——高性能PHP应用开发之

第1讲 面向对象深度剖析

目录

CONTENTS

开发前的准备

■ 准备工作

✓ PHP运行环境：PHP7，Composer，环境变量

✓ 开发工具：PHPStorm、Sublime Text、Visual Studio Code

✓ 类的自动加载机制：

- 传统加载类的方法：require 和 require_once
- 自动加载类：__autoload( )
- SPL自动加载：spl_autoload_register( )
- Composer自动加载

✓ 命名空间

✓ 编码规范：PSR规范

目录

CONTENTS

PHP中的内存分区

内存分区

- ✓ **栈区**：编译器**自动分配和管理**的内存单元，**基本数据类型**（数值型、字符串型、bool值）变量存储在该区域，**对象类型（数组、对象）中某一部分**也存储在该内存单元。
- ✓ **堆区**：由**程序员手工管理**的内存单元，对象类型的真实存储位置。
- ✓ **全局静态区**：全局变量和静态变量（属性）的存储位置。
- ✓ 文字常量区：常量、类常量、字面值的存储位置。
- ✓ 代码区：程序代码的存储位置。

栈区

- 基本数据类型

✓ 数值型数据、字符串、bool值

```php
// 数值型数据
$num1 = 25;
$num2 = 3.14;
// 字符串数据
$str = "hello";
// bool值
$flag = true;
```

✓ 数组类型：PHP中数组类型按照哈希表结构存储，也存储在栈区。

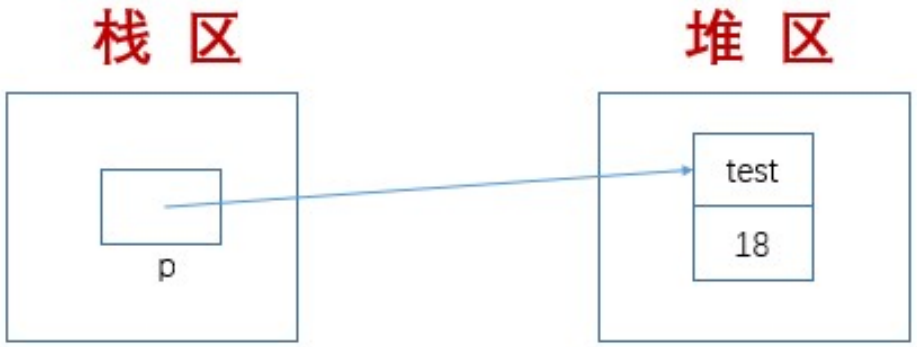
✓ 资源类型（数据库连接资源）：变量名存储在栈区，资源内容存储在堆区。

✓ 对象类型：对象名存储在栈区，对象实际内容存储在堆区。

堆区

- 堆区：由程序员手工管理的内存空间，当程序结束（一次HTTP请求执行完成）时，会自动销毁该段内存空间；程序员也可以手工删除该段内存空间中的内容。

```php
class Person {
    private $_name;
    private $_age;
    public function __construct($name, $age) {
    }
}
$p = new Person('test', 18);
```

