Prestashop - Développement d'un module

David Annebicque

Décembre 2016

1 Objectif de cette deuxième partie sur Prestashop

Cette partie est un aperçu de la création d'un module dans Prestashop.

Il faut avoir en tête que depuis cette version de Prestashop (la 1.7), il y a eu un changement important avec l'intégration de Symfony. L'ensemble de Prestashop n'est pas encore transposée en Symfony, et nous avons donc quelque chose d'un peu hybride.

Néanmoins, cette transition implique que tout ce qui fonctionnait sur la 1.6, et notamment les thèmes, ne vont plus fonctionner sur la 1.7 [1]. Mais du coup, il est théoriquement possible d'utiliser Twig plutôt que SMarty comme moteur de template.

La documentation officielle sur le développement de module pour la version 1.7 se trouve ici `http://developers.prestashop.com/module/index.html`

2 Pré-requis

2.1 Installation

Nous partirons sur l'installation de la première séance (version 1.7 de prestashop).

2.2 Configuration

Il est nécessaire de configurer Prestashop pour avoir un environnement de développement efficace.

1. Lire `http://build.prestashop.com/news/module-development-changes-in-17/`

1. Désactiver le cache (Administration > Paramètres Avancés > Performances)
2. Cocher "Forcer la compilation à chaque appel" (Administration > Paramètres Avancés > Performances)
3. Activer le "mode Debug" (Administration > Paramètres Avancés > Performances)
4. N'oubliez pas de sauvegarder.

3 Travail à faire

3.1 Création du module

1. Créer un répertoire dans le répertoire module. Nommer ce répertoire du nom de votre module (exemple monmodule).
2. Créer un fichier monmodule.php (si votre module se nomme monmodule)
3. Mettre le code ci-dessous, en adaptant.

```php
<?php
if (!defined('_PS_VERSION_'))
{
        exit;
}

class MyModule extends Module
{
        public function __construct()
        {
                $this->name = 'mymodule';
                $this->tab = 'front_office_features';
                $this->version = '1.0.0';
                $this->author = 'Firstname Lastname';
                $this->need_instance = 0;
                $this->ps_versions_compliancy =
                ↪ array('min' => '1.6', 'max' =>
                ↪ _PS_VERSION_);
                $this->bootstrap = true;

                parent::__construct();

```

```
21                  $this->displayName = $this->l('My
                    ↪ module');
22                  $this->description = $this->l('Description
                    ↪ of my module.');
23
24                  $this->confirmUninstall = $this->l('Are
                    ↪ you sure you want to uninstall?');
25
26                  if (!Configuration::get('MYMODULE_NAME'))
27                  $this->warning = $this->l('No name
                    ↪ provided');
28          }
29  }
```

Ce code permet uniquement de configurer les informations sur le module que vous souhaitez créer. Il est maintenant possible de le trouver dans la liste des modules disponibles de prestashop.

Vous devez l'installer depuis l'administration. Il est possible de personnaliser le processus d'installation (méthode install) et de désinstallation (méthode uninstall). `http://developers.prestashop.com/module/05-CreatingAPrestaShop17Modu 02-CreatingAFirstModule.html`

Une fois le module installé, celui-ci ne fait rien... Il faut maintenant ajouter des méthodes qui vont venir écrire des éléments dans le front ou le back office. Pour cela, il faut manipuler les "hooks" de prestashop, c'est à dire associer des éléments à des points d'ancrage.

3.2 Affichage sur le front-office

Pour affecter notre module à un ou plusieurs hook, il est nécessaire d'écrire la méthode install. Il est possible d'autoriser l'accrochage de notre module sur plusieurs "hook". Ceci laissera la possibilité aux administrateur de placer le module où ils le souhaitent.

```
1   public function install()
2   {
3           if (Shop::isFeatureActive())
4                   Shop::setContext(Shop::CONTEXT_ALL);
5
6           return parent::install() &&
7                   $this->registerHook('leftColumn') &&
8                   $this->registerHook('header') &&
```

```
9                   Configuration::updateValue('MYMODULE_NAME',
    ↪  'my friend');
10  }
```

Le code ci-dessus est un exemple de méthode *install* qui autorise le module à être sur la colonne de gauche ou dans le header. **Pour l'exercice qui nous intéresse vous veillerez à n'autoriser que la colonne de droite ou de gauche.**

A ce stade, notre module ne fait toujours rien. Il faut maintenant définir pour chacun des hook "autorisé" le comportement de votre module, et faire appel aux templates.

A noter que les templates doivent se trouver dans un premier temps dans le répertoire views/templates/hook/ et porter le nom du module.

Citation de la documentation officielle : Attaching code to a hook requires a specific method for each:

- hookDisplayLeftColumn(): will hook code into the left column - in our case, it will fetch the MYMODULE_NAME module setting and display the module's template file, mymodule.tpl, which must be located in the /views/templates/hook/ folder.
- hookDisplayRightColumn(): will simply do the same as hookDisplayLeftColumn(), but for the right column.
- hookDisplayHeader(): will add a link to the module's CSS file, /css/mymodule.css.

