



Développement

MVC en PHP Objet

EEMI, Ecole Européenne des Métiers de l'Internet

Par Philippe Giraud

philippe.giraud@eemi.com



Ambitions du cours

- Implémenter le MVC sous une forme Objet
 - Construire une nouvelle architecture
- Préparer l'utilisation des Frameworks
 - Comprendre leurs fonctionnements internes
- Remarque :
 - L'architecture proposée dans ce support est simplifiée
 - Elle sera à améliorer par la suite afin de coller aux frameworks du marché

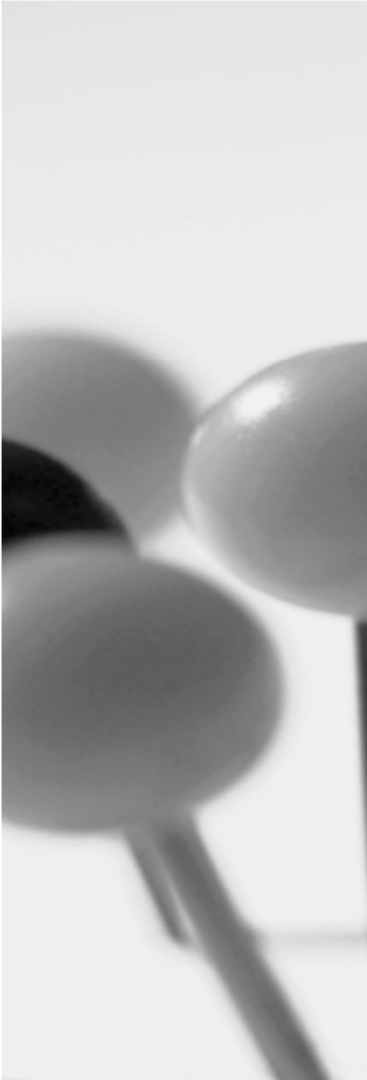


MVC_2 : exemple objet

Développement –MVC en PHP Objet – EEMI par Philippe Giraud

Structure des dossiers

- 4 dossiers principaux :
 - App : les ressources du projet
 - Config : paramétrage de l'application
 - Controller/model/view
 - Core : les ressources de l'architecture
 - Lib : les ressources externes
 - Webroot : les ressources statiques



```
▼ cours_MVC_2
  ▼ app
    ► config
    ► controller
    ► model
    ► view
    app.php
  ► core
  ► lib
  ► webroot
  index.php
```



Controller main

- Fichier : index.php
 - Son rôle est de « lancer la machine »
 - Quelque soit l'application
 - On trouve donc dedans :
 - Du code complètement standard
 - Des initialisations
 - Paramétrages des erreurs par exemple

Controller main

```
<?php
// Paramétrage des erreurs
ini_set('display_errors', 1);
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// Chargement des paramètres
include_once('app/config/config.inc.php');

// Lancement de l'application
include_once('app/app.php');
```



A

pplication

- Fichier app.php
 - Chargement des éventuelles librairies spécifiques
 - Chargement des classes
 - Classe utilitaire : load.php
 - Classe controller : en fonction du paramètre de l'url
 - Instanciation de l'objet controller

Application

```
<?php
// Chargement du moteur d'affichage
include_once('core/load.php');

// Récupération du paramètre de l'url
if (!isset($_GET['module'])) {
    $module = DEFAULT_MODULE;
} else {
    $module = $_GET['module'];
}

// Appel du controller du module demandé
include_once('app/controller/' . $module . '.php');

// Instanciation du controller
new Controller();
```




Controller

- Fichier posts.php
 - Chargement du model nécessaire
 - Instanciations nécessaires
 - Moteur d'affichage
 - Model
 - Dans le constructeur :
 - Aiguillage vers l'action en fonction des paramètres de l'url
 - Les différentes actions

Controller

```
<?php
// Chargement du model
include_once('app/model/Post.php');

class Controller
{
    public $load;
    public $model;

    function __construct()
    {
        $this->load = new Load();
        $this->model = new Model();

        // Récupération des paramètre de L'url
        if (isset($_GET['page'])) {
            if ($_GET['page']=='article') {
                if (isset($_GET['id'])) {
                    // Action view
                    $this->view($_GET['id']);
                } else {
                    $this->page404();
                }
            } else {
                $this->page404();
            }
        } else {
            // Action index
            $this->index(0, 5);
        }
    }
}
```

```
function index($limite, $offset)
{
    $data = $this->model->postList($limite, $offset);
    define("PAGE_TITLE" , "Liste articles");
    $this->load->view('posts', 'index.php', $data);
}

function view($id)
{
    $data = $this->model->postRead($id);
    define("PAGE_TITLE" , "Détail article");
    $this->load->view('posts', 'view.php', $data);
}

function page404()
{
    $this->load->view('vue_404.php');
}
```



