

Conception avancée d'une galerie d'images générée à la volée - utilisation d'un système de templates

par [Pierre-Baptiste Naigeon](#)

Date de publication : 15 Septembre 2006

Dernière mise à jour : 15 Septembre 2006

Suite au premier article de la série, "Exemple de conception d'une galerie d'images générée à la volée", nous allons voir comment adapter le code afin d'optimiser le fonctionnement de l'application en utilisant les templates (dans cet exemple, nous utiliserons le moteur Smarty).

- I - Introduction
 - I-A - Préambule
 - I-B - Les templates, qu'est ce que c'est ?
 - I-C - Ce qu'il nous faut modifier
- II - Transformation du PHP
 - II-A - affichage-galerie.php
 - Fonction ajoute_lien()
 - Fonction affichage()
 - Nettoyage du code
 - II-B - affichage-image.php
- III - Création des gabarits HTML
 - III-A - affichage-galerie.tpl
 - III-B - affichage-image.tpl
- IV - Conclusion

I - Introduction

I-A - Préambule

Pour aborder cet article, vous devez préalablement avoir (au moins) récupéré le code final du premier article : [Exemple de conception d'une galerie d'images générée à la volée](#).

Pour la gestion de l'affichage, nous allons utiliser le [moteur de templates Smarty](#). Je vous laisse suivre les instructions détaillées sur le site pour l'installer.

I-B - Les templates, qu'est ce que c'est ?

Les templates sont un moyen simple de séparer logique de traitement de la logique d'affichage.

Ils permettent par exemple à un graphiste et à un développeur de collaborer sur un même projet sans aucun soucis et sans avoir besoin de compétences dans le domaine du collaborateur.

De plus, séparer le traitement de la présentation permet de faciliter la maintenance du code, même si vous faites office à la fois de développeur et de graphiste : le partie traitement ne contient que du PHP et la partie affichage ne contiendra que du HTML/CSS ainsi que des balises propres au système de templates.

I-C - Ce qu'il nous faut modifier

Essayons de faire les choses proprement.

Maintenant que Smarty est installé, et puisque nous avons plein de fichier, il ne coûte rien d'en rajouter un qui contiendra nos styles. Nous l'appellerons "style.css" (original non ?), et nous le placerons dans le répertoire "Include" situé à la racine de notre arborescence.

Voici le contenu de ce fichier :

```
a img {  
  border-color:transparent;  
}  
body {  
  text-align:center;  
}
```

Je me suis contenté de recopier le style que j'avais défini pour enlever les bordures bleues, et de rajouter un style au body afin que l'ensemble de l'affichage soit centré. Libre à vous d'en rajouter en fonction du rendu final que vous souhaitez obtenir.

Nous allons également séparer l'affichage de la galerie de l'affichage des images. Ainsi, "index.php" va devenir "affichage-galerie.php", et nous allons créer un second fichier, "affichage-image.php", qui sera simplement chargé d'afficher l'image choisie.

Il va maintenant falloir 'intégrer' Smarty à ces deux pages.

Il va suffire d'intégrer le code suivant dans nos deux pages, juste après le "error_reporting(E_ALL | E_STRICT);" :

```
require_once('Smarty/libs/Smarty.class.php');
$smarty = new Smarty();

$smarty->template_dir = 'Smarty/templates/';
$smarty->compile_dir = 'Smarty/templates_c/';
$smarty->config_dir = 'Smarty/configs/';
$smarty->cache_dir = 'Smarty/cache/';
```

Chez moi, le répertoire de Smarty est situé dans le même répertoire que ma page PHP, pensez à adapter les chemins en fonction de votre installation.

Profitions de la création de nouveaux fichiers pour créer les deux fichiers template : "affichage-galerie.tpl" et "affichage-image.tpl", à placer dans le répertoire "templates" de Smarty. Commençons au passage à les remplir :

```
{* Smarty *}
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Affichage des miniatures</title>
<link rel="stylesheet" type="text/css" href="Include/style.css">
</head>
<body>

</body>
</html>
```

Le titre sera bien entendu à modifier en fonction de la page.

La première ligne ne sert absolument à rien, si ce n'est à signaler que Smarty a été utilisé dans ce fichier (utile pour reprendre un code longtemps après).

Lors de l'inclusion du CSS, il faut l'inclure relativement au fichier PHP d'origine, et non pas relativement au fichier TPL lié.

En regardant plus attentivement le contenu de notre page "affichage-galerie.php", nous nous apercevons qu'il nous faudra modifier la fonction `ajoute_lien()`, ainsi que la fonction `affichage()` (le but d'un système de templates étant, je le rappelle, de dissocier complètement le code PHP de l'affichage, il suffit de repérer l'ensemble des éléments produisant du code HTML).

