Módulos, Clases y Reemplazo del Controlador

Módulos, Clases y Reemplazo del Controlador

Este artículo fue escrito por Julien Breux y fue publicado por primera vez el 10 de agosto de 2011, en el blog PrestaShop.

Introducción

PrestaShop le permite reemplazar diversos componentes y comportamientos. PrestaShop versión 1.4 consta de dos puntos principales:

- El primero es reemplazar las partes visibles de los módulos (plantillas, JavaScript y lenguaje de hojas de estilo) para que los temas pueden adaptarse mejor a ellos.
- El segundo es reemplazar el comportamiento del software (los archivos de clase y los archivos de controlador) para apuntar a una sección específica de los componentes necesarios.

Reemplazo de módulo

Los módulos se encuentran generalmente en el siguiente formato:

- /modules/my_module/my_module.tpl
- /modules/my_module/my_module.css
- /modules/my_module/my_module.js

PrestaShop le permite reemplazar o sustituir ciertos archivos visibles del módulo por otros nuevos con el mismo tema. No podría ser más simple, haga lo siguiente:

- /themes/prestashop/modules/my_module/my_module.tpl
- /themes/prestashop/css/modules/my_modules/my_module.css
- /themes/prestashop/js/modules/my_modules/my_module.js

Los nuevos archivos se utilizarán cuando muestre su sitio web.

Reemplazo de clase

Reemplazar es una forma de "suplantar" los archivos de clase y los archivos de controlador.
La ingeniosa función de auto-carga de clase de PrestaShop, hace el "cambio" a otros archivos bastantes simples.
Use la herencia para modificar y agregar nuevos comportamientos utilizando las diversas propiedades de clases y métodos.

Por lo general se construyen de la siguiente manera (ejemplo de producto):

- /classes/Product.php
  Esta clase será denominada "ProductCore"
- /controllers/ProductController.php
  Este controlador será denominado "ProductControllerCore"

Usted tendrá que crear un archivo en la carpeta "override/classes/" para "reemplazar" el modelo de clase resultante, tal como:

- /override/classes/Product.php

Esta clase será denominada "producto" y se extiende la clase "ProductCore". ¡Sólo un movimiento de la varita mágica PrestaShop y el hechizo está funcionando!

También puede usar este principio con los controladores y el "reemplazo" de la siguiente manera:

- /override/controllers/ProductController.php

Este controlador será denominado "ProductController" y extiende la clase "ProductControllerCore".

PrestaShop cuenta con ciertas carpetas que puede utilizar para reemplazar elementos tales como mostrar redireccioness (`_Tools.php`) y medir el tiempo de ejecución del hook (`_Module.php`) etc.

Ejemplo 1

Utilizando datos de clase MySQL.php es simplemente imposible tratar de introducir datos en una base de datos diferente a PrestaShop en el mismo servidor MySQL. (¡En serio!)

La solución es utilizar el siguiente reemplazo de la clase MySQLCore:

```
<?php
class MySQL extends MySQLCore
{
  public function __construct($server, $user, $password, $database, $newlink = false)
  {
    $this->_server = $server;
    $this->_user = $user;
    $this->_password = $password;
    $this->_type = _DB_TYPE_;
    $this->_database = $database;

    $this->connect($newlink);
  }

  public function connect($newlink = false)
  {
    if (!defined('_PS_DEBUG_SQL_'))
      define('_PS_DEBUG_SQL_', false);

    if ( $this->_link = mysql_connect($this->_server, $this->_user, $this->_password, $newlink) )
    {
      if(!$this->set_db($this->_database))
        die(Tools::displayError('The database selection cannot be made.'));
    }
    else
      die(Tools::displayError('Link to database cannot be established.'));

    /* UTF-8 support */
    if (!mysql_query('SET NAMES \'utf8\'', $this->_link))
      die(Tools::displayError('PrestaShop Fatal error: no utf-8 support. Please check your server
configuration.'));
    // removed SET GLOBAL SQL_MODE : we can't do that (see PSCFI-1548)
    return $this->_link;
  }
}
?>
```

Para utilizarlo, debe crear una instancia de clase de la siguiente manera:

- For local connection : `new MySQL(`*`DB_SERVER`*`, `*`DB_USER`*`, `*`DB_PASSWD`*`, 'DB_name', true);`
- For remote connection : `new MySQL(`*`DB_SERVER`*`, `*`DB_USER`*`, `*`DB_PASSWD`*`, 'DB_name', true);`

El último parámetro obliga a la creación de una conexión MySQL.

Ejemplo 2

```
/*
 * This override allows you to use ajax-tab.php to make any admin action.
 * Use it for crontask for example
*/
class AdminTab extends AdminTabCore {
    public function ajaxProcess()
    {
        return $this->postProcess();
    }
}
```

Ejemplo 3

```
/*
 * Create a cron task to make periodical database backup
 * (please test before use, I didn't tested it yet)
*/
class AdminTab extends AdminTabCore{
    public function ajaxProcess()
    {
        // here we call the same thing as if we do the old way
        // + with "if" : maybe we want to limit its use to only adding backup :
        // note : find yourself a way to get the file link if you want to send it by mail !
        if (isset($_REQUEST['addbackup']))
            return $this->postProcess();
}

public function displayAjax()
{
    if (sizeof($this->_errors) > 0)
    {
        // handle errors
        // for example, send mail with all error msg
        $content = '';
        foreach($this->_errors as $errorMsg)
           $content .= $errorMsg;

           $lang =  Configuration::get('PS_LANG_DEFAULT');
           // here we send a mail to give the result of the process
           // notice : you have to create template mails files
           Mail::Send($lang, 'backuptaskdone', '[autobackup] report backup error', array('backup_link'=>)),
$to);
    }
    else
    {
        // no error, but maybe we want a mail ?
        if(Configuration::get('PS_NOTICE_SUCCEED_BACKUP'))
        {
           // fileAttachment available, see 9th param of Send() method in classes/Mail.php
               // + we can add a condition "if(Configuration::get('PS_AUTOBACKUP_SEND_FILE'))"
           Mail::Send($lang, 'backuptaskerror', '[autobackup] report backup error', array('vars to use in
tpl'), $to);
        }
    }
    return true;
}
}
```

Ejemplo 4

```
/*
 * with this override, you have a new Smarty variable "currentController"
 * available in header.tpl
 * This allows you to use a different header if you are
 * on a product page, category page or home.
 */
class FrontController extends FrontControllerCore{
  public function displayHeader()
  {
    self::$smarty->assign('currentController',get_class($this));
    return parent::displayHeader();
  }
}
```

Conclusiones

Los archivos del núcleo no son modificados al reemplazar. Esta técnica le permite personalizar su tienda PrestaShop y monitorear cómo evoluciona el software. Las actualizaciones están facilitadas.