Tutoriel AJAX Chat Partie 1: Introduction, le Zend Framework

Introduction

Développer une application de type "chat" n'est pas une tâche difficile, honnêtement !

Ce tutoriel est une introduction étape par étape pour créer un chat léger, utilisant XML comme support de stockage. Ma motivation personnelle m'a amené à développer des jeux PHP comme un hobby. En poursuivant ce hobby, j'ai découvert qu'en offrant un forum plat, même sans contenu, dans ces jeux, peut avoir comme conséquence une utilisation lourde des joueurs qui s'en servent comme salon de chat. La raison de ce problème est l'utilisation intensive des nouveaux supports de communication comme IRC et la messagerie instantanée. Les utilisateurs continuent de voir ces solutions comme attractive pour plusieurs raisons s'ils ont un accès limité à leur messagerie instantanée à travers un proxy, ou simply for convenience sake.

Vous pouvez télécharger la version actuelle du code source pour votre application depuis : http://game.patternsforphp.com/chat.tar.gz ou http://game.patternsforphp.com/chat.zip. Ce sera la version finale du code après avoir terminé le tutoriel.

Pour créer cette application, je vais utiliser une librairie tierce standard. Je n'ai pas l'intention de créer de nouvelles librairies puisque celles du Zend Framework sont déjà très bien et couvrent amplement mes besoins. Beaucoup de travaux inutiles peuvent être associés à la mentalité "Non Fait Ici", alors laissons cette mentalité derrière nous.

Côté serveur, je vais utiliser le Zend Framework 0.20 (sorti le 31 octobre 2006). Côté client, Ajax et Javascript vont être simplifiés en utilisant la librairie Prototype. Aucun effet n'est requis, j'utiliserai Scriptaculous. Les partisans d'une solution alternative comme jQuery, Dojo ou de n'importe quelle autre solution peuvent appliquer les mêmes principes en utilisant leur librairie préférée.

Ce tutoriel se focalisera en mélangeant Zend Framework, Javascript et la librairie Protoype dans le but de créer une solution simple et élégante. Ce code est sous licence "New BSD License", les lecteurs sont libres de modifier (et alléger) le code sous cette licence comme bon leur semble. Les formalités faites, entrons dans le vif du sujet !

Le Zend Framework

Le Zend Framework a été rapidement adopté comme l'une de mes librairies favorites en PHP. Personnellement, je n'aime pas les vastes frameworks qui établissent une structure artificielle où on est forcé d'adhérer. Le Zend Framework est cependant empaqueté comme une librairie de classes indépendantes la plupart du temps, qui restent simple d'adapter et d'associer à sa propre librairie personnelle.

Vous pouvez télécharger la librairie depuis le site officiel http://framework.zend.com/. La documentation de l'actuelle version est suffisante, et je sais qu'en suivant la "mailing list", elle grandira sans cesse (il y a une version wiki à l'étude). Hormis la documentation officielle, je vous recommande vivement ce tutoriel http://g-rossolini.developpez.com/tutoriels/php/zend-framework/debuter/ qui est une excellente introduction au Zend Framework et de très bonne lecture. Kudos to Rob Allen for putting it together.

Structure des répertoires

Pour commencer, télécharger la version 0.20 du Zend Framework depuis http://framework.zend.com/download. Créer un nouveau répertoire à la racine de votre serveur et nommez le "chat-tutorial". A l'intérieur de ce répertoire, nous devrons installer la structure de base requise pour organiser les fichiers de l'application "chat". La structure des répertoires recommandées (vous êtes libre de changement) est la suivante :

chat-tutorial/
/application
/controllers
/views
/library
/incubator
/public
/javascript
/styles
/javascript
/data

Veuillez copier le répertoire "/library/Zend", que vous trouverez dans l'archive que vous venez de télécharger sur le site du Zend Framework, dans le dossier "chat-tutorial". Le dossier "chat-tutorial/library" doit maintenant contenir un nouveau répertoire nommé Zend, à l'intérieur duquel il y a le coeur de la librairie et le fichier Zend.php. Il y a aussi un dossier séparé "incubator" dans l'archive téléchargée du Zend Framework. Ce dossier sert à placer de nouveaux composants jusqu'à ce que la version publique du Zend Framework sorte. Dans la version 0.20, l'incubateur contient les nouveaux composants MVC que nous utiliserons. Copiez le dossier "/incubator/library" de l'archive téléchargée vers le répertoire "/incubator/library" dans le dossier "chat-tutorial".

