Surcharge et override

Surcharge et override

Cet article a été écrit par Julien Breux, et publié sur le blog de PrestaShop le 9 août 2011.

Introduction

PrestaShop vous permet de surcharger différents éléments et comportements. Il existe en réalité deux grands principes dans la version 1.4 de PrestaShop :

- Le premier consiste à surcharger les éléments de vue des modules (Templates, JavaScript et feuilles de styles) afin que les thèmes puissent s'adapter au mieux à ceux-ci.
- Le second principe consiste quant à lui à surcharger les comportements métiers (fichiers classes et fichiers contrôleurs) afin de ne cibler qu'une partie des éléments désirés.

Surcharge des modules

Les modules sont généralement constitués de cette façon :

- /modules/mon_module/mon_module.tpl
- /modules/mon_module/mon_module.css
- /modules/mon_module/mon_module.js

PrestaShop vous permet de « surcharger », autrement dit de remplacer certains fichiers de vue des modules par de nouveaux situés dans le thème. Pour cela, rien de plus simple, il vous suffit de procéder de la manière suivante :

- /themes/prestashop/modules/mon_module/mon_module.tpl
- /themes/prestashop/css/modules/mon_modules/mon_module.css
- /themes/prestashop/js/modules/mon_modules/mon_module.js

Ainsi, lorsque vous allez afficher votre site, ces nouveaux fichiers seront utilisés.

Surcharge des classes

L'override est un concept qui permet de "surcharger" les fichiers classes et les fichiers contrôleurs. L'utilisation ingénieuse de l'auto-chargement des classes de PrestaShop permet donc un "switch" assez facile de ces différents fichiers.
Il vous est donc possible grâce à l'héritage de modifier ou ajouter de nouveaux comportements via les propriétés et les méthodes de ces différentes classes.

En temps normal, ceux-ci sont architecturés de cette façon (exemple pour la partie produit) :

- /classes/Product.php
  Cette classe se nommera "ProductCore"
- /controllers/ProductController.php
  Ce contrôleur se nommera "ProductControllerCore"

Afin de "surcharger" la classe modèle produit, vous devez créer un fichier situé dans le dossier "override/classes/" comme ceci :

- /override/classes/Product.php

Cette classe se nommera "Product" et étendra la classe "ProductCore".
Ainsi, la magie de PrestaShop peut opérer !

De la même façon, vous pouvez utiliser ce principe avec les contrôleurs et donc "surcharger" de cette façon :

- /override/controllers/ProductController.php

Cette classe se nommera "ProductController" et étendra la classe "ProductControllerCore".

PrestaShop met à votre disposition certains fichiers vous permettant de mettre en œuvre quelques overrides comme l'affichage des redirections (_Tools.php), le calcul du temps d'exécution des "hooks" (_Module.php), etc...

Exemple 1

Lorsque l'on essaye de taper dans une base de données différente de celle de Presta, sur le même serveur MySQL, c'est simplement impossible avec la classe MySQL.php de base (et oui !).

La solution : utiliser cette surcharge de la classe 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;
	}
}
?>
```

Pour l'utiliser, il suffit juste d'instancier la classe de la manière suivante :

- Pour une connexion locale : new MySQL(*DB_SERVER*, *DB_USER*, *DB_PASSWD*, 'le_nom_de_la_bdd', true);
- Pour une connexion distante : new MySQL($ip_or_server, $user, $pass, $dbname, true);

Le dernier paramètre force la création d'une nouvelle connexion MySQL.

Exemple 2

Cette surcharge vous permet d’utiliser ajax-tab.php pour toute action d’administration.

A utiliser pour les taches cron par exemple.

```
/*
 * 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();
    }
}
```

Exemple 3

Créer une tache cron pour faire une sauvegarde périodique de la base de données.
Merci de faire un test avant utilisation !

```
/*
 * 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;
}
}
```

Exemple 4

Avec cette surcharge, vous avez une nouvelle variable Smarty « currentController » disponible dans header.tpl
Ceci vous permet d'utiliser un header différent selon que vous soyez sur une page produit, une page catégorie ou sur la home.

```
/*
 * 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();
  }
}
```

Conclusions

Avec la surcharge, les fichiers « core » ne sont pas modifiés. Cette technique vous permet donc de personnaliser votre boutique PrestaShop tout en ayant la possibilité de suivre l'évolution du logiciel. Les mises à jour en sont facilitées