Hyla Tpl

Table des matières

1. Introduction....2
2. Fonctionnalités....3
  2.1. L'Api....3
    2.1.1. Le constructeur....3
    2.1.2. Liste des méthodes....3
  2.2. Les variables....4
    2.2.1. Tableau de variables....4
    2.2.2. Fonctions pour les variables....4
    2.2.3. Valeur par défaut pour les variables....5
    2.2.4. Un mix de tout....5
    2.2.5. Ajouter des fonctions de variables....5
    2.2.6. Modification de variables dans la vue....6
  2.3. Les blocs....6
    2.3.1. Les blocs else....7
  2.4. Les Fonctions....7
    2.4.1. Liste des fonctions prédéfinies....8
    2.4.2. Ajouter des fonctions....8
  2.5. La traduction....9
3. Exemples....9
  3.1. Hello World!....9
  3.2. Hello World! Amélioré....10
  3.3. Variable par défaut et bloc else....10
  3.4. Listage de données....11
  3.5. Un explorateur de fichiers....11

Hyla TPL, un moteur de template pour PHP simple et efficace.

1. Introduction

Hyla Tpl est un moteur de template (ou de gabarit) en PHP avec un syntaxe similaire à celle du moteur de la PhpLib sous license GPL.

Les principales caractéristiques de ce nouveau moteur de gabarit sont les performances et la simplicité d'utilisation.

En voici d'autres :

- Concept de variables par défaut
- Modification de variable depuis la vue
- Appel de fonction pour les variables
- Méthode de traduction intégrée
- Bloc else
- Importation de fichier depuis la vue
- Appel de fonctions
- Déclaration des blocs au préalable inutile

Le moteur de la PhpLib dispose de nombreux avantages, simplicité des modèles : pas de code (foreach, if, etc...) respectant ainsi la logique qui veut que les graphistes ne touchent pas une brindille de code, les blocs sont en fait des commentaires, ils apparaissent donc cachés aux graphistes travaillant sur le modèle.

Mais ce moteur de template souffre de 2 grosses lacunes :

- Sa complexité de mise en oeuvre du côté php, on arrive très vite à du code relativement lourd : il est très verbeux.
- Plus aucun support depuis bien longtemps, il lui manque donc des fonctionnalités qui sont, pourtant, devenues courantes.

Hyla Tpl permetde garder la simplicité de la vue avec quelques nouveautés et améliorer le contrôleur.

La syntaxe est relativement simple, pour les variables, on utilise les accolades {}, à cela, vient s'ajouter 4 opérateurs :

- {$mavariable} : Défini la variable nommée « mavariable », Chapitre sur les variables
- {_myvariable} : Défini une chaine de traduction, Chapitre sur la traduction
- {&mavariable:Cette valeur} : Permet de donner une valeur à une variable, Chapitre sur la modification de variables dans la vue
- {#Mon commentaire} : Défini un commentaire

Télécharger le code source : 

2. Fonctionnalités

2.1. L'Api

2.1.1. Le constructeur

Le constructeur s'utilise très simplement, dans l'exemple suivant, on va instancier Hyla_Tpl avec en paramètre le chemin vers le dossier contenant les fichier gabarit.

Ensuite, sur la ligne suivante, à l'aide de la méthode importFile, on importe le fichier test.tpl qui portera le nom de test.

```
<?php
require 'hyla_tpl.class.php';

$tpl = new Hyla_Tpl('tpl');
$tpl->importFile('test', 'test.tpl');

[...]
```

2.1.2. Liste des méthodes

- importFile(name, file, path)

  Déclare un fichier de gabarit « file » portant le nom « name » dans le dossier « path » si il ne se trouve pas dans le dossier courant.

- setCurrentFile(name)

  Spécifie le fichier de gabarit courant, attention, il s'agit bien du nom du fichier donné en premier argument à la méthode importFile.

- setVar(name, value)

  Donne la valeur « value » à la variable de gabarit nommée « name », voir les variables

- setVars(array)

  Donne un tableau de valeurs de variables, voir les tableaux de variables

- setL10nCallback(callback)

  Déclare la fonction de rappel pour la traduction, voir principes de la traduction.

