Architecture MVC

L’architecture MVC

Pour mieux comprendre les avantages du modèle MVC, nous allons voir comment convertir une application PHP standard en une application basée sur l’architecture MVC. Pour cela, nous allons reprendre la page d'affichage des liens de la séance précédente.

La programmation standard

En programmation PHP standard, l’affichage d’une liste d’éléments d’une base de données pourrait ressembler au code ci-dessous.

liste1.php

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Atelier PHP - MySQL - simple - MVC</title>
  </head>
  <body>
  <?php
      $db=new PDO('mysql:host=localhost;dbname=sites;charset=utf8', "root",
"");
      $result = $db->query ("select * from liens");
  ?>

      <h1 >Sommaire</h1>

      <table >
          <tr>
              <th>Titre</th>
              <th>URL</th>
          </tr>
          <?php while ( $lien = $result->fetch(PDO::FETCH_OBJ)  ) { ?>
              <tr>
                <td>
                   <?php echo $lien->titre; ?>
                </td>
                <td>
                   <a href="<?php echo $lien->url; ?>"><?php echo $lien->url;
?></a>
                </td>
              </tr>
          <?php } ?>
      </table>
```

```
  <?php
      $result->closeCursor();
  ?>

  </body>
</html>
```

C’est rapide à écrire et à exécuter mais difficile à maintenir. Les principaux problèmes de ce code sont :

- La syntaxe s’appuie sur du PHP et du HTML entremêlés.
- Le designer et le développeur doivent travailler sur le même fichier.
- Pas de contrôle d’erreur (que se passe-t-il si la connexion échoue ?).
- Le code n’est adapté qu’à la base de données MySQL.

Dans l'architecture MVC Web:

Une vue représente une page HTML à afficher.

Le modèle contient les données à manipuler (la vue utilise le modèle).

Le contrôleur est chargé de construire la page, il initialise le modèle et déclenche l'affichage de la vue.

Isoler la présentation

Sur le code précédent, l'utilisation des ordres `echo` et `print` complexifie la lisibilité du code et tout changement de présentation, via une modification du source HTML, devient périlleux.

Par conséquent, nous diviserons ce script en deux parties. La première partie liste2.php contiendra le code PHP, la connexion à la base de données, la création d'un variable $model et le chargement de la vue.

liste2.php

```
  <?php

      $db=new PDO('mysql:host=localhost;dbname=sites;charset=utf8', "root",
"");

      $result = $db->query ("select * from liens");

      $model = Array();
      while ( $lien = $result->fetch(PDO::FETCH_OBJ)  ) {
          $model[] = $lien;
      }

      $result->closeCursor();


      require("vue2.php");
  ?>
```

Le code HTML de la vue vue2.php, contient le code dédié à la présentation.

vue2.php

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Atelier PHP - MySQL - simple - MVC</title>
  </head>
  <body>

      <h1 >Sommaire</h1>

      <table >
          <tr>
              <th>Titre</th>
```

```
            <th>URL</th>
          </tr>
          <?php foreach ($model as $lien) { ?>
            <tr>
              <td><?php echo $lien->titre; ?></td>
              <td><a href="<?php echo $lien->url; ?>"><?php echo $lien->url;
?></a></td>
            </tr>
          <?php } ?>
      </table>

  </body>
</html>
```

Pour qu'un script de vue soit de bonne qualité, il doit contenir le moins de code PHP possible et **aucun ordres PHP encapsulant des balises HTML** de manière à être compréhensible par un graphiste démuni de connaissances en PHP. Les ordres les plus communs dans un script vue sont echo, if, foreach.

Toute la logique de traitement des données est placée dans le script Contrôleur (titre2.php) et ne contient que la syntaxe PHP, sans aucun code HTML. Dans l'idéal, il faut pouvoir réutiliser le même contrôleur pour une présentation complètement différente.

Isoler la manipulation des données

La majeure partie du code d'un script contrôleur est dédiée à la manipulation des données. Mais que se passerait-il si vous aviez besoin de la même liste de liens pour un autre contrôleur ? Comment faire si vous souhaitez conserver toutes les requêtes de base de données dans un même fichier pour éviter de dupliquer le code ? Que faire si vous décidez de changer la structure de la base de données et que la table `liens` devient `weblog_liens` ?

