PHP : SPL

L3 Informatique - UE Développement Web

David Lesaint

david.lesaint@univ-angers.fr

Janvier 2020

Introduction à la SPL

Standard PHP Library

Un ensemble de classes et d'interfaces :

- Structures de données dédiées (liste chaînée, tas, ...).
- Itérateurs.
- Exceptions (débordements, ...).
- Fonctions (réflexion, autochargement, ...).
- Gestion des répertoires et fichiers.
- Autres (gestion des tableaux sous forme d'objets, patron Observer).

Intégrée au langage depuis PHP 5.0.

Pour connaître les classes et interfaces implantées par la SPL

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

Structures de données

Les classes SPL

`SplDoublyLinkedList`	Liste doublement chaînée.
`SplStack`	Pile (hérite de `SplDoublyLinkedList`).
`SplQueue`	File (hérite de `SplDoublyLinkedList`).
`SplHeap`	Tas.
`SplMaxHeap`	Tas (maximum sauvergardé en haut).
`SplMinHeap`	Tas (minimum sauvegardé en haut).
`SplPriorityQueue`	File de priorité.
`SplFixedArray`	Tableau indicé de taille fixe.
`SplObjectStorage`	Ensemble ou dictionnaire d'objets.

Itérateurs

Itérateur

Objet permettant de parcourir tous les éléments d'un conteneur (tableau, liste, pile, file, dictionnaire, ...)

Les classes SPL implémentent

- Des interfaces prédéfinies : `Traversable`, `Iterator`, `IteratorAggregate`, `ArrayAccess`
- Des interfaces de la SPL : `Countable`, `OuterIterator`, `RecursiveIterator`, `SeekableIterator`

L'itération sur un conteneur peut se faire avec `foreach` sur un objet itérateur.

L'interface `Iterator`

`Iterator`

- Pour itérateur bidirectionnel utilisable avec constructeur `foreach`.
- Etend `Traversable` (interface "interne" ne pouvant être implémentée seule).

interface-iterator.php

```php
interface Iterator extends Traversable {
    public function current(); // valeur de l'élément courant
    public function key(); // clé de l'élément courant
    public function next(); // passe à l'élément suivant
    public function rewind(); // revient sur le 1er élément
    public function valid(); // TRUE si on n'est pas à la fin
}
```

Implémentation et usage avec `foreach`

interface-iterator-use.php

```php
class myIterator implements Iterator {
        private $position = 0;
        private $array = array("first","second","last");
        public function __construct() { $this->position = 0; }
        function current() { var_dump(__METHOD__);
                return $this->array[$this->position];
        }
        function key() { var_dump(__METHOD__);
                return $this->position;
        }
        function next() { var_dump(__METHOD__);
                ++$this->position;
        }
        function rewind() { var_dump(__METHOD__);
                $this->position = 0;
        }
        function valid() { var_dump(__METHOD__);
                return isset($this->array[$this->position]);
        }
}
$it = new myIterator();
foreach($it as $key => $value) { var_dump($key, $value); }
```

L'interface `ArrayAccess`

Permet de manipuler comme un tableau un objet encapsulant un conteneur, eg. accès via `$obj[2]`.

interface-arrayaccess.php

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

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 l'élément à l'indice donné.

L'interface `ArrayAccess`

interface-arrayaccess-use.php

```php
class Obj implements ArrayAccess {
   private $cnr = array();
   public function __construct() {
      $this->cnr = array("un"=>1,"deux"=>2);
   }
   public function offsetExists($offset) {
      return isset($this->cnr[$offset]);
   }
   public function offsetGet($offset) {
      return isset($this->cnr[$offset]) ? $this->cnr[$offset] : null;
   }
   public function offsetSet($offset, $value) {
      if (is_null($offset)) {
         $this->cnr[] = $value;
      } else {
         $this->cnr[$offset] = $value;
      }
   }
   public function offsetUnset($offset) {
      unset($this->cnr[$offset]);
   }
}
$obj = new Obj();
var_dump($obj["deux"]);
$obj["deux"] = "Une valeur";
$obj[] = 'Ajout 1';///<=>$obj[0]<=>$obj["0"]
print_r($obj);
```

La classe `ArrayIterator` (SPL)

Permet d'itérer sur un tableau ou un objet et d'en modifier ou supprimer les éléments.

class-arrayiterator.php

```php
$tab=range('a','c');
$itr=new ArrayIterator($tab);
$itr[0]='z';
unset($itr[1]);
// z c
foreach ($itr as $key=>$val) { $itr[$key]=strtoupper($val); }
foreach ($itr as $val) { echo $val."\n"; }
```

L'interface `IteratorAggregate`

Permet à un objet conteneur de fournir un itérateur.

interface-iteratoraggregate.php

```php
interface IteratorAggregate extends Traversable {
    public function getIterator();
}
```

interface-iteratoraggregate-use.php

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

Countable et LimitIterator (SPL)