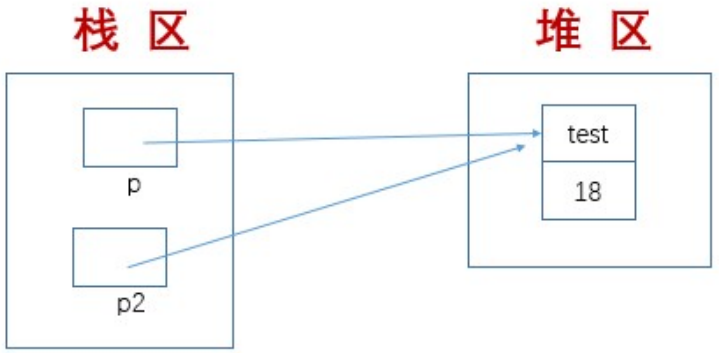


对象复制

■ **对象赋值**

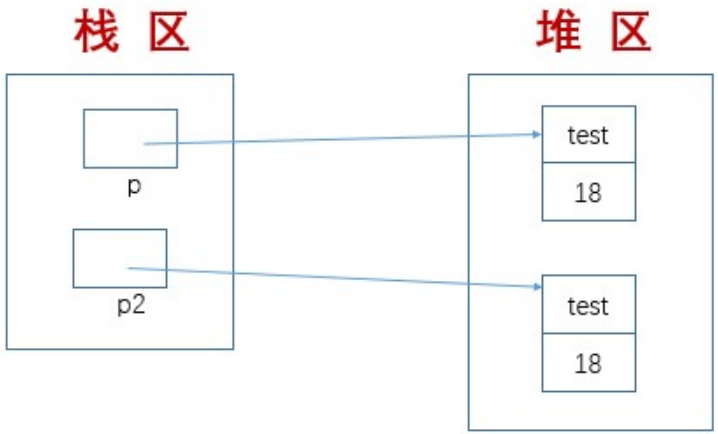
✓ 简单对象赋值：共享堆区同一块内存。

```
$p = new Person('test', 18);
$p2 = $p;
```



✓ **clone**复制：在堆区建立不同的内存区域。

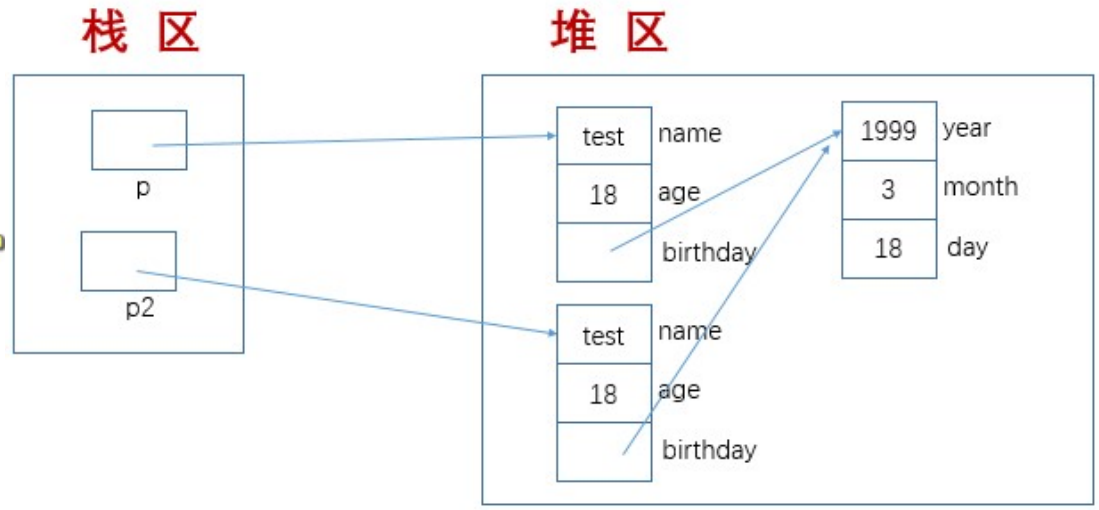
```
$p = new Person('test', 18);
$p2 = clone $p;
```



对象复制

- **浅拷贝**：使用clone复制时，只针对简单对象有效；对于复杂对象（“has-a”类型对象），只是较浅层次的拷贝。

```php
class Person {
    private $_name;
    private $_age;
    private $_birthday;
    public function __construct($name, $age, $birthday) {
    }
}
$p = new Person('test', 18, new Birthday(1999,1,1));
$p2 = clone $p;
```

