Développement Web
PHP Avancé

Jean-Michel Richer
jean-michel.richer@univ-angers.fr
http://www.info.univ-angers.fr/pub/richer

FACULTÉ
DES SCIENCES
Unité de formation
et de recherche

2008

Objectif

Objectif du cours

Entrevoir des fonctionnalités avancées de PHP ainsi que

- frameworks
- conventions de codage
- documentation
- SPL
- gestion des fichiers

Plan

Frameworks

Frameworks et PHP

PEAR - PHP Extension and Application Repository

Definition (PEAR `http://pear.php.net/`)

La mission de PEAR est de fournir :

- une librairie structurée de code source libre pour les utilisateurs de PHP
- un système de distribution du code source et de maintenance des paquets.
- un style de codage pour les programmes écrit en PHP
- une bibliothèque d'extensions de PHP (PECL - PHP Extension Code Library),
- un site Web, des listes de diffusion et des sites mirroirs pour supporter la communauté PHP/PEAR

PECL - PHP Extension Code Library

Definition (PECL `http://pecl.php.net/`)

- PECL est un projet distinct de PEAR pour distribuer les extensions de PHP (code écrit en C et compilé, telle que l'extention **PDO**).
- les extensions PECL sont aussi distribuées en paquets et peuvent être installées avec l'installeur de PEAR.

Zend Framework

Definition (Zend `http://framework.zend.com/`)

Le framework Zend (ZF) est un framework open source, orienté objet de PHP 5 axé sur le développement de sites web. Il repose notamment sur une implantation MVC.

Zend Framework

Definition (Zend `http://framework.zend.com/`)

Le framework Zend (ZF) est un framework open source, orienté objet de PHP 5 axé sur le développement de sites web. Il repose notamment sur une implantation MVC.

Definition (framework)

un framework est un espace de travail modulaire : ensemble de bibliothèques, d'outils et de conventions permettant le développement d'applications (on traduit par cadre d'applications ou cadriciel)

Zend Framework

Pour info (wikipedia)

Les fondateurs de Zend Technologies, Zeev Suraski et Andi Gutmans, ont commencé à travailler sur PHP en 1997 lorsqu'ils ont créé une nouvelle implémentation de PHP, basée sur son prédécesseur PHP2/FI, créé par Rasmus Lerdorf. Cette implémentation a donné naissance à PHP3, qui a révolutionné le monde PHP et jeté les bases de ce qu'allait devenir le langage tel qu'il est connu en 2008. En 1999, ils ont grandement amélioré le moteur de script de PHP4 nommé *Zend Engine*.

Note : Zend Studio

Autres Frameworks

Wikipedia

Wikipedia donne une liste assez importante de framework PHP :

`http://fr.wikipedia.org/wiki/Liste_de_frameworks_PHP`

Conventions de codage

Conventions de codage

Conventions de codage

Definition (Conventions de codage)

Elles servent à mettre en place des normes de manière à ce qu'un programmeur puisse relire et comprendre facilement le code produit par d'autres programmeurs.

- certaines conventions sont dictées par le langage
- d'autres sont établies par les programmeurs et concernent :
  - le nommage des variables et fonctions
  - la dispositions des blocs (if-then-else, while, ...)
  - le nommage des paramètres, ...

nommage des variables et fonctions

Conventions à appliquer

- les noms doivent être courts et explicites
- les noms de variables et fonctions ne contiennent que des lettres minuscules et le symbole souligné `'_'`
- un nom de variable est mis au pluriel s'il contient plusieurs éléments
- une méthode qui retourne un attribut d'une classe doit commencer par `get_`
- une méthode qui fixe un attribut d'une classe doit commencer par `set_`

Disposition des blocs

On utilisera préférentiellement la syntaxe :

Disposition des blocs

```
1 function divise($a, $b) {
2    if ($b==0) {
3       throw new exception("zero");
4    } else {
5       return $a/$b;
6    }
7 }
```

Autres conventions

- on supprime le plus possible les espaces inutiles
- on déclare les variables au moment où on les utilise
- on sort des boucles les calculs inutiles ou redondants (refactoring)
- on documente les fonctions, méthodes et variables / attributs

Exemple

Convention en application

```
1 function moyenne($arr_notes) {
2     $nbr=count($arr_notes);
3     $somme=0;
4     for ($i=0;$i<$nbr;++$i) {
5         $somme+=$arr_notes[$i];
6     }
7     return $somme/$nbr;
8 }
```

Documentation

A quoi sert la documentation du code ?

