<http://www.php.cn/php-weizijiaocheng-372376.html>

面向对象编程（OOP)是我们编程的一项基本技能，PHP5对OOP提供了良好的支持。如何使用OOP的思想来进行PHP的高级编程，对于提高PHP编程能力和规划好Web开发构架都是非常有意义的。下面我们就通过实例来说明使用PHP的OOP进行编程的实际意义和应用方法。

　　我们通常在做一个有数据库后台的网站的时候，都会考虑到程序需要适用于不同的应用环境。和其他编程语言有所不同的是，在PHP中，操作数据库的是一系列的具体功能函数（如果你不使用ODBC接口的话）。这样做虽然效率很高，但是封装却不够。如果有一个统一的数据库接口，那么我们就可以不对程序做任何修改而适用于多种数据库，从而使程序的移植性和跨平台能力都大大提高。

**下面就来给大家介绍一下它们的区别：**

面向对象专注于由哪个对象来处理一个问题。

其最大特点是由一个一个具有属性和功能的类，从类中拿到对象，进而处理问题。

面向过程专注于解决一个问题的过程。其最大特点是由一个一个的函数去解决处理这个问题的一系列过程。

|  |
| --- |
| **一、面向对象基础** |

**一面向对象**

1、什么是类？  
具有相同属性（特征）和方法（行为）的一系列个体的集合，类是一个抽象的概念。

2、什么是对象？  
从类中，拿到的具有具体属性值的个体，称为对象。对象是一个具体的个体。  
eg:人类；张三

3、类和对象的关系？  
类是对象的抽象化！对象是类的具体化！  
类仅仅表明这类对象有哪些属性，但是不能有具体的值，所以类是抽象的。  
对象是将类的所有属性赋值后，产生具体的个体，所有对象是具体的。

**二类的声明与实例化**

1、如何声明一个类：

class 类名{  
访问修饰符 $属性[=默认值];  
[访问修饰符] function 方法(){}  
}

2、声明一个类的注意事项：  
①类名只能有字母数字下划线组成，开头不能是数字，必须符合大驼峰法则;  
②类名必须使用class修饰，类名后面一定不能有();  
③属性必须要带访问修饰符，方法可以不带访问修饰符。

3、实例化对象及对象属性方法的调用：  
$对象名 = new 类名(); //()可以不带  
  
类外部调用属性和方法：  
$对象名 -> $属性名; //使用->调用属性时，属性名不能带$符号  
  
类内部调用属性和方法：  
$this -> $属性名;

**三构造函数**

1、什么是构造函数？  
构造函数是类中的一个特殊函数，当我们使用new关键字实例化对象时，相当于调用了类的构造函数。

2、构造函数有什么作用？  
实例化对象时，自动调用，用于给对象的属性赋初值！

3、构造函数的写法：  
①构造函数名，必须与类同名  
[public] function Person($name){  
$this -> name = $name;  
}  
②使用魔术方法\_\_construct  
[public] function \_\_construct($name){  
$this -> name = $name;  
}  
4、构造函数注意事项：  
①第一种写法，构造函数名必须与类同名！！！！  
②如果一个类没有手写构造函数，则系统默认会有一个空参构造，因此可以使用new Person();  
如果我们写了带参数的构造函数，则将不会再有空参构造，也就是不能直接使用new Person();  
Person后面的()中的参数列表，必须符合构造函数的要求！！！！  
③如果两种构造函数同时存在，将使用\_\_construct。

5、析构函数：\_\_destruct():  
①析构函数在对象被销毁释放之前自动调用；  
②析构函数不能带有任何的参数；  
③析构函数常用于对象使用完以后，释放资源，关闭资源等。

6、魔术方法：  
PHP中，给我们提供一系列用\_\_开头的函数，这些函数无需自己手动调用，  
会在合适的时机自动调用，这类函数称为魔术称为魔术函数。  
eg:function \_\_construct(){} 在类new一个对象时自动调用  
function \_\_destruct(){} 在对象被销毁时自动调用