- get(name)

  Retourne le contenu du bloc nommé « name »

- render(name)

  Effectue un rendu du bloc nommé « name »

- registerVarFunction(name, function_name)

  Enregistre la fonction « function_name » appelable depuis la gabarit par « name », voir ajouter des fonctions de variables

- registerFunction(name, function_name)

  Enregistre la fonction « function_name » appelable depuis la gabarit par « name », voir ajouter des fonctions

- getFunctionList(bool)

  Retourne la liste des fonctions définies sous la forme d'un tableau, si la second argument est à true, les fonctions définies par l'utilisateurs seront aussi retournées.

- removeUnknowVar(bool)

  Permet de spécifier si le moteur de gabarit supprime les variables vides du gabarit ou non.

- displayError(bool)

  Spécifie si les erreurs doivent être affichée ou non

- logError(bool)

  Enregistre les erreurs

- getVersion()

  Retourne le numéro de la version (ex: 0.7.0)

2.2. Les variables

Depuis le contrôleur, il est bien sûr possible de donner des valeurs à des variables dans votre fichier template.

Cela se fait avec la méthode setVar :

```
$tpl->setVar('toto', 'Coucou');
```

Dans le fichier gabarit, on déclare la variable simplement comme ceci :

```
{$toto}
```

2.2.1. Tableau de variables

Il est tout à fait possible de spécifier à une variable un tableau ou un objet.

Dans ce cas, on accède aux éléments du tableau ou de l'objet à l'aide du . dans les 2 cas, on s'abstrait ainsi dans le gabarit du type de données.

```
// En spécifiant un tableau
$user = array('name' => 'Marc', 'age' => 23);
$tpl->setVar('user', $user);

// En spécifiant un objet
$obj = new user();
$obj->name = 'Paul';
$obj->age = 34;
$tpl->setVar('user', $obj);

Dans le fichier gabarit, on fera la même chose dans les 2 cas :

Je suis {$user.name} et mon age est de {$user.age} ans !
```

2.2.2. Fonctions pour les variables

Exemple 1 : Appel de fonction pour une variable

Il est possible d'envoyer le contenu d'une variable à des fonctions avec l'opérateur barre verticale (|) de la manière suivante :

```
<p>{$toto|upper}</p>
```

Si la variable toto vaut « coucou la terre ! », le résultat sera :

> COUCOU LA TERRE !

Exemple 2 : Appel de plusieurs fonctions

Il est possible de chainer les fonctions avec l'opérateur barre verticale | :

```
<p>{$toto|upper|ucfirst}</p>
```

Donnera ainsi :

> Coucou La Terre !

2.2.3. Valeur par défaut pour les variables

Il est possible de définir des valeurs par défaut pour les variables qui ne serait pas remplies depuis le contrôleur.

```
{$variable (Ma valeur par défaut) }
```

Renverra :

> Ma valeur par défaut

Il est également possible de spécifier la valeur par défaut avec une fonction de traduction de cette manière, voir la documentation sur la traduction pour plus d'information :

```
{$variable _(Hello World !) }
```

2.2.4. Un mix de tout

Mixons tout ce que nous venons de voir sur les variables :

```
{$variable (<p>Ma valeur par défaut</p>)|upper|trim|escape}
```

Renverra ceci si la variable $variable est vide :

> <P>MA VALEUR PAR DéFAUT</P>

Renverra ceci si la variable $variable vaut « Valeur de variable » :

> <P>VALEUR DE VARIABLE</P>

2.2.5. Ajouter des fonctions de variables

Hyla Tpl vous permet de définir des fonctions pour les variables directement depuis le contrôleur à l'aide de la méthode registerVarFunction, voici un exemple d'utilisation.

```
[...]

function quote($var)
{
	return "« $var »";
}

$tpl->registerVarFunction('quote', 'quote');

[...]

{$citation|quote}
```

Si la variable citation vaut : Hello world !, le résultat sera la suivant :

« Hello World ! »

2.2.6. Modification de variables dans la vue

Il est possible modifier la contenu d'une variable depuis la vue de deux manières différentes.

Utilisation de la fonction setVar :

```
<p>{$toto}</p>
{!setvar:toto,'Variable toto créée !'}
<p>{$toto}</p>
{!setvar:toto,'Variable toto modifiée !'}
<p>{$toto}</p>
```

Affichera ceci :

Variable toto créée !

Variable toto modifiée !

