Programmation orientée objet en PHP

Le ServiceLocator

Nous allons modifier notre application pour l'adapter à une architecture orientée service. Chaque classe de notre application sera un "service", c'est-à-dire une unité fonctionnelle possédant un rôle déterminé. Par exemple, la classe PersonneDao assurera la persistance des Personne(s), ou encore, la classe Router assurera le routing de notre application. Ce "service" pourra être sollicité par un autre service (une classe de service pourra posséder des dépendances vers d'autres classes de service), ou être utilisé dans les controllers.
Pour gérer l'instanciation des services, l'injection des dépendances et la récupération des services par les controllers, nous allons utiliser une classe ServiceLocator. Cette classe fournira au composant qui le demande le service prêt à l'emploi. Le ServiceLocator stockera dans un tableau les instances des services, et renverra l'instance du service demandé. Les services ne seront instanciés et stockés dans le tableau qu'à la première demande. Ensuite c'est l'instance stockée dans le tableau qui sera toujours renvoyée. Il n'y aura donc qu'une instance de chaque service.
Pour décrire les services et leurs dépendances, nous utiliserons un fichier service.json au format JSON qui contiendra, pour chaque service, le nom du service, le nom de la classe représentant ce service, les arguments à passer au constructeur :

```
[
    {
        "name": "router",
        "class": "util/Router"
    },
    {
        "name": "personne_dao",
        "class": "dao/PersonneDao"
    },
    {
        "name": "personne_service",
        "class": "business/PersonneService",
        "arguments": "@personne_dao"
    }
]
```

Le caractère @ devant un argument signifie que cet argument est lui-même un service.

Nous allons ajouter au fichier setup.php une constante qui indiquera le fichier de service :

```
<?php

// setup.php
define('SITE_URL', 'http://localhost:8080');
define('ROOT', realpath(__DIR__));
define('CONFIG', ROOT . '/config');
define('CONTROLLER', ROOT . '/controller');
define('ENTITY', ROOT . '/entity');
define('DAO', ROOT . '/dao');
define('VIEW', ROOT . '/view');
define('INI_FILE', CONFIG . '/conf.ini');
define('ROUTING_FILE', CONFIG . '/routing.json');
define('SERVICE_FILE', CONFIG . '/service.json');

spl_autoload_register(function($class) {
    list($dir, $file) = explode('\\', $class);
    include_once ROOT . "/$dir/$file.php";
```

```
});
```

Les attributs d'un service pourront être encapsulés dans un objet de type Service :

```php
<?php

// util/Service.php

namespace util;

class Service {

    private $name;
    private $class;
    private $arguments;

    function __construct($name, $class = '', $arguments = []) {
        $this->name = $name;
        $this->class = $class;
        $this->arguments = $arguments;
    }

    public function getName() {
        return $this->name;
    }

    public function getClass() {
        return $this->class;
    }

    public function getArguments() {
        return $this->arguments;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function setClass($class) {
        $this->class = $class;
    }

    public function setArguments($arguments) {
        $this->arguments = $arguments;
    }

}
```

La classe ServiceLocator possédera une méthode publique get() qui prendra en argument le nom du service, et qui renverra l'instance de ce service :

```php
<?php

// util/ServiceLocator.php

namespace util;

use ReflectionClass;
use util\Service;

class ServiceLocator {

    private $services;

    public function __construct() {
```

```
        $this->services = [];
    }

    public function get($serviceName) {
        $conf_text = file_get_contents(SERVICE_FILE);
        $conf = json_decode($conf_text, true);
        return $this->getService($conf, $serviceName);
    }

    private function getService($conf, $serviceName) {
        if (array_key_exists($serviceName, $this->services)) {
            return $this->services[$serviceName];
        } else {
            return $this->findService($conf, $serviceName);
        }
    }

    private function findService($conf, $serviceName) {
        foreach ($conf as $service) {
            if ($service['name'] == $serviceName) {
                $serviceObj = $this->createService($conf, $service);
            }
        }
        $class = new ReflectionClass($serviceObj->getClass());
        $instance = $class->newInstanceArgs($serviceObj->getArguments());
        $this->services[$serviceObj->getName()] = $instance;
        return $instance;
    }

    private function createService($conf, $service) {
        $serviceObj = new Service($service['name']);
        $className = str_replace('/', '\\', $service['class']);
        $serviceObj->setClass($className);
        if (array_key_exists('arguments', $service)) {
            $arguments = $this->getArguments($conf, $service);
            $serviceObj->setArguments($arguments);
        }
        return $serviceObj;
    }

    private function getArguments($conf, $service) {
        $arguments = explode(',', $service['arguments']);
        for ($i = 0; $i < count($arguments); $i++) {
            if ($arguments[$i][0] == '@') {
                $arguments[$i] = $this->getService($conf, substr($arguments[$i], 1));
            }
        }
        return $arguments;
    }

}
```

