O'REILLY
ONLamp.com
LAMP: THE OPEN SOURCE WEB PLATFORM

Published on **ONLamp.com** (http://www.onlamp.com/)
http://www.onlamp.com/pub/a/php/2003/04/17/pear_smarty.html
See this if you're having trouble printing code examples

The Dynamic Duo of PEAR::DB and Smarty

by Joao Prado Maia
04/17/2003

Our Goals

Total separation of business and template logic is an often-sought goal in advanced web application development. Many times, PHP leads to a compromise. Some code sets implement features by embedding markup in the business logic. Others put business logic in the template side.

While this was true even for my own software development, recently I came to the conclusion that it didn't need to be that way. By simple trial and error, developing new versions of my own libraries and reusing other libraries for my projects, I have found a better way to integrate a database-backed application with a powerful template engine.

I'm obviously talking about PEAR::DB and Smarty here, and I will show throughout this article my experiences and best practices in integrating these two amazing libraries. You will end up with a really nice setup, having an excellent separation between business logic and template logic.

Database abstraction packages are a common part to most programming languages, such as Java's JDBC or Perl's DBI. PHP is no different -- there are countless database abstraction libraries available. PEAR::DB is one of the newer data access abstraction packages, and it was described in my article PEAR::DB Primer. Template libraries are also a pretty common feature of most web related technologies, such as Perl or Cold Fusion. Smarty is a template engine maintained by Monte Ohrt. Andrei Zmievski, a core PHP developer, also contributed to its development. Smarty is different from the existing solutions because of its simplicity and available features, such as the ability to cache templates into PHP scripts which gives a big boost in performance.

The Setup

While I do love to work with PEAR::DB and Smarty, I always try to make my code as "future-proof" as possible. I want to develop my PHP scripts in a way that is forward-compatible (as much as possible) with whatever template engine I wish to use.

That basically means creating a generic wrapper class that hides away the specific features of Smarty. This is the class that I usually use in my personal projects:

```
<?php
require_once(APP_SMARTY_PATH . "Smarty.class.php");

class Template_API
{
    var $smarty;
    var $tpl_name = "";

    function Template_API()
```

```
    {
        $this->smarty = new Smarty;
        $this->smarty->template_dir = APP_PATH . "templates/";
        $this->smarty->compile_dir = APP_PATH . "templates_c";
        $this->smarty->config_dir = '';
    }

    function setTemplate($tpl_name)
    {
        $this->tpl_name = $tpl_name;
    }

    function assign($var_name, $value = "")
    {
        if (!is_array($var_name)) {
            $this->smarty->assign($var_name, $value);
        } else {
            $this->smarty->assign($var_name);
        }
    }

    function bulkAssign($array)
    {
        while (list($key, $value) = each($array)) {
            $this->smarty->assign($key, $value);
        }
    }

    function displayTemplate()
    {
        $this->smarty->display($this->tpl_name);
    }

    function getTemplateContents()
    {
        return $this->smarty->fetch($this->tpl_name);
    }
}
?>
```

APP_PATH and APP_SMARTY_PATH are constants defined in a configuration file. They specify the path on the server where my scripts are stored and where the Smarty library is located, respectively. I usually like to bundle Smarty and PEAR::DB with my own applications, to prevent against problems arising if the system administrator decides to upgrade the system-wide PEAR library.

In any case, the wrapper class above allows me to have scripts that would look similar to the one below:

```
<?php
include_once("config.inc.php");
include_once(APP_INC_PATH . "class.template.php");

$tpl = new Template_API();
$tpl->setTemplate("example.tpl.html");

$tpl->assign("links", "list of links goes here");

$tpl->displayTemplate();
?>
```

APP_INC_PATH is, as you probably guessed, the path on the server where my classes are stored--my "include" directory.

You also probably noticed that the script above is totally clear of any markup or related layout logic.

This is still a very simple script, but you will continue to see the same type of code when using this kind of system.

Smarty Template Engine Functions

Some of the most useful features of Smarty are in the wide variety of functions available to use in the template itself. It might seem odd to be able to use functions from inside of the templates, since the whole idea is to keep the logic separate from your layout. This is template logic however, not business logic.

