

Piece Network 1

# SPL入門

- SPLで学ぶPHP5のオブジェクト指向 -

関山隆介 <[rsky0711@gmail.com](mailto:rsky0711@gmail.com)>

<http://d.hatena.ne.jp/rsky/>

# SPLとは

- ✿ Standard PHP Library
- ✿ PHP5から導入された標準クラスライブラリ
  - ✿ 各種イテレータ
  - ✿ ファイル、ディレクトリ
  - ✿ 例外クラス

# SPLの基盤

- ✿ PHP5 (Zend Engine 2) で強化されたオブジェクト指向機能
- ✿ インターフェイスとイテレータをよく使っている
- ✿ メソッド実装の強制、APIの共通化
- ✿ foreachによる反復処理

# 内容

- I. SPL関数紹介
- II. SPL例外クラス紹介
- III. SPLインターフェイス紹介
- IV. SPLクラス紹介&使用例
- V. まとめ

# SPL関数

# SPL関数

## \_\_autoload系

<b>spl_autoload</b>	__autoload()のデフォルト実装
<b>spl_autoload_extensions</b>	spl_autoload()用の拡張子を登録する
<b>spl_autoload_register</b>	指定した関数を__autoload()の実装として登録する
<b>spl_autoload_unregister</b>	指定した関数の__autoload()の実装としての登録を解除する
<b>spl_autoload_functions</b>	登録済み__autoload()関数を返す
<b>spl_autoload_call</b>	要求されたクラスを読み込むために、すべての登録済み__autoload()関数を試す

# SPL関数

## イテレータ系

<b>iterator_to_array</b>	イテレータを配列にコピーする ArrayIteratorの逆
<b>iterator_count</b>	イテレータにある要素をカウントする Countableを実装していないイテレータ用
<b>iterator_apply</b>	イテレータの要素の数だけ関数をコールする 各要素はコールバック関数に渡されず、引数はオプションで指定する コールバック関数がfalseを返したら反復停止 戻り値は関数をコールした回数

# SPL関数

## ユーティリティ系

<b>spl_classes</b>	利用可能なSPLクラス名を配列で返す
<b>class_parents</b>	クラスまたはオブジェクトの親クラス名を配列で返す
<b>class_implements</b>	クラスまたはオブジェクトが実装しているインターフェイス名を配列で返す
<b>spl_object_hash</b>	オブジェクトの一意な識別子を返す オブジェクトを要素とする連想配列のキーに