Model

- Fichier : post.php
 - Contient les différentes requêtes relatives au module
 - Une méthode par requête
- Fichier : pdo.inc.php
 - Etablissement de la connexion PDO

Model

```
public function postList($limit, $offset)
{
    try {
        // On constitue la requête
        $query = $this->pdo->prepare('SELECT * FROM blog_posts
                                     ORDER BY post_date ASC LIMIT :limit, :offset');
        // On initialise les paramètres
        $query->bindParam(':offset', $offset, PDO::PARAM_INT);
        $query->bindParam(':limit', $limit, PDO::PARAM_INT);
        // On exécute la requête
        $query->execute();
        // On récupère tous les résultats
        $posts = $query->fetchAll();
        // Supprime le curseur
        $query->closeCursor();
        // On retourne tous les articles sélectionnés
        return $posts;
    } catch (Exception $e) {
        return false;
    }
}
```

Model

```
public function postRead($id)
{
    try {
        // On constitue la requête
        $query = $this->pdo->prepare('SELECT * FROM blog_posts
                                     WHERE post_ID = :id');
        // On initialise les paramètres
        $query->bindParam(':id', $id, PDO::PARAM_INT);
        // On exécute la requête
        $query->execute();
        // On récupère tous les résultats
        $posts = $query->fetchAll();
        // Supprime le curseur
        $query->closeCursor();
        // On retourne tous les articles sélectionnés
        return $posts;
    } catch (Exception $e) {
        return false;
    }
}
```

Model

```
function __construct()
{
    try {
        $dns = 'mysql:host=' . DB_HOST . ';dbname=' . DB_NAME;
        // Options de connexion
        $options = array(
            PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
        $this->pdo = new PDO( $dns, DB_USER, DB_PASSWORD, $options );
    } catch (Exception $e) {
        die("Database error : " . $e->getMessage());
    }
}
```



View

- Principe de fonctionnement :
 - La classe load contient une méthode d’affichage view
 - Elle est instanciée et rattachée au controller
 - Cette méthode possède 3 paramètres :
 - Module : le nom du sous-dossier dans le dossier view
 - View : le nom du fichier de la view
 - Data : le tableau des données à intégrer dans la view
 - Ce tableau est facultatif, certaines views n’affichent pas de données

View : Load

```
<?php
class Load
{
    function view( $module, $view, $data = null )
    {
        include 'app/view/' . $module . '/' . $view;
    }
}
```

- On pourrait ajouter un test pour vérifier l'existence de la view demandée

View : Load

- Exemple index

```
<?php include("app/view/layouts/header.inc.php"); ?>

<h1>Les derniers articles</h1>
<div>
    <?php foreach($data as $article) { ?>
        <div>
            Date = <?php echo $article['post_date']; ?>
            <br/>
            <a href="?action=view&id=<?= $article['post_ID']; ?>">Title = <?= $article['post_title']; ?></a>
        </div>
    <?php } ?>
</div>

<?php include("app/view/layouts/footer.inc.php"); ?>
```

View : Load

- Exemple view

```
<?php include("app/view/layouts/header.inc.php"); ?>

<h1>Détail d'un article</h1>
<?php if (!empty($data)) {
    // Pour chaque article du tableau des articles
    foreach($data as $article) { ?>
        <div>
            Date = <?= $article['post_date']; ?><br/>
            Title = <?= $article['post_title']; ?><br/>
            Content = <?= $article['post_content']; ?>
        </div>
    <?php }
} else { ?>
    <p>Article introuvable !</p>
<?php } ?>

<?php include("app/view/layouts/footer.inc.php"); ?>
```

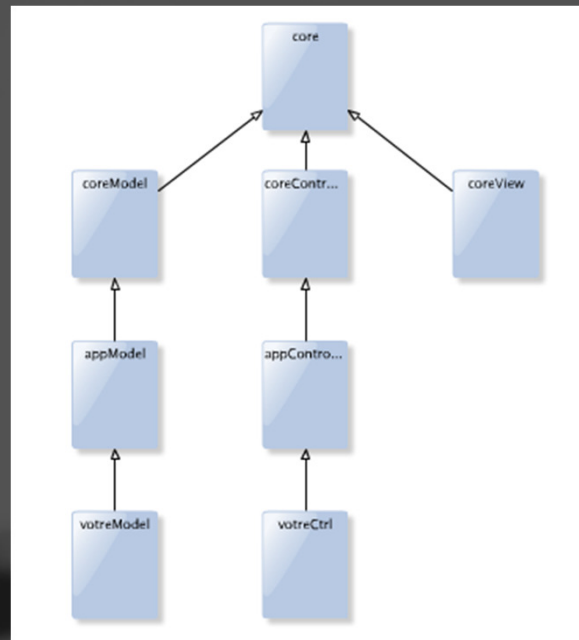


Core : Mise en place

- Les possibilités de l'objet permettent facilement de centraliser du code réutilisable
- Grâce à l'héritage, il est possible de prévoir du code « générique » qui sera donc présent dans chaque classe
- Classes concernées :
 - Controller
 - Model
 - View

