[IMAC S2] Programmation Back-end

TD 04 : Accès à la base de données et manipulations avec PHP Objet

Support

Pascale Ho - 2018 - Ingénieur IMAC

Introduction

Un des avantages de PHP est de pouvoir s'interfacer avec à peu près tous les SGBD relationnels. Pour se connecter à une bdd en PHP on va se servir d'une extension PHP qui s'appelle **PDO** (PHP Data Objects). Ce module contient entre autre une classe PDO qui contient des méthodes très utiles pour manipuler n'importe quelle bdd. PDO est orienté Objet, j'espère que vous avez retenu les anciens cours sur PHP Objet !! Et puisqu'on est en plein dans la POO, on va voir comment "recréer" notre bdd en objet.

TL;DR

- Exemple d'utilisation de requête SELECT avec PDO :

```php
<?php
try {
    // instanciation
    $pdo = new PDO(
        'mysql:host=localhost;dbname=imac',
        'mylogin',
        'mypassword!!'
    );
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // préparation et exécution
    $pdostat = $pdo->prepare("SELECT ...;");
    $values = array(
        "nom_tag" => "value",
        ...
    );
    $pdostat->execute($values);
```

```
    // exploitation des résultats
    $pdostat->setFetchMODE(PDO::FETCH_...);
    $results = $pdostat->fetchAll();
    foreach ($results as $row) {
        ...
    }

    // fin de connexion
    $pdo=null; // ou unset($pdo);
}
catch (PDOException $e) {
    echo "<div class='error-message'>Erreur:
".$e->getMessage()."</div>";
    die(); // fin brutale du script
}
```

- Servez-vous du singleton pour se connecter qu'une seule fois à la bdd dans le script PHP.

PDO

Cette extension existe depuis PHP 5 et supporte beaucoup de bdd différentes dont MySQL ! Pratique.

Présentation des classes PDO

- **PDO** : classe principale d'interaction avec le SGBD
- **PDOStatement** : classe gérant les résultats des requêtes, préparées ou exécutées
- **PDOException** : classe d'exception

Connexion et gestion des erreurs

Pour se connecter, il faut d'abord **instancier un objet PDO** avec comme paramètres un DSN (Data Source Name, en gros ce paramètre contient l'hôte et le nom de notre base), le login de l'utilisateur et le mot de passe (!). Pour finir la connexion, il suffit de "détruire" l'objet.

Par contre il serait bien de pouvoir gérer des erreurs de connexion. Dans ce cas on pourrait lancer une objet de classe PDOException (qui hérite de Exception).

```
<?php
try {
    // instanciation
    $pdo = new PDO(
```

```
        'mysql:host=localhost;dbname=imac',
        'mylogin',
        'mypassword!!'
    );
    /* Options du pilote PDO : ici permet d'afficher dans le message de
       l'exception plus de détails */
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    ...

    // fin de connexion
    $pdo=null; // ou unset($pdo);
}
catch (PDOException $e) {
    echo "<div class='error-message'>Erreur:
".$e->getMessage()."</div>";
    die(); // fin brutale du script
}
```

Requêtes préparées

Alors, pour celles et ceux qui ont déjà utilisé PDO, vous vous souvenez peut-être de la méthode query() pour exécuter une requête SQL. Et bah je vous le demande de ne plus l'utiliser et de jeter cette méthode à la poubelle !!! (On verra pourquoi dans le paragraphe sur les injections SQL). Mais si vous y tenez vraiment à vous servir de query(), et bah ne comptez-pas sur moi pour vous l'apprendre !

L'intérêt des requêtes préparées est qu'elles sont performantes lorsqu'on répète plusieurs fois la même exécution, et elles nous **protègent des attaques par injection** de code SQL (oui on va voir cela plus tard, patience).

Comme le nom l'indique, on va d'abord préparer la requête SQL avec la fonction **prepare()**. L'avantage est de pouvoir donner des **paramètres anonymes (?) ou nommés (:nom)** à notre requête qui pourrait servir de "template" lorsqu'on veut exécuter à peu près la même requête (mais avec des valeurs différentes).

On utilise **bindValue()** pour associer les paramètres de la requête préparée. Et on exécute avec **execute()**.

```
$pdo = new PDO(
    'mysql:host=localhost;dbname=imac',
    'mylogin',
    'mypassword!!'
);
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```php
// préparation avec un paramètre anonyme ?
$pdostat = $pdo->prepare("SELECT * FROM Utilisateur WHERE prenom=?");

// association et exécution
$pdostat->bindValue(1, 'Pascale'); // 1 : premier ? de la requête
$pdostat->execute();

// exécution de la même requête avec d'autres valeurs
$pdostat->bindValue(1, 'Victor');
$pdostat->execute();
```

```php
// la même chose mais avec un paramètre nommé
$pdostat = $pdo->prepare(
    "SELECT * FROM Utilisateur WHERE prenom=:prenom"
);

// association et exécution
$pdostat->bindValue(':prenom', 'Pascale');
$pdostat->execute();

// exécution de la même requête avec d'autres valeurs
$pdostat->bindValue(':prenom', 'Victor');
$pdostat->execute();
```