Une fois les services implémentés, nous pourrons tester notre ServiceLocator avec une classe de test ServiceLocatorTest :

```
<?php

include './setup.php';
include './util/ServiceLocator.php';

use util\ServiceLocator;

class ServiceLocatorTest extends PHPUnit_Framework_TestCase {

    private $locator;

```

```
12      protected function setUp() {
13          $this->locator = new ServiceLocator();
14      }
15
16      public function testGet() {
17          $router = $this->locator->get('router');
18          $router2 = $this->locator->get('router');
19          $personneDao = $this->locator->get('personne_dao');
20          $personneService = $this->locator->get('personne_service');
21          $this->assertInstanceOf('util\Router', $router);
22          $this->assertInstanceOf('util\Router', $router2);
23          $this->assertSame($router, $router2);
24          $this->assertInstanceOf('dao\PersonneDao', $personneDao);
25          $this->assertInstanceOf('business\PersonneService', $personneService);
26          $this->assertSame($personneDao, $personneService->getDao());
27      }
28
29  }
```

Fin

NAVIGATION

Accueil
- Ma page
  - Pages du site
  - Mon profil
  - Cours actuel
    - POO PHP
      - Participants
      - Généralités
      - La programmation orientée objet : premiers pas
      - L'héritage
      - Les interfaces
      - Le typage
      - Les namespaces
      - Les exceptions
      - Les bases de données avec PDO
      - Les tests avec PHPUnit
      - Petite application version 2
      - Petitie application version 3
      - **Le ServiceLocator**
      - La dao
      - Les classes de traitement métier
      - Les contrôleurs
  - Mes cours

ADMINISTRATION

Administration du cours

Réglages de mon profil

Programmation orientée objet en PHP

La dao

Nous allons renommer notre classe MysqlDao en PersonneDao puisque le rôle de cette classe est de gérer la persistance des Personne(s).
Ensuite nous allons créer une interface IPersonneDao, qui déclarera les méthodes qui doivent être implémentées par une classe de service dont le rôle sera la persistance des Personne(s) :

```php
<?php

namespace dao;

use entity\Personne;

interface IPersonneDao {

    public function getAllPersonnes();

    public function addPersonne(Personne $personne);
}
```

Afin d'alléger les responsabilités de la classe PersonneDao, et de lui permettre de travailler avec n'importe quel SGBD, nous créons une classe qui gérera la création des connexions. Cette classe de connexion utilisera les configurations du fichier conf.ini. Nous ajoutons à ce fichier une entrée correspondant au type de driver utilisé :

```
[prod]
driver = mysql
host = localhost
db_name = petite_application
user = root
password = root

[dev]
driver = mysql
host = localhost
db_name = test_petite_application
user = root
password = root

[settings]
env = dev
```