我们要求，除了魔术方法之外，自定义的函数与方法不能使用\_\_开头。

最后，一般对于功能比较复杂的类，我们会单独的写到一个类文件中。

类文件的命名，同一小写，使用"类名小写.class.php"的方式命名。  
在其他文件中使用这个类时，可以使用include导入这个".class.php"文件。

|  |
| --- |
| **二、封装和继承** |

1、什么是封装？  
通过访问修饰符，将类中不需要外部访问的属性和方法进行私有化处理，以实现访问控制。

\*注意：是实现访问控制，而不是拒绝访问。也就是说，我们私有化属性后，需要提供对应的方法，让用户通过我们提供的方法处理属性。

2、封装的作用？  
①使用者只关心类能够提供的功能，不关心功能实现的细节！（封装方法）  
②对用户的数据进行控制，防止设置不合法数据，控制返回给用户的数据（属性封装+set/get方法）

3、实现封装操作？  
①方法的封装  
对于一些只在类内部使用的方法，而不像对外部提供使用，那么，这样的方法我们可以使用private进行私有化处理。

|  |  |
| --- | --- |
| 1 | 1 private function formatName(){} //这个方法仅仅能在类内部使用$this调用2 function showName(){3 $this -> formatName();4 } |

②属性的封装+set/get方法  
为了控制属性的设置以及读取，可以将属性进行私有化处理，并要求用户通过我们提供的set/get方法进行设置

|  |  |
| --- | --- |
| 1 | 1 private $age;2 //set方法3 function setAge($age){4 $this->age=$age;5 }6 //get方法7 function getAge(){8 return $this->age;9 } |

$对象->getAge();  
$对象->setAge(12);

③属性的封装+魔术方法

|  |  |
| --- | --- |
| 1 | \_\_get(  ->   \_\_set(, ->= } |

$对象->age; //访问对象私有属性时，自动调用\_\_get()魔术方法，并且将访问的属性名传给\_\_get()方法；  
$对象->age=12; //设置对象私有属性时，自动调用\_\_set()魔术方法，并且将设置的属性名以及属性值传给\_\_set()方法；

注意：在魔术方法中，可以使用分支结构，判断$key的不同，进行不同操作。

4、关于封装的魔术方法：  
①\_\_set($key,$value):给类私有属性赋值时自动调用，调用时给方法传递两个参数：需要设置的属性名，属性值。  
②\_\_get($key,$value):读取类私有属性时自动调用，调用时给方法传递一个参数，需要读取的属性名；  
③\_\_isset($key):外部使用isset()函数检测私有属性时，自动调用。  
>>>类外部使用isset();检测私有属性，默认是检测不到的。false  
>>>所以，我们可以使用\_\_isset();函数，在自动调用时，返回内部检测结果。

|  |  |
| --- | --- |
| 1 | 1 function \_\_isset($key){2 return isset($this -> $key);3 } |

当外部使用isset($对象名->私有属性);检测时，将自动调用上述\_\_isset()返回的结果！

④\_\_unset($key):外部使用unset()函数删除私有属性时，自动调用；  
1 function \_\_unset($key){ 2 unset($this -> $key); 3 }   
当外部使用unset($对象名->私有属性);删除属性时，自动将属性名传给\_\_unset(),并交由这个魔术方法处理。

**继承的基础知识：**

1、如何实现继承？  
给子类使用extends关键字，让子类继承父类；  
class Student extends Person{}

2、实现继承的注意事项？  
①子类只能继承父类的非私有属性。  
②子类继承父类后，相当于将父类的属性和方法copy到子类，可以直接使用$this调用。  
③PHP只能单继承，不支持一个类继承多个类。但是一个类进行多层继承。  
class Person{}  
class Adult extends Person{}  
class Student extends Adult{}   
//Student 类就同时具有了Adult类和Person类的属性和方法