Core : Mise en place

- Schéma d'héritage proposé :



Core : classes du core

```
<?php
class core
{
}
```

```
<?php
class coreController extends core
{
}
```

```
<?php
class coreModel extends core
{
}
```

```
<?php
class coreView
{
    function view( $module, $view, $data = null )
    {
        include 'app/view/' . $module . '/' . $view;
    }
}
```

Core : classes de l'app

```
<?php
class appController extends coreController
{
}
```

```
<?php
class appModel extends coreModel
{
}
```

```
<?php
// Récupération du paramètre de l'url
if (!isset($_GET['module'])) {
    $module = DEFAULT_MODULE;
} else {
    $module = $_GET['module'];
}

// Appel du controller du module demandé
$controller = 'app/controller/' . $module . '.php';
if (file_exists($controller)) {
    include_once('app/controller/' . $module . '.php');
    // Instanciation du controller
    new Controller();
} else {
    include_once('app/view/layouts/404.php');
}
```

Core : modification du MC

```
<?php
// Paramétrage des erreurs
ini_set('display_errors', 1);
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// Chargement des paramètres
include_once('app/config/config.inc.php');

// Chargement du core
include_once('core/core.php');
include_once('core/coreController.php');
include_once('core/coreModel.php');
include_once('core/coreView.php');

// Chargement de l'application
include_once('app/appController.php');
include_once('app/appModel.php');

// Lancement de l'application
include_once('app/app.php');
```


Core : modification des classes

```
<?php
// Chargement du model
include_once('app/model/post.php');

class Controller extends appController
{
    public $load;
    public $model;

    function __construct()
    {
        $this->load = new coreView();
        $this->model = new Model();
    }
}
```

```
<?php
class Model extends appModel
{
    function __construct()
    {
        .
    }
}
```




Core : que faire avec ?

- A partir de maintenant, à chaque besoin de développement, il faudra se demander où va le code
 - Est-ce du Core ou de l'App ?
 - Est-ce spécifique Controller/Model/View ou générique ?
- Avec le code que nous avons déjà, nous pouvons nous poser des questions afin de le réorganiser



Core : la connexion BD

- Pour l'instant, la connexion à la BD :
 - Est dans le constructeur du Model
 - Doit donc être dupliquée dans chaque Model
- Il faut donc la remonter dans l'architecture :
 - Pas dans App, car pas spécifique à l'application
 - Donc dans coreModel
 - Tous les Models auront ainsi une connexion active

Core : la connexion BD

```
<?php
class coreModel extends core
{
    function __construct()
    {
        try {
            $dns = 'mysql:host=' . DB_HOST . ';dbname=' . DB_NAME;
            // Options de connexion
            $options = array(
                PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
            $this->pdo = new PDO( $dns, DB_USER, DB_PASSWORD, $options );
        } catch (Exception $e) {
            die("Database error : " . $e->getMessage());
        }
    }
}
```

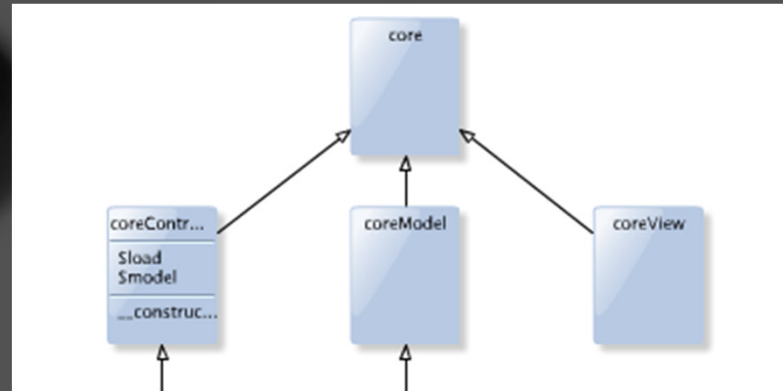
```
<?php
class Model extends appModel
{
    public function postList($limit, $offset)
    {
```