La classe Connection ne possédera qu'une méthode de classe getConnection() qui renverra un objet PDO :

```php
<?php

// dao/Connection.php

namespace dao;

use PDO;

class Connection {

    public static function getConnection() {
```

```
        $conf = parse_ini_file(INI_FILE, true);
        $settings = $conf['settings']['env'];
        $host = $conf[$settings]['host'];
        $db_name = $conf[$settings]['db_name'];
        $db_user = $conf[$settings]['user'];
        $db_password = $conf[$settings]['password'];
        $db_driver = $conf[$settings]['driver'];
        $db_datasource = null;
        switch ($db_driver) {
            case 'mysql':
                $db_datasource = "mysql:host=$host;dbname=$db_name";
                break;
        }
        $conn = new PDO($db_datasource, $db_user, $db_password
                , [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);

        return $conn;
    }

}
```

La classe PersonneDao utilisera donc cette classe Connection, et implémentera l'interface IPersonneDao :

```
<?php

// dao/PersonneDao.php

namespace dao;

use PDO;
use dao\IPersonneDao;
use entity\Personne;
use dao\Connection;

class PersonneDao implements IPersonneDao {

    private $conn;

    public function __construct() {
        $this->conn = Connection::getConnection();
    }

    public function getAllPersonnes() {
        $statement = 'SELECT * FROM personne';
        $stmt = $this->conn->prepare($statement);
        $stmt->execute();
        $result = [];
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $pers = new Personne($row['nom'], $row['prenom'], $row['id']);
            $result[] = $pers;
        }
        return $result;
    }

    public function addPersonne(Personne $personne) {
        $statement = 'INSERT INTO personne (nom,prenom) values '
                . '(:nom, :prenom)';
        $stmt = $this->conn->prepare($statement);
        $stmt->bindValue(':nom', $personne->getNom());
        $stmt->bindValue(':prenom', $personne->getPrenom());
        $result = $stmt->execute();
        return $result;
    }

}
```

Dans le répertoire test/dao nous créons une classe de test ConnectionTest :

```php
<?php

// test/dao/ConnectionTest

include_once './setup.php';
include './dao/Connection.php';

use dao\Connection;

class ConnectionTest extends PHPUnit_Framework_TestCase {

    public function testGetConnection() {
        $conn = Connection::getConnection();
        $this->assertInstanceOf('PDO', $conn);
    }

}
```

Et une classe PersonneDaoTest utilisant les mêmes datastets que le TP précédent :

```php
<?php

// test/dao/PersonneDaoTest.php

include_once './setup.php';
include_once './entity/Personne.php';
include_once './dao/PersonneDao.php';

use entity\Personne;
use dao\PersonneDao;

class PersonneDaoTest extends PHPUnit_Extensions_Database_TestCase {

    protected $connection;
    protected $dao;

    protected function getConnection() {
        if ($this->connection === null) {
            $conf = parse_ini_file(INI_FILE, true);
            $settings = $conf['settings']['env'];
            $host = $conf[$settings]['host'];
            $db_name = $conf[$settings]['db_name'];
            $db_user = $conf[$settings]['user'];
            $db_password = $conf[$settings]['password'];
            $db_driver = $conf[$settings]['driver'];
            $connectionString = "$db_driver:host=$host;dbname=$db_name";
            $this->connection = $this->createDefaultDBConnection(
                    new PDO($connectionString, $db_user
                    , $db_password));
        }
        return $this->connection;
    }

    protected function getDataSet() {
        return new PHPUnit_Extensions_Database_DataSet_YamlDataSet(
                './test/dataset/personne_base.yml');
    }

    protected function setUp() {
        $conn = $this->getConnection();
        // désactivation des contraintes de clés étrangères pour permettre
        //le chargement des données via le dataset
        $conn->getConnection()->query('set foreign_key_checks=0');
        parent::setUp();
```

```
45          // activation des contraintes de clés étrangères
46          $conn->getConnection()->query('set foreign_key_checks=1');
47          $this->dao = new PersonneDao();
48      }
49
50      public function testAddPersonne() {
51          $pers = new Personne("Kent", "Clark");
52          $this->dao->addPersonne($pers);
53          // création d'un objet dataset à partir de la connexion
54          $actualDataset = new PHPUnit_Extensions_Database_DataSet_QueryDataSet(
55                  $this->getConnection());
56          // ajout à ce dataset des enregistrements de la table personne
57          // de notre base de données de test
58          $actualDataset->addTable('personne');
59          // récupération d'un dataset à partir de notre fichier
60          $expectedDataset = new PHPUnit_Extensions_Database_DataSet_YamlDataSet(
61                  './test/dataset/add_personne.yml');
62          // comparaison des deux datasets
63          $this->assertDataSetsEqual($expectedDataset, $actualDataset);
64      }
65
66  }
```