Explication : La première fois que l'affichage de toto est demandée {$toto}, rien n'est retourné car $toto n'est pas déclaré et n'a pas de valeur par défaut, ensuite, le premier appel à setVar créée la variable $toto, et enfin, le second appel à setVar modifie $toto.

Utilisation de & :

setVar et l'opérateur « & » diffère dans le fait que ce dernier change une fois pour toute la valeur d'une variable, ainsi, c'est le dernier appel qui l'emportera.

```
<p>{$toto}</p>
{&toto,'Variable toto créée !'}
<p>{$toto}</p>
{&toto,'Variable toto modifiée !'}
<p>{$toto}</p>
```

Affichera ceci :

Variable toto modifiée !

Variable toto modifiée !

Variable toto modifiée !

2.3. Les blocs

Un bloc se défini comme ceci :

```
<!-- BEGIN line -->
<p>
	Contenu du bloc
</p>
<!-- END line -->
```

La génération d'un bloc se fait en appelant la méthode render.

```
[...]
$tpl->render('line');
$tpl->render('line');
[...]
```

Les 2 appels à render ci-dessus généreront le code ci-dessous :

Contenu du bloc

Contenu du bloc

Ce sont les blocs du fichier de gabarit courant qui sont rendus, vous pouvez modifier le fichier courant avec la méthode setCurrentFile.

Vous pouvez aussi sélectionner le fichier de la manière suivante :

```
[...]
$tpl->render('menu:line');
[...]
```

Dans cet exemple, c'est le bloc line du fichier nommé menu qui sera rendu.

2.3.1. Les blocs else

Nouveauté très utile pour ceux qui viennent de la PhpLib, les blocs else permettent d'afficher un contenu d'un bloc else uniquement lorsque le bloc principal n'a pas été appelé, simplifiant alors le code php.

Un bloc else se défini comme ceci :

```
<!-- BEGIN line -->
<p>
	Contenu du bloc
</p>
<!-- ELSE line -->
<strong>
	Jamais appelé !
</strong>
<!-- END line -->
```

Si le bloc line n'est jamais appelé, le contenu du bloc else sera affiché :

Jamais appelé !

2.4. Les Fonctions

Il est possible d'appeler des fonctions directement depuis la vue afin de réaliser certaine actions.

Exemple avec la fonction cycle :

```
<table>
	<!-- BEGIN line -->
	<tr>
		<td>{!cycle:'Rouge','Jaune'}</td>
	</tr>
	<!-- END line -->
</table>
```

Si le rendu de line est appelé 4 fois, cela produira le source suivant :

```
<table>
	<tr>
		<td>Rouge</td>
	</tr>
	<tr>
```

```
            <td>Jaune</td>
        </tr>
        <tr>
            <td>Rouge</td>
        </tr>
        <tr>
            <td>Jaune</td>
        </tr>
</table>
```

2.4.1. Liste des fonctions prédéfinies

Voici la liste des fonctions prédéfinies, en bleu, le nom de la fonction, en vert, le ou les arguments :

- cycle:odd,even,cycle

  Permet d'afficher le contenu des paramètres odd et even de manière cyclique avec la fréquence cycle

- include:file

  Inclu un fichier tpl depuis le dossier défini de tpl défini dans le constructeur (footer.tpl)

- import:file

  Importe un fichier tpl depuis n'importe quel location (/home/user/tpl/info.tpl)

- errors:html

  Renvoie la liste des erreurs, si le premier paramètre est à true (par défaut), l'affichage se fera sous forme de html.

- setvar:name,value

  Modifie une variable

- l10n:var

  Fonction de traduction

2.4.2. Ajouter des fonctions

Hyla Tpl vous permet de définir des fonctions directement depuis le contrôleur à l'aide de la méthode registerFunction, voici un exemple d'utilisation.

```
[...]

function hello($name = null)
{
        return "Hello $name!";
}

$tpl->registerFunction('hello', 'hello');

[...]
```

Un appel depuis le gabarit à la fonction hello :

```
{!Hello}
```

Provoquera alors l'affichage suivant :

Hello!

Un appel depuis le gabarit à la fonction hello avec un paramètre :

```
{!Hello:'world'}
```

Provoquera alors l'affichage suivant :

Hello world!