Elle a pour but d'expliquer au programmeur qui ne connait pas le code :

- ce que contiennent les variables
- ce que font les fonctions (rôle, comportement, paramètres en entrée / sortie, valeur de retour)

Documentation

A quoi sert la documentation du code ?

Elle a pour but d'expliquer au programmeur qui ne connait pas le code :

- ce que contiennent les variables
- ce que font les fonctions (rôle, comportement, paramètres en entrée / sortie, valeur de retour)

PHP + Doc

C'est d'autant plus important en PHP étant donné l'absence de typage et la non distinction entre procédure et fonction.

Documentation et commentaires

Commentaires

Il existe trois types de commentaires

- `// commentaire sur une ligne`
- `/* commentaire sur plusieurs lignes */`
- `/** commentaire pour documentation */`

Documentation et commentaires

Commentaires

Il existe trois types de commentaires

- `// commentaire sur une ligne`
- `/* commentaire sur plusieurs lignes */`
- `/** commentaire pour documentation */`

A noter

il ne faut pas mettre de ligne blanche entre le commentaire de documentation et la variable ou la fonction à laquelle il se rapporte.

Format des commentaires (1/2)

tags généralistes

On peut introduire des *tag* (mots-clés) au sein des commentaires pour la documentation qui seront ensuite utilisés pour générer la documentation au format html :

- `@author Author Name` nom de l'auteur
- `@copyright Copyright Information`
- `@deprecated [version|information]`
- `@license url [license]`
- `@link url [description]`
- `@version description`

Format des commentaires (2/2)

autres tags

- `@param datatype $variablename description`
- `@return datatype description`
- `@see reference`
- `@todo description`
- `@var datatype` variables de classes

Commentaire des classes

Commentaires des classes

```
1  /**
2   * classe utilisée pour représenter une personne
3   * une personne est définie par son nom sous forme de
4   * chaîne de caractère
5   */
6  class Personne {
7      /**
8       * nom de la persone
9       */
10     protected $nom;
11 }
```

Commentaire des fonctions

Commentaires des fonctions

```php
/**
 * calcul de la moyenne
 * on calcule la moyenne d'un ensemble de notes donnée
 *
 * @param array $arr_notes tableau de notes entières
 * @return float moyenne des notes
 */
function moyenne($arr_notes) {
    $nbr=count($arr_notes);
    $somme=0;
    // parcours séquentiel du tableau suivant la clé entière
    for ($i=0;$i<$nbr;++$i) {
        $somme+=$arr_notes[$i];
    }
    return $somme/$nbr;
}
```

Générer la documentation

PHPDoc

`phpdoc` permet de générer de la documentation à partir de fichiers sources PHP (comme on le ferait avec doxygen).

Générer la documentation

PHPDoc

`phpdoc` permet de générer de la documentation à partir de fichiers sources PHP (comme on le ferait avec doxygen).

Installer phpdoc

```
sudo pear install
channel ://pear.php.net/phpdoc-0.1.0
```

Générer la documentation

PHPDoc

`phpdoc` permet de générer de la documentation à partir de fichiers sources PHP (comme on le ferait avec doxygen).

Installer phpdoc

```
sudo pear install
channel ://pear.php.net/phpdoc-0.1.0
```

Générer la documentation

```
phpdoc -s srcdir -d docdir
```

PEAR Conventions

PEAR Coding Conventions

Les conventions de codage et commentaires de PEAR sont disponibles à l'adresse suivante :

`http://pear.php.net/manual/en/standards.php`

SPL

Standard PHP Library

Introduction à la SPL

Definition (Standard PHP Library)

Il s'agit d'un ensemble de classes et d'interfaces sensées apporter à PHP un comportement Orienté Objet dans le cadre de la programmation. Elle repose sur

- des itérateurs
- gestion des répertoires et fichiers
- support XML
- gestion des tableaux sous forme de classe
- définition d'exceptions

SPL Classe et Interfaces

Classes et interfaces

Pour connaître les classes et interfaces implantées par la SPL :

```
1 print_r(spl_classes());
```

Définition d'un itérateur

Definition (Itérateur)

Un itérateur est généralement une classe dont la fonction est de parcourir une autre classe stockant des objets (ex : tableau, liste, pile, file, dictionnaire, ...)

