Développer un framework PHP (suite)

Smarty, comme Smart.

Dans notre précédent article sur les méthodes de développement en PHP, nous avons abordé l'aspect organisation. Nous avons laissé de côté tout l'aspect présentation, pourtant pierre angulaire d'un site web réussi. Si nous n'affichons rien, je doute que notre positionnement au hit parade soit des plus brillants (même si, réflexion faite, un site comme perdu.com a gagné sa célébrité de par son absence de contenu). Templates et moteurs de templates en PHP, ou "comment procéder pour afficher correctement des pages HTML".

L'art de l'affichage

La question n'est pas là, nous voulons (si, nous le voulons), afficher des informations à l'écran.

Nous pouvons le faire, via deux méthodes:

- echo, printf & autres variantes.
- Contenu hors des balises php.

Notre démonstration en 5 lignes de code :

```
<?php
echo "avec un echo", "toujours avec un echo";
printf ("avec un printf c'est pareil");
?>
<!- du html ->
Et voilà comment faire en dehors de php sans echo ni printf.
<!- encore du html ->
```

(Note aux hackers chevronnés : oui, nous pouvons aussi ouvrir un socket sur un serveur distant pour rediriger ce que nous lisons sur le poste client, nous pouvons utiliser la gdlib, générer du pdf, des images, lire des fichiers, mais soyez raisonnables, nous cherchons à démontrer un point important)

Maintenant que nous sommes rassurés sur notre capacité à afficher des éléments sur le navigateur de nos internautes, interrogeons-nous sur la façon de procéder. La première considération : "pourquoi ne pas dire ce que j'ai à dire au moment où j'ai envie de le dire ?". La deuxième, plus réfléchie : "pensons à tout ce que nous avons à dire et réfléchissons sur la forme et sur la manière de l'annoncer". Malgré mes efforts pour ne pas entamer le suspens, vous vous doutez que nous allons opter pour la deuxième solution. Tentons de démontrer pourquoi, avec un exemple volontairement dédié, mais proche de la réalité.

Le scénario d'un problème

Vous développez le site personnel d'un ami, non informaticien, qui s'étonne tous les jours de l'étendue de vos talents (profitons-en, nous sommes entre nous). Cet ami a tenu, devant la somme de travail vous étant assignée, à prendre en charge la partie graphique. Ce n'est pas plus mal me direz-vous, vous êtes informaticien, pas graphiste... quoique, à la vue de la charte produite par votre ami, vous nourrissez des doutes (mais, après tout, c'est son site).

Vous voici muni d'un ensemble de code HTML pour le transformer en un système de news.

Courageusement, vous commencez à insérer au milieu des balises le code PHP pour vous connecter à la base, lancer vos requêtes, transformer les multiples tables imbriquées, pour finir par obtenir la première page de news du site. Excellent résultat, félicitations ! Votre ami, par souci de pouvoir satisfaire tous les anciens élèves de son école à qui le site est destiné, souhaite disposer d'une fonctionnalité d'export pour que ces nouveaux travailleurs (les anciens élèves... suivez un peu) puissent télécharger les nouvelles dans leur PDA. Idée comme une autre, vous vous exécutez (dans le sens "vous procédez au développement"). Vous récupérez les bribes de code PHP de connexion à la base, en extrayez les parties graphiques et obtenez un nouveau fichier *news_export.php*.

Maintenant que tout commence à fonctionner, votre ami souhaite renommer le champ "content" en "contenu". Même si cela ne représente rien de bien compliqué, vous en convenez, il vous faut changer vos deux fichiers (celui affichant les news et celui les exportant) pour une seule modification qui n'a rien à voir avec la façon d'afficher les éléments.

Votre ami, toujours dans un souci de ne pas vous accabler, essaie de lui-même de changer la charte graphique, et le site tombe en panne. Après avoir passé la moitié de la soirée qui suit à lui expliquer que echo n'est pas un attribut de style, vous réintégrez tous les éléments supprimés.

L'histoire peut continuer indéfiniment, mais il est évident qu'une telle éthique de programmation n'est pas satisfaisante car vous passez votre temps à refaire ou à corriger ce que vous avez déjà fait, et si vous maintenez plusieurs sites de front, vous n'aurez bientôt plus de temps à vous, ni d'amis d'ailleurs. Vous noterez également que dans des conditions d'entreprise, ce problème est d'autant plus prononcé, dans le sens où les acteurs (graphistes, clients, développeurs et chefs de projets) sont tous distants et que les avis comme les besoins changent (ou sont affinés) tout au long de la phase de développement. Nous ne pouvons pas nous permettre de remettre en cause nos procédés à chaque changement d'interface (un peu comme si nous devions ressaisir tout le contenu d'un document texte pour pouvoir changer la couleur d'un titre). De même, dans le milieu professionnel, vous serez souvent amené à vouloir réutiliser du travail que vous avez déjà fait, à la condition que cette tâche de reprise ne représente pas plus de temps que la refonte du système.

