sono usati per includere uno script php nel template. Se la security è abilitata, lo script php si deve trovare nel percorso di $trusted_dir. Il tag include_php deve avere l'attributo "file", che contiene il percorso al file da includere, che può essere assoluto relativo alla directory $trusted_dir.

include_php è un ottimo modo per gestire template a componenti, e tiene il codice PHP separato dai file dei template. Diciamo che abbiamo un template che mostra la navigazione del nostro sito, che viene prelevata dinamicamente da un database. Possiamo tenere la logica PHP che ottiene il contenuto del database in una directory separata, ed includerla in cima al template. Ora possiamo includere questo template ovunque senza preoccuparci che l'applicazione abbia preventivamente caricato i dati del database.

Per default, i file php sono inclusi una sola volta, anche se richiesti più volte nel template. Potete specificare che devono essere inclusi ogni volta con l'attributo *once*. Se impostate once a false, lo script verrà incluso tutte le volte che viene richiesto nel template.

Opzionalmente potete passare l'attributo *assign*, che specifica un nome di variabile cui sarà assegnato l'output di *include_php*, invece di essere visualizzato.

L'oggetto smarty è disponibile come $this all'interno dello script PHP che viene incluso.

Example 7.9. funzione include_php

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

<?php

 // carichiamo le variabili da un db mysql e le assegnamo al template
 require_once("MySQL.class.php");
 $sql = new MySQL;
 $sql->query("select * from site_nav_sections order by name",SQL_ALL);
 $this->assign('sections',$sql->record);

?>


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

{* percorso assoluto, o relativo a $trusted_dir *}
{include_php file="/path/to/load_nav.php"}

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

insert

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
name	stringa	sì	*nessuno*	Nome della funzione di insert (insert_name)
assign	stringa	no	*nessuno*	Nome della variabile del template cui verrà assegnato l'output
script	stringa	no	*nessuno*	Nome dello script php che viene incluso prima della chiamata alla funzione di insert
[variabile ...]	[tipo variabile]	no	*nessuno*	Variabile da passare alla funzione di insert

I tag insert funzionano praticamente come i tag include, ad eccezione del fatto che i tag insert non vengono messi in cache quando avete il caching del template abilitato. Verranno quindi eseguiti ad ogni chiamata del template.

Diciamo che abbiamo un template con uno spazio banner in cima alla pagina. Il banner può contenere qualsiasi mescolanza di HTML, immagini, flash, ecc., quindi non possiamo usare un link statico, e non

vogliamo che questo contenuto sia messo in cache con la pagina. Ecco quindi l'utilità del tag insert: il template conosce i valori di #banner_location_id# e #site_id# (presi da un file di configurazione), e ha bisogno di chiamare una funzione per ottenere il contenuto del banner.

Example 7.10. funzione insert

```
{* esempio di caricamento di un banner *}
{insert name="getBanner" lid=#banner_location_id# sid=#site_id#}
```

In questo esempio stiamo usando il nome "getBanner" e passiamo i parametri #banner_location_id# e #site_id#. Smarty cercherà una funzione chiamata insert_getBanner() nell'applicazione PHP, passandole i valori di #banner_location_id# e #site_id# come primo argomento in un array associativo. Tutti i nomi di funzioni di insert nell'applicazione devono essere prefissati con "insert_", per evitare possibili conflitti nei nomi di funzione. La nostra funzione insert_getBanner() farà qualcosa con i valori passati e restituirà il risultato, che verrà visualizzato nel templat al posto del tag insert. In questo esempio, Smarty chiamerebbe questa funzione: insert_getBanner(array("lid" => "12345","sid" => "67890")); e visualizzerebbe il risultato restituito al posto del tag insert.

Se fornite l'attributo "assign", l'output del tag insert verrà assegnato a questa variabile invece di essere mostrato nel template. NOTA: assegnare l'output ad una variabile non è molto utile se il caching è abilitato.

Se fornite l'attributo "script", questo script verrà incluso (una volta sola) prima dell'esecuzione della funzione di insert. Questo caso può presentarsi quando la funzione di insert può non esistere ancora, e uno script php deve essere quindi incluso per farla funzionare. Il percorso può essere assoluto o relativo a $trusted_dir. Se la security è abilitata, lo script deve trovarsi in $trusted_dir.

Come secondo argomento viene passato l'oggetto Smarty. In questo modo potete ottenere e modificare informazioni nell'oggetto Smarty dall'interno della funzione di insert.

Nota tecnica

E' possibile avere porzioni di template non in cache. Se avete il caching abilitato, i tag insert non verranno messi in cache. Verranno quindi eseguiti dinamicamente ogni volta che la pagina viene creata, anche se questa si trova in cache. Questo viene utile per cose come banner, sondaggi, situazione del tempo, risultati di ricerche, aree di feedback utenti, ecc.

if,elseif,else

Le istruzioni *{if}* in Smarty hanno praticamente la stessa flessibilità delle istruzioni if PHP, con qualche caratteristica aggiuntiva per il motore di template. Ogni *{if}* deve essere chiuso con un *{/if}*. Sono previsti anche *{else}* e *{elseif}*. Sono riconosciuti tutti gli operatori condizionali di PHP, come ||, *or*, *&&*, *and*, ecc.

Quella che segue è una lista degli operatori riconosciuti, che devono essere separati con degli spazi dagli elementi circostanti. Notate che gli elementi mostrati fra [parentesi quadre] sono opzionali. Quando esistono sono mostrati gli equivalenti in PHP.

Operatore	Alternative	Esempio di sintassi	Significato	Equivalente PHP
==	eq	$a eq $b	uguale	==
!=	ne, neq	$a neq $b	diverso	!=
>	gt	$a gt $b	maggiore di	>
<	lt	$a lt $b	minore di	<
>=	gte, ge	$a ge $b	maggiore o uguale	>=

Operatore	Alternative	Esempio di sintassi	Significato	Equivalente PHP
<=	lte, le	$a le $b	minore o uguale	<=
!	not	not $a	negazione (unario)	!
%	mod	$a mod $b	modulo (resto della divisione)	%
is [not] div by		$a is not div by 4	divisibile per	$a % $b == 0
is [not] even		$a is not even	[non] è un numero pari (unario)	$a % 2 == 0
is [not] even by		$a is not even by $b	livello di raggruppamento [non] pari	($a / $b) % 2 == 0
is [not] odd		$a is not odd	[non] è un numero dispari (unario)	$a % 2 != 0
is [not] odd by		$a is not odd by $b	livello di raggruppamento [non] dispari	($a / $b) % 2 != 0

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

Example 7.11. Istruzioni if

```
{* un esempio con "or" logico *}
{if $name eq "Fred" or $name eq "Wilma"}
 ...
{/if}

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

{* questa sintassi NON funziona, gli operatori condizionali
   devono essere separati con spazi dagli elementi circostanti *}
{if $name=="Fred" || $name=="Wilma"}
 ...
{/if}


{* si possono usare le parentesi *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}
 ...
{/if}

{* potete anche incorporare chiamate a funzioni php *}
{if count($var) gt 0}
 ...
{/if}

{* test su valori pari o dispari *}
{if $var is even}
 ...
{/if}
{if $var is odd}
 ...
{/if}
{if $var is not odd}
 ...
{/if}

{* test se var è divisibile per 4 *}
{if $var is div by 4}
 ...
{/if}

{* test se var è pari, raggruppato per due. Ad es.:
0=pari, 1=pari, 2=dispari, 3=dispari, 4=pari, 5=pari, etc. *}
{if $var is even by 2}
 ...
{/if}

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

ldelim,rdelim

ldelim e rdelim si usano per fare l'escape dei delimitatori del template, nel nostro caso "{" o "}". Potete usare anche {literal}{/literal} per fare l'escape su blocchi di testo. Vedere anche {$smarty.ldelim} e {$smarty.rdelim}

Example 7.12. ldelim, rdelim

```
{* questo stamperà i delimitatori *}

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

L'esempio sopra produrrà:

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

literal

I tag literal vi consentono di far sì che un blocco di dati venga letto "letteralmente". Ciò è utile tipicamente quando avete un blocco javascript o CSS nel quale le parentesi graffe si confonderebbero con i delimitatori del template. Tutto ciò che si trova fra {literal} e {/literal} non viene interpretato, ma visualizzato così com'è. Se avete bisogno di usare tag del template all'interno del blocco literal, considerate la possibilità di usare invece {ldelim}{rdelim} per fare l'escape dei singoli delimitatori.

Example 7.13. tag literal

```
{literal}
 <script type="text/javascript">

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

 </script>
{/literal}
```

php

I tag php vi consentono di incorporare codice php direttamente nel template. Non sarà fatto l'escape, indipendentemente dall'impostazione di $php_handling. Questa funzione è solo per utenti avanzati, normalmente non dovreste averne bisogno.

Example 7.14. tag php

```
{php}
  // inclusione di uno script php
                // direttamente dal template.
  include("/path/to/display_weather.php");
{/php}
```

section,sectionelse

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
name	stringa	sì	*nessuno*	Nome della sezione
loop	[$variable_name]	sì	*nessuno*	Nome della variabile che determina il numero di iterazioni del ciclo
start	intero	no	*0*	L'indice dal quale inizierà il ciclo. Se

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
				il valore è negativo, la posizione di partenza è calcolata dalla fine dell'array. Ad esempio, se ci sono sette valori nell'array da ciclare e start è -2, l'indice di partenza sarà 5. Valori non validi (cioè al di fuori della lunghezza dell'array da ciclare) saranno automaticamente convertiti al valore valido più vicino.
step	intero	no	*1*	Il valore di passo da usare per attraversare l'array da ciclare. Ad esempio, step=2 ciclerà sugli indici 0,2,4, ecc. Se step è negativo il ciclo procederà sull'array all'indietro.
max	intero	no	*nessuno*	Massimo numero di cicli per la sezione.
show	booleano	no	*true*	Stabilisce se mostrare o no la sezione

Le sezioni sono usate per ciclare su array di dati. Tutti i tag *section* devono essere chiusi con */section*. I parametri obbligatori sono *name* e *loop*. Il nome della sezione può essere quello che preferite, formato da lettere, numeri e underscore. Le sezioni possono essere nidificate, ed i nomi delle sezioni nidificate devono essere diversi fra loro. La variabile loop (di solito un array di valori) determina quante volte sarà eseguito il ciclo. Quando stampate una variabile all'interno di una sezione, il nome della sezione deve essere indicato a fianco del nome della variabile fra parentesi quadre []. *sectionelse* viene eseguito quando non ci sono valori nella variabile loop.

Example 7.15. section

```
{* questo esempio stamperà tutti i valori dell'array $custid *}
{section name=customer loop=$custid}
 id: {$custid[customer]}<br>
{/section}

OUTPUT:

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

Example 7.16. variabile loop

```
{* la variabile loop determina soltanto il numero di cicli da ripetere.
   Potete accedere a qualsiasi variabile dal template della sezione.
   In questo esempio presumiamo che $custid, $name e $address siano
   tutti array contenenti lo stesso numero di valori *}
{section name=customer loop=$custid}
 id: {$custid[customer]}<br>
 name: {$name[customer]}<br>
 address: {$address[customer]}<br>
 <p>
{/section}


OUTPUT:

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

Example 7.17. nomi delle sezioni

```
{* come nome della sezione potete usare quello che preferite,
   e viene usato per riferirsi ai dati all'interno della sezione *}
{section name=mydata loop=$custid}
 id: {$custid[mydata]}<br>
 name: {$name[mydata]}<br>
 address: {$address[mydata]}<br>
 <p>
{/section}
```

Example 7.18. sezioni nidificate

```
{* le sezioni possono essere nidificate a qualsiasi profondità. Con
   le sezioni nidificate potete accedere a strutture di dati complesse,
   ad esempio array multidimensionali. In questo esempio, $contact_type[customer]
   è un array di tipi di contatto per il cliente corrente. *}
{section name=customer loop=$custid}
 id: {$custid[customer]}<br>
 name: {$name[customer]}<br>
 address: {$address[customer]}<br>
 {section name=contact loop=$contact_type[customer]}
  {$contact_type[customer][contact]}: {$contact_info[customer][contact]}<br>
 {/section}
 <p>
{/section}


OUTPUT:

id: 1000<br>
name: John Smith<br>
address: 253 N 45th<br>
home phone: 555-555-5555<br>
cell phone: 555-555-5555<br>
e-mail: john@myexample.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@myexample.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@myexample.com<br>
<p>
```

Example 7.19. sezioni e array associativi

```
{* questo è un esempio di stampa di un array associativo
   di dati in una sezione *}
{section name=customer loop=$contacts}
 name: {$contacts[customer].name}<br>
 home: {$contacts[customer].home}<br>
 cell: {$contacts[customer].cell}<br>
 e-mail: {$contacts[customer].email}<p>
{/section}


OUTPUT:

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

Example 7.20. sectionelse

```
{* sectionelse viene eseguito se non ci sono valori in $custid *}
{section name=customer loop=$custid}
 id: {$custid[customer]}<br>
{sectionelse}
 there are no values in $custid.
{/section}
```

Le sezioni hanno anche le proprie variabili di gestione delle proprietà. Vengono indicate così: {$smarty.section.nomesezione.nomevariabile}

Note

A partire da Smarty 1.5.0, la sintassi per le variabili delle proprietà di sessione è cambiata da {%nomesezione.nomevariabile%} a {$smarty.section.sectionname.varname}. La vecchia sintassi è ancora supportata, ma negli esempi del manuale troverete solo riferimenti alla nuova.

index

index si usa per visualizzare l'attuale indice del ciclo, partendo da zero (o dall'attributo start se presente), e con incrementi di uno (o dell'attributo step se presente).

Nota tecnica

Se le proprietà step e start non vengono modificate, index funziona allo stesso modo della proprietà iteration, ad eccezione del fatto che parte da 0 invece che da 1.

Example 7.21. proprietà index

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


OUTPUT:

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

index_prev

index_prev visualizza l'indice del ciclo precedente. Sul primo ciclo è impostata a -1.

Example 7.22. proprietà index_prev

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{* nota: $custid[customer.index] e $custid[customer] hanno identico significato *
{if $custid[customer.index_prev] ne $custid[customer.index]}
    The customer id changed<br>
{/if}
{/section}


OUTPUT:

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

index_next

index_next visualizza l'indice del prossimo ciclo. Sull'ultimo ciclo ha sempre il valore maggiore dell'attuale (rispettando l'attributo step, quando presente).

Example 7.23. proprietà index_next

```
{section name=customer loop=$custid}
{$smarty.section.customer.index} id: {$custid[customer]}<br>
{* nota: $custid[customer.index] e $custid[customer] hanno identico significato *
{if $custid[customer.index_next] ne $custid[customer.index]}
    The customer id will change<br>
{/if}
{/section}


OUTPUT:

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

iteration

iteration visualizza l'iterazione attuale del ciclo.

Note

Al contrario di index, questa proprietà non è influenzata dalle proprietà start, step e max. Inoltre iteration comincia da 1 invece che da 0 come index. rownum è un alias di iteration, e funziona in modo identico.

Example 7.24. proprietà iteration

```
{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>
{* nota: $custid[customer.index] e $custid[customer] hanno identico significato *
{if $custid[customer.index_next] ne $custid[customer.index]}
    The customer id will change<br>
{/if}
{/section}


OUTPUT:

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

first

first vale true se l'iterazione attuale è la prima.

Example 7.25. proprietà first

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

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

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


OUTPUT:

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

last

last vale true se l'attuale iterazione è l'ultima.

Example 7.26. proprietà last

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

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

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


OUTPUT:

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

rownum

rownum visualizza l'iterazione attuale del ciclo, partendo da uno. E' un alias di iteration, e funziona in modo identico.

Example 7.27. proprietà rownum

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


OUTPUT:

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

loop

loop visualizza l'index dell'ultimo ciclo visualizzato dalla sezione. Può essere usato all'interno o dopo la sezione.

Example 7.28. proprietà index

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

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

OUTPUT:

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

There were 3 customers shown above.
```

show

show è usato come parametro per la sezione. *show* è un valore booleano, true o false. Se false, la sezione non verrà visualizzata. Se è presente un sectionelse, verrà visualizzato questo.

Example 7.29. attributo show

```
{* $show_customer_info potrebbe essere stato passato dall'applicazione
       PHP, per stabilire se questa sezione deve essere visualizzata o no *}
{section name=customer loop=$custid show=$show_customer_info}
{$smarty.section.customer.rownum} id: {$custid[customer]}<br>
{/section}

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


OUTPUT:

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

the section was shown.
```

total

total visualizza il numero totale di iterazioni che la sezione eseguirà. Può essere usato all'interno o dopo la sezione.