2.5. La traduction

Il est possible, simplement, de rendre votre gabarit prêt pour une diffusion en plusieurs langues.

Voici un exemple :

```
[...]

function l10n($var)
{
	global $lang;

	if ($lang == 'en')
		return $var;

	$l10n = array(
		'Hello'			=>	'Bonjour',
		'Hello world!'		=>	'Bonjour monde !',
		);

	return (array_key_exists($var, $l10n) ? $l10n[$var] : $var);
}


$tpl->setL10nCallback('l10n');

[...]
```

Dans le fichier gabarit :

```
<p>{_Hello}</p>
<p>{_Hello world!}</p>
```

Le comportement est très simple, lorsque Hyla Tpl rencontre une balise {_XXxxxXXX}, elle appelle la fonction définie par setL10nCallback et affiche le résultat renvoyé par cette dernière.

Par défaut, c'est la méthode _l10n de la classe hyla_tpl qui est appelé, celle-ci retourne tout simplement ce qu'on lui passe dans le premier paramètre.

3. Exemples

3.1. Hello World!

Fichier PHP

```
<?php

require 'hyla_tpl.class.php';

// Créé l'objet Hyla_Tpl
```

Fichier tpl

```
<h1>{$var}</h1>
```

```php
$tpl = new Hyla_Tpl('.');

// Import le fichier hello.tpl
$tpl->importFile('hello.tpl');

// La variable de template var vaut Hello World !
$tpl->setVar('var', 'Hello World !');

// Affiche le résultat
echo $tpl->render();

?>
```

3.2. Hello World! Amélioré

Le fonctionnement est simple, si la variable name n'est pas déclarée, c'est la valeur par défaut qui est chargée, dans l'exemple, « world ».

Fichier PHP

```php
<?php

require 'hyla_tpl.class.php';

// Créé l'objet Hyla_Tpl
$tpl = new Hyla_Tpl('.');

// Import le fichier hello.tpl
$tpl->importFile('hello2.tpl');

// Change la valeur du cookie une fois sur 2
setcookie('test', !@$_COOKIE['test'], time() + 3600);

// Assigne you à la variable name une fois sur 2
if (@$_COOKIE['test']) {
    $tpl->setVar('name', 'you');
}

// Affiche le résultat
echo $tpl->render();

?>
```

Fichier tpl

```
<h1>Hello {$name (world)} !</h1>
```

3.3. Variable par défaut et bloc else

Voici un exemple mettant en application les variables par défaut et les blocs else.

Le bloc user_status n'est jamais appelé et la variable $user ne vaut rien, le contenu du bloc ELSE est alors affiché et la valeur par défaut de la variable $user est alors affichée.

Fichier PHP

```php
<?php

header('Content-type: text/html; charset=utf-8');

require 'hyla_tpl.class.php';

// Créé l'objet Hyla_Tpl
$tpl = new Hyla_Tpl('.');
$tpl->importFile('default', 'default.tpl');

class User {
    var $first_name;
```

Fichier tpl

```
<p>
    <!-- BEGIN user_status -->
    <strong>
        Vous êtes loggué en tant que
        {$user.first_name}
        {$user.last_name} !
    </strong>
    <!-- ELSE user_status -->
    Veuillez vous connecter !
    <!-- END user_status -->
</p>
```

```
    var $last_name;
}

$user = new User;
$user->first_name = 'George';
$user->last_name = 'Orwell';

// Change la valeur du cookie une fois sur 2
setcookie('test', !@$_COOKIE['test'], time() + 3600);

// Assigne you à la variable name une fois sur 2
if (@$_COOKIE['test']) {
    $tpl->setVar('user', $user);
    $tpl->render('user_status');
}

// Affiche le résultat
echo $tpl->render();

?>
```

3.4. Listage de données

Voici un exemple de listage de données sous forme tabulaire.