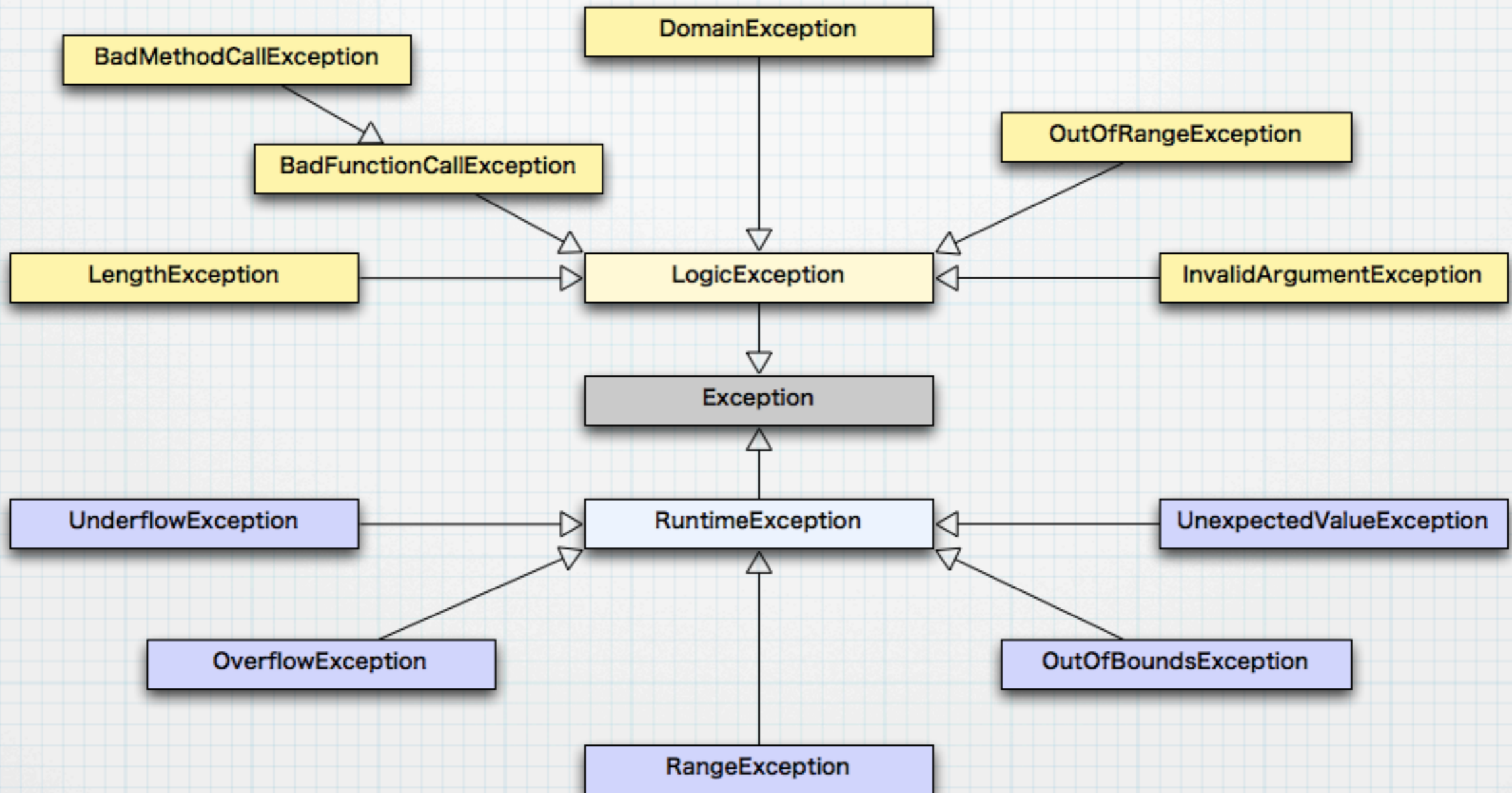
# SPL例外クラス

# PHP5の例外処理

- ✿ try-catch
- ✿ catchは好きなだけ書ける
- ✿ finallyはない
- ✿ 通常の処理とエラー処理を分離できる
- ✿ 例外の種類に応じて処理を個別に書ける

```
try {  
    if ($foo != "Foo" ) {  
        throw new FooException( "Not Foo!" );  
    }  
    if ($bar != "Bar" ) {  
        throw new BarException( "Not Bar!" );  
    }  
    if ($bar != "Baz" ) {  
        throw new BazException( "Not Baz!" );  
    }  
} catch (FooException $e) {  
    echo $e->getMessage(), "\n" ;  
} catch (BarException $e) {  
    echo $e->getMessage(), "\n" ;  
    whenNotBarCallback($e->getCode());  
} catch (BazException $e) {  
    echo $e->getMessage(), "\n" ;  
    exit($e->getCode());  
}  
  
echo "Maybe OK!\n" ;
```

# SPL例外クラス継承図



# SPL例外クラス

## LogicException系

<b>LogicException</b>	プログラムのロジックに問題があり コードの修正が必要な類の例外
<b>BadFunctionCallException</b>	不正な関数呼び出し
<b>BadMethodCallException</b>	不正なメソッド呼び出し
<b>InvalidArgumentException</b>	不正な引数
<b>DomainException</b>	不正な範囲の数値
<b>LengthException</b>	不正な大きさの値 (文字列、ファイル、配列など)
<b>OutOfRangeException</b>	不正なインデックス