Ce qui donne le code ci-dessous :

```
1           public function hookDisplayLeftColumn($params)
2           {
3                   $this->context->smarty->assign(
4                   array(
5                   'my_module_name' =>
    ↪  Configuration::get('MYMODULE_NAME'),
6                   'my_module_link' =>
    ↪  $this->context->link->getModuleLink('mymodule',
    ↪  'display')
7                   )
8                   );
9                   return $this->display(__FILE__,
    ↪  'mymodule.tpl');
10          }
11
```

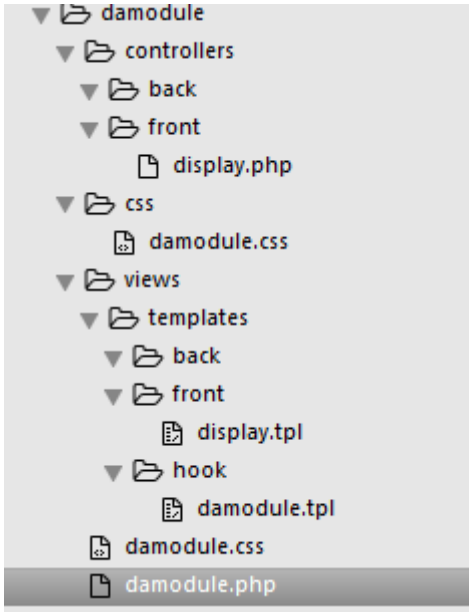
```
12          public function hookDisplayRightColumn($params)
13          {
14                  return
    ↪  $this->hookDisplayLeftColumn($params);
15          }
16
17          public function hookDisplayHeader()
18          {
19                  $this->context->controller->addCSS(
20                          $this->_path.'css/mymodule.css',
    ↪  'all'
21                  );
22          }
```

Une fois tout le code en place (ajoutez un peu de CSS et quelques lignes HTML dans le template), vous devez retourner dans le back-office, dans l'onglet apparence, puis position, rechercher votre module dans la liste déroulante (il peut être nécessaire de le désinstaller et le réinstaller pour la prise en compte des paramètres d'installation), vous devez voir qu'il peut être installé dans les deux colonnes (droite et gauche). Activez le dans une des colonnes.

Le code ci-dessous est un exemple de template pour afficher le lien (my_module_link) sur le front office de la boutique.

```
1  <div id="damodule_block_home" class="block">
2          <h4>Welcome!</h4>
3          <div class="block_content">
4                  <p>Hello,
5                  {if isset($da_module_name) &&
    ↪  $da_module_name}
6                          {$da_module_name}
7                  {else}
8                          World
9                  {/if}
10                 !
11                 </p>
12         <ul>
13                 <li><a href="{$da_module_link}"
    ↪  title="Click this link">Click me!</a></li>
14         </ul>
15 </div>
16 </div>
```

Ci-dessous l'arborescence que vous devriez avoir. Tous les fichiers ne sont pas encore réalisé a ce stade du TP.



A ce stade, vous devriez voir votre module apparaître dans votre boutique. Si vous cliquez sur le lien, vous obtiendrez un message d'erreur.

Pour réaliser cette partie il faut ajouter un contrôleur et une vue.

Ci-dessous le code du contrôleur. Un contrôleur front hérite de la classe : ModuleFrontController.

```
1  <?php
2  class damoduledisplayModuleFrontController extends
   ↪  ModuleFrontController
3  {
4          public function initContent()
5          {
6                  parent::initContent();
7
8                  $this->context->smarty->assign(array(
9                          'test' => 'toto',
10                 ));
11                 $this->setTemplate('module:damodule/views/templates/front/d
                   ↪
12         }
13 }
```

Il faut ensuite ajouter la vue dans le bon répertoire avec, par exemple le code ci-dessous.

```
{extends file='page.tpl'}

{block name="page_content"}
        {$test}
        Welcome!
{/block}
```

3.3 Accès aux données

Accès à la base de données :

```
$sql = new DbQuery();
```

Création de la requête, par exemple un select sur tous les produits. On n'indique pas le préfixe, juste la partie située après le _ du nom de la table.

```
$sql->select('*');
$sql->from('product', 'p');
```

Vous trouverez sur ce lien, des exemples et les possibilités disponibles pour écrire une requète : http://developers.prestashop.com/module/04-DivingIntoPSCoreDevelop 01-AccessingTheDatabase.html

Ensuite, pour exécuter cette requête, il faut écrire :

```
$exe = Db::getInstance()->executeS($sql);
```

3.4 Travail demandé

Le travail demandé pour cette séance est de créer et personnaliser un module qui affichera un lien vers une page présentant des statistiques sur votre boutique.

Vous afficherez par exemple : le nombre de produit, le prix moyens, le nombre de fournisseur, ou encore le nombre de commande. Vous êtes libre de la forme (tableau, graphique, ...), mais soyez lisible et propre dans la présentation des données.