Maintenant que l'ensemble des choses à modifier à été défini (et même bien entamé), passons de suite à la modification des fonctions.

II - Transformation du PHP

II-A - affichage-galerie.php

Fonction ajoute_lien()

A l'exception de la création du HTML, il va falloir modifier une seule chose dans cette fonction : il faut définir `$smarty` comme étant une variable globale. Placez donc le code suivant juste sous la définition de la fonction :

```
global $smarty;
```

Attaquons-nous maintenant à la partie de création du HTML. Quelles sont les choses intéressantes à garder, celles dont nous avons besoin ?

Il s'agit des quatre variables suivantes :

- `$chemin_image`
- `$chemin_miniaiture`
- `$taille_html_miniaiture`
- `$file`

Ces quatre variables étaient stockées sous forme de tableau. Aucune raison pour que cela change, nous allons donc utiliser la méthode `append()` de Smarty :

```
$smarty->append('imgs_prop',
    array(
        'CHEMIN_MINIATURE' => $chemin_miniaiture,
        'CHEMIN_GRANDE'   => $chemin_image,
        'NOM_IMAGE'       => $file,
        'TAILLE_HTML_MIN' => $taille_html_miniaiture
    )
);
```

Nous venons simplement de créer un tableau de tableaux associatifs. A chaque itération de la fonction "ajoute_lien()", un tableau associatif contenant les différentes propriétés de l'image en cours sera ajouté au tableau "imgs_prop".

Une fois ces modifications effectuées, le travail sur la fonction "ajoute_lien()" est terminé. Voici le code complet de la 'nouvelle' fonction :

```
// Ajoute un tableau associatif contenant les propriétés de l'image en cours
// au tableau imgs_prop (propre à Smarty)
function ajoute_lien($chemin_image, $chemin_miniaiture, $file) {
    global $smarty;
    // Récupère la taille de la miniature sous forme HTML (width="xxx" height="yyy")
    $taille_html_miniaiture = getimagesize($chemin_miniaiture);
    $taille_html_miniaiture = $taille_html_miniaiture[3];

    $smarty->append('imgs_prop',
        array(
            'CHEMIN_MINIATURE' => $chemin_miniaiture,
            'CHEMIN_GRANDE'   => $chemin_image,
            'NOM_IMAGE'       => $file,
            'TAILLE_HTML_MIN' => $taille_html_miniaiture
        )
    );
}
```

Il ne nous reste plus qu'à nous attaquer à la fonction "affichage()".

Fonction affichage()

Si la section précédente vous a paru facile, celle-ci le sera encore plus : la fonction "affichage()" n'a tout bonnement plus de raison d'exister, puisque tout l'affichage sera géré dans notre fichier "affichage-galerie.tpl".

Il vous suffit donc de la supprimer, et de remplacer son appel par la méthode d'affichage du template de Smarty : `display()` :

```
$smarty->display('affichage-galerie.tpl');
```

Nous nous contentons ici de demander à Smarty d'afficher le template, en se basant sur le fichier "affichage-galerie.tpl". Vous pouvez d'ores et déjà tester votre galerie, mais rien ne s'affichera. Ceci est parfaitement normal, puisque nous n'avons pas géré l'affichage dans "affichage-galerie.tpl".

Nettoyage du code

Maintenant que nous avons effectué ces petites modifications, re-regardons un peu en détail notre code.

Vous ne remarquez rien ? Rien ne vous semble inutile désormais ?

Si vous avez trouvé, c'est bien, vous avez le coup d'oeil. Effectivement, on se rend compte que deux variables ne nous sont plus d'aucune utilité : la constante **NBRE_COLONNES**, qui nous servait à définir le nombre de colonnes pour l'affichage (bah oui, tout se fait dans le template maintenant), et la variable globale **\$tabl_liens**, qui nous servait à stocker les liens vers les différentes images de notre galerie.

Vous pouvez donc supprimer toute référence à ces variables de votre code source, elles ne nous serviront plus jamais.

II-B - affichage-image.php

Encore une fois, rien de très difficile.

J'ai décidé que l'image à afficher sera passée en paramètre à la page 'affichage-image.php' par l'URL.