Example 7.30. proprietà total

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

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

OUTPUT:

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

There were 3 customers shown above.
```

strip

Molte volte i progettisti di pagine web si trovano davanti al problema causato da spazi e "a capo" che influiscono sull'output HTML generato (a causa delle "caratteristiche" del browser), per cui si trovano costretti a mettere tutti insieme i tag del template per ottenere il risultato voluto. Questo di solito significa ritrovarsi con un template illeggibile o ingestibile.

Tutto ciò che è compreso fra i tag {strip}{/strip} in Smarty viene ripulito dagli spazi extra o dai caratteri di ritorno a capo all'inizio e alla fine delle righe, prima di essere visualizzato. In questo modo potete mantenere la leggibilità dei vostri template senza preoccuparvi dei problemi causati dagli spazi.

Nota tecnica

{strip}{/strip} non modificano il contenuto delle variabili del template. Vedere la funzione strip modifier.

Example 7.31. tag strip

```
{* il codice seguente uscirà in output su una riga unica *}
{strip}
<table border=0>
 <tr>
  <td>
   <A HREF="{$url}">
   <font color="red">This is a test</font>
   </A>
  </td>
 </tr>
</table>
{/strip}


OUTPUT:

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

Notate che nell'esempio qui sopra tutte le righe iniziano e finiscono con tag HTML. Tenete presente che tutte le linee vengono "attaccate", per cui se avete del testo all'inizio o alla fine di qualche riga, questo verrà attaccato, e probabilmente non è ciò che volete.

Chapter 8. Custom Functions

Smarty è fornito di numerose funzioni utente che potete utilizzare nei template.

assign

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
var	stringa	sì	*nessuno*	Nome della variabile valorizzata
value	stringa	sì	*nessuno*	Valore assegnato alla variabile

assign è usato per assegnare valori alle variabili del template durante l'esecuzione dello stesso.

Example 8.1. assign

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

The value of $name is {$name}.

OUTPUT:

The value of $name is Bob.
```

counter

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
name	stringa	no	*default*	Nome del contatore
start	numerico	no	*1*	Valore di partenza del contatore
skip	numerico	no	*1*	Passo del contatore
direction	stringa	no	*up*	Direzione del conteggio (up/ down)
print	booleano	no	*true*	Se stampare il valore oppure no
assign	stringa	no	*nessuno*	la variabile del template a cui assegnare il valore

counter si usa per stampare un conteggio. counter terrà il conto del valore ad ogni iterazione. Potete impostare il valore di partenza, l'intervallo e la direzione del conteggio, così come decidere se stampare il valore oppure no. Potete utilizzare più contatori contemporaneamente indicando un nome diverso per ciascuno. Se non indicate un nome, verrà usato il nome 'default'.

Se fornite lo speciale attributo "assign", l'output della funzione contatore verrà assegnato a questa variabile invece di essere stampata in output.

Example 8.2. counter

```
{* inizializzazione del contatore *}
{counter start=0 skip=2}<br />
{counter}<br />
{counter}<br />
{counter}<br />
```

questo stamperà:

```
0<br />
2<br />
4<br />
6<br />
```

cycle

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
name	stringa	no	*default*	Nome del ciclo
values	misto	sì	*nessuno*	Valori da usare nel ciclo: può essere una lista delimitata da un separatore (vedere attributo delimiter), oppure un array di valori.
print	booleano	no	*true*	Se stampare il valore oppure no.
advance	booleano	no	*true*	Se avanzare o no al prossimo valore.
delimiter	stringa	no	,	Delimitatore per l'attributo values.
assign	stringa	no	*nessuno*	Variabile del template cui assegnare l'output.

Cycle si usa per effettuare un ciclo alternato fra un insieme di valori. Ci dà la possibilità di alternare facilmente due o più colori in una tabella, o di effettuare un ciclo su un array di valori.

Potete effettuare il ciclo su più di un insieme di valori nel template fornendo l'attributo name, se date ad ogni insieme un nome diverso.

Potete evitare che il valore corrente venga stampato impostando l'attributo set a false. Può essere utile per saltare un valore.

L'attributo advance serve per ripetere un valore. Se lo impostate a false, l'iterazione successiva del ciclo stamperà lo stesso valore.

Se fornite lo speciale attributo "assign", l'output della funzione cycle verrà assegnato a questa variabile invece di essere stampato in output.

Example 8.3. cycle

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

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

debug

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
output	stringa	no	*html*	tipo di output: html o javascript

{debug} produce un dump sulla pagina della console di debug. Funziona indipendentemente dall'impostazione debug di Smarty. Siccome viene eseguita a runtime, è in grado di mostrare soltanto le variabili, non i template che state utilizzando. Comunque vedrete tutte le variabili attualmente disponibili nello scope di questo template.

eval

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
var	misto	sì	*nessuno*	variabile (o stringa) da valorizzare

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
assign	stringa	no	*nessuno*	la variabile cui verrà assegnato l'output

eval si usa per valorizzare una variabile come se fosse un template. Si può usare per incorporare tag o variabili di template dentro altre variabili, oppure tag o variabili nelle variabili dei file di configurazione.

Se fornite lo speciale attributo "assign" l'output della funzione eval sarà assegnato a questa variabile invece di essere stampato in output.

Nota tecnica

La variabili valorizzate con eval sono trattate allo stesso modo dei template. Seguono le stesse regole di escape e di sicurezza, come se fossero template

Nota tecnica

Le variabili valorizzate con eval vengono compilate ad ogni chiamata: la versione compilata non viene salvata! Comunque, se avete il caching abilitato, l'output verrà messo in cache con il resto del template.

Example 8.4. eval

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

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


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

{config_load file="setup.conf"}

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

OUTPUT:

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

fetch

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
file	stringa	sì	*nessuno*	il file o l'indirizzo http o ftp da caricare
assign	stringa	no	*nessuno*	la variabile del template cui assegnare l'output

fetch si usa per recuperare file dal filesystem locale, oppure da un indirizzo http o ftp, e visualizzarne il contenuto. Se il nome del file inizia per "http://", la pagina web verrà letta e visualizzata. Se il nome del file inizia per "ftp://", il file verrà recuperato dal server ftp e visualizzato. Per i file locali deve essere indicato l'intero percorso sul filesystem oppure un percorso relativo all'indirizzo dello script php in esecuzione.

Se fornite lo speciale attributo "assign", l'output della funzione fetch verrà assegnato a questa variabile invece di essere stampato in output. (novità di Smarty 1.5.0)

Nota tecnica

I redirect http non sono supportati, quindi assicuratevi di mettere lo slash finale sull'indirizzo della pagina web quando necessario.

Nota tecnica

Se è attivata la security del template e state cercando di caricare un file dal filesystem locale, saranno consentiti soltanto file compresi in una delle directory definite sicure ($secure_dir).

Example 8.5. fetch

```
{* inclusione di un javascript nel template *}
{fetch file="/export/httpd/www.example.com/docs/navbar.js"}

{* incorporazione nel template del testo relativo al tempo proveniente da un altro
{fetch file="http://www.myweather.com/68502/"}

{* lettura via ftp dei titoli delle ultime notizie *}
{fetch file="ftp://user:password@ftp.example.com/path/to/currentheadlines.txt"}

{* assegnazione del contenuto letto ad una variabile del template *}
{fetch file="http://www.myweather.com/68502/" assign="weather"}
{if $weather ne ""}
 <b>{$weather}</b>
{/if}
```

html_checkboxes

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
name	stringa	no	*checkbox*	nome della lista di checkbox

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
values	array	sì, a meno che si usi l'attributo options	*nessuno*	array di valori per le checkbox
output	array	sì, a meno che si usi l'attributo options	*nessuno*	array di output per le checkbox
selected	stringa/array	no	*vuoto*	la/le checkbox preselezionata/e
options	array associativo	sì, a meno che si usino values e output	*nessuno*	array associativo di valori e output
separator	stringa	no	*vuoto*	stringa di testo da usare come separatore fra le checkbox
labels	booleano	no	*true*	aggiunge i tag <label> all'output

html_checkboxes è una funzione utente che usa i dati forniti per creare un gruppo di checkbox html. Si occupa anche di impostare la casella selezionata per default. Gli attributi obbligatori sono values e output, a meno che non usiate invece options. Tutto l'output generato è compatibile XHTML.

Tutti i parametri non compresi nella lista qui sopra vengono stampati come coppie nome/valore all'interno di ogni tag <input>.

Example 8.6. html_checkboxes

```
<?php

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

?>
```

dove index.tpl è:

```
{html_checkboxes name="id" values=$cust_ids selected=$customer_id output=$cust_nam
```

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

dove index.tpl è:

```
{html_checkboxes name="id" options=$cust_checkboxes selected=$customer_id separato
```

entrambi gli esempi produrranno in output:

```
<label><input type="checkbox" name="id[]" value="1000" />Joe Schmoe</label><br />
<label><input type="checkbox" name="id[]" value="1001" checked="checked" />Jack Sm
<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

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
file	stringa	sì	*nessuno*	nome/percorso dell'immagine
border	stringa	no	*0*	dimensione del bordo dell'immagine
height	stringa	no	*altezza effettiva dell'immagine*	altezza con cui visualizzare l'immagine
width	stringa	no	*larghezza effettiva dell'immagine*	larghezza con cui visualizzare l'immagine
basedir	stringa	no	*doc root del web server*	directory di base per percorsi relativi
alt	stringa	no	""	descrizione alternativa dell'immagine
href	stringa	no	*nessuno*	valore di href per il link dell'immagine

html_image è una funzione utente che genera un tag HTML per una immagine. L'altezza e la larghezza, quando non indicate, vengono calcolate automaticamente dal file dell'immagine.

basedir è la directory di riferimento per percorsi relativi. Se non viene indicata, viene usata come base la document root del web server (variabile di ambiente DOCUMENT_ROOT). Se la security è abilitata, il percorso dell'immagine deve trovarsi in una directory considerata sicura.

`href` è l'indirizzo del link a cui collegare l'immagine. Se viene fornito, verrà creato un tag <a href="LINKVALUE"><a> attorno al tag image.

Tutti i parametri non compresi nella lista qui sopra vengono stampati come coppie nome/valore all'interno del tag <img> generato.

Nota tecnica

html_image richiede un accesso al disco per leggere il file dell'immagine e calcolarne altezza e larghezza. Se non usate il caching dei template, è generalmente consigliabile evitare html_image e lasciare i tag image statici per ottenere prestazioni ottimali.

Example 8.7. esempio di html_image

```
<?php

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

?>
```

dove index.tpl è:

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

un possibile output potrebbe essere:

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

html_options

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
values	array	sì, a meno che si usi l'attributo options	*nessuno*	array di valori per il menù a discesa
output	array	sì, a meno che si usi l'attributo options	*nessuno*	array di output per il menù a discesa
selected	stringa/array	no	*vuoto*	l'elemento/gli elementi selezionato/i
options	array associativo	sì, a meno che si usino values e output	*nessuno*	array associativo di valori e output
name	stringa	no	*vuoto*	nome del gruppo select

html_options è una funzione utente che usa i dati forniti per creare un gruppo di opzioni, cioè di valori option per un menù a discesa (casella select). Si occupa anche di quale o quali valori devono essere preselezionati. Gli attributi obbligatori sono values e output, a meno che non usiate invece options.

Se uno dei valori forniti è un array, verrà trattato come un gruppo di opzioni (OPTGROUP), e visualizzato di conseguenza. E' possibile creare gruppi ricorsivi (a più livelli). Tutto l'output generato è compatibile XHTML.

Se viene fornito l'attributo opzionale *name*, la lista di opzioni verrà racchiusa con il tag <select name="groupname"></select>. In caso contrario verrà generata solo la lista di opzioni.

Tutti i parametri non compresi nella lista qui sopra verranno stampati come coppie nome/valore nel tag <select>. Saranno ignorati se l'attributo *name* non è presente.

Example 8.8. html_options

```
index.php:

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

index.tpl:

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


index.php:

require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('cust_options', array(
   1001 => '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>


OUTPUT: (per entrambi gli esempi)

<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

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
name	stringa	no	*radio*	nome dell'insieme di pulsanti radio

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
values	array	sì, a meno che si usi l'attributo options	*nessuno*	array di valori per i pulsanti radio
output	array	sì, a meno che si usi l'attributo options	*nessuno*	array di output per i pulsanti radio
selected	stringa	no	*vuoto*	l'elemento preselezionato
options	array associativo	sì, a meno che si usino values e output	*n/a*	array associativo di valori e output
separator	stringa	no	*vuoto*	stringa di testo da usare come separatore fra le diverse voci

html_radios è una funzione utente che usa i dati forniti per creare un gruppo di pulsanti radio html. Si occupa anche di quale deve essere selezionato per default. Gli attributi obbligatori sono values e output, a meno che non usiate invece options. Tutto l'output generato è compatibile XHTML.

Tutti i parametri non compresi nella lista qui sopra verranno stampati come coppie nome/valore in ciascuno dei tag <input> creati.

Example 8.9. html_radios

```
index.php:

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


index.tpl:

{html_radios name="id" values=$cust_ids selected=$customer_id output=$cust_names s


index.php:

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


index.tpl:

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


OUTPUT: (per entrambi gli esempi)

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

html_select_date

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
prefix	stringa	no	Date_	prefisso per i nomi delle variabili
time	timestamp/ YYYY-MM-DD	no	data attuale in formato unix	data preselezionata

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
			timestamp o YYYY-MM-DD	
start_year	stringa	no	anno corrente	primo anno visualizzato: può essere in valore assoluto o relativo all'anno corrente(+/- N)
end_year	stringa	no	uguale a start_year	ultimo anno visualizzato: può essere in valore assoluto o relativo all'anno corrente(+/- N)
display_days	booleano	no	true	se visualizzare i giorni oppure no
display_months	booleano	no	true	se visualizzare i mesi oppure no
display_years	booleano	no	true	se visualizzare gli anni oppure no
month_format	stringa	no	%B	formato per i mesi in output (strftime)
day_format	stringa	no	%02d	formato per i giorni in output (sprintf)
day_value_format	string	no	%d	formato per il valore dei giorni (sprintf)
year_as_text	booleano	no	false	se visualizzare gli anni in forma testuale oppure no
reverse_years	booleano	no	false	se visualizzare gli anni in ordine inverso
field_array	stringa	no	null	se viene fornito un nome, le caselle select verranno create in modo che il risultato venga fornito a PHP nella forma nome[Day], nome[Year], nome[Month].
day_size	stringa	no	null	se presente aggiunge l'attributo size al tag select

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
month_size	stringa	no	null	se presente aggiunge l'attributo size al tag select
year_size	stringa	no	null	se presente aggiunge l'attributo size al tag select
all_extra	stringa	no	null	se presente aggiunge attributi extra a tutti i tag select
day_extra	stringa	no	null	se presente aggiunge attributi extra ai tag select/ input
month_extra	stringa	no	null	se presente aggiunge attributi extra ai tag select/ input
year_extra	stringa	no	null	se presente aggiunge attributi extra ai tag select/ input
field_order	stringa	no	MDY	ordine di visualizzazione dei campi (mese, giorno, anno)
field_separator	stringa	no	\n	stringa di separazione fra i campi
month_value_format	stringa	no	%m	formato strftime per i valori dei mesi
year_empty	stringa	no	null	Se presente, il primo elemento della casella select per gli anni conterrà questo valore come output e "" come valore. E' utile per mostrare, ad esempio, sul menù a discesa la frase "Selezionare l'anno". Notate che potete utilizzare valori del tipo "-MM-DD" nell'attributo time per indicare che l'anno non deve

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
				essere preselezionato.
month_empty	stringa	no	null	Se presente, il primo elemento della casella select per i mesi conterrà questo valore come output e "" come valore. Notate che potete utilizzare valori del tipo "YYYY---DD" nell'attributo time per indicare che il mese non deve essere preselezionato.
day_empty	stringa	no	null	Se presente, il primo elemento della casella select per i giorni conterrà questo valore come output e "" come valore. Notate che potete utilizzare valori del tipo "YYYY-MM-" nell'attributo time per indicare che il giorno non deve essere preselezionato.

html_select_date è una funzione utente che crea per voi menù a discesa per le date. Può mostrare anno, mese e giorno o solo qualcuno di questi valori.

L'attributo time può avere diversi formati: può essere un timestamp UNIX o una stringa di tipo Y-M-D (anno-mese-giorno). Il formato più comune sarebbe YYYY-MM-DD, ma vengono riconosciuti anche mesi e giorni con meno di due cifre. Se uno dei tre valori (Y,M,D) è una stringa vuota, il campo select corrispondente non avrà nessuna preselezione. Ciò è utile in special modo con gli attributi year_empty, month_empty e day_empty.