3、方法覆盖(方法重写)  
条件一： 子类继承父类  
条件二：子类重写父类已有方法

符合上述两个条件，称为方法覆盖。覆盖之后，子类调用方法，将调用子类自己的方法。  
同样，除了方法覆盖，子类也可以具有与父类同名的属性，进行属性覆盖。

如果，子类重写了父类方法，如何在子类中调用父类同名方法？

partent::方法名();  
所以，当子类继承父类时，需在子类的构造中的第一步，首先调用父类构造进行复制。

|  |  |
| --- | --- |
| 1 | 1 function \_\_construct($name,$sex,$school){2 partent::\_\_construct($name,$sex);3 $this -> school = $school;4 } |
| **三、PHP关键字** | |

1、final  
①final修饰类，此类为最终类，不能被继承！  
②final修饰方法，此方法为最终方法，不能被重写！  
③final不能修饰属性。

2、static  
①可以修饰属性和方法，分别称为静态属性和静态方法，也叫类属性，类方法；  
②静态属性，静态方法，只能使用类名直接调用。  
使用"类名::$静态属性" , "类名::静态方法()"  
Person::$sex; Person::say();  
③静态属性和方法，在类装载时就会声明，先于对象产生。  
④静态方法中，不能调用非静态属性或方法；  
非静态方法，可以调用静态属性和方法。   
（因为静态属性和方法在类装载时已经产生，而非静态的属性方法，此时还没有实例化诞生）  
⑤在类中，可以使用self关键字，代指本类名。

|  |  |
| --- | --- |
| 1 | 1 class Person{2 static $sex = "nan";3 function say(){4 echo self::$sex;5 }6 } |

⑥静态属性是共享的，也就是new出很多对象，也是共用一个属性。

3、const关键字：  
在类中声明常量，不能是define()函数！必须使用const关键字。  
与define()声明相似，const关键字声明常量不能带$，必须全部大写！  
常量一旦声明，不能改变。调用时与static一样，使用类名调用Person::常量。

4、instanceof操作符：  
检测一个对象，是否是某一个类的实例。（包括爹辈，爷爷辈,太爷爷辈……）

|  |  |
| --- | --- |
| 1 | $zhangsan instanceof Person; |

**【小总结】几种特殊操作符：**

① . 只能连接字符串； "".""  
② => 声明数组时，关联键与值["key"=>"value"]  
③ -> 对象($this new出的对象)调用成员属性，成员方法；  
④ :: ①使用parent关键字，调用父类中的同名方法：parent::say();  
②使用类名(和self)调用类中的静态属性，静态方法，以及常量。

|  |
| --- |
| **四、单例** |

单例模式也叫单态模式。可以保证，一个类只能有一个对象实例。

实现要点：  
①构造函数私有化，不允许使用new关键字创建对象。  
②对外提供获取对象的方法，在方法中判断对象是否为空。  
如果为空，则创建对象并返回；如果不为空则直接返回。  
③实例对象的属性以及获取对象的方法必须都是静态的。  
④之后，创建对象只能使用我们提供的静态方法。  
eg:$s1 = Singleton::getSingle();

|  |
| --- |
| **五、对象串行化和魔术方法** |

\*\*\*关键词：clone与\_\_clone、\_\_antoload()、串行化与反串行化（序列化与反序列化）、类型约束、魔术方法小总结（12个）

**一clone与\_\_clone**

1、当使用=讲一个对象，赋值给另一个对象时，赋的实际是对象的地址。  
两个对象指向同一地址，所以一个对象改变，另一个也会变化。  
eg: $lisi = $zhangsan;  
2、如果想要将一个对象完全克隆出另一个对象，两个对象是独立的，互不干扰的，  
则需要使用clone关键字；  
eg: $lisi = clone $zhangsan; //两个对象互不干扰  
3、\_\_clone():  
①当使用clone关键字，克隆对象时，自动调用clone函数。  
②\_\_clone()函数，类似于克隆时使用的构造函数，可以给新克隆对象赋初值。  
③\_\_clone()函数里面的$this指的是新克隆的对象  
某些版本中，可以用$that代指被克隆对象，绝大多数版本不支持。  
4、\_\_toString()  
当使用echo等输出语句，直接打印对象时调用echo $zhangsan;  
那么，可以指定\_\_toString()函数返回的字符串；