# SPL例外クラス

## RuntimeException系

<b>RuntimeException</b>	実行時にのみ判定できる類の例外
<b>OverflowException</b>	数値orバッファオーバーフロー
<b>UnderflowException</b>	数値orバッファアンダーフロー
<b>UnexpectedValueException</b>	予期しない値 (InvalidArgumentExceptionのRuntime版)
<b>RangeException</b>	不正な範囲の数値 (DomainExceptionのRuntime版)
<b>OutOfBoundsException</b>	不正なインデックス (OutOfRangeExceptionのRuntime版)

# SPLインターフェイス

# PHP5組み込みインターフェイス Traversable

- ✿ 通常のオブジェクトをforeachで使うと、プロパティに順番にアクセスする
- ✿ Traversableを実装するオブジェクトはどのように反復されるかを決められる
- ✿ 実際にはIteratorかIteratorAggregateのどちらかを実装する
- ✿ SimpleXMLElementのようにIteratorでないけれどTraversableなクラスもある

# PHP5組み込みインターフェイス Iterator

- ✿ Iteratorを実装するクラスはメソッド  
current, key, next, rewind, validを実装  
しなければならない
- ✿ foreachで使うとこれらのメソッドが自  
動で呼ばれる

# PHP5組み込みインターフェイス

## IteratorAggregate

- ✿ IteratorAggregateを実装するクラスはメソッドgetIteratorを実装し、getIteratorはIteratorを実装したオブジェクトを返さなければならない
- ✿ foreachで使うとgetIteratorが自動で呼ばれ、getIteratorが返したIteratorについて反復処理する

# PHP5組み込みインターフェイス ArrayAccess

- ✿ ArrayAccessを実装するオブジェクトは配列のように [] (角括弧) でアクセスできる
- ✿ ArrayAccessを実装するクラスはメソッド offsetExists, offsetGet, offsetSet, offsetUnsetを実装しなければならない

# PHP5組み込みインターフェイス Serializable

- ✿ Serializableを実装するオブジェクトはシリアライズ/アンシリアライズされる際のフォーマットを独自のものにできる
- ✿ Serializableを実装するクラスはメソッド `serialize`, `unserialize`を実装しなければならない
- ✿ マジックメソッド `__wakeup`, `__sleep`は無効になる

# SPLインターフェイス Countable

- ✿ Countableを実装するオブジェクトは配列のようにcount関数で要素の数などを取得することができる
- ✿ Countableを実装するクラスはメソッドcountを実装しなければならない
- ✿ 組み込みインターフェイスのような特殊なポジション。SPLなのは言語仕様に深く関係しないから？

# SPLインターフェイス

# SeekableIterator

- ✿ Iteratorを継承し、要素へのアクセス性を高めたインターフェイス
- ✿ SeekableIteratorを実装するオブジェクトはseekメソッドでイテレータの位置を移動して要素にアクセスできる

# SPLインターフェイス

## RecursiveIterator

- ✿ Iteratorを継承した、再帰的なデータ構造のためのインターフェイス
- ✿ RecursiveIteratorを実装するオブジェクトはhasChildrenメソッドで現在の要素が子要素を持つかどうかを調べ、getChildrenメソッドで子要素（一般には同じクラスのオブジェクト）を取得できる

# SPLインターフェイス OuterIterator

- ✿ Iteratorを継承したインターフェイス
- ✿ SPLのOuterIteratorを実装したクラスはプロパティとしてイテレータを持ち、getInnerIteratorメソッドでそのイテレータを返している
- ✿ Iterator系クラスにIteratorAggregateのgetIteratorと同等のメソッドを実装させるためのインターフェイスと言える