Example 8.10. html_select_date

```
{html_select_date}
```

Questo stampera: _

```
+
<select name="Date_Month">
<option value="1">January</option>
<option value="2">February</option>
<option value="3">March</option>
<option value="4">April</option>
<option value="5">May</option>
<option value="6">June</option>
<option value="7">July</option>
<option value="8">August</option>
<option value="9">September</option>
<option value="10">October</option>
<option value="11">November</option>
<option value="12" selected="selected">December</option>
</select>
<select name="Date_Day">
<option value="1">01</option>
<option value="2">02</option>
<option value="3">03</option>
<option value="4">04</option>
<option value="5">05</option>
<option value="6">06</option>
<option value="7">07</option>
<option value="8">08</option>
<option value="9">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13" selected="selected">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
</select>
<select name="Date_Year">
<option value="2001" selected="selected">2001</option>
</select>
```

Example 8.11. html_select_date

```
{* l'anno iniziale e finale possono essere relativi a quello corrente *}
{html_select_date prefix="StartDate" time=$time start_year="-5" end_year="+1" disp
+
```

Questo stamperà: (l'anno corrente è il 2000)

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

html_select_time

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
prefix	stringa	no	Time_	prefisso per i nomi delle variabili
time	timestamp	no	ora corrente	ora preselezionata
display_hours	booleano	no	true	se mostrare o no le ore
display_minutes	booleano	no	true	se mostrare o no i minuti
display_seconds	booleano	no	true	se mostrare o no i secondi
display_meridian	booleano	no	true	se mostrare o no il valore "am/

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
				pm"(antimeridiano/ pomeridiano). Questo valore non viene mai mostrato (e quindi il parametro ignorato) se use_24_hours è true.
use_24_hours	booleano	no	true	se usare o no l'orologio di 24 ore
minute_interval	intero	no	1	intervallo dei minuti nel menù a discesa relativo
second_interval	intero	no	1	intervallo dei secondi nel menù a discesa relativo
field_array	stringa	no	nessuno	imposta i valori in un array con questo nome
all_extra	stringa	no	null	se presente aggiunge attributi extra a tutti i tag select/input
hour_extra	stringa	no	null	se presente aggiunge attributi extra al tag select/ input
minute_extra	stringa	no	null	se presente aggiunge attributi extra al tag select/ input
second_extra	stringa	no	null	se presente aggiunge attributi extra al tag select/ input
meridian_extra	stringa	no	null	se presente aggiunge attributi extra al tag select/ input

html_select_time è una funzione utente che crea per voi menù a discesa per la selezione di un orario. Potete scegliere quali campi visualizzare fra ore, minuti, secondi e antimeridiano/postmeridiano.

L'attributo time può avere vari formati. Può essere un timestamp o una stringa nel formato YYYYMMDDHHMMSS o una stringa leggibile dalla funzione php strtotime().

Example 8.12. html_select_time

```
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23" selected="selected">23</option>
<option value="24">24</option>
<option value="25">25</option>
<option value="26">26</option>
<option value="27">27</option>
<option value="28">28</option>
<option value="29">29</option>
<option value="30">30</option>
<option value="31">31</option>
<option value="32">32</option>
<option value="33">33</option>
<option value="34">34</option>
<option value="35">35</option>
<option value="36">36</option>
<option value="37">37</option>
<option value="38">38</option>
<option value="39">39</option>
<option value="40">40</option>
<option value="41">41</option>
<option value="42">42</option>
<option value="43">43</option>
<option value="44">44</option>
<option value="45">45</option>
<option value="46">46</option>
<option value="47">47</option>
<option value="48">48</option>
<option value="49">49</option>
<option value="50">50</option>
<option value="51">51</option>
<option value="52">52</option>
<option value="53">53</option>
<option value="54">54</option>
<option value="55">55</option>
<option value="56">56</option>
<option value="57">57</option>
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Meridian">
<option value="am" selected="selected">AM</option>
<option value="pm">PM</option>
</select>
```

html_table

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
loop	array	sì	*nessuno*	array di dati da visualizzare nella tabella
cols	intero	no	*3*	numero di colonne della tabella
table_attr	stringa	no	*border="1"*	attributi per il tag table
tr_attr	stringa	no	*vuoto*	attributi per i tag tr (gli array vengono alternati)
td_attr	stringa	no	*vuoto*	attributi per i tag td (gli array vengono alternati)
trailpad	stringa	no	* *	valore per le celle aggiuntive dell'ultima riga, se presenti
hdir	stringa	no	*right*	direzione di riempimento delle righe. Valori possibili: *left*/*right*
vdir	stringa	no	*down*	direzione di riempimento delle colonne. Valori possibili: *up*/*down*

html_table è una funzione utente che formatta un array di dati in una tabella HTML. L'attributo *cols* determina il numero di colonne che formeranno la tabella. I valori di *table_attr*, *tr_attr* e *td_attr* determinano gli attributi dei tag table, tr e td. Se *tr_attr* o *td_attr* sono array, la funzione userà un ciclo per alternarne i valori. *trailpad* è il valore da usare nelle ultime celle da aggiungere all'ultima riga, nel caso in cui il numero di valori nell'array loop non sia divisibile per il numero di colonne.

Example 8.13. html_table

```
index.php:

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

index.tpl:

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

OUTPUT:

<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

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
equation	stringa	sì	*nessuno*	l'equazione da eseguire
format	stringa	no	*nessuno*	formato del risultato (sprintf)
var	numerico	sì	*nessuno*	valore di una variabile dell'equazione
assign	stringa	no	*nessuno*	variabile del template cui verrà assegnato il risultato

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
[var ...]	numerico	sì	*nessuno*	valore di una variabile dell'equazione

La funzione math permette al progettista di effettuare equazioni matematiche nel template. Qualsiasi variabile numerica del template può essere utilizzata nell'equazione; il risultato verrà stampato al posto del tag. Le variabili usate nell'equazione vengono passate come parametri, che possono essere variabili del template o valori statici. +, -, /, *, abs, ceil, cos, exp, floor, log, log10, max, min, pi, pow, rand, round, sin, sqrt, srans e tan sono tutti operatori validi. Controllate la documentazione di PHP per ulteriori informazioni su queste funzioni matematiche.

Se fornite lo speciale attributo "assign", l'output della funzione verrà assegnato a questa variabile del template, invece di essere stampato in output.

Nota tecnica

math è una funzione costosa in termini di prestazioni, a causa dell'uso che fa della funzione php eval(). Fare i calcoli matematici in PHP è molto più efficiente, quindi, quando possibile, fate i calcoli in PHP ed assegnate i risultati al template. Evitate decisamente chiamate ripetitive alla funzione math, ad esempio in cicli section.

Example 8.14. math

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

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

OUTPUT:

9


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

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

OUTPUT:

100


{* potete usare le parentesi *}

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

OUTPUT:

6


{* potete indicare un parametro format in formato sprintf *}

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

OUTPUT:

9.44
```

mailto

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
address	stringa	sì	*nessuno*	l'indirizzo e-mail
text	stringa	no	*nessuno*	il testo da visualizzare sul link; il default è l'indirizzo e-mail
encode	stringa	no	*none*	Come codificare l'indirizzo. Può

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
				essere `none`, `hex` o `javascript`.
cc	stringa	no	*nessuno*	indirizzi e-mail da mettere 'per conoscenza'. Separateli con una virgola.
bcc	stringa	no	*nessuno*	indirizzi e-mail da mettere 'in copia nascosta'. Separateli con una virgola.
subject	stringa	no	*nessuno*	oggetto della e-mail.
newsgroups	stringa	no	*nessuno*	newsgroups a cui scrivere. Separateli con una virgola.
followupto	stringa	no	*n/a*	indirizzi per il follow up to. Separateli con una virgola.
extra	stringa	no	*nessuno*	qualsiasi informazione ulteriore che vogliate passare al link, ad esempio classi per i fogli di stile

La funzione mailto automatizza la creazione di link mailto e, opzionalmente, li codifica. Codificare gli indirizzi e-mail rende più difficile per i web spider raccoglierli dal vostro sito.

Nota tecnica

javascript è probabilmente il metodo più completo di codifica, ma potete usare anche la codifica esadecimale.

Example 8.15. mailto

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

OUTPUT:

<a href="mailto:me@example.com" >me@domain.com</a>
<a href="mailto:me@example.com" >send me some mail</a>
<script type="text/javascript" language="javascript">eval(unescape('%64%6f%63%75%6
9%74%65%28%27%3c%61%20%68%72%65%66%3d%22%6d%61%69%6c%74%6f%3a%6d%65%40%64%6f%6d%
61%69%6e%2e%63%6f%6d%22%20%3e%6d%65%40%64%6f%6d%61%69%6e%2e%63%6f%6d%3c%2f%61%3e
%27%29%3b'))</script>
<a href="mailto:%6d%65@%64%6f%6d%61%69%6e.%63%6f%6d" >me@domain.com</a>
<a href="mailto:me@example.com?subject=Hello%20to%20you%21" >me@domain.com</a>
<a href="mailto:me@example.com?cc=you@domain.com%2Cthey@domain.com" >me@domain.com
<a href="mailto:me@example.com" class="email">me@domain.com</a>
```

popup_init

popup è un'integrazione di overLib, una libreria usata per le finestre popup. Tali finestre (si tratta di finestre interne al documento, non finestre di programma come quelle che si aprono con "javascript:window.open...") si usano per informazioni relative al contesto, ad esempio aiuto o suggerimenti. popup_init deve essere chiamata una volta all'inizio di ogni pagina in cui pensate di utilizzare la funzione popup. overLib è stata scritta da Erik Bosrup, e la sua homepage si trova all'indirizzo http://www.bosrup.com/web/overlib/.

A partire dalla versione di Smarty 2.1.2, overLib NON fa più parte della release. Quindi scaricate overLib, piazzate il file overlib.js sotto la vostra document root e indicate il percorso relativo a questo file come parametro "src" di popup_init.

Example 8.16. popup_init

```
{* popup_init deve essere chiamata una volta in cima alla pagina *}
{popup_init src="/javascripts/overlib.js"}
```

popup

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
text	stringa	sì	*nessuno*	testo o codice html da visualizzare nel popup

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
trigger	stringa	mo	*onMouseOver*	evento usato per attivare il popup. Può essere onMouseOver oppure onClick
sticky	booleano	no	*false*	fa sì che il popup rimanga visibile fino a quando non viene chiuso
caption	stringa	no	*nessuno*	imposta il titolo del popup
fgcolor	stringa	no	*nessuno*	colore dell'interno del popup
bgcolor	stringa	no	*nessuno*	colore del bordo del popup
textcolor	stringa	no	*nessuno*	colore del testo del popup
capcolor	stringa	no	*nessuno*	colore del titolo del popup
closecolor	stringa	mo	*nessuno*	colore del link di chiusura
textfont	stringa	no	*nessuno*	carattere del testo
captionfont	stringa	no	*nessuno*	carattere del titolo
closefont	stringa	mo	*nessuno*	carattere del link di chiusura
textsize	stringa	no	*nessuno*	dimensione del carattere del testo
captionsize	stringa	no	*nessuno*	dimensione del carattere del titolo
closesize	stringa	no	*nessuno*	dimensione del carattere del link di chiusura
width	intero	no	*nessuno*	larghezza del box
height	intero	no	*nessuno*	altezza del box
left	boolean	No	*false*	posiziona il popup a sinistra del mouse
right	booleanp	no	*false*	posiziona il popup a destra del mouse
center	booleano	no	*false*	posiziona il popup centrato rispetto al mouse
above	booleano	no	*false*	posiziona il popup al di sopra del mouse. NOTA: possibile solo se

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
				è stata impostata l'altezza
below	booleano	no	*false*	posiziona il popup al di sotto del mouse
border	intero	no	*nessuno*	rende il bordo del popup più grosso o più sottile
offsetx	intero	no	*nessuno*	distanza orizzontale del popup rispetto al mouse
offsety	intero	no	*nessuno*	distanza verticale del popup rispetto al mouse
fgbackground	url di un'immagine	no	*nessuno*	definisce un'immagine da usare invece del colore di sfondo nel popup.
bgbackground	url di un'immagine	no	*nessuno*	definisce un'immagine da usare invece del colore per il bordo del popup. NOTA: dovete impostare il bgcolor a "", altrimenti il colore si vedrà comunque. NOTA: quando è presente un link di chiusura, Netscape ridisegnerà le celle della tabella, rendendo la visualizzazione non corretta
closetext	stringa	no	*nessuno*	imposta un testo come link di chiusura invece di "Close"
noclose	booleano	no	*nessuno*	non mostra il link di chiusura sui popup "sticky" con un titolo
status	stringa	no	*nessuno*	imposta il testo sulla barra di stato del browser
autostatus	booleano	no	*nessuno*	imposta il testo della barra di

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
				stato uguale a quello del popup. NOTA: prevale sull'impostazione di status
autostatuscap	stringa	no	*nessuno*	imposta il testo della barra di stato uguale a quello del titolo. NOTA: prevale sull'impostazione di status e autostatus
inarray	intero	no	*nessuno*	comunica ad overLib di leggere il testo da questo indice dell'array ol_text, che si trova in overlib.js. Questo parametro può essere usato al posto di text
caparray	intero	no	*nessuno*	comunica ad overLib di leggere il titolo da questo indice nell'array ol_caps
capicon	url	no	*nessuno*	mostra l'immagine indicata prima del titolo
snapx	intero	no	*nessuno*	aggancia il popup ad una posizione in una griglia orizzontale
snapy	intero	no	*nessuno*	aggancia il popup ad una posizione in una griglia verticale
fixx	intero	no	*nessuno*	blocca la posizione orizzontale del popup. Nota: prevale su qualsiasi altro posizionamento orizzontale
fixy	intero	no	*nessuno*	blocca la posizione verticale del popup. Nota: prevale su qualsiasi altro posizionamento verticale

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
background	url	no	*nessuno*	imposta un'immagine da utilizzare al posto dello sfondo della tabella
padx	intero,intero	no	*nessuno*	imposta un padding orizzontale sull'immagine di sfondo per il testo. Nota: l'attributo richiede due valori
pady	intero,intero	no	*nessuno*	imposta un padding verticale sull'immagine di sfondo per il testo. Nota: l'attributo richiede due valori
fullhtml	booleano	no	*nessuno*	consente di utilizzare codice html per l'immagine di sfondo. Il codice html dovrà trovarsi nell'attributo text
frame	stringa	no	*nessuno*	controlla il popup in un altro frame. Vedere la documentazione di overlib per maggiori informazioni su questa funzione
timeout	stringa	no	*nessuno*	chiama la funzione javascript specificata e prende il valore restituito come testo da mostrare nel popup
delay	intero	no	*nessuno*	fa sì che il popup si comporti come un tooltip. Verrà visualizzato solo dopo questo ritardo in millisecondi.
hauto	booleano	no	*nessuno*	determina automaticamente se il popup deve apparire a sinistra o a destra del mouse.

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
vauto	booleano	no	*nessuno*	determina automaticamente se il popup deve apparire sopra o sotto il mouse.

popup si usa per creare finestre popup javascript.

Example 8.17. popup

```
{* popup_init deve essere chiamata una volta in cima alla pagina *}
{popup_init src="/javascripts/overlib.js"}

{* crea un link con un popup che appare al passaggio del mouse *}
<a href="mypage.html" {popup text="This link takes you to my page!"}>mypage</a>

{* potete usare html, links, etc nel testo del popup *}
<a href="mypage.html" {popup sticky=true caption="mypage contents"
text="<ul><li>links</li><li>pages</li><li>images</li></ul>" snapx=10
snapy=10}>mypage</a>
```

textformat

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
style	stringa	no	*nessuno*	stile predefinito
indent	numero	no	*0*	numero di caratteri da rientrare ad ogni riga
indent_first	numero	no	*0*	numero di caratteri da rientrare alla prima riga
indent_char	stringa	no	*(spazio singolo)*	carattere (o stringa di caratteri) da usare come rientro
wrap	numero	no	*80*	a quanti caratteri spezzare ogni riga
wrap_char	stringa	no	*\n*	caratteri (o stringa di caratteri) da usare per spezzare le righe
wrap_cut	booleano	no	*false*	se vero, le righe verranno spezzate al carattere esatto invece che al

Nome Attributo	Tipo	Obbligatorio	Default	Descrizione
				termine di una parola
assign	stringa	no	*nessuno*	variabile del template cui assegnare l'output

textformat è una funzione di blocco usata per formattare il testo. Fondamentalmente rimuove spazi e caratteri speciali, e formatta i paragrafi spezzando le righe ad una certa lunghezza ed inserendo dei rientri.

Potete impostare i parametri esplicitamente oppure usare uno stile predefinito. Attualmente "email" è l'unico stile disponibile.

```
This is foo.

This is bar.

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

{/textformat}

OUTPUT:

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

        This is bar.

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

{textformat style="email"}

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

This is bar.

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

{/textformat}

OUTPUT:

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

This is bar.

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

Example 8.18. textformat

Chapter 9. File di configurazione

I file di configurazione sono utili ai progettisti per gestire le variabili globali del template in un unico file. Un esempio è quello dei colori. Normalmente, se volete cambiare lo schema dei colori di un'applicazione, dovreste andare in ogni template a cambiare i colori. Con un file di configurazione, i colori possono essere tenuti in un unico punto, e solo un file deve essere modificato.

