[IMAC S2] Programmation Back-end

TD 02 : PHP Objet

Compléments

Pascale Ho - 2018 - Ingénieur IMAC

Programmation Objet Avancé

Interface

Notion assez proche de la classe abstraite, une **interface** permet de spécifier quelles méthodes doivent être implémentées par une classe utilisant cette interface. Une classe peut implémenter plusieurs interfaces. C'est un moyen certain qu'un objet particulier doit posséder des méthodes particulières. Dans les interfaces **on indique uniquement les prototypes des méthodes** que les classes l'implémentant doivent les définir ensuite.

Comme pour les classes abstraites, on utilise les interfaces dans le but de répondre à des besoins précis.

```php
// interface
interface Comparable {
    public function compare($a); // méthode de l'interface
}
// classe implémentant l'interface
class Personne implements Comparable {
    private $age;
    public function __construct($age) {
        $this->age = $age;
    }
    // méthode de l'interface
    public function compare($a) {
        return $this->age <=> $a->age; // opérateur spaceship <=>
    }
}
```

Quelles différences entre une interface et une classe abstraite ?

Les interfaces et les classes abstraites servent le même objectif : créer des templates de structure pour des classes qui en ont besoin. Il existe cependant des différences (mineures):

- Une interface ne peut contenir uniquement des méthodes contrairement à une classe abstraite
- Les méthodes des interfaces sont en **public** uniquement
- Une classe ne peut étendre qu'une seule classe abstraite tandis qu'elle peut implémenter plusieurs interfaces

Auto-chargement de classes

Lorsque vous faîtes une application "objet", vous allez devoir à chaque fois que vous codez un script devoir inclure les fichiers contenant les classes que vous avez besoin (ce qui peut devenir fastidieux). C'est là qu'intervient la fonction **__autoload()** : elle permet d'inclure automagiquement une classe lorsqu'on souhaite l'utiliser et qu'elle n'est pas encore définie.

```php
<?php
function __autoload($class_name) {
    require_once $class_name . '.class.php';
    // ou include, include_once, require
}
$obj = new Personne(); // -> require_once 'Personne.class.php'
```

Il est possible d'enregistrer une (ou des) fonction(s) d'autoload avec **spl_autoload_register**.

```php
spl_autoload_register(
    function ($class_name) {
        include "src/{$class_name}.class.php";
    }
);
```

Namespace

Le but de créer des espaces de noms (**namespace**) est d'éviter les collisions de noms pour les classes, interfaces, fonctions et constantes. On peut créer des alias afin de simplifier l'accès aux namespace.

fichier mysql.php :

```php
<?php
namespace Projet\MySQL;
// une classe de connexion à la bdd
class Connexion { ...}
```

fichier login.php

```php
<?php
// une autre façon
namespace  Projet\App {
// une classe des connexion (session)
class Connexion {...}
}
```

utilisation des namespaces :

```php
<?php
require_once 'mysql.php';
require_once 'login.php';

$mysql = new Projet\MySQL\Connexion();
$login = new Projet\App\Connexion();
```

utilisation d'alias :

```php
use Projet\MySQL;
$mysql = new MySQL\Connexion();
```

```php
use Projet\MySQL as BD;
$mysql = new BD\Connexion();
```

Méthodes magiques

`__construct()`	constructeur
`__destruct()`	destructeur
`__set()`	fonction déclenchée lorsqu'on souhaite accéder en écriture à un attribut inaccessible ou inexistant
`__get()`	fonction déclenchée lorsqu'on souhaite accéder en lecture à un attribut inaccessible ou inexistant
`__call()`	fonction déclenchée lorsqu'on appelle une méthode inaccessible ou inexistante
`__callstatic()`	fonction déclenchée lorsqu'on appelle une méthode statique inaccessible ou inexistante
`__isset()`	fonction déclenchée lorsqu'on lance un isset() sur un attribut inaccessible ou inexistant
`__unset()`	fonction déclenchée lorsqu'on lance un unset() sur un attribut inaccessible ou inexistant

`__toString()`	fonction déclenchée lorsqu'on veut afficher directement l'objet (echo $objet)
`__clone()`	fonction déclenchée lorsqu'on essaie de cloner l'objet

Fonctions sur les classes utiles

`get_class()`	retourne la classe d'un objet
`class_parents()`	retourne un tableau de la classe parent et de tous ses parents
`class_implements()`	retourne un tableau de toutes les interfaces implémentées par la classe et par tous ses parents
`class_exists()`	vérifie qu'une classe a bien été définie
`get_declared_classes()`	retourne une liste de classes définies
`get_class_methods()`	retourne une liste des méthodes d'une classe
`get_class_vars()`	retourne une liste des attributs d'une classe