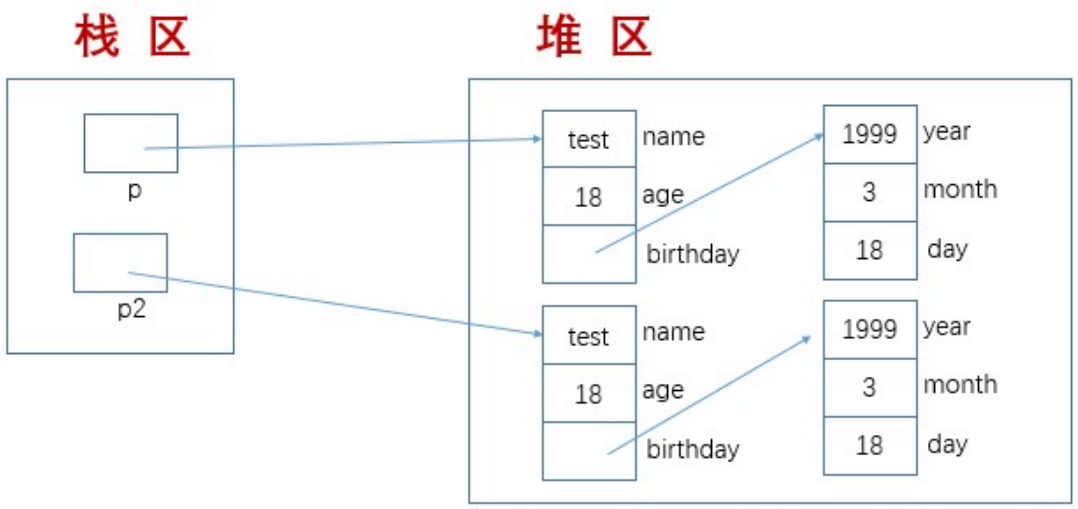


对象复制

- **深拷贝**：复制时，同时复制子对象数据内容。

✓ 在类中，编写魔术方法 **__clone( )**，实现子对象的拷贝。

```php
class Person {
    public $_name;
    public $_age;
    public $_birthday;
    public function __construct($name, $age, $birthday)
    }
    public function __clone() {
        $this->_birthday = clone $this->_birthday;
    }
}
$p = new Person('test', 18, new Birthday(1999,1,1));
$p2 = clone $p;
$p2->_birthday->_year = 1995;
echo $p2->_birthday->_year;
```



目录

CONTENTS

PHP面向对象

- 面向对象三大基本特征

✓ **封装**：一个类就是一个封装了数据以及操作这些数据的代码的逻辑实体。在一个对象内部，某些代码或某些数据可以是私有的，不能被外界访问。

✓ **继承**：也称之为扩展，在原有类功能的基础上，添加新的功能或新的属性。要实现继承，可以通过“继承”（Inheritance）或“组合”（Composition）来实现。

✓ **多态**：一个类实例的相同方法在不同情形有不同表现形式，虽然针对不同对象的具体操作不同，但通过一个公共的类，它们（那些操作）可以通过相同的方式予以调用。

PHP面向对象

- 面向对象五大基本原则：SOLID

✓ **单一职责原则**：Single Responsibility Principle，一个类应该有且只有一个去改变它的理由，这意味着一个类应该只有一项工作。

✓ 实例：

• 若一个类中包含了多项职责（如计算订单金额、订单支付、生成订单报表等功能），当修改某一项功能时，需要重新测试该类的整个功能以防止bug发生。

• 解决思路：把当前类分割成多个类，每个类只负责一项功能，当某一项功能发生改变时，只需要测试某一个类即可

PHP面向对象

- 面向对象五大基本原则：SOLID

✓ **开放封闭原则**：Open-Closed principle，对象或实体应该对扩展开放，对修改封闭。

✓ 实例：

• 订单支付方法中，显式使用信用卡支付（信用卡支付类提供的方法），当切换支付方式时（切换为微信支付），需要手动修改订单支付方法。

• 建立 支付接口，在订单支付方法中，直接调用支付接口的支付方法即可，后续修改时，不需要修改原始支付方法，只是传递不同支付接口对象而已。

PHP面向对象

- 面向对象五大基本原则：SOLID

✓ **里氏替换原则**：Liskov-Substituion Principle，对父类的调用同样适用于子类。

✓ 实例：

• User类包含有修改密码、修改手机号、修改用户角色等功能；Admin类继承User类，自动具有这些功能；而Guest类也继承User类，显然Guest类不具备修改密码、修改角色等功能，意外调用时会出现系统错误。

✓ 解决方案：重新设计 is-a 关系，只有满足里氏替换的继承才是真正的 is-a 关系。

PHP面向对象

- 面向对象五大基本原则：SOLID

✓ **接口隔离原则**：Interface-Segregation Principle，不应强迫客户端实现一个它用不上的接口，使用多个小的专门的接口，而不要使用一个大的总接口。

✓ 实例：

• 设计一个接口 Telephone，具备发送短信、打电话、录音、拍照等功能；现有两个子类：手机类（Mobile）和固定电话类（LandlinePhone），显示固定电话类不具有拍照、发送短信等功能。