Example 9.1. Esempio di sintassi di file di configurazione

```
# variabili globali
pageTitle = "Main Menu"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

[Customer]
pageTitle = "Customer Info"

[Login]
pageTitle = "Login"
focus = "username"
Intro = """Questo è un valore che occupa più
           di una riga. Dovete racchiuderlo
           fra triple virgolette."""

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

I valori delle variabili dei file di configurazione possono essere fra virgolette, ma non è necessario. Potete usare sia gli apici singoli ('), sia le virgolette doppie ("). Se avete un valore che occupa più di una riga, racchiudete l'intero valore fra triple virgolette ("""). Potete mettere commenti usando qualsiasi sintassi che non sia valida per il file di configurazione. Noi consigliamo l'uso di un cancelletto (#) all'inizio della riga.

Questo esempio di file di configurazione ha due sezioni. I nomi di sezione sono racchiusi fra parentesi quadre []. I nomi di sezioni possono essere stringhe dal contenuto arbitrario, purché non comprenda `[` o `]`. Le quattro variabili in alto sono variabili globali, non contenute in alcuna sezione. Queste variabili vengono sempre caricate dal file di configurazione. Se viene caricata una particolare sezione, allora saranno caricate le variabili globali e quelle di quella sezione. Se una variabile esiste sia come globale che in una sezione, verrà usata la variabile di sezione. Se date lo stesso nome a due variabili nella stessa sezione verrà usato l'ultimo valore.

I file di configurazione vengono caricati nel template con la funzione **config_load**.

Potete nascondere variabili o intere sezioni anteponendo un punto al nome della variabile o della sezione. Questo è utile se la vostra applicazione legge dai file di configurazione dati sensibili di cui il motore di

template non ha bisogno. Se affidate a terzi la modifica del template, potete stare sicuri che non potranno leggere dati sensibili dal file di configurazione caricandolo nel template.

Chapter 10. Console di Debugging

C'è una console di debugging inclusa in Smarty. La console vi informa di tutti i template che sono stati inclusi, le variabili assegnate e quelle dei file di configurazione per la chiamata attuale del template. Nella distribuzione di Smarty è incluso un template chiamato "debug.tpl" che controlla la formattazione della console. Impostate $debugging a true in Smarty, e se necessario impostate $debug_tpl con il percorso del file debug.tpl (di default si trova nella SMARTY_DIR). Quando caricate la pagina, dovrebbe apparire in pop up una console creata con javascript che vi informa di tutti i nomi dei template inclusi e delle variabili assegnate nella pagina attuale. Per vedere le variabili disponibili per un particolare template, consultate la funzione {debug}. Per disabilitare la console di debugging impostate $debugging a false. Potete anche attivare temporaneamente la console mettendo SMARTY_DEBUG nell'URL, se abilitate questa opzione con $debugging_ctrl.

Nota tecnica

La console di debugging non funziona quando usate la API fetch(), funziona solo con display(). E' un insieme di istruzioni javascript aggiunte in fondo al template generato. Se non vi piace l'uso di javascript, potete modificare il template debug.tpl per formattare l'output come preferite. I dati di debug non vengono messi in cache e i dati relativi a debug.tpl non sono inclusi nell'output della console di debug.

Note

I tempi di caricamento di ogni template e file di configurazione sono in secondi o frazioni di secondo.

Part III. Smarty Per Programmatori

Table of Contents

Chapter 11. Costanti

SMARTY_DIR

Questo dovrebbe essere il percorso completo sul sistema dei file di classe di Smarty. Se la costante non è definita, Smarty cercherà di determinare automaticamente il valore appropriato. Se è definita, il percorso deve terminare con una barra.

Example 11.1. SMARTY_DIR

```
<?php
// imposta il percorso della directory di Smarty
define("SMARTY_DIR","/usr/local/lib/php/Smarty/");

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

Chapter 12. Variabili

$template_dir

Questo è il nome della directory di default dei template. Se non ne indicate una quando includete i file, verranno cercati qui. Per default è "./templates", che significa che Smarty cercherà la directory dei template nella stessa directory dello script php in esecuzione.

Nota tecnica

E' sconsigliato mettere questa directory sotto la document root del web server.

$compile_dir

Questo è il nome della directory dove vengono messi i template compilati. Per default è "./templates_c", che significa che Smarty cercherà la directory di compilazione sotto la stessa directory dello script php in esecuzione.

Nota tecnica

Questa impostazione deve essere un percorso relativo o assoluto. include_path non viene usata per i file in scrittura.

Nota tecnica

E' sconsigliato mettere questa directory sotto la document root del web server.

$config_dir

Questa è la directory usata per memorizzare i file di configurazione usati nei template. Il default è "./configs2, che significa che Smarty cercherà la directory dei file di configurazione nella stessa directory dello script php in esecuzione.

Nota tecnica

E' sconsigliato mettere questa directory sotto la document root del web server.

$plugins_dir

Queste sono le directory dove Smarty andrà a cercare i plugin di cui ha bisogno. Il default è "plugins" sotto la SMARTY_DIR. Se fornite un percorso relativo, Smarty cercherà prima di tutto sotto la SMARTY_DIR, poi sotto la directory corrente, infine sotto ogni directory compresa nell'include_path di PHP.

Nota tecnica

Per migliori prestazioni, non costringete Smarty a cercare le plugins_dir usando l'include path di PHP. Usate un percorso assoluto, o relativo alla SMARTY_DIR o alla directory corrente.

$debugging

Questa variabile abilita la console di debugging. La console è una finestra javascript che vi informa sui template inclusi e sulle variabili valorizzate per la pagina attuale.

$debug_tpl

Questo è il nome del file di template usato per la console di debugging. Per default, il nome è debug.tpl ed il file si trova nella SMARTY_DIR.

$debugging_ctrl

Questa variabile consente modi alternativi per abilitare il debugging. NONE significa che non sono consentiti metodi alternativi. URL significa che quando la parola chiave SMARTY_DEBUG viene trovata nella QUERY_STRING, il debugging viene abilitato per quella chiamata dello script. Se $debugging è true, questo valore viene ignorato.

$autoload_filters

Se ci sono alcuni filtri che volete caricare ad ogni chiamata del template, potete specificarli usando questa variabile e Smarty li caricherà automaticamente. La variabile è un array associativo dove le chiavi sono i tipi di filtro ed i valori sono array con i nomi dei filtri. Ad esempio:

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

$compile_check

Ad ogni chiamata dell'applicazione PHP, Smarty controlla se il template corrente è stato modificato (cioè se il timestamp è cambiato) dall'ultima volta che è stato compilato. Se è cambiato, Smarty ricompila il template. Se il template non è stato mai compilato, sarà compilato indipendentemente da questa impostazione. Per default questa variabile è impostata a true. Una volta che l'applicazione viene messa in produzione (quind i template non cambieranno più), il passo di compile_check non è più necessario. Assicuratevi di impostare $compile_check a "false" per massimizzare le prestazioni. Notate che se impostate questo valore a "false" e un file di template viene modificato, *non* vedrete la modifica fino a quando il template non viene ricompilato. Se sono abilitati il caching e il compile_check, i file della cache verranno rigenerati quando un file di template o un file di configurazione fra quelli interessati vengono modificati. Consultate $force_compile o clear_compiled_tpl.

$force_compile

Questo valore forza Smarty a (ri)compilare i template ad ogni chiamata. Questa impostazione prevale su $compile_check. Per default è disabilitata. E' utile per lo sviluppo ed il debug. Non dovrebbe essere mai usata in un ambiente di produzione. Se il caching è abilitato, i file della cache verranno rigenerati ogni volta.

$caching

Questa variabile dice a Smarty se mettere in cache oppure no l'output dei template. Per default è impostata a 0, o disabilitata. Se i vostri template generano contenuto ridondante, è consigliabile attivare il caching. Ne deriveranno significativi guadagni di prestazioni. Potete anche avere più di una cache per lo stesso template. I valori 1 e 2 abilitano il caching. 1 dice a Smarty di usare l'attuale variabile $cache_lifetime per determinare se la cache è scaduta. Il valore 2 dice a Smarty di usare il valore di cache_lifetime del momento in cui la cache è stata generata. In questo modo potete impostare il cache_lifeteime subito prima di caricare il template per avere un controllo granulare su quando quella particolare cache scadrà. Consultate anche is_cached.

Se $compile_check è abilitato, il contenuto in cache verrà rigenerato quando i file del template o di configurazione che fanno parte di questa cache vengono modificati. Se è abilitato $force_compile, il contenuto in cache verrà rigenerato ogni volta.

$cache_dir

Questo è il nome della directory dove vengono salvati i file della cache. Per default è "./cache", che significa che Smarty cercherà la directory della cache nella stessa directory dello script php in esecuzione. Potete anche usare una funzione personalizzata di gestione della cache, che ignorerà questa impostazione.

Nota tecnica

Questa impostazione deve essere un percorso relativo o assoluto. include_path non viene usato per i file in scrittura.

Nota tecnica

E' sconsigliato mettere questa directory sotto la document root del web server.

$cache_lifetime

E' la durata in secondi della validità di un file di cache. Una volta che questo tempo è scaduto, la cache verrà rigenerata. $caching deve essere impostato a "true" perché $cache_lifetime abbia significato. Il valore -1 forza la cache a non scadere mai. Il valore 0 farà sì che la cache venga sempre rigenerata (è utile solo in fase di test, per disabilitare il caching un metodo più efficiente è impostare $caching a false.)

Se $force_compile è abilitato, i file della cache verranno rigenerati ogni volta, disabilitando in effetti il caching. Potete eliminare tutti i file della cache con la funzione clear_all_cache(), oppure singoli file (o gruppi di file) con la funzione clear_cache().

Nota tecnica

Se volete dare a certi template un particolare tempo di vita della cache, potete farlo impostando $caching = 2, quindi dando il valore che vi interessa a $cache_lifetime subito prima di chiamare display() o fetch().

$cache_handler_func

Potete fornire una funzione personalizzata di gestione dei file della cache invece di usare il metodo incorporato che usa la $cache_dir. Consultate la sezione funzione di gestione della cache per i dettagli.

$cache_modified_check

Se è impostato a true, Smarty rispetterà l'header If-Modified-Since spedito dal client. Se il timestamp del file in cache non è cambiato dall'ultima visita, verrà inviato un header "304 Not Modified" invece del contenuto. Questo funziona solo sul contenuto in cache senza tag **insert**.

$config_overwrite

Se è impostato a true, le variabili lette dai file di configurazione si sovrascriveranno l'una con l'altra. Diversamente, verranno messe in un array. E' utile se volete memorizzare array di dati nei file di configurazione, è sufficiente elencare più volte ogni elemento. true per default.

$config_booleanize

Se impostato a true, le variabili dei file di configurazione con i valori on/true/yes e off/false/no verranno convertite automaticamente in valori booleani. In questo modo potete usare questi valori nei template in questo modo: {if #foobar#} ... {/if}. Se foobar è on, true o yes, l'istruzione {if} verrà eseguita. true per default.

$config_read_hidden

Se impostato a true, le sezioni nascoste (col nome che inizia con un punto) dei file di configurazione possono essere lette dai template. Tipicamente lascerete questo valore a false, in modo da poter memorizzare dati sensibili nei file di configurazione (ad esempio parametri per l'accesso a un database) senza preoccuparvi che vengano caricati sul template. false per default.

$config_fix_newlines

Se impostato a true, i caratteri di 'a capo' mac e dos (\r e \r\n) nei file di configurazione vengono convertiti a \n quando sono analizzati. true per default.

$default_template_handler_func

Questa funzione viene chiamata quando Smarty non riesce a caricare un template.

$php_handling

Questa variabile dice a Smarty come gestire il codice PHP incorporato nei template. Ci sono quattro possibili impostazioni: il default è SMARTY_PHP_PASSTHRU. Notate che questa variabile NON ha effetto sul codice php che si trova fra i tag {php}{/php}.

- SMARTY_PHP_PASSTHRU - Smarty stampa il contenuto dei tag così com'è.
- SMARTY_PHP_QUOTE - Smarty trasforma i tag in entità html.
- SMARTY_PHP_REMOVE - Smarty rimuove i tag dal template.
- SMARTY_PHP_ALLOW - Smarty esegue il codice PHP.

Note

Incorporare codice PHP nei template è altamente sconsigliato. Usate invece le funzioni utente o i modificatori.

$security

E' una variabile booleana, per default è false. Security viene utile per situazioni in cui avete affidato la modifica dei template a terzi (ad esempio via ftp) di cui non vi fidate completamente, e volete quindi ridurre il rischio di compromettere la sicurezza del sistema attraverso il linguaggio del template. Attivare security comporta l'applicazione delle seguenti regole al linguaggio del template, a parte ciò che può essere modificato con $security_settings:

- Se $php_handling è impostato a SMARTY_PHP_ALLOW viene implicitamente modificato a SMARTY_PHP_PASSTHRU
- Non sono ammesse funzioni PHP nelle istruzioni IF, ad esclusione di quelle specificate in $security_settings
- I file dei template possono essere inclusi solo dalle directory elencate nell'array $secure_dir
- I file locali possono essere letti con {fetch} solo dalle directory elencate nell'array $secure_dir
- I tag {php}{/php} non sono consentiti
- Non è possibile usare funzioni PHP come modificatori, ad esclusione di quelle specificate in $security_settings

$secure_dir

E' un array contenente tutte le directory locali che sono considerate sicure. {include} e {fetch} lo usano quando $security è abilitata.

$security_settings

Sono valori usati per modificare o specificare le impostazioni di sicurezza quando $security è abilitata. Queste sono le impostazioni possibili:

- PHP_HANDLING - true/false. Se è impostato a true, l'impostazione di $php_handling non viene verificata per la sicurezza.
- IF_FUNCS - E' un array con i nomi delle funzioni PHP consentite nelle istruzioni IF.
- INCLUDE_ANY - true/false. Se impostata a true, qualsiasi template può essere incluso dal filesystem, indipendentemente dalla lista di $secure_dir.
- PHP_TAGS - true/false. Se impostato a true, è consentito l'uso dei tag {php}{/php} nei template.
- MODIFIER_FUNCS - E' un array coi nomi delle funzioni PHP di cui è consentito l'uso come modificatori delle variabili.

$trusted_dir

$trusted_dir viene usata solo quando $security è abilitata. E' un array di tutte le directory che sono considerate affidabili. Le directory affidabili sono quelle dalle quali possono essere eseguiti script php direttamente dai template con {include_php}.

$left_delimiter

E' il delimitatore di sinistra usato dal linguaggio dei template. Per default è "{".

$right_delimiter

E' il delimitatore di destra usato dal linguaggio dei template. Per default è "}".

$compiler_class

Specifica il nome della classe del compilatore che Smarty userà per compilare i template. Il default è 'Smarty_Compiler'. Solo per utenti avanzati.

$request_vars_order

L'ordine in cui sono registrate le variabili della richiesta http, simile a variables_order in php.ini.

$request_use_auto_globals

Specifica se Smarty deve usare gli array di php $HTTP_*_VARS[] ($request_use_auto_globals=false che è il valore di default) o $_*[] ($request_use_auto_globals=true). Ciò ha effetto sui template che usano {$smarty.request.*}, {$smarty.get.*} ecc. . Attenzione: Se impostate $request_use_auto_globals a true, $request_vars_order non ha effetto, e viene usato il valore di configurazione di php `gpc_order`.

$error_reporting

Quando a questa variabile viene dato un valore non-null, il suo valore viene usato come livello di error_reporting di php all'interno di display() e fetch(). Quando il debugging è abilitato questo valore è ignorato e il livello degli errori viene lasciato invariato.

$compile_id

Identificatore persistente di compilazione. In alternativa a passare lo stesso compile_id ad ogni chiamata di funzione, potete impostare questa variabile ed il suo valore verrà usato implicitamente da quel momento in poi.

$use_sub_dirs

Impostate questo valore a false se il vostro ambiente PHP non consente la creazione di sottodirectory da parte di Smarty. Le sottodirectory sono più efficienti, quindi usatele se potete.

Nota tecnica

A partire da Smarty-2.6.2 `use_sub_dirs` per default vale false.

$default_modifiers

E' un array di modificatori da applicare implicitamente ad ogni variabile in un template. Ad esempio, per fare l'escape HTML ad ogni variabile per default, usate array('escape:"htmlall"'); per rendere una variabile esente dai modificatori di default, passatele lo speciale modificatore "smarty" con il parametro "nodefaults", così: {$var|smarty:nodefaults}.

$default_resource_type

Dice a Smarty che tipo di risorsa usare implicitamente. Il valore di default è 'file', il che significa che $smarty->display('index.tpl'); e $smarty->display('file:index.tpl'); hanno identico significato. Leggete il capitolo risorse per i dettagli.

Chapter 13. Methods

Name

append

```
void append(mixed var);

void append(string varname,
            mixed var,
            bool merge);
```

Si usa per aggiungere un elemento ad un array. Se aggiungete un valore stringa, verrà convertito in un elemento di array e aggiunto. Potete passare esplicitamente coppie nome/valore, oppure array associativi contenenti le coppie nome/valore. Se passate il terzo parametro opzionale a true, il valore verrà fuso nell'array corrente invece che aggiunto.

Nota tecnica

Il parametro *`merge`* rispetta le chiavi degli array, quindi se fate un merge su due array a indici numerici rischiate che alcuni valori vengano sovrascritti, o di avere indici in ordine non sequenziale. Questo comportamento è diverso da quello della funzione array_merge() di PHP che elimina le chiavi numeriche ed effettua una rinumerazione.

Example 13.1. append

```
<?php
// passaggio di coppie nome/valore
$smarty->append("Name", "Fred");
$smarty->append("Address", $address);

// passaggio di un array associativo
$smarty->append(array("city" => "Lincoln", "state" => "Nebraska"));
?>
```

Name

append_by_ref

```
void append_by_ref(string varname,
                   mixed var,
                   bool merge);
```

Si usa per aggiungere valori al template per riferimento. Se aggiungete una variabile per riferimento e poi cambiate il suo valore, il template vedrà il valore modificato. Per gli oggetti, append_by_ref() evita anche la copia in memoria dell'oggetto aggiunto. Consultate il manuale di PHP sui riferimenti alle variabili per una spiegazione approfondita. Se passate il terzo parametro opzionale a true, il valore verrà fuso nell'array corrente invece che aggiunto.