Fin

NAVIGATION

Accueil
- Ma page
  - Pages du site
  - Mon profil
  - Cours actuel
    - POO PHP
      - Participants
      - Généralités
      - La programmation orientée objet : premiers pas
      - L'héritage
      - Les interfaces
      - Le typage
      - Les namespaces
      - Les exceptions
      - Les bases de données avec PDO
      - Les tests avec PHPUnit
      - Petite application version 2
      - Petitie application version 3
      - Le ServiceLocator
      - **La dao**
      - Les classes de traitement métier
      - Les contrôleurs
  - Mes cours

ADMINISTRATION

Administration du cours

Réglages de mon profil

Connecté sous le nom « Arnaud Lemais » (Déconnexion)
POO PHP

Programmation orientée objet en PHP

Les classes de traitement métier

Nous allons implémenter des classes de service métier dont le rôle va être de contenir les traitements "métiers" de notre application. Pour l'instant, lorsque nous ajoutons une personne, le traitement métier de vérification du nom et du prénom est contenu dans le controller. Ce dernier possède donc, dans cette méthode, une double responsabilité : vérifier les champs de la Personne et rediriger vers la bonne page.
Notre classe de service métier va nous permettre d'extraire ce traitement du controller.

Dans un répertoire business, nous allons créer une classe PersonneService qui contiendra tous les traitements métiers liés à l'entité Personne. Cette classe ne contiendra que deux méthodes :

1. addPersonne() qui vérifiera la validité de la personne et permettra sa persistance
2. getAllPersonnes() qui ne fera qu'appeler la méthode du même nom de la DAO (on appelle cela de la délégation d'appel)

Cette classe possédera une dépendance vers un objet de type IPersonneDao. Cette dépendance lui sera fournie par le constructeur. Notre classe de service implémentera aussi une interface qui déclarera les méthodes qu'un service gérant les Personne(s) doit implémenter :

```php
<?php

// business/IPersonneService.php

namespace business;

interface IPersonneService {

    public function addPersonne($nom, $prenom);

    public function getAllPersonnes();
}
```

La classe PersonneService implémentera cette interface :

```php
<?php

// business/PersonneService.php

namespace business;

use entity\Personne;
use dao\IPersonneDao;
use business\IPersonneService;

class PersonneService implements IPersonneService {

    private $dao;

    public function __construct(IPersonneDao $dao) {
        $this->dao = $dao;
    }

    public function addPersonne($nom, $prenom) {
        $nom = filter_var($nom, FILTER_VALIDATE_REGEXP
                , ['options' => ['regexp' => '/^[A-Za-z- ]{1,20}$/']]);
        $prenom = filter_var($prenom, FILTER_VALIDATE_REGEXP
```

```
                , ['options' => ['regexp' => '/^[A-Za-z0-9]{4,10}$/']]);
        if ($nom && $prenom) {
            $personne = new Personne($nom, $prenom);
            $result = $this->dao->addPersonne($personne);
            return $result;
        } else {
            return false;
        }
    }

    public function getAllPersonnes() {
        $personnes = $this->dao->getAllPersonnes();
        return $personnes;
    }

    public function getDao() {
        return $this->dao;
    }

}
```