I often use the `html_options` function. Below you will find a few simple examples of its use:

```
<?php
include_once("config.inc.php");
include_once(APP_INC_PATH . "class.template.php");

$tpl = new Template_API();
$tpl->setTemplate("example.tpl.html");

// first select box here
$states = array(
    "CA" => "California",
    "TX" => "Texas",
    "NY" => "New York"
);
$tpl->assign("states", $states);

// second select box here
$abbreviations = array("CA", "TX", "NY");
$titles = array("California", "Texas", "New York");
$tpl->assign(array(
    "abbreviations" => $abbreviations,
    "titles"        => $titles
));

$tpl->displayTemplate();
?>
```

This example builds the options straight from the associative array:

```
<select name="states">
  {html_options options=$states}
</select>
```

This example builds the options from two separate arrays of values:

```
<select name="other_states">
  {html_options values=$abbreviations output=$titles}
</select>
```

This idea is easily integrated with features available from PEAR::DB. The use of the `get*()` functions is very handy, and appropriate in these cases. Notice the example below:

```
<?php
include_once("config.inc.php");
include_once(APP_INC_PATH . "class.template.php");

$tpl = new Template_API();
$tpl->setTemplate("example.tpl.html");

include_once("DB.php");
$dbh = DB::connect("mysql://user:passwd@server/database");
```

```
$stmt = "SELECT abbrev, title FROM states";
$states = $dbhl->getAssoc($stmt);
$tpl->assign("states", $states);

$tpl->displayTemplate();
?>
```

That's it--instant integration between your database abstraction package and your template engine library.

Handling More Complex Result Sets

While `html_options` is a very convenient way to build select boxes straight from a database result set, Smarty has several other interesting features that will be valuable when using a more complex setup. It's often useful to fetch a complex database result set and pass it to the template. The following example does exactly that:

```
<?php
include_once("config.inc.php");
include_once(APP_INC_PATH . "class.template.php");

$tpl = new Template_API();
$tpl->setTemplate("example.tpl.html");

include_once("DB.php");
$dbh = DB::connect("mysql://user:passwd@server/database");

$stmt = "SELECT * FROM news ORDER BY news_id DESC";
$news = $dbhl->getAll($stmt, DB_FETCHMODE_ASSOC);
$tpl->assign("news", $news);

$tpl->displayTemplate();
?>
```

Here is the companion template:

```
<table>
  <tr>
    <td>ID</td>
    <td>Title</td>
    <td>Author</td>
  </tr>
  {section name="i" loop=$news}
  <tr>
    <td>{$news[i].news_id}</td>
    <td>{$news[i].news_date}</td>
    <td>{$news[i].news_title}</td>
    <td>{$news[i].news_author}</td>
  </tr>
  {/section}
</table>
```

The `section` function works like the `for` loop construct in PHP. A common enhancement is to show things in your templates only when a specific condition happens, like so:

```
<table>
  <tr>
    <td>ID</td>
    <td>Title</td>
    <td>Author</td>
  </tr>
  {section name="i" loop=$news}
```

```
  {cycle values="#CCCCCC,#999999" assign="row_color"}
  <tr bgcolor="{$row_color}">
    <td>{$news[i].news_id}</td>
    <td>{$news[i].news_date}</td>
    <td>{$news[i].news_title}</td>
    <td>{$news[i].news_author}</td>
  </tr>
  {if $smarty.section.i.last}
  <tr>
    <td colspan="4">
      Total of {$smarty.section.i.total} news entries found.
    </td>
  </tr>
  {/if}
  {sectionelse}
  <tr>
    <td colspan="4">
      No news could be found.
    </td>
  </tr>
  {/section}
</table>
```

This example introduced several new features:

- The `cycle` function takes a comma-separated list of values and assigns each to the `row_color` variable. This produces a different row color for each loop iteration.
- The `$smarty` reserved variable holds loop information. You can test if the current iteration is the last one by examining `$smarty.section.i.last`, and can get the total number of rows in the `$news` array of data with `$smarty.section.i.total`.
- The `sectionelse` part shows different information if the `$news` data array is empty. You can use this to show a "There were no results found"-type message in your templates.

Separating Template Files

Like any PHP script, many applications grow so complex that it's best to separate small features into their own scripts. The same thing happens in templates. Often, it's easier to include other snippets than to duplicate the same HTML in several files.

A typical Smarty-based application might resemble:

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

<!-- content goes here -l->

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

This is very useful to make your application more modular and flexible to changes. You usually want to show the copyright notice in all pages of your web site, but you don't want it in pop-up windows or other simple pages. Moving the copyright notice to its own template can make your applications more flexible in regards to template organization.

The `include` template function also supports some advanced features. You can pass flags to your templates to control various output options, for example:

```
{include file="header.tpl.html" title="special title"}
```

Let's say you usually have:

```
<title>website name - slogan goes here</title>
```

In your *header.tpl.html* template, you could have the following:

```
<html>
<head>
{if $title != ""}
        <title>website name - {$title}</title>
{else}
        <title>website name - slogan goes here</title>
{/if}
</head>
<body>
```

That's a pretty simple example, but it shows you what kind of logic you can put in your templates.

Conclusion

I hope this article demonstrated some of the best features of integrating PEAR::DB and Smarty. Have fun and most of all, remember to read the manuals!

- PEAR Manual
- Smarty Manual

Joao Prado Maia is a web developer living in Houston with more than four years of experience developing web-based applications and loves learning new technologies and programming languages.

Return to the PHP DevCenter.