Nota tecnica

Il parametro *`merge`* rispetta le chiavi degli array, quindi se fate un merge su due array a indici numerici rischiate che alcuni valori vengano sovrascritti, o di avere indici in ordine non sequenziale. Questo comportamento è diverso da quello della funzione array_merge() di PHP che elimina le chiavi numeriche ed effettua una rinumerazione.

Example 13.2. append_by_ref

```
<?php
// aggiunta di coppie nome/valore
$smarty->append_by_ref("Name", $myname);
$smarty->append_by_ref("Address", $address);
?>
```

Name

assign

```
  void assign(mixed var);


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

Si usa per assegnare valori ai template. Potete passare esplicitamente coppie nome/valore, o array associativi contenenti le coppie nome/valore.

Example 13.3. assign

```
<?php
// passaggio di coppie nome/valore
$smarty->assign('Name', 'Fred');
$smarty->assign('Address', $address);

// passaggio di un array associativo
$smarty->assign(array("city" => "Lincoln", "state" => "Nebraska"));
?>
```

Name

assign_by_ref

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

Si usa per assegnare valori ai template per riferimento invece di farne una copia. Consultate il manuale PHP sui riferimenti alle variabili per una spiegazione.

Nota tecnica

Questo metodo si usa per assegnare valori ai template per riferimento. Se assegnate una variabile per riferimento e poi cambiate il suo valore, il template vedrà il valore modificato. Per gli oggetti, assign_by_ref() evita anche la copia in memoria dell'oggetto assegnato. Consultate il manuale PHP sui riferimenti alle variabili per una spiegazione approfondita.

Example 13.4. assign_by_ref

```
<?php
// passaggio di coppie nome/valore
$smarty->assign_by_ref('Name', $myname);
$smarty->assign_by_ref('Address', $address);
?>
```

Name

clear_all_assign

```
  void clear_all_assign();
```

Annulla i valori di tutte le variabili assegnate.

Example 13.5. clear_all_assign

```
<?php
// annulla tutte le variabili assegnate
$smarty->clear_all_assign();
?>
```

Name

clear_all_cache

```
  void clear_all_cache(int expire_time);
```

Annulla l'intera cache del template. Come parametro opzionale potete fornire un'età minima in secondi che i file della cache devono avere prima di essere eliminati.

Example 13.6. clear_all_cache

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

Name

clear_assign

```
  void clear_assign(mixed var);
```

Annulla il valore di una variabile assegnata in precedenza. Può essere un valore singolo, o un array di valori.

Example 13.7. clear_assign

```
<?php
// annullamento di una singola variabile
$smarty->clear_assign("Name");

// annullamento di più variabili
$smarty->clear_assign(array("Name", "Address", "Zip"));
?>
```

Name

clear_cache

```
  void clear_cache(string template,
                   string cache_id,
                   string compile_id,
                   int expire_time);
```

Elimina la cache per un *`template`* specifico. Se avete più cache per questo template, potete eliminarne una specifica fornendo il *`cache_id`* come secondo parametro. Potete anche passare un *`compile_id`* come terzo parametro. Potete "raggruppare" i template in modo da rimuoverli in gruppo. Leggete la sezione sul caching per maggiori informazioni. Come quarto parametro opzionale potete fornire un'età minima in secondi che il file di cache deve avere prima di essere eliminato.

Example 13.8. clear_cache

```
<?php
// eliminazione della cache per un template
$smarty->clear_cache("index.tpl");

// eliminazione di una particolare cache in un template a più cache
$smarty->clear_cache("index.tpl", "CACHEID");
?>
```

Name

clear_compiled_tpl

```
  void clear_compiled_tpl(string tpl_file,
                          string compile_id,
                          int exp_time);
```

Elimina la versione compilata dello specifico template indicato, o tutti file di template compilati se non ne viene specificato uno. Se passate un compile_id solo il template compilato relativo a questo compile_id viene eliminato. Se passate un exp_time, solo i template compilati con un'età maggiore di exp_time (in secondi) vengono eliminati; per default tutti i template compilati vengono eliminati, indipendentemente dalla loro età. Questa funzione è solo per uso avanzato, normalmente non ne avrete bisogno.

Example 13.9. clear_compiled_tpl

```
<?php
// eliminazione di uno specifico template
$smarty->clear_compiled_tpl("index.tpl");

// eliminazione di tutti i template compilati
$smarty->clear_compiled_tpl();
?>
```

Name

clear_config

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

Elimina tutte le variabili di configurazione assegnate. Se viene fornito un nome di variabile, soltanto quella variabile viene eliminata.

Example 13.10. clear_config

```
<?php
// eliminazione di tutte le variabili di configurazione
$smarty->clear_config();

// eliminazione di una variabile
$smarty->clear_config('foobar');
?>
```

Name

config_load

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

Carica i dati del file di configurazione *`file`* e li assegna al template. Funziona esattamente come la funzione del template config_load.

Nota tecnica

A partire da Smarty 2.4.0, le variabili dei template vengono mantenute fra le diverse chiamate di fetch() e display(). Le variabili di configurazione caricate con config_load() hanno sempre uno scope globale. Anche i file di configurazione vengono compilati per una esecuzione più veloce, e rispettano le impostazioni di force_compile e compile_check.

Example 13.11. config_load

```
<?php
// caricamento delle variabili di configurazione e loro assegnazione al template
$smarty->config_load('my.conf');

// caricamento di una sezione
$smarty->config_load('my.conf', 'foobar');
?>
```

Name

display

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

Visualizza il template. Dovete fornire un tipo e percorso corretti per la risorsa del template. Come secondo parametro opzionale potete passare una cache id. Consultate la sezione sul caching per maggiori informazioni.

Come terzo parametro opzionale, potete passare un *`compile_id`*. Questo nel caso in cui vogliate compilare versioni diverse dello stesso template, oppure avere template diversi per lingue diverse. Un altro uso di compile_id è quando usate più di una $template_dir ma soltanto una $compile_dir. Impostate un *`compile_id`* diverso per ogni $template_dir, altrimenti i template con lo stesso nome si sovrascriveranno a vicenda. Potete anche impostare la variabile $compile_id una volta sola invece di passarla ogni volta che chiamate questa funzione.

Example 13.12. display

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

// faccio le chiamate al db solo se
// non esiste la cache
if(!$smarty->is_cached("index.tpl")) {

  // dummy up some data
  $address = "245 N 50th";
  $db_data = array(
              "City" => "Lincoln",
              "State" => "Nebraska",
              "Zip" => "68502"
            );

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

}

// visualizzo l'output
$smarty->display("index.tpl");
?>
```

Usate la sintassi delle risorse dei template per visualizzare file che si trovano al di fuori della directory $template_dir.

Example 13.13. esempi di visualizzazione di risorse di template

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

// percorso assoluto (equivale al precedente)
$smarty->display("file:/usr/local/include/templates/header.tpl");

// percorso assoluto windows (OBBLIGATORIO il prefisso "file:")
$smarty->display("file:C:/www/pub/templates/header.tpl");

// inclusione dalla risorsa di template di nome "db"
$smarty->display("db:header.tpl");
?>
```

Name

fetch

```
string fetch(string template,
             string cache_id,
             string compile_id);
```

Questo metodo restituisce l'output del template invece di visualizzarlo. Dovete fornire un tipo e percorso corretti per la risorsa del template. Come secondo parametro opzionale potete passare una cache id. Consultate la sezione sul caching per maggiori informazioni.

Come terzo parametro opzionale, potete passare un *`compile_id`*. Questo nel caso in cui vogliate compilare versioni diverse dello stesso template, oppure avere template diversi per lingue diverse. Un altro uso di compile_id è quando usate più di una $template_dir ma soltanto una $compile_dir. Impostate un *`compile_id`* diverso per ogni $template_dir, altrimenti i template con lo stesso nome si sovrascriveranno a vicenda. Potete anche impostare la variabile $compile_id una volta sola invece di passarla ogni volta che chiamate questa funzione.

Example 13.14. fetch

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

$smarty->caching = true;

// faccio le chiamate al db solo se
// non esiste la cache
if(!$smarty->is_cached("index.tpl")) {

  // dummy up some data
  $address = "245 N 50th";
  $db_data = array(
              "City" => "Lincoln",
              "State" => "Nebraska",
              "Zip" => "68502"
            );

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

}

// catturo l'output
$output = $smarty->fetch("index.tpl");

// qui faccio qualcosa con $output

echo $output;
?>
```

Name

get_config_vars

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

Restituisce il valore della variabile di configurazione data, se è stata caricata. Se non viene passato un parametro viene restituito un array di tutte le variabili di configurazione caricate.

Example 13.15. get_config_vars

```
<?php
// recupero la variabile di configurazione del template 'foo'
$foo = $smarty->get_config_vars('foo');

// recupero tutte le variabili di configurazione caricate
$config_vars = $smarty->get_config_vars();

// diamo un'occhiata
print_r($config_vars);
?>
```

Name

get_registered_object

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

Restituisce un riferimento a un oggetto registrato. E' utile quando, dall'interno di una funzione utente, avete bisogno di accedere direttamente a un oggetto registrato.

Example 13.16. get_registered_object

```
<?php
function smarty_block_foo($params, &$smarty)
{
  if (isset($params['object'])) {
    // ottengo il riferimento all'oggetto registrato
    $obj_ref = &$smarty->get_registered_object($params['object']);
    // $obj_ref ora è un riferimento all'oggetto
  }
}
?>
```

Name

get_template_vars

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

Restituisce il valore della variabile data assegnata al template. Se non viene fornito il parametro viene restituito un array di tutte le variabili assegnate.

Example 13.17. get_template_vars

```
<?php
// recupero la variabile assegnata al template 'foo'
$foo = $smarty->get_template_vars('foo');

// recupero tutte le variabili assegnate al template
$tpl_vars = $smarty->get_template_vars();

// diamo un'occhiata
print_r($tpl_vars);
?>
```

Name

is_cached

```
  bool is_cached(string template,
                 string cache_id,
                 string compile_id);
```

Restituisce TRUE se è presente una cache valida per questo template. Funziona soltanto se caching è impostato a true.

Example 13.18. is_cached

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

if(!$smarty->is_cached("index.tpl")) {
// faccio le chiamate al database, assegno le variabili
}

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

Potete passare anche una cache id come secondo parametro opzionale, nel caso vogliate cache multiple per il template dato.

Potete fornire un compile id come terzo parametro opzionale. Se lo omettete, viene usato il valore della variabile persistente $compile_id.

Se non volete passare una cache id ma volete passare un compile id dovete passare null come cache id.

Example 13.19. is_cached con template a cache multiple

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

if(!$smarty->is_cached("index.tpl", "FrontPage")) {
  // faccio le chiamate al database, assegno le variabili
}

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

Nota tecnica

Se is_cached restituisce true, in realtà carica l'output in cache e lo memorizza internamente. Ogni chiamata successiva a display() o a fetch() restituirà questo output memorizzato

internamente, e non cercherà di ricaricare il file della cache. Questo evita una situazione che potrebbe verificarsi quando un secondo processo elimina la cache nell'intervallo fra la chiamata a is_cached e quella a display, nell'esempio visto prima. Questo significa anche che le chiamate a clear_cache() ed altre modifiche fatte sulle impostazioni della cache potrebbero non avere effetto dopo che `is_cached` ha restituito true.

Name

load_filter

```
  void load_filter(string type,
                   string name);
```

Questa funzione può essere usata per caricare un plugin filtro. Il primo parametro specifica il tipo di filtro da caricare e può avere uno di questi valori: 'pre', 'post' o 'output'. Il secondo parametro specifica il nome del plugin filtro, ad esempio 'trim'.

Example 13.20. caricamento di plugin filtro

```
<?php
$smarty->load_filter('pre', 'trim');        // carico un prefiltro di nome 'trim'
$smarty->load_filter('pre', 'datefooter');  // carico un altro prefiltro di nome '
$smarty->load_filter('output', 'compress'); // carico un filtro di output di nome
?>
```

Name

register_block

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

Si può usare questa funzione per registrare dinamicamente funzioni plugin per i blocchi. Dovete fornire il nome della funzione di blocco, seguito dalla funzione PHP da richiamare che implementa tale funzione.

Il parametro *`impl`*, contenente la funzione callback, può avere uno dei seguenti valori: (a) una stringa contenente il nome della funzione (b) un array nella forma `array(&$oggetto, $metodo)`, dove `&$oggetto` è il riferimento ad un oggetto e `$metodo` è una stringa contenente il nome di un metodo (c) un array nella forma `array(&$classe, $metodo)` dove `$classe` è un nome di classe e `$metodo` è un metodo statico della classe.

`cacheable` e *`cache_attrs`* possono essere omessi nella maggioranza dei casi. Consultate Controllo della Cache per l'output dei Plugins per capire come usarli.

Example 13.21. register_block

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

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

dove il template è:

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

Name

register_compiler_function

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

Si può usare questa funzione per registrare dinamicamente una funzione plugin di compilazione. Dovete fornire il nome della funzione di compilazione, seguito dalla funzione PHP da richiamare che la implementa.

Il parametro *`impl`*, contenente la funzione callback, può avere uno dei seguenti valori: (a) una stringa contenente il nome della funzione (b) un array nella forma `array(&$oggetto, $metodo)`, dove `&$oggetto` è il riferimento ad un oggetto e `$metodo` è una stringa contenente il nome di un metodo (c) un array nella forma `array(&$classe, $metodo)` dove `$classe` è un nome di classe e `$metodo` è un metodo statico della classe.

`cacheable` può essere omesso nella maggioranza dei casi. Consultate Controllo della Cache per l'output dei Plugins per capire come usarlo.

Name

register_function

```
  void register_function(string name,
                         mixed impl,
                         bool cacheable,
                         mixed cache_attrs);
```

Si può usare questa funzione per registrare dinamicamente funzioni plugin per i template. Dovete fornire il nome della funzione di template, seguito dalla funzione PHP da richiamare che implementa tale funzione.

Il parametro *`impl`*, contenente la funzione callback, può avere uno dei seguenti valori: (a) una stringa contenente il nome della funzione (b) un array nella forma `array(&$oggetto, $metodo)`, dove `&$oggetto` è il riferimento ad un oggetto e `$metodo` è una stringa contenente il nome di un metodo (c) un array nella forma `array(&$classe, $metodo)` dove `$classe` è un nome di classe e `$metodo` è un metodo statico della classe.

`cacheable` e *`cache_attrs`* possono essere omessi nella maggioranza dei casi. Consultate Controllo della Cache per l'output dei Plugins per capire come usarli.

Example 13.22. register_function

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

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

// ora potete usare questa funzione in Smarty per stampare la data attuale: {date_
// oppure {date_now format="%Y/%m/%d"} per formattarla.
?>
```

Name

register_modifier

```
  void register_modifier(string name,
                         mixed impl);
```

Potete usarla per registrare dinamicamente plugin modificatori. Passate il nome del modificatore del template, seguito dalla funzione PHP che lo implementa.

Il parametro *`impl`*, contenente la funzione callback, può avere uno dei seguenti valori: (a) una stringa contenente il nome della funzione (b) un array nella forma `array(&$oggetto, $metodo)`, dove `&$oggetto` è il riferimento ad un oggetto e `$metodo` è una stringa contenente il nome di un metodo (c) un array nella forma `array(&$classe, $metodo)` dove `$classe` è un nome di classe e `$metodo` è un metodo statico della classe.

Example 13.23. register_modifier

```
<?php
// mappiamo la funzione PHP stripslashes a un modificatore Smarty.