# “OuterIterator” という 名前は良くないような...

- ◆ 繰り返しの「記述」でなく「対象」がオブジェクトの外部にある
- ◆ プログラミング用語の「外部イテレータ」とは意味が異なる
- ◆ そもそもIterator自体が内部イテレータとしても使える外部イテレータである
- ♥ でもOuterIteratorを実装したRecursiveIteratorとか超便利

## SPLインターフェイス

# SplObserver, SplSubject

- ✿ デザインパターン、Observerパターンのためのインターフェイス
- ✿ SplSubjectを実装するクラスはnotifyメソッドで自身(\$this)を引数としてattachされた全てのSplObserver達のupdateメソッドを呼び出す
- ✿ Subjectに登録されたObserverの管理にはSPL関数spl\_object\_hashで得られる値をキーとした連想配列が適している

# SPLクラス

## SPLクラス

# IteratorIterator

- ✿ Iterator以外のTraversableなオブジェクトをIteratorと同じように扱うためのクラスで、OuterIteratorを実装
- ✿ Iterator, IteratorAggregate以外のTraversableなクラスは拡張モジュールでしか定義できず、数も少ないので使う機会は少ないと思われる

## SPLクラス

# RecursiveIteratorIterator

- ✿ RecursiveIteratorをフラットなIteratorのように反復処理するためのクラスで、OuterIteratorを実装
- ✿ 再帰処理を書くことなくRecursiveIteratorの子要素や孫要素をたどることができ、非常に便利
- ✿ コンストラクタはRecursiveIteratorかIteratorAggregateしか受け付けない

(注) IteratorAggregate::getIterator()がRecursiveIterator以外を返した場合はInvalidArgumentExceptionが発生する

# SimpleXMLElement + IteratorIterator

```
<?php
$string = <<<XML
<?xml version='1.0'?>
<document>
  <title>Forty What?</title>
  <from>Joe</from>
  <to>Jane</to>
  <body>I know that's the answer -- but what's the question?
  </body>
</document>
XML;

// SimpleXMLElementはIteratorではないがTraversableなのでforeach文で使える
$xml = new SimpleXMLElement($string);
foreach ($xml as $tag => $content) {
    printf("%s: %s\n", $tag, $content);
}

// IteratorIteratorを使うとIteratorを引数に取る関数・メソッドで使えるようになる
$iter = new IteratorIterator($xml);
var_dump(iterator_count($iter));
print_r(iterator_to_array($iter));

// 後で紹介するSimpleXMLIteratorがあるのでSimpleXMLElementを
// IteratorIteratorと組み合わせる必要は無かったりする...
```

# SPLクラス

# SplFileInfo

- ✿ SPLのファイルシステム系クラスの基底クラス
- ✿ 各種ファイル情報にアクセスするためのメソッドを実装

# SplFileInfo使用例

```
<?php
// SplFileInfoを生成
$fi = new SplFileInfo(__FILE__);

// ファイル情報を取得する例。行末のコメントは同じ働きをする関数
echo $fi->getRealPath(), "\n"; // realpath()
echo '-r ? ' ; var_dump($fi->isReadable()); // is_readable()
echo '-w ? ' ; var_dump($fi->isWritable()); // is_writable()
echo '-x ? ' ; var_dump($fi->isExecutable()); // is_executable()
echo '-f ? ' ; var_dump($fi->isFile()); // is_file()
echo '-d ? ' ; var_dump($fi->isDir()); // is_dir()
echo '-l ? ' ; var_dump($fi->isLink()); // is_link()
echo 'owner: ', $fi->getOwner(), "\n"; // stat()
echo 'group: ', $fi->getGroup(), "\n"; // stat()
echo 'mtime: ', date('Y-m-d H:i:s', $fi->getMTime()), "\n"; // filemtime()
```

# SPLクラス

## SplFileObject

- ✿ SplFileInfoのサブクラスで、ファイルを読み書きするためのメソッドを実装
- ✿ RecursiveIteratorとSeekableIteratoを実装しており、各行に順番にアクセスしたりもできる
- ✿ 一時ファイルを扱うためのサブクラスのSplTempFileObjectもある