|  |  |
| --- | --- |
| 1 | 1 function \_\_toString(){2 return "haha";3 }4 echo $zhangsan; //结果为：haha |

5、\_\_call()  
调用类中未定义或未公开的方法时，会自动执行\_\_call()方法。  
自动执行时，会给\_\_call()方法传递两个参数；  
参数一：调用的方法名  
参数二：（数组）调用方法的参数列表。

**二\_\_antoload()**

①这是唯一一个不在类中使用的魔术方法；  
②当实例化一个不存在的类时，自动调用这个魔术方法；  
③调用时，会自动给\_\_autoload()传递一个参数：实例化的类名  
所以可以使用这个方法实现自动加载文件的功能。

|  |  |
| --- | --- |
| 1 | 1 function \_\_autoload($className){2 include    "class/".strtolower($className).".class.php";3 }4 $zhangsan=new Person();//本文件内没有Person类，会自动执行\_\_autoload()加载person.class.php文件 |

**三面向对象串行化与反串行化（序列化与反序列化）**

1、串行化：将对象通过一系列操作，转化为一个字符串的过程，称为串行化。

　　　　　（对象通过写出描述自己状态的数值来记录自己）

2、反串行化：将串行化后的字符串，再转为对象的过程，称为反串行化；  
3、什么时候使用串行化？  
①对象需要在网络中传输的时候  
② 对象需要在文件或数据库中持久保存的时候  
4、怎样实现串行化与反串行化  
串行化： $str=serialize($zhangsan);  
反串行化：$duixiang=unserialize($str);  
5、\_\_sleep()魔术方法：  
①当执行对象串行化的时候，会自动执行\_\_sleep()函数；  
②\_\_sleep()函数要求返回一个数组，数组中的值，就是可以串行化的属性；不在数组中的属性，不能被串行化；  
function \_\_sleep(){  
return array("name","age"); //只有name/age两个属性可以串行化。  
}  
  
6、\_\_wakeup()魔术方法  
①当反串行化对象时，自动调用\_\_wakeup()方法；  
②自动调用时，用于给反串行化产生的新对象属性，进行重新赋值。  
1 function \_\_wakeup(){ 2 $this -> name = "李四"; 3 }

**四类型约束**

1、类型约束：是指在变量时，加上数据类型，用于约束此变量只能存放对应的数据类型。  
（这种操作常见于强类型语言，在PHP中，只能实现数组和对象的类型约束）  
2、如果类型约束为某一个类，则本类以及本类的子类对象，都可以通过。  
3、在PHP中，类型约束，只能发生在函数的形参中。

|  |  |
| --- | --- |
| 1 | 1 class Person{}2 class Student extends Person{}3 function func(Person $p){ //约束函数的形参，只接受Person类及Person子类4 echo "1111";5 echo $p -> name;6 } |

func(new Person()); √  
func(new Student()); √  
func("111"); ×

形如new Person();的形式，我们称其为"匿名对象";