Lorsqu'on a plusieurs valeurs à associer, on peut faire un bindValue() pour chaque paramètre puis execute() ou alors se servir directement de execute() avec comme paramètre un **tableau** (associatif pour les paramètres nommés).

```php
$pdostat = $pdo->prepare(
    "SELECT * FROM Utilisateur WHERE prenom=? AND age<=?"
);

// première exécution
$pdostat->execute(array('Pascale', 25));

// deuxième
$pdostat->execute(array('Victor', 100));
```

```php
$pdostat = $pdo->prepare(
    "SELECT * FROM Utilisateur WHERE prenom=:prenom AND age<=:age"
);

// première exécution
```

```php
$pdostat->execute(
    array(
        ':prenom' => 'Pascale',
        ':age' => 25
    )
);

// deuxième
$pdostat->execute(
    array(
        ':prenom' => 'Victor',
        ':age' => 100
    )
);
```

Résultats d'une requête

Pour une requête de sélection, PDO va récupérer grâce à la méthode **fetch** un jeu de résultats de la requête (une liste de tableaux indexés ou associatifs, une liste d'objets). Le choix du format du résultat se fait avec **fetch_style** qui peut être contrôlé en argument de la méthode fetch, ou avec la méthode **setFetchMode()** :

- PDO::FETCH_ASSOC : retourne chaque ligne dans un **tableau indexé par les noms des colonnes** retournés par la requête.
- PDO::FETCH_NUM : retourne chaque ligne dans un **tableau indexé par les numéros des colonnes** retournés par la requête.
- **PDO::FETCH_BOTH** (par défaut) : ASSOC et NUM en même temps.
- **PDO::FETCH_OBJ** : retourne chaque ligne dans un **objet avec les noms de propriétés correspondant aux noms des colonnes** retournés par la requête.
- **PDO::FETCH_CLASS** : retourne une **instance de la classe demandée**, liant les colonnes aux propriétés nommées dans la classe.
- …

Exemples d'exploitation des résultats d'une requête :

```php
// PDO::FETCH_ASSOC
// récupération des résultats sous forme d'une liste de tableaux associatifs
...
$pdostat = $pdo->prepare("SELECT id FROM Utilisateur;");
$pdostat->execute(...);
$pdostat->setFecthMode(PDO::FETCH_ASSOC);

// une façon de récupérer les résultats avec fetch()
while (($row = $pdostat->fetch()) !== false) {
```

```
    echo $row['id']
}

// une autre avec fetchAll()
$results = $pdostat->fetchAll();
foreach ($results as $result) {
    echo $result['id']
}

// PDO::FETCH_CLASS
...
$pdostat = $pdo->prepare("SELECT id FROM Utilisateur;");
$pdostat->execute(...);
$pdostat->setFecthMode(PDO::FETCH_CLASS, "Utilisateur");
if (($object = $pdostat->fetch()) !== false) {
    // instancie que le premier objet de classe Utilisateur ayant un
attribut id
    echo $object->id;
}
```

Sécurité : injections SQL

Prenons l'exemple suivant : notre site a besoin d'un login/pass pour pouvoir accéder à ses services. Ces informations sont stockées dans la bdd, et lorsqu'on se connecte on vérifie si le login et le mot de passe correspondent bien dans notre bdd. Si on utilisait query() au lieu des requêtes préparées, notre requête ressemblerait comme ceci (d'ailleurs ne le refaites pas car niveau sécurité c'est zéro, déjà avoir un mdp en clair dans notre bdd, c'est pas bon)

```
// un exemple
$req = <<<SQL
SELECT id
FROM Utilisateur
WHERE login = '$login' AND password='$passwd';
SQL;
```

Si on rentre ce login/mdp:

```
login: p.ho
password: whatever
```

et que le mdp n'est pas le bon dans notre bdd, le site devrait interdire l'accès.

Mais que se passe-t-il si on fournit un mot de passe particulier, du genre

```
login: p.ho
password: whatever' OR true != '
```

Notre requête deviendrait :

```
SELECT id
FROM Utilisateur
WHERE login = 'p.ho' AND password = 'whatever' OR true != '';
```

Souvenez vous que l'opérateur OR passe d'abord avant AND, du coup `password = 'whatever' OR true != ''` devient vrai ! Ce genre d'attaque malveillante s'appelle une **injection SQL**. Pour se protéger, on se sert des requêtes préparées !!

Singleton PDO

Lorsqu'on développe un site nécessitant une bdd, on doit à chaque fois devoir se connecter en début de chaque page PHP, faire sa requête et se déconnecter en fin de chaque page PHP. En prog objet c'est à chaque fois qu'une méthode de classe a besoin de faire une requête dans une bdd. La problématique est la suivante : **Quand doit-on se connecter et se déconnecter ?**

Une première solution est de se servir d'un **singleton**. Vous verrez en détail en imac2 mais il s'agit d'un design pattern dont le but est de rendre un objet à instance unique (dans notre cas, cela va faire en sorte qu'on ait qu'une connexion unique à la bdd). Du coup on ne se connectera une fois lors de la première requête et on se déconnectera lorsqu'on quitte le script (c'est beaucoup plus propre ainsi).

Comme je suis gentille, en TD **vous aurez un singleton de PDO tout fait** par votre serviteure ! Donc pas besoin de coder, en plus vous pouvez vous en resservir dans vos projets ultérieurs (ou professionnellement, la preuve, je m'en suis servie à l'époque où je faisais du PHP).

Ressources Supplémentaires

PHP :

- Documentation sur PDO : http://php.net/manual/fr/book.pdo.php
- Fichiers de singleton qui vous seront fournis