# SplFileObject使用例

```
<?php
// SplFileObjectを生成
$fo = new SplFileObject('sample.txt', 'w+');

// ロックして書き込む
$fo->flock(LOCK_EX);
$fo->fwrite("1,orange\n");
$fo->fwrite("2,lemon\n");
$fo->fwrite("3,apple\n");
$fo->flock(LOCK_UN);

// 読み込みモードか読み書き両用で開かれたSplFileObjectは
// file関数で得た配列のようにforeachで各行にアクセスできる
// (書き込みモードや追記モードでは固まるので注意!)
foreach ($fo as $line) {
    echo $line;
}

// CSVとして読み込むこともできる
$fo->setFlags(SplFileObject::READ_CSV);
foreach ($fo as $row) {
    print_r($row);
}

-- このようにも書ける
$fo->seek(0);
while (!$fo->eof()) {
    print_r($fo->fgetcsv());
}
```

## SPLクラス

# DirectoryIterator

- ✿ ディレクトリの中身について反復処理するためのSplFileInfoのサブクラスで、Iteratorを実装
- ✿ keyメソッドはパス、currentメソッドはSplFileInfoオブジェクトを返す
- ✿ RecursiveIteratorを実装したサブクラスのRecursiveDirectoryIteratorもある

# 従来のコード

```
<?php
// ディレクトリハンドルを使ってディレクトリを再帰的にたどる関数
function findConfigPhp($path)
{
    $dir = opendir($path);

    while (false !== ($entry = readdir($dir))) {
        if ($entry == '.' || $entry == '..') {
            continue;
        }

        $newpath = $path . DIRECTORY_SEPARATOR . $entry;
        if (is_dir($newpath)) {
            findConfigPhp($newpath);
        } elseif (is_file($newpath) && basename($newpath) == 'Config.php') {
            echo realpath($newpath), "\n";
        }
    }

    closedir($dir);
}

// /usr/local/share/pear/pieceからConfig.phpを検索
findConfigPhp('/usr/local/share/pear/piece');
```

# RecursiveDirectoryIteratorを使うと

```
<?php
// RecursiveDirectoryIteratorを使ってディレクトリを再帰的にたどる関数
function findConfigPhp(RecursiveDirectoryIterator $iter)
{
    // (Recursive)DirectoryIteratorをforeachで使うと
    // 自分自身のポインタも進めながら同じファイル/ディレクトリを
    // 指すSplFileInfoオブジェクトを$entryに代入する
    foreach ($iter as $entry) {
        if ($iter->isDot()) {
            continue;
        } elseif ($iter->isDir() && $iter->hasChildren()) {
            findConfigPhp($iter->getChildren());
        } elseif ($iter->isFile() && $iter->getBasename() == 'Config.php') {
            echo $iter->getRealPath(), "\n";
        }
    }
}

// RecursiveDirectoryIteratorを生成、Config.phpを検索
findConfigPhp(new RecursiveDirectoryIterator('/usr/local/share/pear/piece'));
```

# RecursiveIteratorIteratorを使うと

```
<?php
// RecursiveIteratorIteratorを使うとRecursiveIteratorの
// getChildren()を自動で呼んでくれるので再帰を意識せずに書ける
// ただし、 hasChildren()がtrueを返した要素は無視されるので
// 目的によってはRecursiveIteratorIterator+foreachが適さないこともある
$diter = new RecursiveDirectoryIterator('/usr/local/share/pear/piece');
$riter = new RecursiveIteratorIterator($diter);

// $entryに代入されるのはRecursiveIteratorが返した値
foreach ($riter as $entry) {
    if ($entry->isFile() && $entry->getBasename() == 'Config.php') {
        echo $entry->getRealPath(), "\n";
    }
}
```