Core : les attributs

- Pour l'instant, les attributs \$load et \$model :
 - Sont dans le Controller
 - Doivent donc être dupliqués dans chaque Controller
- Il faut donc les remonter dans l'architecture :
 - Pas dans App, car pas spécifique à l'application
 - Donc dans coreController
 - Tous les Controllers auront ainsi les attributs pour afficher des Views et relier un Model

Core : les attributs



```
<?php
class coreController extends core
{
    public $load;
    public $model;
}
```

```
<?php
// Chargement du model
include_once('app/model/post.php');

class Controller extends appController
{
    function __construct()
    {
        $this->load = new coreView();
        $this->model = new Model();
    }
}
```



Core : les liens entre classes

- Pour l'instant, le controller :
 - Relie le model et le gestionnaire de view
 - Il serait étonnant qu'un Controller n'ait absolument pas besoin de ces sujets
 - Donc on peut les remonter dans l'architecture
 - Faire les includes
 - Faire les instantiations

Core : les liens entre classes

- Pour le gestionnaire de view, c'est facile :

```
<?php
class coreController extends core
{
    public $load;
    public $model;

    function __construct()
    {
        $this->load = new coreView();
    }
}
```

```
class Controller extends appController
{
    function __construct()
    {
        parent::__construct();
        $this->model = new Model();
    }
}
```

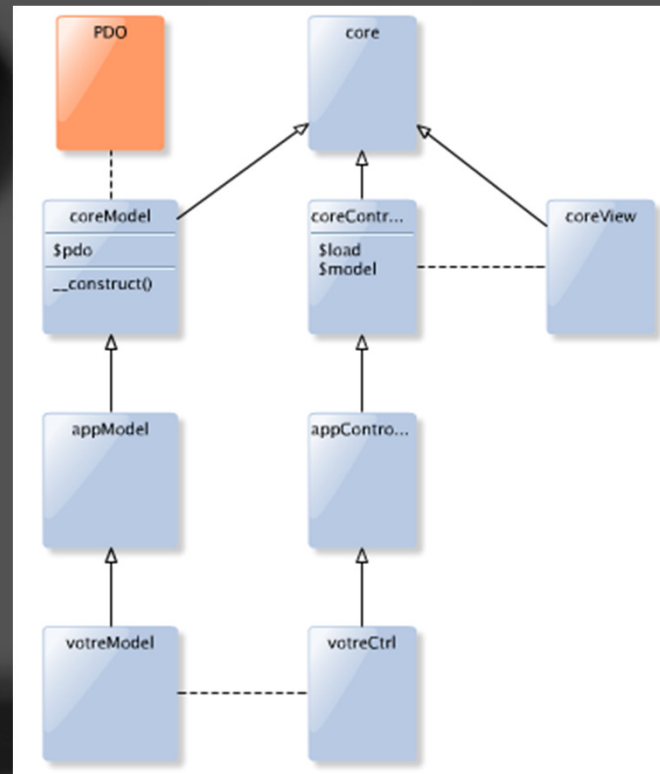
- Attention à la « surcharge » du constructeur !



Core : les liens entre classes

- Pour le lien avec le Model
 - Il est possible de le remonter aussi dans coreController
 - Mais il n'est pas possible de deviner le nom du Model
 - Il faudrait ajouter une convention de nommage
 - Exemple : postModel.php
 - En reprenant le nom du controller
 - Du coup, l'appel pourrait être générique
 - Dans l'exemple de ce cours, nous conserverons l'existant

Core : Schéma complet





Core : Requêtes SQL

- Un certain nombre de requêtes SQL se retrouvent régulièrement dans les projets
- Il est possible de les rendre génériques
- Et de les faire rentrer dans le Core
- Par exemple :
 - Nombre d'enregistrements dans une table
 - Select toutes les lignes toutes les colonnes
 - ...

Core : Requêtes SQL

- Exemple Count :

```
function coreModelCount($table)
{
    try {
        // On envoie la requête
        $query = $this->pdo->query('SELECT count(*) as theCount FROM ' . $table);
        // On récupère le résultat
        $theCount = $query->fetch();
        // Supprime le curseur
        $query->closeCursor();
        // On retourne le nombre d'enregistrements
        return $theCount['theCount'];
    }
    catch ( Exception $e ) {
        die("Erreur SQL ici : " . $e->getMessage());
    }
}
```

```
$count = $this->model->coreModelCount("blog_posts");
var_dump($count);
```



Bibliographie/**W**ebographie

- Certains contenus, concepts ou schémas de ce document sont extraits des sites et livres ci-dessous :
 - <http://www.php.net>
 - <http://www.w3schools.com>



Merci pour votre attention !

Philippe Giraud

Mail : philippe.giraud@eemi.com

Développement –MVC en PHP Objet – EEMI par Philippe Giraud