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

// ora potete usare  {$var|sslash} per togliere gli slash dalle variabili
?>
```

Name

register_object

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

Serve a registrare un oggetto per poterlo usare nei template. Consultate la sezione oggetti del manuale per gli esempi.

Name

register_outputfilter

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

Usatela per registrare dinamicamente filtri di output che devono operare sull'output di un template prima che venga visualizzato. Consultate i filtri di output sui template per maggiori informazioni su come impostare una funzione di filtro di output.

Il parametro *`function`*, contenente la funzione callback, può avere uno dei seguenti valori: (a) una stringa contenente il nome della funzione (b) un array nella forma `array(&$oggetto, $metodo)`, dove `&$oggetto` è il riferimento ad un oggetto e `$metodo` è una stringa contenente il nome di un metodo (c) un array nella forma `array(&$classe, $metodo)` dove `$classe` è un nome di classe e `$metodo` è un metodo statico della classe.

Name

register_postfilter

```
void register_postfilter(mixed function);
```

Usatela per registrare dinamicamente filtri da eseguire sui template dopo la compilazione ("postfiltri"). Consultate postfiltri sui template per maggiori informazioni su come impostare una funzione postfiltro.

Il parametro *`function`*, contenente la funzione callback, può avere uno dei seguenti valori: (a) una stringa contenente il nome della funzione (b) un array nella forma `array(&$oggetto, $metodo)`, dove `&$oggetto` è il riferimento ad un oggetto e `$metodo` è una stringa contenente il nome di un metodo (c) un array nella forma `array(&$classe, $metodo)` dove `$classe` è un nome di classe e `$metodo` è un metodo statico della classe.

Name

register_prefilter

```
void register_prefilter(mixed function);
```

Usatela per registrare dinamicamente filtri da eseguire sui template prima della compilazione ("prefiltri"). Consultate prefiltri sui template per maggiori informazioni su come impostare funzioni prefiltro.

Il parametro *`function`*, contenente la funzione callback, può avere uno dei seguenti valori: (a) una stringa contenente il nome della funzione (b) un array nella forma `array(&$oggetto, $metodo)`, dove `&$oggetto` è il riferimento ad un oggetto e `$metodo` è una stringa contenente il nome di un metodo (c) un array nella forma `array(&$classe, $metodo)` dove `$classe` è un nome di classe e `$metodo` è un metodo statico della classe.

Name

register_resource

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

Usatelo per registrare dinamicamente un plugin risorsa per Smarty. Passate il nome della risorsa e l'array delle funzioni PHP che la implementano. Consultate risorse per i template per maggiori informazioni su come impostare una funzione per caricare i template.

Nota tecnica

Il nome di una risorsa deve avere un minimo di due caratteri di lunghezza. Nomi di risorsa di un solo carattere verranno ignorati ed usati come parte del percorso del file; ad es. $smarty->display('c:/path/to/index.tpl');

L'array di funzioni php *`resource_funcs`* deve avere 4 o 5 elementi. Con 4 elementi, questi saranno le funzioni callback per le rispettive funzioni "source", "timestamp", "secure" e "trusted" della risorsa. Con 5 elementi, il primo deve essere il riferimento all'oggetto oppure il nome della classe relativi all'oggetto o alla classe che implementano la risorsa, mentre i 4 elementi successivi saranno i nomi dei metodi che implementano "source", "timestamp", "secure" e "trusted".

Example 13.24. register_resource

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

Name

trigger_error

```
void trigger_error(string error_msg,
                   int level);
```

Questa funzione può essere usata per produrre in output un messaggio di errore attraverso Smarty. Il parametro *`level`* può contenere uno dei valori usati per la funzione PHP trigger_error(), cioè E_USER_NOTICE, E_USER_WARNING, ecc. Per default il suo valore è E_USER_WARNING.

Name

template_exists

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

Questa funzione verifica se il template specificato esiste. Accetta il percorso del template sul filesystem oppure una stringa che identifica la risorsa del template.

Name

unregister_block

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

Usatela per eliminare dinamicamente una funzione plugin per i blocchi. Passate in *`name`* il nome della funzione di blocco.

Name

unregister_compiler_function

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

Usatela per eliminare dinamicamente una funzione di compilazione. Passate in *name* il nome della funzione.

Name

unregister_function

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

Usatela per eliminare dinamicamente una funzione plugin per i template. Passate il nome della funzione.

Example 13.25. unregister_function

```
<?php
// non vogliamo che i progettisti del template abbiano accesso al filesystem

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

Name

unregister_modifier

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

Usatela per eliminare dinamicamente plugin modificatori. Passate il nome del modificatore del template da eliminare.

Example 13.26. unregister_modifier

```
<?php
// non vogliamo che i progettisti del template eliminino i tag dal contenuto

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

Name

unregister_object

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

Usatela per eliminare un oggetto.

Name

unregister_outputfilter

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

Usatela per eliminare dinamicamente un filtro di output.

Name

unregister_postfilter

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

Usatela per eliminare dinamicamente un postfiltro.

Name

unregister_prefilter

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

Usatela per eliminare dinamicamente un prefiltro.

Name

unregister_resource

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

Usatela per eliminare dinamicamente un plugin risorsa. Passate il nome della risorsa.

Example 13.27. unregister_resource

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

Chapter 14. Caching

Il caching si usa per velocizzare una chiamata a display() o fetch() salvando il suo output su un file. Se una versione della chiamata è disponibile in cache, viene visualizzata questa invece di rigenerare l'output. Il caching può velocizzare tremendamente le cose, specialmente con i template che richiedono maggiori tempi di elaborazione. Se l'output di display() o fetch() viene salvato in cache, un file della cache può concettualmente essere composto di diversi file di template, di configurazione ecc.

Siccome i template sono dinamici, è importante stare attenti a ciò che mettete in cache e per quanto tempo. Ad esempio, se state visualizzando la home page del vostro sito, i cui contenuti non cambiano troppo spesso, può essere utile mettere in cache questa pagina per un'ora o più. D'altra parte, se state visualizzando una pagina con una mappa del tempo atmosferico che viene aggiornata di minuto in minuto, non avrebbe senso mettere in cache questa pagina.

Impostare il Caching

La prima cosa da fare è abilitare il caching. Per farlo bisogna impostare $caching = true (o 1.)

Example 14.1. abilitare il caching

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

$smarty->caching = true;

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

Col caching abilitato, la chiamata alla funzione display('index.tpl') causa la normale generazione del template, ma oltre a questo salva una copia dell'output in un file (la copia in cache) nella $cache_dir. Alla chiamata successiva di display('index.tpl'), verrà usata la copia in cache invece di generare di nuovo il template.

Nota tecnica

I file nella $cache_dir vengono chiamati con nomi simili al nome del template. Sebbene abbiano l'estensione ".php", in realtà non sono script php eseguibili. Non editateli!

Ogni pagina in cache ha un tempo di vita limitato, determinato da $cache_lifetime. Il valore di default è 3600 secondi, cioè 1 ora. Dopo questo tempo, la cache viene rigenerata. E' possibile dare a file singoli il proprio tempo di scadenza impostando $caching = 2. Consultate la documentazione di $cache_lifetime per i dettagli.

Example 14.2. impostare cache_lifetime per singolo file di cache

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

$smarty->caching = 2; // la durata è per singolo file

// impostiamo il cache_lifetime per index.tpl a 5 minuti
$smarty->cache_lifetime = 300;
$smarty->display('index.tpl');

// impostiamo il cache_lifetime per home.tpl a 1 ora
$smarty->cache_lifetime = 3600;
$smarty->display('home.tpl');

// NOTA: l'impostazione seguente di $cache_lifetime non funzionerà
// con $caching = 2. La scadenza per home.tpl è stata già impostata
// a 1 ora, e non rispetterà più il valore di $cache_lifetime.
// La cache di home.tpl scadrà sempre dopo 1 ora.
$smarty->cache_lifetime = 30; // 30 seconds
$smarty->display('home.tpl');
?>
```

Se $compile_check è abilitato, tutti i file di template e di configurazione che sono coinvolti nel file della cache vengono verificati per vedere se sono stati modificati. Se qualcuno dei file ha subito una modifica dopo che la cache è stata generata, il file della cache viene rigenerato. Questo provoca un piccolo sovraccarico, quindi, per avere prestazioni ottimali, lasciate $compile_check a false.

Example 14.3. abilitare $compile_check

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

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

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

Se $force_compile è abilitato, i file della cache verranno sempre rigenerati. Di fatto questo disabilita il caching. $force_compile normalmente serve solo per scopi di debug, un modo più efficiente di disabilitare il caching è di impostare $caching = false (o 0.)

La funzione is_cached() può essere usata per verificare se un template ha una cache valida oppure no. Se avete un template in cache che necessita di qualcosa come una lettura da un database, potete usare questa funzione per saltare quella parte.

Example 14.4. uso di is_cached()

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

$smarty->caching = true;

if(!$smarty->is_cached('index.tpl')) {
    // Non c'è cache disponibile, assegnamo le variabili qui.
    $contents = get_database_contents();
    $smarty->assign($contents);
}

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

Potete mantenere parti di una pagina dinamiche con la funzione del template insert. Diciamo che l'intera pagina può essere messa in cache eccetto un banner che viene visualizzato in fondo a destra nella page. Usando la funzione insert per il banner, potete tenere questo elemento dinamico all'interno del contenuto in cache. Consultate la documentazione su insert per dettagli ed esempi.

Potete eliminare tutti i file della cache con la funzione clear_all_cache(), o singoli file della cache (o gruppi di file) con la funzione clear_cache().

Example 14.5. eliminare la cache

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

$smarty->caching = true;

// eliminiamo tutti i file della cache
$smarty->clear_all_cache();

// eliminiamo solo la cache di index.tpl
$smarty->clear_cache('index.tpl');

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

Cache multiple per una pagina

Potete avere più file di cache per una singola chiamata a display() o fetch(). Diciamo che una chiamata a display('index.tpl') può avere diversi output in base a una certa condizione, e volete cache separate per ciascun caso. Potete farlo passando alla funzione un cache_id come secondo parametro.

Example 14.6. passare un cache_id a display()

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

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

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

Qui sopra passiamo la variabile $my_cache_id a display() come cache_id. Per ogni valore di $my_cache_id verrà generato un file di cache per index.tpl. In questo esempio, "article_id" proveniva dall'URL e viene usato come cache_id.

Nota tecnica

Siate molto prudenti quando passate valori ricevuti da un client (come un browser) a Smarty (o qualsiasi applicazione PHP). Sebbene nell'esempio qui sopra l'uso di article_id proveniente dall'URL sembri molto comodo, potrebbe avere brutte conseguenze. Il valore di cache_id viene usato per creare una directory sul filesystem, quindi se l'utente passa un valore molto lungo come article_id, o se scrive uno script che spedisce velocemente valori casuali, potremmo avere dei problemi sul server. Assicuratevi di validare qualsiasi dato ricevuto in input prima di usarlo. In questo caso, potreste sapere che article_id ha una lunghezza di 10 caratteri, è composto solo di caratteri alfanumerici, e deve essere un article_id valido sul database. Verificatelo!

Assicuratevi di passare lo stesso valore di cache_id come secondo parametro a is_cached() e clear_cache().

Example 14.7. passare un cache_id a is_cached()

```
<?php
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)) {
    // Non c'è un file di cache disponibile, assegnamo le variabili qui.
    $contents = get_database_contents();
    $smarty->assign($contents);
}

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

Potete eliminare tutti i file di cache per un determinato cache_id passando null come primo parametro di clear_cache().

Example 14.8. eliminare tutte le cache per un determinato cache_id

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

$smarty->caching = true;

// eliminiamo tutti i file di cache con "sports" come cache_id
$smarty->clear_cache(null,"sports");

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

In questo modo, potete "raggruppare" i vostri file di cache dando loro lo stesso valore di cache_id.

Gruppi di Cache

Potete raggruppare le cache in modo più elaborato impostando gruppi di cache_id. Per fare questo separate ogni sottogruppo con una barra verticale "|" nel valore di cache_id. Potete usare tutti i sottogruppi che volete.

Potete pensare ai gruppi di cche come ad una gerarchia di directory. Ad esempio, un gruppo di cache "a|b|c" può essere concepito come la struttura di directory "/a/b/c". clear_cache(null,"a|b|c") equivale a

cancellare i file "/a/b/c/*". clear_cache(null,"a|b") sarebbe come cancellare i file "/a/b/*". Se specificate un compile_id, ad esempio clear_cache(null,"a|b","foo"), sarà considerato come un ulteriore sottogruppo "a/b/c/foo/". Se specificate un nome di template, ad esempio clear_cache("foo.tpl","a|b|c"), Smarty tenterà di cancellare "/a/b/c/foo.tpl". NON POTETE cancellare un template specifico sotto più gruppi di cache, ad es. "a/b/*/foo.tpl"; i gruppi di cache funzionano SOLO da sinistra a destra. Dovrete raggruppare i vostri template sotto un singolo sottogruppo di cache per poterli cancellare tutti insieme.

I gruppi di cache non vanno confusi con la gerarchia della vostra directory dei template: i gruppi di cache infatti non sanno qual è la struttura dei template. Ad esempio, se avete una struttura di template tipo "themes/blu/index.tpl" e volete avere la possibilità di cancellare tutti i file di cache per il tema "blue", dovrete creare un gruppo di cache che riproduce la struttura dei template, ad esempio display("themes/blue/index.tpl","themes|blue"), e poi eliminarli con clear_cache(null,"themes|blue").

Example 14.9. gruppi di cache_id

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

$smarty->caching = true;

// eliminiamo tutti i file di cache che hanno "sports|basketball" come primi due g
$smarty->clear_cache(null,"sports|basketball");

// eliminiamo tutti i file di cache che hanno "sports" come primo gruppo di cache
// questo include "sports|basketball", nonché "sports|(anything)|(anything)|(anyth
$smarty->clear_cache(null,"sports");

// eliminiamo il file di cache foo.tpl con "sports|basketball" come cache_id
$smarty->clear_cache("foo.tpl","sports|basketball");


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

Mettere in Cache l'output dei Plugin

A partire dai plugin di Smarty-2.6.0 la possibilità di mettere in cache il loro output può essere dichiarata nel momento in cui li si registrano. Il terzo parametro da passare a register_block, register_compiler_function e register_function si chiama *`$cacheable`* e per default vale true, il che equivale al comportamento dei plugin di Smarty nelle versioni precedenti alla 2.6.0

Quando si registra un plugin con $cacheable=false il plugin viene chiamato tutte le volte che la pagina viene visualizzata, anche se la pagina stessa arriva dalla cache. La funzione del plugin funziona così un poco come una funzione insert.

Al contrario di ciò che avviene in {insert}, gli attributi passati al plugin non vengono, per default, messi in cache. E' possibile però dichiarare che devono essere messi in cache con il quarto parametro *`$cache_attrs`*. *`$cache_attrs`* è un array di nomi di attributi che devono essere messi in cache, in

modo che la funzione del plugin ottenga il valore dell'attributo qual era al momento in cui la pagina è stata salvata sulla cache ogni volta che la cache stessa viene riletta.

Example 14.10. Evitare che l'output di un plugin vada in cache

```
<?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')) {
    // leggiamo $obj dal db e lo assegnamo al template...
    $smarty->assign_by_ref('obj', $obj);
}

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

dove index.tpl è:

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

Il numero di secondi che mancano alla scadenza di $obj cambia ad ogni visualizzazione della pagina, anche se questa è in cache. Siccome l'attributo endtime è in cache, l'oggetto deve essere letto dal database solo quando la pagina viene scritta sulla cache, ma non nelle richieste successive.

Example 14.11. Evitare che un intero blocco di template vada in cache

index.php:

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

function smarty_block_dynamic($param, $content, &$smarty) {
    return $content;
}
$smarty->register_block('dynamic', 'smarty_block_dynamic', false);

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

dove index.tpl è:

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

{dynamic}

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

... qui facciamo altre cose ...

{/dynamic}
```

Quando ricaricate lapagina vedrete che le due date sono diverse. Una è "dinamica", l'altra è "statica". Potete mettere qualsiasi cosa fra {dynamic} e {/dynamic}, sicuri che non verrà messa in cache col resto della pagina.

Chapter 15. Funzioni avanzate

Oggetti

Smarty consente di accedere agli oggetti PHP attraverso i template. Ci sono due modi per farlo. Uno è registrare gli oggetti al template, quindi accedere ad essi attraverso una sintassi simile a quella delle funzioni utente. L'altro modo è di assegnare gli oggetti ai template ed accedere loro come ad una qualsiasi variabile assegnata. Il primo metodo ha una sintassi del template migliore. E' anche più sicuro, perché su un oggetto registrato potete impedire l'accesso a certi metodi o proprietà. D'altra parte, su un oggetto registrato non potete effettuare dei cicli o metterlo in un array di oggetti, ecc. Il metodo che sceglierete dipenderà dalle vostre necessità, ma quando possibile usate sempre il primo metodo, per mantenere la sintassi del template al massimo della semplicità.

Se la security è abilitata, non è possibile accedere a metodi o funzioni private (che cominciano con "_") dell'oggetto. Quando esistono un metodo e una proprietà con lo stesso nome, verrà usato il metodo.

Potete impedire l'accesso a certi metodi e proprietà elencandoli in un array come terzo parametro di registrazione.

Per default, i parametri passati agli oggetti attraverso i template sono passati nello stesso modo in cui li leggono le funzioni utente. Il primo parametro è un array associativo, e il secondo è l'oggetto smarty. Se volete i parameteri passati uno alla volta per ogni argomento come nel tradizionale passaggio di parametri per gli oggetti, impostate il quarto parametro di registrazione a false.

Il quinto parametro opzionale ha effetto soltanto quando *`format`* è `true` e contiene una lista di metodi che devono essere trattati come blocchi. Ciò significa che questi metodi hanno un tag di chiusura nel template (`{foobar->meth2}...{/foobar->meth2}`) e i parametri passati al metodo hanno la stessa struttura di quelli per le funzioni plugin per i blocchi. Questi metodi quindi ricevono 4 parametri *`$params`*, *`$content`*, *`&$smarty`* e *`&$repeat`* e si comportano come funzioni plugin per i blocchi.

Example 15.1. usare un oggetto registrato o assegnato

```
<?php
// the object

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

$myobj = new My_Object;
// registriamo l'oggetto (sarà usato per riferimento)
$smarty->register_object("foobar",$myobj);
// se vogliamo impedire l'accesso a metodi o proprietà, elenchiamoli
$smarty->register_object("foobar",$myobj,array('meth1','meth2','prop1'));
// se vogliamo usare il formato tradizionale per i parametri, passiamo un false
$smarty->register_object("foobar",$myobj,null,false);

// Possiamo anche assegnare gli oggetti. Facciamolo per riferimento quando possibi
$smarty->assign_by_ref("myobj", $myobj);

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

Ed ecco come accedere all'oggetto in index.tpl:

```
{* accediamo all'oggetto registrato *}
{foobar->meth1 p1="foo" p2=$bar}

{* possiamo anche assegnare l'output *}
{foobar->meth1 p1="foo" p2=$bar assign="output"}
the output was {$output}

{* accediamo all'oggetto assegnato *}
{$myobj->meth1("foo",$bar)}
```

Prefiltri

I prefiltri sui template sono funzioni PHP che vengono eseguite sui template prima della compilazione. Sono utili per pre-processare i template allo scopo di rimuovere commenti non desiderati, tenere d'occhio ciò che i progettisti mettono nei template, ecc. I prefiltri possono essere registrati oppure caricati dalla directory plugins con la funzione load_filter() o impostando la variabile $autoload_filters. Smarty passerà il codice sorgente del template come primo parametro, e si aspetterà che la funzione restituisca il codice sorgente risultante.