A noter

Tous les itérateurs héritent de l'interface abstraite **Traversable**

L'interface iterator (1/2)

Interface iterator

```
1 interface Iterator implements Traversable {
2   public function current();
3   public function key();
4   public function next();
5   public function rewind();
6   public function valid();
7 }
```

L'interface iterator (2/2)

Description des méthodes

current	valeur de l'élément courant
key	clé de l'élément courant
next	passe à l'élément suivant
rewind	retourne au début
valid	vrai si on est pas à la fin

Fonctions agissant sur les itérateurs

`iterator_to_array(iterator)`

convertion d'un itérateur en tableau

`iterator_count(iterator)`

compte le nombre d'éléments

`iterator_apply(iterator,callback)`

applique une fonction sur chaque élément de l'itérateur

Exemple

Exemple avec des itérateurs

```php
$tableau=array(1,2,3);
$iterator=new ArrayIterator($tableau);
echo "il y a ". iterator_count($iterator)
   ." elements\n";
print_r(iterator_to_array($iterator));
print_r($iterator);
```

Exemple et résultat

Exemple avec des itérateurs

```
il y a 3 elements
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
ArrayIterator Object
(
    [0] => 1
    [1] => 2
    [2] => 3
)
```

Comportement d'un tableau

Interface ArrayAccess

Elle permet de manipuler un objet sous forme de tableau

Interface Countable

Elle permet de connaître le nombre d'éléments d'une classe

Interface ArrayAccess (1/2)

Interface ArrayAccess

```php
interface ArrayAccess {
  public function offsetExists($offset);
  public function offsetSet($offset, $value);
  public function offsetGet($offset);
  public function offsetUnset($offset);
}
```

L'interface ArrayAccess (2/2)

Description des méthodes	
offsetExists	détermine si l'indice existe
offsetSet	attribue une valeur à l'indice donné
offsetGet	retourne la valeur à l'indice donné
offsetUnset	supprime la donnée à l'indice donné

Interface Countable

Interface Countable

```
1 Interface Countable {
2   public function count();
3 }
```

Interface IteratorAggregate

Interface IteratorAggregate

Elle permet à un objet container de fournir un iterator qui permettra de parcourir les objets qu'il contient

Interface IteratorAggregate

```
1 interface IteratorAggregate extends
Traversable {
2   public function getIterator();
3 }
```

Exemple d'utilisation d'IteratorAggregate

IteratorAggregate

```php
class MyContainer implements IteratorAggregate {
  protected $tab;
  public function __construct() {
    $this->tab=array(1,2,3);
  }
  public function getIterator() {
    return new ArrayIterator($this->tab);
  }
}
foreach(new MyContainer() as $value) {
  echo $value."\n";
}
```

La classe ArrayIterator

class ArrayIterator(array)

Elle permet de générer un itérateur sur un tableau PHP

class ArrayIterator

```php
$tableau=range(1,10);
$iterator=new ArrayIterator($tableau);
foreach ($iterator as $val) {
    echo $val."\n";
}
```

La classe LimitIterator

class LimitIterator(iterator,index,length)

Elle permet de générer un itérateur sur un tableau PHP dont on fixe la plage de valeurs

class LimitIterator

```php
$tableau=range('c','r');
$iterator=new ArrayIterator($tableau);
$limit=new LimitIterator($iterator,3,2);
// donne f et g
foreach ($limit as $val) {
    echo $val."\n";
}
```

La classe AppendIterator

class AppendIterator()

Elle permet de générer un itérateur sur plusieurs tableaux

class AppendIterator

```php
$it1=new ArrayIterator(range(1,5));
$it2=new ArrayIterator(range(10,15));
$appiterator=new AppendIterator();
$appiterator->append($it1);
$appiterator->append($it2);
// 1 2 3 4 5 10 11 12 13 14 15
foreach ($appiterator as $value) {
    echo $value." ";
}
```

La classe FilterIterator

class FilterIterator(iterator)

Elle permet de filtrer les valeurs en redéfinissant la méthode `accept`

class FilterIterator

```php
class PlusGrandQue12 extends FilterIterator {
  public function accept() {
    return ($this->current() > 12);
  }
}
$iterator=new ArrayIterator(range(1,15));
$filter=new PlusGrandQue12($iterator);
print_r(iterator_to_array($filter));
```

La classe RegexIterator

class RegexIterator(iterator,expreg)