## SPLクラス

# ArrayObject, ArrayIterator

- ✿ ともにArrayAccess, Countableを実装
- ✿ ArrayObjectはIteratorAggregateを、ArrayIteratorはSeekableIteratorを実装
- ✿ ArrayObjectはArrayIteratorを集約する
- ✿ SPLのインターフェイスを使った配列オブジェクトの実装
- ✿ Iteratorを引数にとるSPLクラスと配列の橋渡しの役割

# ArrayObject使用例 (ArrayIteratorも同様)

```
<?php
// ArrayObjectを生成
$ao = new ArrayObject(array('orange', 'lemon', 'apple'));

// IteratorAggregateを実装しているのでforeachできる
foreach ($ao as $key => $value) {
    printf("%d: %s\n", $key, $value);
}

// ArrayObjectの実体はArrayIterator
echo $ao->getIteratorClass(), "\n";

// countableを実装しているのでcount()できる ($ao->count() でも同じ)
echo count($ao), "\n";

// ArrayAccessを実装しているので[]で要素を操作できる
echo $ao[1], "\n";
$ao[2] = 'peach';
unset($ao[0]);
$ao[] = 'grape';

// ソートしてもう一度foreach
$ao->asort();
foreach ($ao as $key => $value) {
    printf("%d: %s\n", $key, $value);
}
```

## SPLクラス

# RecursiveArrayIterator

- ✿ RecursiveIteratorを実装したArrayIteratorのサブクラス
- ✿ RecursiveIteratorと組み合わせて使うことが多いと思われる
- ✿ ArrayObject::setIteratorClassメソッドでこのクラス名を指定するとArrayObjectとRecursiveIteratorを組み合わせたことができる

# RecursiveArrayIterator使用例

```
<?php
// 一次元配列のArrayObjectをforeach
$ao = new ArrayObject(array(1, 2, 3));
foreach ($ao as $value) {
    var_dump($value);
}
echo "--\n";

// 多次元配列にしてforeach
$ao->exchangeArray(array(1, array(2, 3), array(4, 5, array(6, 7, array(8, 9))));
foreach ($ao as $value) {
    var_dump($value);
}
echo "--\n";

// RecursiveIteratorとしてforeach
$ao->setIteratorClass('RecursiveArrayIterator');
foreach (new RecursiveIteratorIterator($ao) as $value) {
    var_dump($value);
}
```

## SPLクラス

# SimpleXMLIterator

- ✿ SimpleXMLElementのサブクラスで、RecursiveIterator, Countableを実装
- ✿ RecursiveIteratorと組み合わせてforeachで子要素をたどることができる
- ✿ SimpleXML関数simplexml\_import\_dom, simplexml\_load\_{file,string}の第2引数に指定することもできる

# SimpleXMLIterator使用例

```
<?php
$string = <<<XML
<?xml version='1.0'?>
<documents>
  <document>
    <title>Forty What?</title>
    <from>Joe</from>
    <to>Jane</to>
    <body>I know that's the answer -- but what's the question?
    </body>
  </document>
  <document>
    <title>Re: Forty What?</title>
    <from>Jane</from>
    <to>Joe</to>
    <body>I know that's the question -- but what's the answer?
    </body>
  </document>
</documents>
XML;

// RecursiveIteratorIteratorと組み合わせて子要素を再帰的に表示する
$xml = new SimpleXMLIterator($string);
foreach (new RecursiveIteratorIterator($xml) as $tag => $content) {
    printf("%s: %s\n", $tag, $content);
}
```

# SPLクラス

## FilterIterator

- ✿ IteratorIteratorのサブクラスで、Iteratorの各要素にフィルタを適用した値を取得するための抽象クラス
- ✿ このクラスのサブクラスはIteratorのcurrentメソッドが返す値を検証し、真偽値を返すacceptメソッドを実装する
- ✿ RecursiveIteratorを実装したサブクラスのRecursiveFilterIteratorもある