Example 15.2. uso di un prefiltro

```
<?php
// mettiamo questo nell'applicazione
function remove_dw_comments($tpl_source, &$smarty)
{
    return preg_replace("/<!--#.*-->/U","",$tpl_source);
}

// registriamo il prefiltro
$smarty->register_prefilter("remove_dw_comments");
$smarty->display("index.tpl");
?>
```

Questo rimuoverà tutti i commenti dal sorgente del template.

Postfiltri

I postfiltri sui template sono funzioni PHP che vengono eseguite sui template dopo la compilazione. I postfiltri possono essere registrati oppure caricati dalla directory plugins con la funzione load_filter() o impostando la variabile $autoload_filters. Smarty passerà il codice del template compilato come primo parametro, e si aspetterà che la funzione restituisca il template risultante.

Example 15.3. uso di un postfiltro

```
<?php
// mettiamo questo nell'applicazione
function add_header_comment($tpl_source, &$smarty)
{
    return "<?php echo \"<!-- Created by Smarty! -->\n\"; ?>\n".$tpl_source;
}

// registriamo il postfiltro
$smarty->register_postfilter("add_header_comment");
$smarty->display("index.tpl");
?>
```

Questo farà sì che il template compilato index.tpl appaia così:

```
<!-- Created by Smarty! -->
{* resto del template... *}
```

Filtri di output

Quando il template viene richiamato via display() o fetch(), è possibile eseguire uno o più filtri sul suo output. Ciò è diverso dai postfiltri, perché questi ultimi lavorano sul template compilato prima che venga salvato su disco, mentre i filtri dioutput lavorano sull'output del template quando viene eseguito.

I filtri di output possono essere registrati o caricati dalla directory plugins con la funzione load_filter() oppure impostando la variabile $autoload_filters. Smarty passerà l'output del template come primo argomento, e si aspetterà che la funzione restituisca il risultato dell'esecuzione.

Example 15.4. uso di un filtro di output

```
<?php
// mettiamo questo nell'applicazione
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;
}

// registriamo il filtro
$smarty->register_outputfilter("protect_email");
$smarty->display("index.tpl");

// ora ogni indirizzo email nell'output del template avrà una semplice
// protezione contro gli spambot
?>
```

Funzione di gestione della Cache

Come alternativa all'uso del meccanismo di default per la cache basato sui file, potete specificare una funzione personalizzata di gestione che verrà usata per leggere, scrivere ed eliminare i file in cache.

Create una funzione nella vostra applicazione che Smarty userà come gestore della cache. Impostate il nome di questa funzione nella variabile di classe $cache_handler_func. Smarty ora userà questa funzione per gestire i dati della cache. Il primo parametro è l'azione, che può essere 'read', 'write' o 'clear'. Il secondo parametro è l'oggetto Smarty. Il terzo parametro è il contenuto in cache. In una 'write', Smarty passa il contenuto da mettere in cache in questo parametro. In una 'read', Smarty si aspetta che la funzione prenda questo parametro per riferimento e che lo riempia con i dati della cache. In una 'clear', il parametro non viene usato, quindi passate una variabile dummy. Il quarto parametro è il nome del file del template (necessario per le read e le write), il quinto parametro è il cache_id (opzionale), e il sesto è il compile_id (opzionale).

Nota: l'ultimo parametro ($exp_time) è stato aggiunto in Smarty-2.6.0.

Example 15.5. esempio con l'uso di MySQL per la cache

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

  if($use_gzip && function_exists("gzuncompress")) {
   $cache_content = gzuncompress($row["CacheContents"]);
  } else {
   $cache_content = $row["CacheContents"];
  }
  $return = $results;
  break;
 case 'write':
  // salviamo la cache sul 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 = $results;
  break;
 case 'clear':
  // eliminiamo i dati in cache
  if(empty($cache_id) && empty($compile_id) && empty($tpl_file)) {
   // eliminiamo tutto
   $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:
  // errore, azione non prevista
  $smarty_obj->_trigger_error_msg("cache_handler: unknown action \"$action\"");
  $return = false;
  break;
}
mysql_close($link);
return $return;

}

?>
```

Risorse

I template possono arrivare da varie risorse. Quando fate la display o la fetch di un template, o quando fate la include da un altro template, fornite un tipo di risorsa, seguito dal percorso appropriato e dal nome del template. Se non viene esplicitamente indicato un tipo di risorsa, viene utilizzato il valore di $default_resource_type.

Template della $template_dir

I template provenienti dalla $template_dir non hanno bisogno che indichiate un tipo di risorsa, sebbene possiate indicare file: per coerenza. Basta che forniate il percorso per il template che volete usare, relativo alla directory radice $template_dir.

Example 15.6. uso dei template della $template_dir

```
<?php
$smarty->display("index.tpl");
$smarty->display("admin/menu.tpl");
$smarty->display("file:admin/menu.tpl"); // equivale al precedente
?>
```

And from within Smarty template:

```
{include file="index.tpl"}
{include file="file:index.tpl"} {* equivale al precedente *}
```

Template da qualsiasi directory

I template che si trovano al di fuori della $template_dir richiedono obbligatoriamente che indichiate il tipo di risorsa file: seguito dal percorso assoluto e dal nome del template.

Example 15.7. uso dei template da qualsiasi directory

```
<?php
$smarty->display("file:/export/templates/index.tpl");
$smarty->display("file:/path/to/my/templates/menu.tpl");
?>
```

And from within Smarty template:

```
{include file="file:/usr/local/share/templates/navigation.tpl"}
```

Percorsi su Windows

Se usate una macchina Windows, i percorsi di solito comprendono una lettera di drive (C:) all'inizio del percorso. Accertatevi di usare "file:" nel path, per evitare conflitti di namespace e ottenere i risultati voluti.

Example 15.8. uso di template da percorsi di Windows

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

{* dall'interno di un template *}
{include file="file:D:/usr/local/share/templates/navigation.tpl"}
```

Template da altre risorse

Potete ottenere template da qualsiasi risorsa alla quale sia possibile accedere con PHP: database, socket, directory LDAP, e così via. Potete farlo scrivendo una funzione plugin per le risorse e registrandola a Smarty.

Consultate la sezione plugin risorse per maggiori informazioni sulle funzioni che dovrete creare.

Note

Notate che non è possibile modificare il comportamento della risorsa `file`, ma potete fornire una risorsa che legge i template dal filesystem in maniera diversa registrandola con un altro nome di risorsa.

Example 15.9. uso di risorse personalizzate

```
<?php
// mettete queste funzioni da qualche parte nell'applicazione
function db_get_template ($tpl_name, &$tpl_source, &$smarty_obj)
{
    // fate qui le chiamate al database per ottenere il template
    // e riempire $tpl_source
    $sql = new SQL;
    $sql->query("select tpl_source
                 from my_table
                where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_source = $sql->record['tpl_source'];
        return true;
    } else {
        return false;
    }
}

function db_get_timestamp($tpl_name, &$tpl_timestamp, &$smarty_obj)
{
    // qui facciamo una chiamata al db per riempire $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)
{
    // ipotizziamo che tutti i template siano sicuri
    return true;
}

function db_get_trusted($tpl_name, &$smarty_obj)
{
    // non usata per i template
}

// register the resource name "db"
$smarty->register_resource("db", array("db_get_template",
                                       "db_get_timestamp",
                                       "db_get_secure",
                                       "db_get_trusted"));

// uso della risorsa dallo script php
$smarty->display("db:index.tpl");
?>
```

And from within Smarty template:

```
{include file="db:extras/navigation.tpl"}
```

Funzione di gestione dei template di default

Potete specificare una funzione da usare per ottenere i contenuti del template nel caso in cui non sia possibile leggerlo dalla risorsa appropriata. Un uso possibile di questa funzione è di creare al volo template che non esistono.

Example 15.10. uso della funzione di gestione dei template di default

```
<?php
// mettete questa funzione da qualche parte nell'applicazione

function make_template ($resource_type, $resource_name, &$template_source, &$templ
{
 if( $resource_type == 'file' ) {
  if ( ! is_readable ( $resource_name )) {
   // create il file del template e restituite il contenuto.
   $template_source = "This is a new template.";
   $template_timestamp = time();
   $smarty_obj->_write_file($resource_name,$template_source);
   return true;
  }
    } else {
  // non è un file
  return false;
    }
}

// impostate il gestore di default
$smarty->default_template_handler_func = 'make_template';
?>
```

Chapter 16. Estendere Smarty con i Plugin

La versione 2.0 ha introdotto l'architettura dei plugin, che viene usata per quasi tutte le funzionalità personalizzabili di Smarty. Queste comprendono:

- funzioni
- modificatori
- funzioni di blocco
- funzioni di compilazione
- prefiltri
- postfiltri
- filtri di output
- risorse
- insert

Con l'eccezione delle risorse, viene preservata la compatibilità retroattiva con il vecchio modo di registrare le funzioni di gestione attraverso l'API register_*. Se non usavate questa interfaccia, ma modificavate direttamente le variabili di classe `$custom_funcs`, `$custom_mods` e altre, ora dovrete modificare i vostri script per usare l'API oppure convertire in plugin le vostre funzionalità personalizzate.

Come funzionano i Plugin

I plugin vengono sempre caricati a richiesta. Solo gli specifici modificatori, funzioni, risorse ecc. invocati negli script dei template verranno caricati. Inoltre, ogni plugin viene caricato una volta sola, anche se avete diverse istanze di Smarty in esecuzione nella stessa richiesta.

I pre/postfiltri e i filtri di output sono casi un po' speciali. Siccome non vengono menzionati nei template, devono essere registrati o caricati esplicitamente attraverso le funzioni di interfaccia prima che il template venga eseguito. L'ordine in cui vengono eseguiti più filtri dello stesso tipo dipende dall'ordine in cui sono stati registrati o caricati.

La $plugins_dir può essere una stringa che contiene un percorso oppure un array che ne contiene diversi. Per installare un plugin, è sufficiente installarlo in una delle directory e Smarty lo userà automaticamente.

Convenzioni per i nomi

I file e le funzioni dei plugin devono seguire delle convenzioni molto specifiche per i loro nomi, per poter essere trovati da Smarty.

I file dei plugin devono essere chiamati come segue:

```
tipo.nome.php
```

Dove `tipo` è uno di questi tipi di plugin:

- function
- modifier

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

E `nome` deve essere un identificatore valido (solo lettere, numeri e underscore).

Alcuni esempi: `function.html_select_date.php`, `resource.db.php`, `modifier.spacify.php`.

Le funzioni plugin all'interno dei file dei plugin devono essere chiamate come segue:

```
smarty_tipo, _nome
```

Il significato di `tipo` e `nome` sono gli stessi visti prima.

Smarty produrrà i messaggi di errore appropriati se il file del plugin di cui ha bisogno non viene trovato, o se il file o la funzione hanno un nome non appropriato.

Scrivere Plugin

I plugin possono essere caricati automaticamente dal filesystem da parte di Smarty, oppure possono essere registrati a runtime attraverso le funzioni register_*. Possono anche essere eliminati con le funzioni unregister_*.

Per i plugin che vengono registrati a runtime, i nomi delle funzioni non devono necessariamente rispettare le convenzioni di denominazione.

Se un plugin dipende da qualche funzionalità fornita da un altro plugin (come nel caso di alcuni plugin incorporati in Smarty), il modo corretto di caricare il plugin necessario è questo:

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

Come regola generale, l'oggetto Smarty viene sempre passato ai plugin come ultimo parametro (con due eccezioni: ai modificatori non viene passato l'oggetto Smarty, mentre ai blocchi viene passato `&$repeat` dopo l'oggetto Smarty, per mantenere la compatibilità retroattiva con le vecchie versioni di Smarty).

Funzioni per i template

```
void smarty_function_name($params, &$smarty);

array $params;
```

```
object &$smarty;
```

Tutti gli attributi passati dai template alle funzioni relative sono contenuti in *`$params`* nella forma di un array associativo.

L'output (valore di ritorno) della funzione sostituirà il tag della funzione nel template (ad esempio con la funzione `fetch`). In alternativa, la funzione potrebbe semplicemente svolgere qualche altro compito, senza produrre output (funzione `assign`).

Se la funzione deve assegnare variabili al template, o usare qualche altra funzionalità di Smarty, può usare per questo l'oggetto *`$smarty`* che le viene passato.

Vedere anche: register_function(), unregister_function().

Example 16.1. plugin funzione con output

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     function.eightball.php
 * Type:     function
 * Name:     eightball
 * Purpose:  outputs a random magic answer
 * -------------------------------------------------------------
 */
function smarty_function_eightball($params, &$smarty)
{
    $answers = array('Yes',
                     'No',
                     'No way',
                     'Outlook not so good',
                     'Ask again soon',
                     'Maybe in your reality');

    $result = array_rand($answers);
    return $answers[$result];
}
?>
```

che può essere usata così nel template:

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

Example 16.2. funzione plugin senza output

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     function.assign.php
 * Type:     function
 * Name:     assign
 * Purpose:  assign a value to a template variable
 * -------------------------------------------------------------
 */
function smarty_function_assign($params, &$smarty)
{
    if (empty($params['var'])) {
        $smarty->trigger_error("assign: missing 'var' parameter");
        return;
    }

    if (!in_array('value', array_keys($params))) {
        $smarty->trigger_error("assign: missing 'value' parameter");
        return;
    }

    $smarty->assign($params['var'], $params['value']);
}
?>
```

Modificatori

I modificatori sono piccole funzioni che vengono applicate ad una variabile del template prima che venga visualizzata o usata in qualche altro contesto. I modificatori possono essere concatenati.

```
mixed smarty_modifier_name($value, $param1);

mixed $value;
[mixed $param1, ...];
```

Il primo parametro passato al plugin modificatore è il valore sul quale il modificatore stesso deve operare. Gli altri parametri possono essere opzionali, a seconda del tipo di operazione che deve essere eseguita.

Il modificatore deve restituire il risultato della sua esecuzione.

Vedere anche register_modifier(), unregister_modifier().

Example 16.3. un semplice plugin modificatore

Questo plugin fondamentalmente crea un sinonimo per una delle funzioni incorporate in PHP. Non prevede parametri aggiuntivi.