Elle permet de filtrer les valeurs en utilisant une expression régulière

class RegexIterator

```php
$tableau=array('pomme', 'abricot',
        'poire', 'banane', 'pomelos' );
$iterator=new ArrayIterator($tableau);
$regiterator=new RegexIterator($iterator, '/^po/');
// [0] => pomme [2] => poire [4] => pomelos
print_r(iterator_to_array($regiterator));
```

La classe IteratorIterator

class IteratorIterator(variable)

Elle permet de créer un iterateur sur les classes qui implantent uniquement l'interface Traversable. On l'utilise notamment avec PDO.

class IteratorIterator

```
1 $pdoStatement=$db->query('SELECT * FROM table');
2 $iterator=new IteratorIterator($pdoStatement);
3 $limit=new LimitIterator($iterator,0,10);
4 print_r(iterator_to_array($limit));
```

Autres itérateurs

Autres itérateurs

Il existe de nombreux autres itérateurs :

- CachingIterator
- SeekableIterator
- NoRewindIterator
- EmptyIterator
- InfiniteIterator
- RecursiveArrayIterator
- RecursiveIteratorIterator

Implantation directe : SimpleXMLIterator (1/3)

On dispose de la classe SimpleXMLIterator pour parcourir les fichiers XML.

```
<bibliotheque>
  <livre>
     <titre>PHP pour les mules</titre>
     <auteur>Guy Bouricot</auteur>
  </livre>
  <livre>
     <titre>Le grand livre d'HTML</titre>
     <auteur>Jean Sans Peur</auteur>
  </livre>
</bibliotheque>
```

Implantation directe : SimpleXMLIterator (2/3)

Lecture d'un fichier XML

```php
$bibliotheque=new SimpleXMLIterator(file_get_contents('bibliotheque.xml'));
var_dump($bibliotheque);
foreach($bibliotheque as $livre) {
   echo "$cle \n";
   if ($livre->hasChildren()) {
      foreach($livre->getChildren() as $tag=>$valeur) {
         echo "\t $tag : $valeur\n";
      }
   }
}
```

Implantation directe : SimpleXMLIterator (2/3)

Lecture d'un fichier XML

```php
$bibliotheque=new SimpleXMLIterator(file_get_contents('bibliotheque.xml'));
var_dump($bibliotheque);
foreach($bibliotheque as $livre) {
   echo "$cle \n";
   if ($livre->hasChildren()) {
      foreach($livre->getChildren() as $tag=>$valeur) {
         echo "\t $tag : $valeur\n";
      }
   }
}
```

Résultat

```
titre  : PHP pour les mules
auteur  : Guy Bouricot

titre  : Le grand livre HTML
auteur  : Jean Sans Peur
```

La classe ArrayObject (1/2)

ArrayObject

Elle permet de gérer un tableau (array) sous forme objet.

ArrayObject

```
1 class ArrayObject implements
2   IteratorAggregate, Traversable,
3   ArrayAccess, Countable   {
4 }
```

La classe ArrayObject (2/2)

ArrayObject

```php
$tableau=range(2,5);
$object=new ArrayObject($tableau);
$object->append('hello');
$object[2]='a';
$object['color']='red';
print_r($object->getIterator());
```

Résultat

```
ArrayIterator Object
(
    [0] => 2
    [1] => 3
    [2] => a
    [3] => 5
    [4] => hello
    [color] => red
)
```

Serialisation

Interface Serializable

Elle permet de transmettre des objets entre pages web et elle est définie car les attributs privés des classes ne sont pas sérialisables.

Interface Serializable

```
1 interface Serializable {
2   public function serialize();
3   public function unserialize($serialized);
4 }
```

Exemple Serialisation (1/2)

classe de base

```
1  class Base implements Serializable {
2    private $base_var;
3    public __construct() {
4      $this->base_var='hello';
5    }
6    public function serialize() {
7      return serialize($this->base_var);
8    }
9    public function unserialize($serialized) {
10     $this->base_var=unserialize($serialized);
11   }
12 }
```

Exemple Serialisation (2/2)

sous classe

```php
class SubClass extends Base {
  private $sub_var;
  public __construct() {
    parent::__construct();
    $this->sub_var='world';
  }
  public function serialize() {
    $base=parent::serialize();
    return serialize(array($this->sub_var,$base));
  }
  public function unserialize($serialized) {
    $data=unserialize($serialized);
    $this->sub_var=$data[0];
    parent::unserialize($data[1]);
  }
}
```

Exceptions SPL (1/2)

Exceptions SPL

La SPL définit plusieurs exceptions d'un point de vue sémantique :