※※※基类：父类   
※※※派生类：子类

**五魔术方法小总结**

1、\_\_construct():构造函数，new一个对象时，自动调用。  
2、\_\_destruct():析构函数，当一个对象被销毁前，自动调用。  
3、\_\_get():访问类中私有属性时，自动调用。传递读取的属性名，返回$this->属性名  
4、\_\_set():给类的私有属性赋值时，自动调用。传递需要设置的属性名和属性值；  
5、\_\_isset():使用isset()检测对象私有属性时，自动调用。传递检测的属性名，返回isset($this -> 属性名);  
6、\_\_unset():使用unset()删除对象私有属性时，自动调用。传递删除的属性名，方法中执行unset($this -> 属性名);  
7、\_\_toString():使用echo打印对象时，自动调用。返回想要在打印对象时，显示的内容；返回必须是字符串；  
8、\_\_call():调用一个类中未定义或未公开的方法时，自动调用。传递被调用的函数名，和参数列表数组；  
9、\_\_clone():当使用clone关键字，克隆一个对象时，自动调用。作用是为新克隆的对象进行初始化赋值；  
10、\_\_sleep():对象序列化时，自动调用。返回一个数组，数组中的值就是可以序列化的属性；  
11、\_\_wakeup():对象反序列化时，自动调用。为反序列化新产生的对象，进行初始化赋值；  
12、\_\_autoload():需要在类外部声明函数。当实例化一个未声明的类时，自动调用。传递实例化的类名，可以使用类名自动加载对应的类文件。

|  |
| --- |
| **六、抽象类和抽象方法** |

1、什么是抽象方法？  
没有方法体{}的方法，必须使用abstract关键字修饰。这样的方法，我们称为抽象方法。  
abstract function say(); //抽象方法

2、什么是抽象类？  
使用abstract关键字修饰的类就是抽象类。  
abstract class Person{}

3、抽象类的注意事项：  
① 抽象类可以包含非抽象方法；  
② 包含抽象方法的类必须是抽象类，抽象类并不一定必须包含抽象方法；  
③ 抽象类，不能实例化。（抽象类中可能包含抽象方法，抽象方法没有方法体，实例化调用没有意义）  
我们使用抽象类的目的，就是限制实例化！！！

4、子类继承抽象类，那么子类必须重写父类的所有抽象方法，除非，子类也是抽象类。

5、使用抽象类的作用？  
① 限制实例化。（抽象类是一个不完整的类，里面的抽象方法没有方法体，所以不能实例化）  
② 抽象类为子类的继承提供一种规范，子类继承一个抽象类，则必须包含并且实现抽象类中已定的抽象方法。

|  |
| --- |
| **七、接口与多态** |

**一接口**

1、什么是接口？  
接口是一种规范，提供了一组实现接口的类所必须实现的方法组合。  
接口使用interface关键字声明；  
interface Inter{}

2、接口中的所有方法，必须都是抽象方法。  
接口中的抽象方法不需要也不能使用abstract修饰。

3、接口中不能声明变量，不能有属性，只能使用常量！！！

4、接口可以继承接口，使用extends关键字！  
接口使用extends继承接口，可以实现多继承。  
interface int1 extends Inter,Inter2{}

5、类可以实现接口，使用implements关键字！  
类使用implements实现接口，可同时实现多个接口，多个接口间逗号分隔；  
abstract class Person implements Inter,Inter2{}  
一个类实现一个或多个接口，那么这个类，必须实现所有接口中的所有抽象方法！  
除非，这个类是抽象类。

**【接口&&抽象类区别】：**

①声明方式上，接口使用interface关键字，抽象类使用abstract class。  
②实现/继承方式上，一个类使用extends继承抽象类，使用implements实现接口。  
③抽象类只能单继承，接口可以多实现。（接口extends接口）、多实现（类implements接口）  
④抽象类中可以有非抽象方法，接口中只能有抽象方法，不能有费抽象方法。抽象类中的抽象方法必须使用abstract关键字修饰，接口中抽象方法不能带修饰词。  
⑤抽象类是个类，可以有属性、变量；接口中只能有常量。

**二多态**

二、多态  
1、一个类，被多个子类继承。  
如果，这个类的某个方法，在多个子类中，表现出不同的功能，我们称这种行为为多态。

2、实现多态的必要途径：  
① 子类继承父类；  
② 子类重写父类方法；  
③ 父类引用指向子类对象