• 把接口 Telephone 拆分成多个小的专门的接口。

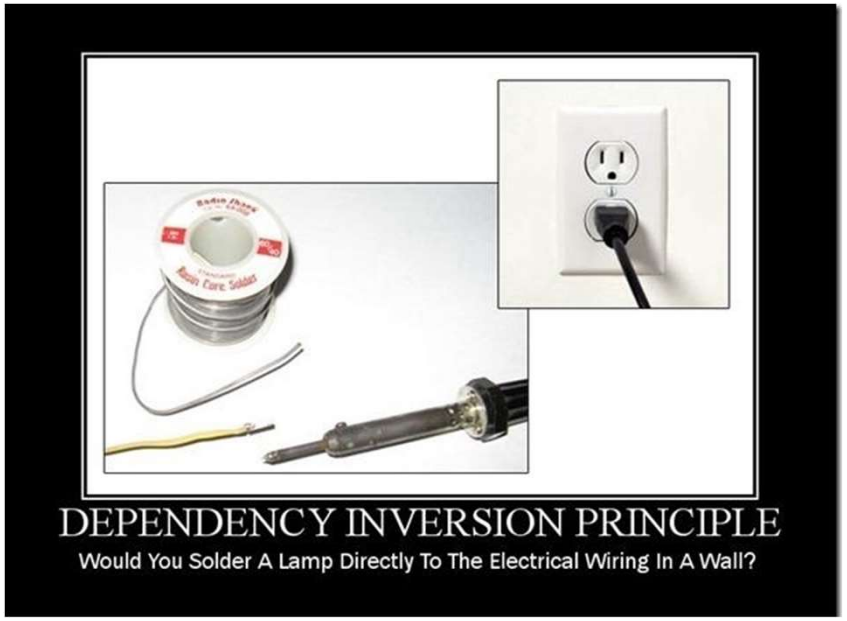
PHP面向对象

- 面向对象五大基本原则：SOLID

✓ **依赖反转原则**：Dependecy-Inversion Principle，高层次的模块不应该依赖于低层次的模块，它们都应该依赖于抽象。

✓ 实例：

- 用户管理类中，当用户修改密码后，需要发送通知消息；User类（高层）中使用 EmailNotify 类（低层）。若通知方式变更为短信发送形式，怎么办?
- 建立统一通知消息接口（抽象），高层依赖于抽象接口实现发送通知。



目录

CONTENTS

SPL简介

- SPL（**S**tandard **P**HP **L**ibrary），是为了解决典型问题而存在，实现的一些有效的数据访问**接口和类**。 它主要包括对常规**数据结构**的访问，**迭代器**，**异常处理**，**文件处理**，**数组处理**和一些**设计模式**的实现。 这些在程序设计的世界中都是一些典型的问题，以这样一种标准库的方式实现可以在很大程度上减少代码的冗余和提高开发的效率。