# SPLクラス

## RegexIterator

- ✿ Iteratorから正規表現にマッチする要素だけを取り出したり、各要素を正規表現で置換して取得する等の処理ができる  
FilterIteratorのサブクラス
- ✿ RecursiveFilterIterator版の  
RecursiveRegexIteratorもある

# 従来のコード

```
<?php
// ファイルの各行を正規表現で処理する
$file = file('regex-sample.txt');

// don't like でない like を含む行を抽出
function filter($l) {
    return (bool)preg_match('/(?<!don\'t )like /', $l);
}
foreach (array_filter($file, 'filter') as $like) {
    echo $like;
}
echo "--\n";

// don't like を like に置換して表示
foreach ($file as $line) {
    echo preg_replace('/ don\'t like /', ' like ', $line);
}
echo "--\n";

// スペースを含む行をスペース区切りで配列にする
foreach ($file as $line) {
    if (count($words = preg_split('/ /', $line))) {
        print_r($words);
    }
}
}
```

# RegexIteratorを使うと

```
<?php
// ファイルの各行を正規表現で処理する
$file = new SplFileObject('regex-sample.txt', 'r');

// don't like でない like を含む行を抽出
foreach (new RegexIterator($file, '/(?<!don\'t )like /') as $like) {
    echo $like;
}
echo "--\n";

// don't like を like に置換して表示
// PHP 5.2.5ではext/spl/spl_iterators.cのバグにより何も起こらない
$repl = new RegexIterator($file, '/ don\'t like /', RegexIterator::REPLACE);
$repl->replacement = ' like ';
foreach ($repl as $line) {
    echo $line;
}
echo "--\n";

// スペースを含む行をスペース区切りで配列にする
foreach (new RegexIterator($file, '/ /', RegexIterator::SPLIT) as $words) {
    print_r($words);
}
```

# SPLクラス

## SplObjectStorage

- ✿ Countable, Iterator, Serializableを実装したオブジェクトを効率良く保管するためのクラス
- ✿ attachされたオブジェクトは一意的なキーで登録され、containsメソッドで登録されているか調べることができる
- ✿ 同じオブジェクトは常に一つしか登録されない

## SPLクラス

# その他のIterator (1)

- ✿ **EmptyIterator** implements Iterator
  - ✿ 空っぽのIterator。ダミー用？
- ✿ **AppendIterator** extends IteratorIterator
  - ✿ 複数のIteratorの要素に連続してアクセス
  - ✿ コンストラクタは引数をとらず、appendメソッドでIteratorを追加する

## SPLクラス

# その他のIterator (2)

- ✿ **LimitIterator** extends IteratorIterator
  - ✿ 開始位置と終了位置を指定できる
- ✿ **InfiniteIterator** extends IteratorIterator
  - ✿ 無限に反復。最後まで進んだら、最初から
- ✿ **NoRewindIterator** extends IteratorIterator
  - ✿ rewindメソッドは何もしない

# SPLクラス

## その他のIterator (3)

- ✿ **ParentIterator** extends RecursiveFilterIterator
  - ✿ RecursiveIteratorから子要素のある  
(hasChildrenメソッドがtrueを返す)要素を抽出する
- ✿ **CachingIterator** extends OuterIterator implements  
ArrayAccess, Countable
  - ✿ hasNextメソッドで次の要素があるか調べることができる等の特徴がある
- ✿ **RecursiveCachingIterator** extends  
CachingIterator implements RecursiveIterator

# まとめ

- ✿ SPLはこれまでできなかったことができるようになるようなものではない
- ✿ ArrayObject系クラスやSplFileInfo系クラスはstandardモジュールの関数をオブジェクト指向でまとめ直したものの
- ✿ PHP5時代のオブジェクト指向プログラミングへの移行を手助けするツール
- ✿ <http://www.php.net/~helly/php/ext/>