Le répertoire "/application" contient la majorité de notre application telle que les contrôleurs et les modèles. Un contrôleur est une classe dans laquelle se trouve le coeur logique de notre application. Cela inclut le code de premier niveau qui utilise les librairies, manipule les modèles, créé les vues, et traite les données utilisateur telles que la vérification et la validation des données. Cette logique est séparée à travers les méthodes Action qui peuvent être accessibles depuis l'url en précisant le nom du contrôleur et le nom de la méthode Action.

Si la terminologie vous trouble, ne vous inquiétez pas. Nous verrons cela dans peu de temps à travers des exemples.

Le répertoire "library" contiendra toutes les librairies dont nous avons besoin telles que le Zend Framework (coeur et incubateur). L'utilisation d'autres librairies PHP n'est pas requise pour ce tutoriel mais gardez ce dossier en mémoire pour d'autres projets que vous pourrez faire avec le Zend Framework, qui nécessitera des librairies PHP additionnelles.

Le répertoire "public" contiendra tous les fichiers qui doivent être accessibles depuis le web. Cela inclut les images, les feuilles de style css, et javascript. En outre, dans le but d'avoir un dossier séparé, j'ai ajouté à la racine un répertoire "javascript" que nous créons nous-mêmes. Il est essentiel de remarquer que les dossiers "public" et "javascript" seront les seuls sous-répertoires dans notre application accessibles depuis le web. Tous les autres sous-répertoires contiendront des fichiers qu'un utilisateur n'a pas nécessairement besoin d'avoir accès (cad nous ne le voulons pas!). Donc ne leur donnez pas accès - c'est une faible pratique de sécurité d'agir ainsi. Nous pouvons gérer ces accès depuis les fichiers .htaccess sous Apache.

Pour finir, nous utiliserons XML comme support de stockage pour garder en mémoire les messages du "chat". Le fichier XML sera placé dans le répertoire "data". Pour s'assurer qu'Apache (et par extension le processus PHP) puisse écrire dans ce répertoire, nous devrons nous assurer que les permissions d'écriture soient actives. La méthode la plus restrictive recommandée est d'autoriser PHP à écrire dans le fichier XML. Cependant, pour plus de simplicité, vous pouvez modifier les accès du répertoire en chmod 777 pour votre plateforme de développement local. Les utilisateurs de Windows peuvent ignorer le sujet des permissions ;)

Débutez avec le Zend Framework

Le Zend Framework fonctionne sur un principe qui structure les urls où une classe s'appelle contrôleur. Comme vous pouvez le deviner, les classes contrôleur sont stockés dans le répertoire "/application/controllers/. Une classe contrôleur contient les méthodes qui exécute des actions, et sont pour cette raison précisément appelées Actions. Un exemple d'url tel que http://www.example.com/chat/refresh se réfèrera à la méthode RefreshAction() de la classe

ChatController, situé dans le dossier "/application/controllers/" dans le fichier ChatController.php. Le point important à remarquer est comment le format de l'url renseigne le Zend Framework sur quel contrôleur et quelle méthode du contrôleur à appeler. Les détails techniques de la structure de l'application ne sont pas requis à ce niveau (ils seront abordés plus en détail dans la documentation officielle et pointeront probablement sur un tutoriel).

La classe Zend_Controller qui gère le processus (le "RewriteRouter") permet d'utiliser des urls propres. Pour faire ceci, toutes les requêtes passent par un fichier central "index.php" (le "Bootstrap"). Le fichier Bootstrap sert à configurer le Framework et contient d'autres codes d'initialisation dont nous aurons besoin.

Il a besoin aussi d'un fichier .htaccess qui s'assure que toutes les requêtes passent par le fichier "index.php". Cette idée d'avoir toutes les requêtes à travers un seul point d'entrée dans une application se réfère au Design Pattern Front Controller. Une recherche google vous offrira toutes les informations nécessaires sur ce Pattern, vous serez sûrement intéressés si vous êtes curieux.