Interface Countable

Permet de connaître le nombre d'éléments d'un objet.

interface-countable.php

```php
Interface Countable { public function count(); }
```

Classe LimitIterator

Permet de générer un itérateur sur un tableau dont on fixe la plage de valeurs.

class-limititerator.php

```php
$tab=range('c','r');
$itr=new ArrayIterator($tab);
$limit=new LimitIterator($itr,3,2);
//fg
foreach ($limit as $val) { echo $val."\n"; }
```

La classe `AppendIterator` (SPL)

Permet de générer un itérateur sur plusieurs tableaux.

class-appenditerator.php

```php
$itr1=new ArrayIterator(range(1,5));
$itr2=new ArrayIterator(range(10,15));
$itr=new AppendIterator();
$itr->append($itr1);
$itr->append($itr2);
// 1 2 3 4 5 10 11 12 13 14 15
foreach ($itr as $value) {
    echo $value." ";
}
```

La classe `FilterIterator` (SPL)

Permet de filtrer les valeurs en redéfinissant la méthode `accept`.

class-filteriterator.php

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

La classe `RegexIterator` (SPL)

Permet de filtrer les valeurs en utilisant une expression régulière.

class-regexiterator.php

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

La classe `IteratorIterator` (SPL)

Permet de créer un itérateur sur toute classe implémentant uniquement `Traversable` (wrapper).
Utilisée notamment avec PDO.

class-iteratoriterator.php

```php
$pdoStatement=$db->query('SELECT * FROM table');
$itr=new IteratorIterator($pdoStatement);
$limit=new LimitIterator($itr,0,10);
print_r(iterator_to_array($limit));
```

Arbre de la classe SPL Iterators

- ArrayIterator
  - RecursiveArrayIterator
- EmptyIterator
- IteratorIterator
  - AppendIterator
  - CachingIterator
    - RecursiveCachingIterator
  - FilterIterator
    - CallbackFilterIterator
      - RecursiveCallbackFilterIterator
    - RecursiveFilterIterator
      - ParentIterator
    - **RegexIterator**
      - RecursiveRegexIterator
  - InfiniteIterator
  - LimitIterator
  - NoRewindIterator
- MultipleIterator
- RecursiveIteratorIterator
  - RecursiveTreeIterator
- DirectoryIterator (extends SplFileInfo)
  - FilesystemIterator
    - GlobIterator
    - RecursiveDirectoryIterator

Fonctions agissant sur les itérateurs

`iterator_apply(iterator,callback)`

Applique une fonction de rappel sur chaque élément parcouru (à la `array_map`).

`iterator_count(iterator)`

Compte le nombre d'éléments de l'itérateur.

`iterator_to_array(iterator)`

Retourne un tableau des éléments de l'itérateur.

Exemple

Exemple (iterator-to-array.php)

```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);
/**/
$it=new InfiniteIterator($iterator);
echo $it->current()."\n";
$it->next();
echo $it->current()."\n";
$it->next();
echo $it->current()."\n";
$it->next();
echo $it->current()."\n";
$it->next();echo $it->current()."\n";
$it->next();
echo $it->current()."\n";
$it->next();
echo $it->current()."\n";
$it->next();
```

Exemple

Résultat

```
ArrayIterator Object
(
    [storage:ArrayIterator:private] => Array
        (
            [0] => 1
            [1] => 2
            [2] => 3
        )

)
il y a 3 elements
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
123123
```

La classe `ArrayObject` (SPL)

Permet de gérer un tableau sous forme d'objet.

class-arrayobject.php

```php
class ArrayObject implements
  IteratorAggregate, Traversable,
  ArrayAccess, Countable   {
//...
}
```

class-arrayobject-use.php

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

La classe `ArrayObject`

Résultat

```
ArrayIterator Object
(
    [storage:ArrayIterator:private] => ArrayObject Object
        (
            [storage:ArrayObject:private] => Array
                (
                    [0] => 2
                    [1] => 3
                    [2] => a
                    [3] => 5
                    [4] => hello
                    [color] => red
                )

        )

)
```

Classes d'exceptions de SPL

`BadFunctionCallException`	Fonction ou arguments manquants.
`BadMethodCallException`	Méthode ou arguments manquants.
`DomainException`	Une valeur n'est pas du domaine.
`InvalidArgumentException`	Un argument n'est pas du type attendu.
`LengthException`	Une taille est invalide.
`LogicException`	Erreur dans la logique du programme.
`OutOfBoundsException`	Clé d'accès invalide.
`OutofRangeException`	Clé d'accès invalide.
`RangeException`	Une valeur est hors de la plage attendue.
`RuntimeException`	Erreur rencontrée à l'exécution.
`OverflowException`	Ajout à un conteneur plein.
`UnderflowException`	Suppression dans un conteneur vide.
`UnexpectedValueException`	Une valeur retour ne fait pas partie d'une liste attendue.