Nous allons pouvoir tester la classe de service en créant un mock de sa dépendance, c'est-à-dire la classe implémentant IPersonneDao. Dans le répertoire test/business nous créons une classe PersonneServiceTest :

```
<?php

include './business/IPersonneService.php';
include './business/PersonneService.php';
include './entity/Personne.php';
include './dao/IPersonneDao.php';

use business\PersonneService;

class PersonneServiceTest extends PHPUnit_Framework_TestCase {

    private $service;

    protected function setUp() {
        $dao = $this->getMock('dao\IPersonneDao');
        $dao->expects($this->any())->method('addPersonne')
                ->with($this->isInstanceOf('entity\Personne'))->willReturn(1);
        $this->service = new PersonneService($dao);
    }

    public function testAddPersonne() {
        $nom = "Sparrow";
        $prenom = "Jack";
        $result = $this->service->addPersonne($nom, $prenom);
        $this->assertEquals(1, $result);
        $result = $this->service->addPersonne('R2D2', 'azerty');
        $this->assertFalse($result);
    }

}
```

Fin

NAVIGATION

Accueil
- Ma page

Pages du site
Mon profil
Cours actuel

ADMINISTRATION

Administration du cours

Réglages de mon profil

Connecté sous le nom « Arnaud Lemais » (Déconnexion)
POO PHP

Programmation orientée objet en PHP

Les contrôleurs

Les contrôleurs vont pouvoir récupérer des services grâce à une instance de la classe ServiceLocator. Cette classe ServiceLocator devra donc être accessible au sein de chaque contrôleur. Chaque contrôleur possédera un attribut d'instance $serviceLocator qui pointera vers l'instance unique et commune de ServiceLocator. Cette instance de ServiceLocator sera créée dans le script index.php (le front controller). Le front controller utilisera le service locator pour récupérer le service de routing :

```php
<?php

// public/index.php

include '../setup.php';

$serviceLocator = new util\ServiceLocator();
$router = $serviceLocator->get('router');
$request_uri = $_SERVER['REQUEST_URI'];
$route = $router->getRoute($request_uri);
if ($route == null) {
    include VIEW . '/index.php';
} else {
    try {
        $controller_name = $route->getController();
        $action_name = $route->getAction();
        $class_name = "controller\\$controller_name";
        $class = new ReflectionClass($class_name);
        $controller = $class->newInstance();
        $method = $class->getMethod("$action_name");
        $method->invoke($controller);
    } catch (Exception $ex) {
        include VIEW . '/errorPage.html';
    }
}
```

L'unique instance de ServiceLocator sera donc créée dans le front controller et devra être aussi attribut d'instance de chaque contrôleur. Pour cela, nous allons créer une classe abstraite qui possédera en attribut d'instance l'unique instance de ServiceLocator :

```php
<?php

// controller/MyController.php

namespace controller;

abstract class MyController {

    protected $serviceLocator;

    public function __construct() {
        global $serviceLocator;
        $this->serviceLocator = $serviceLocator;
    }

}
```

Nos contrôleurs hériteront donc de cette classe abstraite, et pourront utiliser le ServiceLocator qu'ils possèdent en attribut d'instance :

```php
<?php

// controller/PersonneController.php

namespace controller;

use controller\MyController;

class PersonneController extends MyController {

    public function addPersonne() {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $nom = $_POST['nom'];
            $prenom = $_POST['prenom'];
            $personneService = $this->serviceLocator->get('personne_service');
            $result = $personneService->addPersonne($nom, $prenom);
            if ($result) {
                $this->getAllPersonnes();
            } else {
                $msg = 'Champs incorrects';
                include VIEW . '/formulaire.php';
            }
        } else {
            include VIEW . '/formulaire.php';
        }
    }

    public function getAllPersonnes() {
        $personneService = $this->serviceLocator->get('personne_service');
        $result = $personneService->getAllPersonnes();
        include VIEW . '/display_personne.php';
    }

}
```

Les tests fonctionnels des contrôleurs seront les mêmes que ceux du TP précédent.

Fin

La dao
Les classes de traitement métier
Les contrôleurs
Mes cours

ADMINISTRATION
Administration du cours
Réglages de mon profil

Connecté sous le nom « Arnaud Lemais » (Déconnexion)
POO PHP