Premièrement, veuillez créer le fichier .htaccess à la racine dans le dossier "chat-tutorial".

```
RewriteEngine on
RewriteCond %{REQUEST_URI} !/public.*
RewriteCond %{REQUEST_URI} !/javascript.*
RewriteRule .* index.php

php_flag magic_quotes_gpc off
php_flag register_globals off
```

Ce fichier renseigne Apache pour utiliser le module mod_rewrite quand une requête vise le dossier de n'importe quel sous-répertoire. La règle de réécriture ("RewriteRule") indique au module que toutes les requêtes doivent passer dans le fichier "index.php" (cad on fait de ce fichier l'unique point d'entrée de notre application). Les deux lignes php_flag s'assurent simplement que magic_quotes et register_globals soient désactivés pour PHP. Cela réduit potentiellement les risques de sécurité, mais principalement cela assure de n'être pas dépendant de ces paramètres que ce soit par accident ou intentionnellement. Ces deux paramètres controversés finiront par être éradiquer une fois PHP 6 sortie. Les professionnels de la sécurité web fêteront ça toute la nuit le jour où ça arrivera...

En outre, nous avons ajouté deux règles conditionnelles. Tous deux informent le module mod_rewrite que les requêtes des répertoires "/public" ou "/javascript" ne seront pas réécrites. Les fichiers de ces deux répertoires doivent être accessibles au public et ne sont pas traités comme des requêtes par l'application elle-même.

Notre Boostrap "index.php" définit de façon basique le niveau de notre application. C'est son travail de manipuler la phase de démarrage de notre application "chat" telle qu'ajouter les paramètres additionnelles, charger les classes, et initialiser les classes du Zend Framework que nous avons l'intention d'utiliser.

```
<?php
/**
 * Bootstrap file for Chat Tutorial
 */

/*
 * The basics...
 */
error_reporting(E_ALL|E_STRICT);
ini_set('display_errors', 1); //disable on production servers!
date_default_timezone_set('Europe/London');

/*
 * Start our session
 */
session_start();

/*
 * Setup the include_path to the ZF library.
 * We set the incubator first so the
 * incubator classes are loaded in preference
 * to core ZF classes where two versions exist.
 *
 * When 0.21 is released, the MVC classes in
 * Incubator will move to the core library.
 */
set_include_path(
   './library/incubator/library'
            . PATH_SEPARATOR . './library'
);
include 'Zend.php';

/*
 * Use Zend::loadClass() to load essentials
 * Probably a good idea to use require_once()
 * elsewhere to avoid unnecessary coupling.
 */
Zend::loadClass('Zend_Registry');
Zend::loadClass('Zend_Controller_Front');
Zend::loadClass('Zend_Controller_RewriteRouter');
Zend::loadClass('Zend_View');

/*
 * Load and register our View for later use
 */
$view = new Zend_View();
```

```
$view->setScriptPath('./application/views');
Zend_Registry::getInstance()->set('view', $view);

/*
 * Instantiate a Request to set BaseURL
 * See later...
 */
$request = new Zend_Controller_Request_Http();

/*
 * Instantiate a RewriteRouter
 */
$router = new Zend_Controller_RewriteRouter();

/*
 * On my platform, I need to set the BaseURL for ZF 0.20
 * RewriteBase is assumed to be $_SERVER['PHP_SELF'] after
 * removing the trailing "index.php" string.
 *
 * PHP_SELF can be user manipulated. Avoided using SCRIPT_NAME
 * or SCRIPT_FILENAME because they may differ depending on SAPI
 * being used.
 */
$base_url = substr($_SERVER['PHP_SELF'], 0, -9);
$request->setBaseUrl($base_url);

/*
 * Setup and run the Front Controller
 *
 * Set Controller Dir, add the RewriteRouter, dispatch the
 * modified Request (with updated BaseURL) and finally
 * get the resulting Response object.
 */
$controller = new Zend_Controller_Front;
$controller->setControllerDirectory('./application/controllers');
$controller->setRouter($router);
$response = $controller->dispatch($request);

/*
 * By default Exceptions are not displayed
 * That won't do during development.
 * Remove this in a live environment though!
 *
 * $response->renderExceptions(true) will not
 * work, it's broken and was fixed in SVN for
 * next release. Until then...
 */
if($response->isException())
{
   echo $response->getException();
   exit; // Stop here - ;)
}

/*
 * Echo the response (with headers) to client
 * Zend_Controller_Response_Http implements
 * __toString().
```

/
echo $response;

La plupart du fichier Boostrap est simple à comprendre.