Fichier PHP

```
<?php

require 'hyla_tpl.class.php';

$tpl = new Hyla_Tpl('.');
$tpl->importFile('list', 'list.tpl');

// Les données
$data = array(
    array( 'name' => 'Marc',    'age' => 26),
    array( 'name' => 'Paul',    'age' => 18),
    array( 'name' => 'Thomas',  'age' => 54),
    array( 'name' => 'Edgard',  'age' => 31),
);

$tpl->setVar('user_count', count($data};

// Parcours les données
foreach ($data as $user) {
    $tpl->setVar('user', $user);
    $tpl->render('line');
}

// Affiche le résultat
echo $tpl->render();

?>
```

Fichier tpl

```
<p>
    Il y a {$user_count} utilisateur(s) !
</p>

<table>
    <tr>
        <th>Nom</th>
        <th>Age</th>
    </tr>
    <!-- BEGIN line -->
    <tr style="background-color: red">
        <td>{$user.name}</td>
        <td>{$user.age}</td>
    </tr>
    <!-- END line -->
</table>
```

3.5. Un explorateur de fichiers

Voici un exemple de programme réalisé avec Hyla_Tpl, il liste les éléments contenus dans un dossier et permet de naviguer dans l'arborescence, avec des textes en anglais et français afin de démontrer la facilité d'incorporer des éléments à traduire dans les gabarits.

Le code démontre assez bien la facilité d'utilisation de ce moteur de gabarit léger et rapide.

Fichier PHP

```php
<?php

// Inclu les sources de la librairies
require 'hyla_tpl.class.php';

// Créé l'objet Hyla_Tpl
$t = new Hyla_Tpl('.');

// Import du gabarits
$t->importFile('explorer.tpl');

// Fonction de traduction
function traduction($var) {
    global $lang;
    if ($lang == 'en') {
        return $var;
    }
    $l10n = array(
        'Current path'  =>  'Chemin
courant',
        'Size'          =>  'Taille',
        'Name'          =>  'Nom',
        'Switch lang'   =>  'Changer de
langue',
    );
    return (array_key_exists($var, $l10n) ?
$l10n[$var] : $var);
}

// Déclare la fonction de traduction
$t->setL10nCallback('traduction');

// Cette fonction renvoie une taille
facilement lisible (ex: 1024o renverra 1ko)
function get_human_readable_size($bytes) {
    global $lang;
    $types = array(null, 'k', 'm', 'g',
't');
    for ($i = 0; $bytes >= 1024 && $i <
(count($types) -1); $bytes /= 1024, $i++);
    return round($bytes, 2) . $types[$i] .
($lang == 'fr' ? 'o' : 'b');
}

// Enregistre la fonction
get_human_readable_size en tant que
humansize dans le template
$t->registerVarFunction('humansize',
'get_human_readable_size');

// Récupère les variables get
$dir = isset($_GET['dir']) ? $_GET['dir'] :
dirname(__FILE__);
$dir = realpath($dir);

$lang = isset($_GET['lang']) ? $_GET['lang']
: 'fr';

// Assigne quelques variables
$t->setVars(array(
    'dir'           =>  $dir,
    'lang'          =>  $lang,
    'lang_switch'   =>  ($lang == 'en' ?
```

Fichier tpl

```
<html>

<head>
    <title>{$title}</title>
</head>

<body>

<h1>{_Current path} : {$dir}</h1>

<a href="?dir=&lang=">{_Switch lang} :
{$lang_switch}</a>

<table width="50%">
    <tr>
        <th width="90%">{_Name}</th>
        <th width="10%">{_Size}</th>
    </tr>

    <!-- BEGIN table.line -->
    <tr style="background-color: #DDD;">
        <td align="left">
            <!-- BEGIN table.line.dir -->
            <a href="?
dir={$file.path}&lang="><strong>{$file.name}
</strong></a>
            <!-- ELSE table.line.dir -->
            <strong>{$file.name}</strong>
            <!-- END table.line.dir -->
        </td>
        <td align="right">
            {$file.size|humansize}
        </td>
    </tr>
    <!-- END table.line -->

</table>

</body>

</html>
```

```
'fr' : 'en'),
};

// Ouvre le dossier
if (!$files = @scandir($dir} {
    exit("Unable to open « $dir »");
}

// Parcours des dossiers / fichiers
foreach ($files as $file) {

    $path = realpath("$dir/$file");

    $file = array(
        'path'  =>  $path,
        'name'  =>  $file,
        'size'  =>  filesize($path),
    );
    $t->setVar('file', $file);

    // L'élément courant est un dossier ?
    if (is_dir($path} {
        $t->render('table.line.dir');
    }

    // Affiche la ligne
    $t->render('table.line');
}

// Affiche le résultat
echo $t->render();

?>
```