La solution

Arrive une idée de génie, née de ce constat : “séparons la logique de présentation de la logique de récupération des informations.”

Nous décidons donc de procéder à des changements majeurs dans notre façon de faire, et transformons notre code précédent en code séparé. Nous allons, pour ce faire, utiliser un moteur de templates appelé Smarty (http://smarty.php.net) qui a la particularité de compiler pour nous ses templates en PHP (nous reviendrons sur ce que cela implique, et nous reviendrons aussi sur l'utilité d'un moteur de template).

```
<?php
   include ('Smarty.class.php');
   $noMoreHTMLInPhp = & new Smarty ();

   //récupération de nos éléments graphiques.
   mysql_connect ();
   mysql_query ('select id_news, title_news, content_news from news');
   while ($r = mysql_fetch_object ()){
      $noMoreHTMLInPhp->append ('NewsList', $r);
   }
   mysql_close ();

   //nous en avons terminé avec la récupération des news, procédons à l'affichage
   $noMoreHTMLInPhp->display ('news.list.tpl');
?>
```

Ce que nous avons fait est simple. Dans cette page, il n'existe que du code PHP, bien identifiable, pour récupérer des données depuis une base. Au tout début, nous réalisons l'inclusion de la classe Smarty, le moteur de template. Nous nous connectons à notre base, lançons la requête, puis stockons au fur et à mesure les résultats dans l'objet Smarty. Une fois que nous en avons terminé avec l'accumulation des news, nous demandons à Smarty de les afficher dans un template nommé *news.list.tpl.*

```
Fichier news.list.tpl
<table>
  <tr><th>Titre</th><th>Contenu</th></tr>
  {foreach from=$NewsList item=$new}
   <tr><td>{$new->title_news}</td><td>{$new->content_new}</td></tr>
  {/foreach}
</table>
```

Voilà ce que l'on appelle un template, un fichier qui ne contient que de la logique de présentation. Si nous nous souvenons bien de notre problème d'export, il n'y à la rien de bien compliqué pour répondre au besoin : “Il nous suffit de créer un nouveau template”.

```
news.export.tpl
{foreach from=$NewsList item=$new}
   {$new->title_news};{$new->content_new}
{/foreach}
```

Dans notre fichier PHP il nous suffit alors de rajouter un traitement particulier qui choisit comment afficher nos informations, par exemple.

```
<?php
   //[...]
   //Voila la seule ligne de news.php que nous devons modifier
   $noMoreHTMLInPHP->display (isset ($_GET['export']) ? 'news. export
.tpl' :  'news.list.tpl');
?>
```

Dorénavant, lorsque notre système de news rencontrera “export” dans l'URL, il affichera ses données dans un template dédié à cet export, plutôt que de les afficher dans un tableau HTML. Si vous voulez rajouter le champ “résumé”, seul le fichier de récupération des news est à modifier. Pour parfaire notre fonctionnalité de news, nous décidons de proposer une fonctionnalité d'export RSS, pour permettre aux autres sites d'afficher et d'utiliser nos news. Encore une fois, rien de plus simple, il suffit de créer un troisième template pour répondre au besoin. Par la même occasion, nous pouvons même systématiser le choix du template par une variable dédiée dans nos URLs (dans notre exemple kind).

```
<?php
   //Voilà le code de notre récupération de news.
   $noMoreHTMLInPHP->display (Template::get ('myTemplate'););
?>
```

et pour être exhaustif, voici le code de notre méthode Template::get ($fileID);

```
<?php
   class Template {
      function get ($fileID){
            $kind = isset (_GET['kind']) ? _GET['kind'] : '';
            if (is_readable ($fileID'.'.$kind.'.tpl')){
               return $fileID'.'.$kind.'.tpl';
            }else{
               return $fileID.'.tpl';
            }
      }
   }
?>
```

En construisant nos liens correctement, nous sommes maintenant capables de choisir tout type de template pour un contenu donné, sans nous soucier de ces choix dans les appelants de nos méthodes.

Mais pourquoi un “moteur” de templates ?

Les esprits critiques se sont réveillés et s'interrogent de fait sur la nécessité d'un “moteur” de templates. Attention, nous ne nous interrogeons pas sur la nécessité d'utiliser des templates (les fichiers), mais sur la nécessité d'utiliser un moteur de template (l'objet qui effectue le rapprochement donnée / fichier).