Nous avons activé "display_errors", qui nous permettra de voir n'importe quelle erreur ou exception non traitée. Le report d'erreurs a été mis à "E_ALL|E_STRICT" puisque nous ne sommes pas intéressés de tomber sur une erreur de fonctionnalités déconseillées ou de pratiques peu sûres (comme les variables non initialisées). Par défaut, un fuseau horaire a été intégré depuis que PHP 5.1+ ne se fie plus au serveur, il s'exécute dessus pour fournir un horaire correctement (il n'y a pas plus aucune confiance...soupir). Nous avons lancé une session PHP pour garder en mémoire des données spécifiques de l'utilisateur entre plusieurs requêtes. Nous avons aussi réglé notre "include_path" pour inclure le dossier "/library". Puisque nous n'utilisons pas de bases de données, j'ai omis d'inclure le dossier "/applications/models" qui est normalement ajouté au "include_path" aussi.

La prochaine section est où le Zend Framework lui-même est "bootstrapped". Nous incluons la classe de base Zend et l'utilisons pour charger plusieurs classes importantes. Zend::loadClass() est une méthode statique qui remplace le "underscore" en un nom de classe avec des slash pour trouver le chemin du fichier à inclure (cela est généralement connue comme la convention PEAR). Il inclut ensuite ces fichiers. C'est une méthode très maniable à utiliser.

Nous avons aussi chargé la classe Zend_View et avons indiqué le chemin où se trouvent les gabarits (ou vues), cad "/application/views". Une vue peut sonner exotique mais c'est juste un simple gabarit HTML qui peut contenir du code PHP pour contrôler l'insertion de variables de données. Si vous avez déjà utilisé Smarty, Template-Lite, Savant ou quelque chose de similaire, ca reste dans la même optique.

La dernière section charge le contrôleur, ajoute une classe "RewriteRouter", et met en place une url basée sur une nouvelle classe "Request". Le contrôleur final appelé fera correspondre des requêtes d'urls non autorisées à la classe "Controller" appropriée et à la méthode Action (utilisant la règle par défaut du Zend_Controller_RewriteRouter ":controller/:action/*") devant être appelée. Il est possible de définir un schéma de "Rooter" alternatif qui ajoute des paramètres, leurs valeurs par défauts, et n'importe quelle autre condition. Cependant, nous n'avons pas besoin d'un "RewriteRooter" spécifique ici, mais n'hésitez pas à vous intéresser aux "Web Services" et exemples associés au Zend Framework.

Avec cette version 0.20, j'ai trouvé souvent nécessaire d'indiquer au Framework l'endroit situé au-dessus de la racine. La méthode SetBaseUrl() peut être extrêmement portable et se trouve simplement en dehors de l'application pour s'assurer que le Framework analyse n'importe quelle requête correctement.

Le reste étant de manipuler la réponse, qui revient bien entendu à faire des "echo".

Le contrôleur Index

Puisque l'application "chat" est simple, nous aurons juste à inclure un seul contrôleur. Pour valider notre installation du Zend Framework, nous ajouterons une méthode très connue qui est tout simplement echo "Hello World!". La classe ControllerIndex peut être sauvegardée dans le fichier IndexController.php avec notre application dans le répertoire "application/controllers".

<?php

```
class IndexController extends Zend_Controller_Action
{

    public function IndexAction()
    {
        /*
         * echo() would work also, but let's get used
         * to the idea of a Response object.
         * This is plain text, so we'll set the content type
         * also.
         */
        $this->getResponse()->setHeader('Content-Type', 'text/plain');
        $this->getResponse()->setBody('Hello World');
    }

}
```

En naviguant sur l'url http://www.example.com/chat-tutorial/ vous verrez afficher "Hello World!", étant echoed sur notre navigateur. Si vous pouvez voir les en-têtes de réponse (jetez un coup d'oeil sur l'extension Web Developer pour Firefox), vous remarquerez que la valeur "Content-Type" a été mise à "text/plain". Ceci étant chemin compliqué, un simple echo vous indiquera que ça fonctionne - mais il est bon d'avoir comme habitude d'utiliser un objet "Response" comme standard.

Enfin terminé

Ceci finit notre section sur l'installation du Zend Framework ! Nous nous intéresserons à d'autres classes du Framework plus tard dans ce tutoriel, mais puisque ZF n'est pas le seul point important dans ce tutoriel, je vous réfèrerai (une fois de plus encore :)) au tutoriel de Rob Allen traduit par Guillaume Rossolini mentionné plus tôt pour une introduction plus détaillée.

Par la suite, nous examinerons comment installer les librairies Javascript que nous utiliserons en créant notre application "chat".

Le Zend Framework est une communauté basée sur un projet. Si vous êtes intéressé d'en apprendre davantage ou de contribuer à son développement, visitez cette page http://framework.zend.com/wiki/display/ZFDEV/Project+Teams.

Traduction réalisé par R-Benyacoub.