Nous allons donc isoler dans un fichier (dao3.php) la liste des fonctions nous permettant de manipuler notre base de données. Dans notre cas, il n'y a qu'une fonction renvoyant la liste des liens.

dao3.php

```
<?php

function Liste_Liens() {

    $db=new PDO('mysql:host=localhost;dbname=sites;charset=utf8', "root", "");
    $result = $db->query ("select * from liens");

    $model = Array();
    while ( $lien = $result->fetch(PDO::FETCH_OBJ)  ) {
        $model[] = $lien;
    }

    $result->closeCursor();
    return $model;
}

 ?>
```

Liste3.php

```
  <?php

      require("dao3.php");

      $model = Liste_Liens();

      require("vue3.php");
  ?>
```

Le contrôleur devient plus facile à lire. Son principal rôle est de transférer les données du modèle à la vue. Au sein d'applications plus complexes, il s'occupe aussi des requêtes, des sessions utilisateurs, de l'authentification etc..

Le script `dao3.php` est dédié à l'accès aux bases et doit être pensé en ce sens. Toutes les informations qui ne sont pas propres aux bases (comme les paramètres des requêtes) doivent être reçues du contrôleur et non intégrées directement dans le modèle. De cette manière le modèle peut être réutilisé dans un autre contrôleur.

La séparation en couche au-delà de MVC

Pour résumer, le principe de l'architecture MVC est de répartir les codes sur trois niveaux : la logique de données est placée dans le modèle, la présentation dans la vue, et enfin la logique applicative dans le contrôleur.

Afin de simplifier encore le codage, il est possible d'aller plus loin. Ainsi le modèle pourrait utiliser le pattern DAO, la vue pourrait utiliser un système de template.

Le contrôleur lui aussi peut être restructuré.

Action et contrôleur principal (front controller)

Dans les exemples précédents, le contrôleur ne fait pas grand chose, mais dans une application web réelle, il est sûrement celui qui a le plus de travail. De plus, la plupart des contrôleurs partagent en grande partie les mêmes actions comme la gestion des requêtes, la sécurité, la configuration de l'application etc. C'est pourquoi les contrôleurs sont plus souvent représentés par un contrôleur principal unique à l'application et par les actions, qui ne contiennent que le code contrôleur spécifique à certaines pages.

L'un des grands avantages d'un contrôleur frontal est qu'il offre un point d'entrée unique pour toute l'application. Si vous décidez de fermer l'accès à l'application, vous aurez seulement besoin de modifier le script de contrôleur frontal. Dans une application sans un contrôleur frontal, chaque contrôleur doit être désactivé.

index.php

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title></title>
    <link rel="stylesheet" href="/style.css">
    <script src="script.js"></script>
  </head>
  <body>
    <?php
      switch ($_GET["action"]) {
                case "liste":
                    require("dao3.php");
                    $model = Liste_Liens();
                    require("vue3.php");
                    break;
                case "formulaire_ajout":
                    // Code qui permet de gérer l'affichage
                    //du formulaire d'ajout
                    break;

            }
    ?>
  </body>
</html>
```

Cette page index.php fait office de front controller.

Si l'on transmet la paramètre action avec la valeur "liste", le script index.php construit la liste des liens. Il suffit de modifier ce contrôleur pour lui permettre de gérer toutes les pages de votre site.

Orienté objet

Tous les exemples précédents s'appuient sur les principes de la programmation procédurale. Mais les possibilités de la programmation orientée objet (POO) des langages modernes permettent de rendre les développements encore plus simples grâce à l'encapsulation, l'héritage et permettent, en outre, d'avoir des normes de codifications simples.

Implémenter une architecture avec un langage non objet augmente les risques de duplications de codes, de collisions de nom et d'une manière générale rend les sources plus difficiles à lire.

L'orientation objet permet aux développeurs de travailler avec les objets vue, contrôleur et les classes du modèle, et permet de transformer toutes les fonctions des exemples précédents (php standard) en méthodes. C'est un avantage considérable.