✓ 数据结构：SPL提供了一套标准的数据结构，这些都是在应用开发过程中的常用数据结构，如双向链表、堆、栈等。

✓ 迭代器：SPL 提供一系列迭代器以遍历不同的对象。

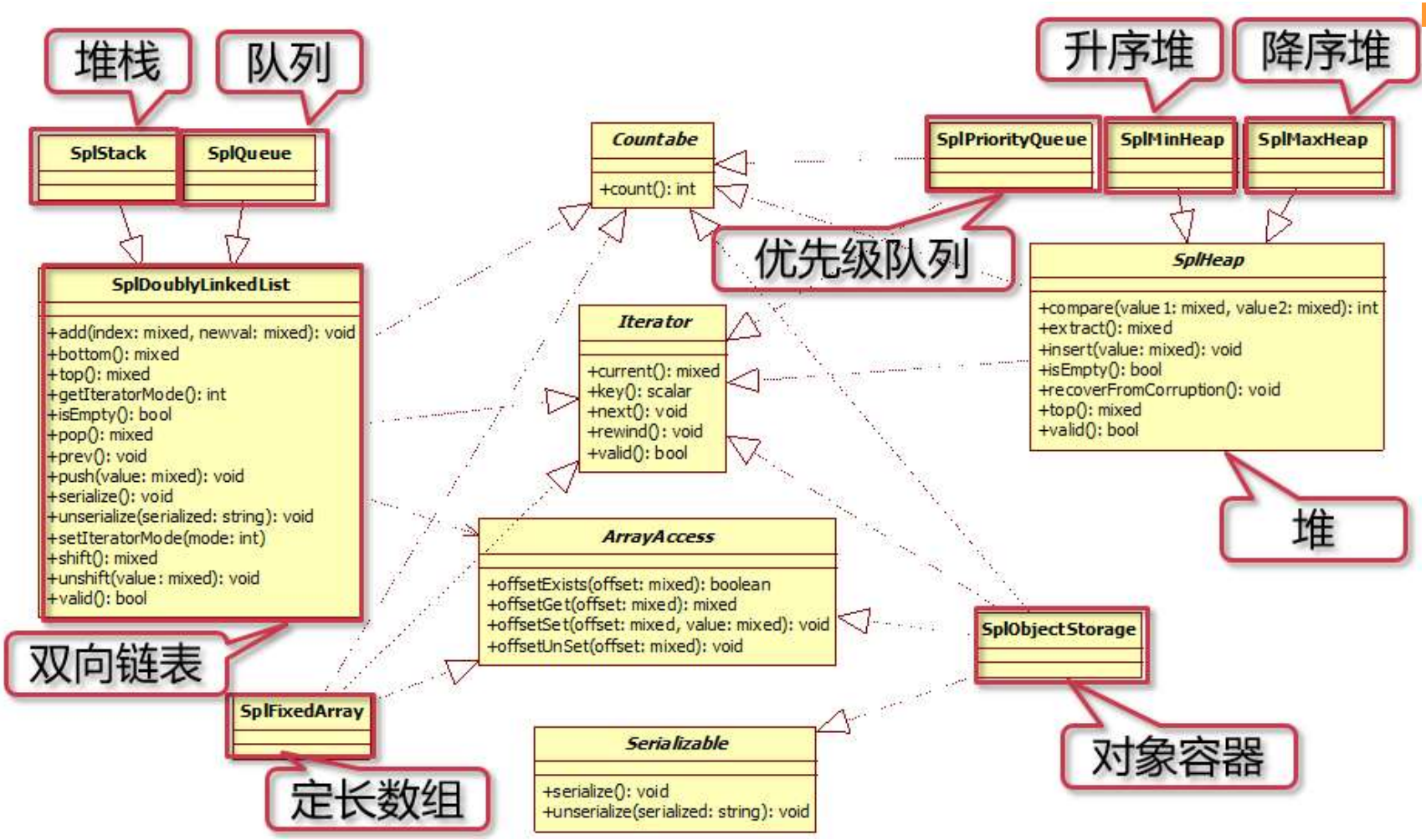
✓ 文件处理和数组处理：文件处理类、数组操作类。

✓ 标准异常处理：SPL提供了一系列的标准异常类，包括逻辑异常和运行时异常。

✓ 典型设计模式：观察者模式。

✓ SPL函数：spl_autoload_register等。

SPL数据结构


堆栈
队列
升序堆
降序堆
SplStack
SplQueue
Countabe
+count(): int
SplPriorityQueue
SplMinHeap
SplMaxHeap
优先级队列
SplDoublyLinkedList
+add(index: mixed, newval: mixed): void
+bottom(): mixed
+top(): mixed
+getIteratorMode(): int
+isEmpty(): bool
+pop(): mixed
+prev(): void
+push(value: mixed): void
+serialize(): void
+unserialize(serialized: string): void
+setIteratorMode(mode: int)
+shift(): mixed
+unshift(value: mixed): void
+valid(): bool
Iterator
+current(): mixed
+key(): scalar
+next(): void
+rewind(): void
+valid(): bool
SplHeap
+compare(value1: mixed, value2: mixed): int
+extract(): mixed
+insert(value: mixed): void
+isEmpty(): bool
+recoverFromCorruption(): void
+top(): mixed
+valid(): bool
堆
ArrayAccess
+offsetExists(offset: mixed): boolean
+offsetGet(offset: mixed): mixed
+offsetSet(offset: mixed, value: mixed): void
+offsetUnSet(offset: mixed): void
SplObjectStorage
双向链表
SplFixedArray
定长数组
Serializable
+serialize(): void
+unserialize(serialized: string): void
对象容器