En effet, en PHP, nous disposons d'un superbe moteur de template : PHP lui même. Qu'est-ce qui nous empêche de préférer la syntaxe <?php

echo $monInformation; ?> à {$monInformation}. Après tout, nous connaissons déjà PHP, inutile d'apprendre un nouveau langage, et niveau performances, inutile d'inclure une bibliothèque volumineuse pour afficher nos informations. De plus, le temps d'analyse du fichier est sûrement long, et si j'utilise PHP, je suis sûr que PHP fonctionnera là où PHP fonctionne ??? (ce qui n'est pas le cas avec une bibliothèque : je ne suis pas sûr que ma bibliothèque fonctionne sur cette "plate-forme PHP et cette configuration" particulière).

Nous aurions donc un système PHP du style:

```
<?php
   include ('Smarty.class.php');
   $noMoreHTMLInPhp = array ();

   //récupération de nos éléments graphiques.
   mysql_connect ();
   mysql_query ('select id_news, title_news, content_news from news');
   while ($r = mysql_fetch_object ()){
      $noMoreHTMLInPhp['NewsList'][] = $r;
   }
   mysql_close ();

   //nous en avons terminé avec la récupération des news, procédons à l'affichage
   include ('templates/news.list.php');
?>
```

Maintenant, notre template "PHP"

```
<table>
  <tr><th>Titre</th><th>Contenu</th></tr>
  <?php foreach ($noMoreHTMLInPhp['NewsList'] as $new) { ?>
   <tr>
     <td><?php echo $new->title_news; ?></td>
     <td><?php echo $new->content_new; ?></td>
   </tr>
  <?php } ?>
</table>
```

Bien, oui, vous avez raison. Toutefois, il existe de nombreux avantages à utiliser un moteur de templates, dans le sens où ce dernier est paramétrable selon vos souhaits. Vous voulez que tous les caractères spéciaux de vos variables soient affichés avec leurs équivalents HTML, sans que vous ne soyez obligé de vous en soucier ? (é => &eacute, â => à, ...). Avec un moteur de template, et notamment avec Smarty, c'est possible, il vous suffit d'utiliser un "modificateur de variable" par défaut, qui se chargera de procéder à ce remplacement au bon moment.

Autre exemple: Vous ne souhaitez afficher que les 30 premiers caractères du contenu de la news dans votre liste ? Avec Smarty, utilisez encore un modificateur de variable: {$contenu|truncate:30}. En PHP, cela devient tout de suite plus long à écrire: <?php echo strlen ($contenu) > 30 ? substr ($contenu, 0, 30).'...' : $contenu; ?>

Vous souhaitez afficher directement un tableau associatif dans un tableau HTML ? Avec Smarty: {html_table loop=$table} en PHP : vous voilà obligé de prendre quelques instants pour y réfléchir.

Bien sûr, vous pouvez développer vos bibliothèques de fonctions pour prendre en charge ces balises particulières, mais alors vous vous retrouvez vous aussi à utiliser de lourdes bibliothèques de fonctions, d'autant plus que ces dernières seront sûrement déjà développées dans un moteur de template classique.

Bref, en un mot, Smarty intègre nombre d'éléments qui vous permettent d'accélérer vos développements en vous facilitant la vie.

De plus, bien que nous ne l'ayons pas vu, il est tout à fait possible avec Smarty (et avec la plupart des moteurs de templates), de développer vos propres balises. Si vous souhaitez avoir un calendrier html en réponse à la balise {monCalendrier}, c'est possible. Vous pouvez aussi définir vos propres modificateurs, vos propres fonctions de bloc et même vos propres fonctions de compilation !

Ce qu'il faut retenir de l'utilisation des templates

L'utilisation des templates va garantir à votre application son indépendance vis-à-vis de son interface. Grâce à cette indépendance, il vous sera possible de réutiliser systématiquement les parties "métiers" des fonctionnalités que vous allez développer. Si vous réalisez un système d'agenda, vous pourrez réutiliser toutes les briques métiers de ce dernier, que l'affichage en découlant se fasse en liste, en calendrier ou par export, en des formats ICalendar.

Ce que nous attendons d'un tel système, c'est de ne pas être lié à cette interface, de pouvoir la modifier quand bon nous semble sans impacts, et de pouvoir réutiliser le "métier" développé.

Maintenant, que vous optiez ou non pour un "moteur" de templates, et si c'est le cas, que vous choisissiez Smarty ou un autre, c'est une question "technique", qui n'entame en rien l'objectif à atteindre.

A suivre. (Configurer et étendre les fonctionnalités de Smarty, Développer avec des objets métier, Compléter notre framework d'après les trois notions abordées, Exemple du framework Copix).

Gérald Croes