- **LogicException** émise si une expression logique est invalide
- **LengthException** émise si une taille est invalide
- **DomainException** émise si une valeurs n'est pas du domaine
- **OutOfBoundsException** émise si une valeurs est invalide

Exceptions SPL (2/2)

Exceptions SPL (suite)

La SPL définit plusieurs exceptions d'un point de vue sémantique :

- **OutofRangeException** émise si une valeur est en dehors de l'intervalle
- **OverflowException** émise si une valeur est trop grande
- **UnderflowException** émise si une valeurs est trop petite
- **InvalidArgumentException** émise si un argument est invalide

Gestion des fichiers

Les Fichiers

Gestion des fichiers

Lecture / Ecriture

On dispose de fonctionnalités avancées pour la lecture et l'écriture des fichiers, notamment grâce à :

- SPL avec la classe `SPLFileInfo`
- `file_get_contents`
- `file_put_contents`

Lecture du contenu

`file_get_contents(filename, [flags, ...])`

Lit le contenu d'un fichier en totalité et le retourne dans une chaîne. Les paramètres de la fonction sont les suivants :

- `filename` nom du fichier
- `flags` : *FILE_USE_INCLUDE_PATH*, *FILE_TEXT*
- `context` à NULL pour les fichiers
- `offset` position de début de lecture
- `maxlen` nombre d'octets à lire, si non spécifié lecture en totalité

On retourne `FALSE` en cas d'erreur

Exemple Lecture 1

Lecture d'un fichier

```php
   // compatibilité PHP 4
$filename='file_get_contents1.php';
if (!function_exists('file_get_contents')) {
   echo "without file_get_contents\n";
   $file=@fopen($filename,'r');
   $string=fread($file,filesize($filename));
   @fclose($file);
} else {
   echo "with file_get_contents\n";
   $string=file_get_contents($filename);
}
if ($string===false) {
   throw excetion('could not read');
} else {
   echo $string;
}
```

Exemple Lecture 2

Lecture d'une page web

```php
$ctx=stream_context_create(array(
    'http' => array(
        'timeout' => 1
        )
    )
);
$file=file_get_contents("http ://www.info.univ-angers.fr/index.php", 0, $ctx);
echo $file ;
```

Ecriture du contenu

`file_put_contents(filename, data [,flags, ...])`

Ecrit dans un fichier la donnée `data`. Les paramètres de la fonction sont les suivants :

- `filename` nom du fichier
- `data` chaine, tableau ou ressource de flux
- `flags` : *FILE_USE_INCLUDE_PATH*, *FILE_TEXT*, *FILE_APPEND*, *FILE_BINARY*
- `context` à NULL pour les fichiers

On retourne `FALSE` en cas d'erreur

Exemple Ecriture 1

Ecriture dans un fichier

```php
$tableau=array('hello', 'world', 2009);
// stocke la chaine 'hello world 2009' dans le fichier log.txt
file_put_contents('log.txt',implode(' ',$tableau)."\n");
```

Exemple Ecriture 2

Remplace les balises `<livre> ... </livre>` par `<book> ... </book>`

Ecriture dans un fichier

```
1 $string=file_get_contents('test.xml');
2 $string=preg_replace("/<(\/?)livre>/","<\\1book>",$string);
3 file_put_contents('test_en.xml',$string);
```

Exemple Ecriture 1

Remplace les balises `<livre> ... </livre>` par `<book> ... </book>`

Ecriture dans un fichier

```
1 $string=file_get_contents('test.xml');
2 $string=preg_replace("/<(\/?)livre>/","<\\1book>",$string);
3 file_put_contents('test_en.xml',$string);
```

Parcours des répertoires

à la manière de PHP

- on utilise la fonction `opendir` pour ouvrir un répertoire
- puis `readdir` pour itérer sur les fichiers

Parcours des répertoires

Parcours PHP

```php
$dir = "./";
if (is_dir($dir)) {
   if ($dh = opendir($dir)) {
      while (($file = readdir($dh)) !== false) {
         echo "fichier : $file : type : " . filetype($dir . $file) . "\n";
      }
      closedir($dh);
   }
}
```

Parcours des répertoires

Parcours SPL

```
1 try {
2   foreach (new DirectoryIterator('./') as
$item) {
3     echo $item."\n";
4   }
5 } catch(Exception $e) {
6   echo "No files Found!\n";
7 }
```

Fin

Fin