SPL数据结构

```php
// spl stack
$stack = new SplStack();
$stack->push('a');
$stack->push('b');
$stack->push('c');
# the last element has index zero
$stack->offsetSet(0, 'C');
$stack->rewind();
while( $stack->valid() ) {
    echo $stack->current(), PHP_EOL;
    $stack->next();
}
```

```php
// spl queue
$q = new SplQueue();
$q->push(1);
$q->push(2);
$q->push(3);
$q->pop();
print_r($q);
```

```php
// SplDoublyLinkedList
$list = new SplDoublyLinkedList();
$list->push('a');
$list->push('b');
$list->push('c');
$list->push('d');

echo "FIFO (First In First Out) :\n";
$list->setIteratorMode(SplDoublyLinkedList::IT_MODE_FIFO);
for ($list->rewind(); $list->valid(); $list->next()) {
 echo $list->current()."\n";
}
```

SPL迭代器

- SPL提供了大量的迭代器，适用于各种应用场景，遍历不同的对象。 包括可以遍历时删除或修改元素的值或key（ArrayIterator）、空迭代器（EmptyIterator）、可以实现多迭代器遍历的MultipleIterator、文件目录迭代器等等。

```php
class RecursiveFileFilterIterator extends FilterIterator {
    // 满足条件的扩展名
    protected $ext = array('jpg','gif');
    // 提供 $path 并生成对应的目录迭代器
    public function __construct($path) {
        parent::__construct(new RecursiveIteratorIterator(
            new RecursiveDirectoryIterator($path)));
    }
    // 检查文件扩展名是否满足条件
    public function accept() {
        $item = $this->getInnerIterator();
        if ($item->isFile() &&
                in_array(pathinfo($item->getFilename(),
                    PATHINFO_EXTENSION), $this->ext)) {
            return TRUE;
        }
    }
}
// 实例化
foreach (new RecursiveFileFilterIterator('/path/to/something') as $item) {
    echo $item . PHP_EOL;
}
```

SPL文件处理

- SPL提供了内置的三个文件操作类：

✓ SplFileInfo类

✓ SplFileObject类

✓ SplTempFileObject类

```php
try{
    // iterate directly over the object
    foreach( new SplFileObject("D:\\WWW") as $line)
    // and echo each line of the file
    echo $line.'<br />';
}
catch (Exception $e) {
    echo $e->getMessage();
}
```

SPL数组处理

- ArrayObject类：可以将Array转化为object 。

```php
/*** a simple array ***/
$array = array('koala', 'kangaroo', 'wombat');

/*** create the array object ***/
$arrayObj = new ArrayObject($array);

/*** iterate over the array ***/
for($iterator = $arrayObj->getIterator();
   /*** check if valid ***/
   $iterator->valid();
   /*** move to the next array member ***/
   $iterator->next()) {
    /*** output the key and current array value ***/
    echo $iterator->key() . ' => ' . $iterator->current() . '<br />';
}
```

SPL标准异常处理

- SPL提供了一系列的标准异常类，包括逻辑异常和运行时异常。

- LogicException (extends Exception)
  - BadFunctionCallException
    - BadMethodCallException
  - DomainException
  - InvalidArgumentException
  - LengthException
  - OutOfRangeException
- RuntimeException (extends Exception)
  - OutOfBoundsException
  - OverflowException
  - RangeException
  - UnderflowException
  - UnexpectedValueException

SPL常用函数

SPL中还包括其它几个常用的spl函数。

- ✓ spl_autoload_register( )：类的自动加载机制。
- ✓ spl_classes( )：返回所有可用的spl类。
- ✓ class_parents( )：返回指定类的父类。
- ✓ class_implements( )：返回指定类所实现的所有接口。
- ✓ class_uses( )：返回当前类所使用的所有Traits。
- ✓ iterator_apply( )：为迭代器中每个元素调用一个用户自定义函数。
- ✓ iterator_count( )：计算迭代器中元素的个数。
- ✓ iterator_to_array( )：将迭代器中的元素拷贝到数组。

感谢聆听！

THANK YOU FOR YOUR ATTENTION