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     modifier.capitalize.php
 * Type:     modifier
 * Name:     capitalize
 * Purpose:  capitalize words in the string
 * -------------------------------------------------------------
 */
function smarty_modifier_capitalize($string)
{
    return ucwords($string);
}
?>
```

Example 16.4. un plugin modificatore più complesso

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     modifier.truncate.php
 * Type:     modifier
 * Name:     truncate
 * Purpose:  Truncate a string to a certain length if necessary,
 *           optionally splitting in the middle of a word, and
 *           appending the $etc string.
 * -------------------------------------------------------------
 */
function smarty_modifier_truncate($string, $length = 80, $etc = '...',
                                  $break_words = false)
{
    if ($length == 0)
        return '';

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

Funzioni sui blocchi

```
void smarty_block_name($params, $content, &$smarty, &$repeat);

array $params;
mixed $content;
object &$smarty;
boolean &$repeat;
```

Le funzioni sui blocchi sono funzioni che appaiono nel template nella forma: {func} .. {/func}. In altre parole, racchiudono un blocco del template e lavorano sul contenuto di questo blocco. Le funzioni di blocco hanno la precedenza sulle funzioni personalizzate con lo stesso nome, il che significa che non potete avere una funzione personalizzata {func} ed allo stesso tempo una funzione di blocco {func} .. {/func}.

Per default la funzione di implementazione viene chiamata due volte da Smarty: una per il tag di apertura, e una per il tag di chiusura (guardate sotto `&$repeat` per capire come modificare questo comportamento).

Solo il tag di apertura della funzione di blocco può avere attributi. Tutti gli attributi passati dal template alle funzioni relative sono contenuti in *`$params`* nella forma di array associativo. Potete accedere a questi valori, ad esempio, con `$params['start']`. Gli attributi del tag di apertura sono accessibili alla funzione anche in fase di elaborazione del tag di chiusura.

Il valore di *`$content`* dipende se la funzione viene chiamata per il tag di apertura o per quello di chiusura. Nel caso del tag di apertura, sarà `null`, mentre nel caso del tag di chiusura sarà il contenuto del blocco di template. Notate che il blocco sarà già stato elaborato da Smarty, quindi ciò che riceverete sarà l'output del template, non il sorgente.

Il parametro *`&$repeat`* è passato alla funzione per riferimento e le fornisce la possibilità di controllare quante volte il blocco viene visualizzato. Per default *`$repeat`* è `true` alla prima chiamata della funzione (al tag di apertura), e `false` per tutte le chiamate successive (al tag di chiusura). Ogni volta che la funzione termina con il valore di *`&$repeat`* a true, il contenuto compreso fra {func} e {/func} viene valorizzato e la funzione viene chiamata di nuovo con il nuovo contenuto del blocco nel parametro *`$content`*.

Se avete funzioni di blocco nidificate, potete scoprire qual è il blocco genitore attraverso la variabile `$smarty->_tag_stack`. Fate un var_dump() su questa variabile e la struttura dovrebbe apparirvi evidente.

Vedere anche: register_block(), unregister_block().

Example 16.5. funzione di blocco

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     block.translate.php
 * Type:     block
 * Name:     translate
 * Purpose:  translate a block of text
 * -------------------------------------------------------------
 */
function smarty_block_translate($params, $content, &$smarty, &$repeat)
{
    if (isset($content)) {
        $lang = $params['lang'];
        // fate qui una traduzione intelligente di $content
        return $translation;
    }
}
?>
```

Funzioni di Compilazione

Le funzioni di compilazione sono chiamate solo durante la compilazione del template. Sono utili per inserire nel template codice PHP o contenuto statico dipendente dal momento (ad es. l'ora). Se esistono una funzione di compilazione e una funzione personalizzata registrate sotto lo stesso nome, la funzione di compilazione ha la precedenza.

```
mixed smarty_compiler_name($tag_arg, &$smarty);

string $tag_arg;
object &$smarty;
```

Alla funzione di compilazione vengono passati due parametri: la stringa che rappresenta l'argomento tag - fondamentalmente, tutto dal nome della funzione fino al delimitatore finale, e l'oggetto Smarty. Ci si aspetta che la funzione restituisca il codice PHP da inserire nel template compilato.

See also register_compiler_function(), unregister_compiler_function().

Example 16.6. semplice funzione di compilazione

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     compiler.tplheader.php
 * Type:     compiler
 * Name:     tplheader
 * Purpose:  Output header containing the source file name and
 *           the time it was compiled.
 * -------------------------------------------------------------
 */
function smarty_compiler_tplheader($tag_arg, &$smarty)
{
    return "\necho '" . $smarty->_current_file . " compiled at " . date('Y-m-d H:M
}
?>
```

Questa funzione può essere chiamata dal template in questo modo:

```
{* questa funzione viene eseguita solo al momento della compilazione *}
{tplheader}
```

Il codice PHP risultante nel template compilato sarà qualcosa di questo tipo:

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

Prefiltri/Postfiltri

I plugin prefiltro e postfiltro sono molto simili concettualmente; la differenza sta nel momento della loro esecuzione.

```
string smarty_prefilter_name($source, &$smarty);

string $source;
object &$smarty;
```

I prefiltri si usano per processare il codice sorgente del template immediatamente prima della compilazione. Il primo parametro passato alla funzione prefiltro è il sorgente del template, eventualmente modificato da qualche altro prefiltro. Ci si aspetta che il plugin restituisca il sorgente modificato. Notate che questo sorgente non viene salvato da nessuna parte, è usato solo per la compilazione.

```
string smarty_postfilter_name($compiled, &$smarty);

string $compiled;
object &$smarty;
```

I postfiltri si usanno per processare l'output compilato del template (il codice PHP) immediatamente dopo la compilazione stessa, ma prima che il template compilato venga salvato sul filesystem. Il primo parametro passato alla funzione postfiltro è il codice compilato, eventualmente modificato da altri postfiltri. Ci si aspetta che il plugin restituisca la versione modificata di questo codice.

Example 16.7. plugin prefiltro

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     prefilter.pre01.php
 * Type:     prefilter
 * Name:     pre01
 * Purpose:  Convert html tags to be lowercase.
 * -------------------------------------------------------------
 */
 function smarty_prefilter_pre01($source, &$smarty)
 {
     return preg_replace('!<(\w+)[^>]+>!e', 'strtolower("$1")', $source);
 }
?>
```

Example 16.8. plugin postfilro

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     postfilter.post01.php
 * Type:     postfilter
 * Name:     post01
 * Purpose:  Output code that lists all current template vars.
 * -------------------------------------------------------------
 */
 function smarty_postfilter_post01($compiled, &$smarty)
 {
     $compiled = "<pre>\n<?php print_r(\$this->get_template_vars()); ?>\n</pre>" .
     return $compiled;
 }
?>
```

Filtri di Output

I plugin filtro di output lavorano sull'output di un template, dopo che il template è stato caricato ed eseguito, ma prima che l'output che venga visualizzato.

```
string smarty_outputfilter_name($template_output, &$smarty);

string $template_output;
object &$smarty;
```

Il primo parametro passato alla funzione filtro è l'output del template che deve essere elaborato, e il secondo parametro è l'istanza di Smarty che sta chiamando il plugin. Ci si aspetta che questo effettui l'elaborazione e restituisca il risultato.

Example 16.9. plugin filtro di output

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     outputfilter.protect_email.php
 * Type:     outputfilter
 * Name:     protect_email
 * Purpose:  Converts @ sign in email addresses to %40 as
 *           a simple protection against spambots
 * -------------------------------------------------------------
 */
 function smarty_outputfilter_protect_email($output, &$smarty)
 {
     return preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.([a-zA-Z]{2,3}|[0-9]{1,3}))!',
                         '$1%40$2', $output);
 }
?>
```

Risorse

I plugin risorsa vanno considerati un modo generico di fornire sorgenti di template o script PHP a Smarty. Alcuni esempi di risorse: database, directory LDAP, memorie condivisse, socket, e così via.

Per ogni tipo di risorsa deve essere registrato un totale di 4 funzioni. Ogni funzione riceverà la risorsa richiesta come primo parametro e l'oggetto Smarty come ultimo parametro. Il resto dei parametri dipende dalla funzione.

```
bool smarty_resource_name_source($rsrc_name, &$source, &$smarty);

string $rsrc_name;
string &$source;
object &$smarty;

bool smarty_resource_name_timestamp($rsrc_name, &$timestamp, &$smarty);

string $rsrc_name;
int &$timestamp;
object &$smarty;

bool smarty_resource_name_secure($rsrc_name, &$smarty);

string $rsrc_name;
object &$smarty;

bool smarty_resource_name_trusted($rsrc_name, &$smarty);

string $rsrc_name;
object &$smarty;
```

Lo scopo della prima funzione è di recuperare la risorsa. Il suo secondo parametro è una variabile passata per riferimento nella quale memorizzare il risultato. Ci si aspetta che la funzione restituisca `true` se è riuscita a recuperare la risorsa e `false` nel caso opposto.

Lo scopo della seconda funzione è di indicare il momento dell'ultima modifica effettuata sulla risorsa richiesta (nel formato timestamp UNIX). Il secondo parametro è una variabile passata per riferimento nella quale memorizzare il timestamp. Ci si aspetta che la funzione restituisca `true` se è riuscita a determinare il timestamp, e `false` nel caso opposto.

La terza funzione deve restituire `true` o `false`, a seconda che la risorsa richiesta sia sicura o no. Questa funzione è usata solo per risorse di template ma deve ancora essere definita.

La quarta funzione deve restituire `true` o `false`, a seconda che la risorsa richiesta sia considerata affidabile o no. Questa funzione è usata solo per script PHP richiesti con i tag **include_php** o **insert** con l'attributo *`src`*. Comunque, deve ancora essere definita per le risorse di template.

Vedere anche register_resource(), unregister_resource().

Example 16.10. plugin risorsa

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     resource.db.php
 * Type:     resource
 * Name:     db
 * Purpose:  Fetches templates from a database
 * -------------------------------------------------------------
 */
function smarty_resource_db_source($tpl_name, &$tpl_source, &$smarty)
{
    // fate qui le chiamate al db per ottenere il template
    // e popolare $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)
{
    // fate qui la chiamata al db per popolare $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)
{
    // diciamo che tutti i template sono sicuri
    return true;
}

function smarty_resource_db_trusted($tpl_name, &$smarty)
{
    // non si usa per i template
}
?>
```

Insert

I plugin Insert sono usati per implementare le funzioni invocate dai tag **insert** nel template.

```
string smarty_insert_name($params, &$smarty);

array $params;
object &$smarty;
```

Il primo parametro è un array associativo di attributi passati all'insert.

Ci si aspetta che la funzione insert restituisca il risultato che sarà posizionato in luogo del tag **insert** nel template.

Example 16.11. plugin insert

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * File:     insert.time.php
 * Type:     time
 * Name:     time
 * Purpose:  Inserts current date/time according to format
 * -------------------------------------------------------------
 */
function smarty_insert_time($params, &$smarty)
{
    if (empty($params['format'])) {
        $smarty->trigger_error("insert time: missing 'format' parameter");
        return;
    }

    $datetime = strftime($params['format']);
    return $datetime;
}
?>
```

Part IV. Appendici

Table of Contents

Chapter 17. Troubleshooting

Errori Smarty/PHP

Smarty è in grado di trovare molti errori, ad esempio attributi mancanti nei tag, o nomi di variabile non corretti. Quando questo succede, vedrete un errore simile al seguente:

Example 17.1. Errori Smarty

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

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

Smarty vi mostra il nome del template, il numero di riga e l'errore. Dopodiché, vi viene mostrato anche il numero reale di riga nella classe Smarty alla quale si è verificato l'errore.

Ci sono alcuni errori che Smarty non riesce a trovare, ad esempio tag di chiusura mancanti. Questi tipi di errore di solito portano ad errori di parsing PHP al momento della compilazione.

Example 17.2. Errori di parsing PHP

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

Quando vi trovate davanti un errore di parsing PHP, il numero di riga indicato corrisponderà allo script PHP compilato, non al template sorgente. Normalmente dando un'occhiata al template si riesce a capire dov'è l'errore di sintassi. Ecco alcuni errori comuni da controllare: mancanza del tag di chiusura per blocchi {if}{/if} o {section}{/section}, oppure problemi di sintassi all'interno di un tag {if}. Se non riuscite a trovare l'errore, andata nel file compilato PHP e trovate il numero di riga indicato per capire dove si trova l'errore corrispondente nel template.

Chapter 18. Tips & Tricks (trucchi e consigli)

Gestione delle variabili vuote

Certe volte potreste voler stampare un valore di default per una variabile vuota invece di stampare niente, ad esempio " " in modo che gli sfondi delle tabelle funzionino regolarmente. Molti userebbero una {if} per gestire questo caso, ma c'è un modo più veloce con Smarty, che è l'uso del modificatore *default*.

Example 18.1. Stampare   quando una variabile è vuota

```
{* il modo lungo *}

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


{* il modo breve *}

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

Gestione dei default delle variabili

Se una variabile viene usata più volte nel template, applicarle ogni volta il modificatore default può diventare pesante. E' possibile rimediare a ciò assegnando alla variabile il suo valore di default con la funzione assign.

Example 18.2. Assegnazione del valore di default a una variabile del template

```
{* mettete questo da qualche parte in cima al template *}
{assign var="title" value=$title|default:"no title"}

{* se $title era vuota, ora contiene il valore "no title" quando la stampate *}
{$title}
```

Passare una variabile titolo ad un template di intestazione

Quando la maggior parte dei template usa gli stessi intestazione e pié di pagina, è abbastanza comune creare dei template a parte per questi ultimi e poi includerli negli altri. Ma cosa succede se l'intestazione ha bisogno di avere un titolo diverso a seconda della pagina in cui ci troviamo? Potete passare il titolo all'intestazione nel momento dell'inclusione.

Example 18.3. Passare la variabile titolo al template dell'intestazione

```
mainpage.tpl
------------

{include file="header.tpl" title="Main Page"}
{* qui va il corpo del template *}
{include file="footer.tpl"}


archives.tpl
------------

{config_load file="archive_page.conf"}
{include file="header.tpl" title=#archivePageTitle#}
{* template body goes here *}
{include file="footer.tpl"}


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


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

Quando viene disegnata la pagina principale, il titolo "Main Page" viene passato a header.tpl, e quindi sarà usato come titolo. Quando viene disegnata la pagina degli archivi, il titolo sarà "Archives". Notate che nell'esempio degli archivi abbiamo usato una variabile del file archives_page.conf invece che una definita nel codice. Notate anche che se la variabile $title non è impostata viene stampato "BC News", attraverso il modificatore di variabile *default*.

Date

Come regola generale, passate sempre le date a Smarty in forma di timestamp. Questo consente ai progettisti di usare date_format per un pieno controllo sulla formattazione delle date, e rende semplice anche il confronto fra date quando necessario.

Note

A partire da Smarty 1.4.0, potete passare date a Smarty come timestamp unix, timestamp mysql, o qualsiasi altro formato leggibile da strtotime().

Example 18.4. uso di date_format

```
{$startDate|date_format}
```

Questo stamperà:

```
Jan 4, 2001
```

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

Questo stamperà:

```
2001/01/04
```

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

Quando usate {html_select_date} in un template, il programmatore probabilmente vorrà convertire l'output del modulo in un formato timestamp. Ecco una funzione che può aiutarvi in questo.

Example 18.5. convertire le date provenienti da un modulo in timestamp

```
<?php

// stabiliamo che gli elementi del modulo si chiamino
// 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

I template WAP/WML richiedono header php di tipo Content-Type che deve essere passato insieme al template. Il modo più semplice per farlo sarebbe scrivere una funzione utente che stampi l'header. Tuttavia, se usate il caching, questo sistema non funziona, per cui lo faremo con il tag insert (ricordate che i tag insert non vanno in cache!). Assicuratevi che nulla sia inviato in output al browser prima del template, altrimenti l'header non potrà essere spedito.

Example 18.6. usare insert per scrivere un header Content-Type WML

```
<?php

// assicuratevi che apache sia configurato per le estensioni .wml!
// mettete questa funzione da qualche parte nell'applicazione, oppure
// in Smarty.addons.php
function insert_header($params)
{
   // la funzione si aspetta un parametro $content
   if (empty($params['content'])) {
       return;
   }
   header($params['content']);
   return;
}

?>
```

il template *deve* iniziare con il tag insert:

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

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

<!-- 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>
```

Template a componenti

Tradizionalmente, programmare le applicazioni a template funziona così: per prima cosa si accumulano le variabili nell'applicazione PHP (magari con query al database). Poi, si istanzia l'oggetto Smarty, si assegnano le variabili e si visualizza il template. Allora supponiamo di avere, ad esempio, un riquadro che visualizza le quotazioni di Borsa (stock ticker) nel nostro template. In questo caso raccoglieremmo i dati sulle azioni nell'applicazione, poi assegneremmo le variabili al template e le visualizzeremmo. Ma non sarebbe bello poter aggiungere questo stock ticker a qualsiasi applicazione semplicemente includendo il template, senza preoccuparci della parte relativa al caricamento dei dati?

E' possibile fare questo scrivendo un plugin personalizzato che recuperi il contenuto e lo assegni ad una variabile del template.

Example 18.7. template a componenti

```
<?php

// mettiamo il file "function.load_ticker.php" nella directory dei plugin

// scriviamo la funzione che carica i dati
function fetch_ticker($symbol)
{
   // qui metteremo la logica che carica $ticker_info da qualche parte
   return $ticker_info;
}

function smarty_function_load_ticker($params, &$smarty)
{
   // chiamiamo la funzione
   $ticker_info = fetch_ticker($params['symbol']);

   // assegnamo la variabile del template
   $smarty->assign($params['assign'], $ticker_info);
}
?>
```

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

{* in index.tpl *}

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

Stock Name: {$ticker.name} Stock Price: {$ticker.price}
```

Offuscare gli indirizzi E-mail

Vi siete mai chiesti come fanno i vostri indirizzi E-mail a finire su così tante mailing list di spam? Uno dei modi che hanno gli spammer per raccogliere indirizzi E-mail è dalle pagine web. Per combattere questo problema, potete fare in modo che gli indirizzi E-mail appaiano in maniera criptata da javascript nel sorgente HTML, anche se continueranno ad essere visti e a funzionare correttamente nel browser. E' possibile farlo con il plugin mailto.

Example 18.8. Esempio di offuscamento di indirizzo E-mail

```
{* in index.tpl *}

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

Nota tecnica

Questo metodo non è sicuro al 100%. Uno spammer, concettualmente, potrebbe programmare il suo raccoglitore di e-mail per decodificare questi valori, ma non è una cosa semplice.

Chapter 19. Risorse

La homepage di Smarty è http://www.smarty.net/. Potete sottoscrivere la mailing list inviando una e-mail a ismarty-discussion-subscribe@googlegroups.com. L'archivio della mailing list è disponibile a http://groups.google.com/group/smarty-discussion.

Chapter 20. BUGS

Verificate il file BUGS compreso nella distribuzione più recente di Smarty, oppure controllate direttamente sul sito web.