Nous allons nous contenter de récupérer le chemin de l'image originale, en extraire la taille ainsi que le nom qui nous servira à remplir la balise ALT. Puis nous enverrons ces données à notre template pour affichage :

```
$chemin_image = $_GET['image'];

$taille = getimagesize($chemin_image);

$nom_img = explode('/', $chemin_image);
$nom_img = array_reverse($nom_img);

$smarty->assign('proprietes', array (
    'CHEMIN_IMAGE' => $chemin_image,
    'NOM_IMAGE'    => $nom_img,
    'TAILLE_HTML_IMAGE' => $taille[3],
));

$smarty->display('affichage-image.tpl');
```

La seule chose spécifique à Smarty est l'affectation des différentes valeurs.

Cette fois-ci, nous avons utilisé la méthode "[assign\(\)](#)".

En effet, nous n'affectons à la variable 'proprietes' qu'un seul tableau associatif, contenant trois paires clé/valeur.

Une fois cette affectation effectuée, il ne nous reste plus qu'à appeler notre template.

III - Création des gabarits HTML

III-A - affichage-galerie.tpl

Avant de commencer, je voulais simplement vous redonner le lien vers la [documentation complète de Smarty](#). N'hésitez pas à en user et à en abuser.

Comme nous l'avons vu dans la section précédente, les données à traiter vont se présenter sous la forme d'un tableau de tableaux associatifs. Il va donc nous falloir faire une boucle pour obtenir l'ensemble des valeurs. Pour cela, nous allons utiliser la méthode "`foreach()`" :

```
{foreach name=liste_imgs item=image from=$imgs_prop}
  <a href="affichage-image.php?image={$image.CHEMIN_GRADE}">
    
  </a>
  {if ($smarty.foreach.liste_imgs.iteration % 4) == 0}
    <br>
  {/if}
{/foreach}
```

Je vous l'accorde, pas très sympathique d'aspect. Mais en regardant un petit peu, ce n'est pas si compliqué que ça.

Voici l'équivalent du code ci-dessus en langage naturel :

```
Pour chaque élément du tableau $imgs_prop (élément que nous appelleront image)
  Lien vers la page affichage-image.php en lui passant en paramètre le chemin de la grande
  image
      Affichage de la miniature avec en paramètres :
          - le chemin de la miniature
          - la taille de la miniature au format HTML
          - le nom de l'image pour servir d'attribut à la balise ALT
      Fin affichage
  Fin du lien
  Si le nombre courant d'itération modulo 4 (nombre de colonnes) vaut 0
      Alors on affiche un retour à la ligne
  Fin si
Fin du pour chaque
```

Nettement plus clair une fois traduit, non ?

Vous l'aurez compris, la seule chose à modifier pour changer le nombre de colonnes est le chiffre 4.

Vous pouvez ainsi changer du tout au tout la manière d'afficher votre galerie, sans rien avoir à retoucher dans votre code PHP.

Maintenant, si vous re-testez, la magie opère... Vous pouvez constater que vos miniatures s'affichent bien sur quatre colonnes, et que les liens sont bien présents.

Cependant, rien ne sert de cliquer sur un lien, puisque la page "affichage-image.tpl" n'a pas été faite.

III-B - affichage-image.tpl

Ce fichier de template va être encore plus simple que le précédent, puisque ici, nous avons une seule variable. Pas besoin de boucle donc :

```

<br>
<a href="./affichage-galerie.php">Retour &agrave; la galerie </a>
```

Nous nous contentons simplement d'afficher une image comme nous le ferions en HTML classique et nous lui passons en paramètre les variables transmises par notre code PHP.

Nous en profitons également pour insérer un lien permettant un retour à la galerie.

Comme vous pouvez le constater, la mise en page est ici minimale. Et encore une fois, l'avantage majeur des templates se fait ressentir : vous pouvez entièrement personnaliser cette page sans devoir toucher de près ou de loin à votre code PHP.

IV - Conclusion

Comme vous avez pu le constater dans cet exemple, adapter une petite application à l'utilisation de templates est plutôt aisé. En revanche, dans le cadre d'applications plus grosses, le travail d'adaptation à un système de templates devient très difficile.

Il convient donc de prendre l'habitude d'utiliser les templates dès le départ, afin de s'épargner beaucoup de peine.

Utiliser les templates doit devenir une habitude, et ce afin d'améliorer la lisibilité du code, et de bien séparer les différentes couches de l'application.

Pour plus d'informations sur la manière de fonctionner des templates, je vous invite à vous reporter aux [différents articles](#) sur [developpez.com](#) :

- [PHP et les templates](#)
- [Gestion d'un système de templates \(gabarits\) en PHP 5](#)

Je souhaitais également remercier l'équipe web de [developpez.com](#) qui m'a poussé à rédiger cette suite, et plus particulièrement [Guillaume Rossolini](#).

[Télécharger le code-source](#)