面向对象

面向过程的程序设计

一条一条代码的在脚本中执行，重复经典的代码可以直接封装成函数，和流程控制

类和对象之间的关系

1、类的实例化结果就是对象

2、对象的抽象就是类

3、类描述了一组有相同特性（属性）和相同行为（方法）的对象。

4、类是一个独立的程序单位，是存放在**硬盘空间的**，放在一个**文件中的**，抽象的定义类是不能有在程序的世界里用的，一定要进行实例化成对象。

5、对象在客观世界里，系统中用来描述客观事物的一个实体，是**存在内存中的。**

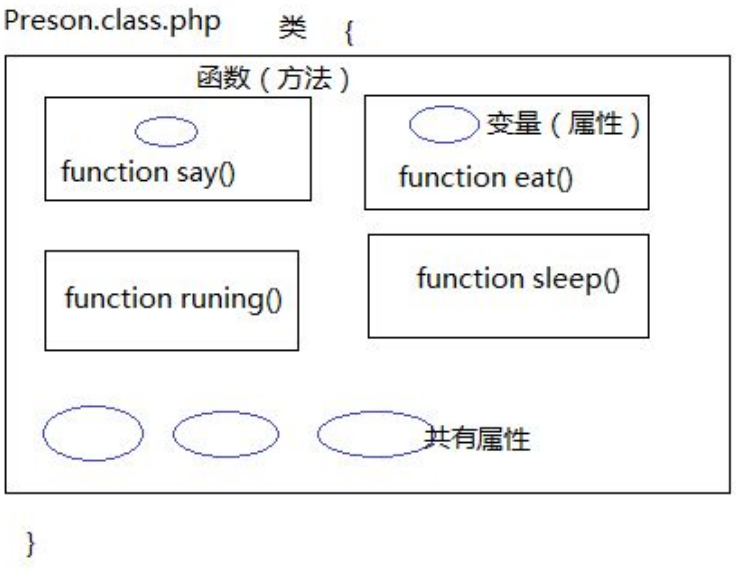
程序的世界就是在内存中的

举例：类就生产汽车的设计图纸，对象是按照着这些图纸进行生产出来的实体

面向对象的程序设计

C 是纯过程化　Java 是存面向对象

简单的说：类就是很多的函数用一个花括号包起来，然后给整体起个名字叫作类，再将这个类创建成一个对象来，我们就可以使用对象中的函数了，一个函数中的变量要想在另一个函数中使用，那就直接提取出来定义成公共的属性，共享，这样比单个函数的使用效率和方法更高。



面向方面的程序设计

举例：餐厅中的每个成员的职责

用户对象

面对象程序设计：　　面向方面：

登陆的方法：　　登陆的方法：

1、获取登陆信息
2、连接数据库
3、准备 SQL 语句
4、执行 SQL 语句
5、处理结果集
6、提示信息

1、直接获取信息
2、查询数据
3、返回信息
不考虑数据信息真实，不考虑拿不拿到
会在之前已经处理好才执行到这步。

面向对象是程序的一种设计方式，它利于提高程序的重用性，使程序结构更加清晰

面向对象的三个主要特性：

对象的行为：可以对对象施加哪些操作；如电视机的开、关、转换频道等。

对象的状态：当施加那些方法时，对象如何响应；如电视机的外形、尺寸、颜色等；

对象的标识：如何区分具有相同行为与状态的不同对象。

总结：

对象是客观存在的一个实体。
类是对对象抽象的一个描述。

概念：对象（实体）、类、 类与对象的关系。

oop 面向对象编程的特点：封装、继承、多态

类和对象的关系：

类的实例化结果就是一个对象（使用 new 关键字）
对对象的抽象描述就是一个类

学习目标：

抽象一个类

1 类的声明
2 成员属性
3 成员方法

类的声明：

简单格式：

```
[修饰符] class  类名{    //使用 class 关键字加空格后加上类名
    [成员属性]          //也叫成员变量
    [成员方法]          //也叫成员函数
}
```

类前的修饰符：final、abstract 可省的

成员属性：修饰符不可以省略

格式：

```
修饰符 $变量名[=默认值];    //如：public $name="zhangsan";
```

注意：成员属性不可以是带运算符的表达式、变量、方法或函数调用。

```
public $var3 = 1+2;                    //错误格式
public $var4 = self::myStaticMethod();  //错误格式
public $var5 = $myVar;                  //错误格式
```

正确定义方式：

```
public $var6 = 100; //普通数值（4个标量：整数、浮点数、布尔、字串）
public $var6 = myConstant;              //常量
public $var7 = self::classConstant;     //静态属性
public $var8 = array(true, false);      //数组
```

常用属性的修饰符：**public**、protected、private、static、var(PHP4 之前,PHP5 不建议使用)

成员方法：修饰符可以省

成员方法格式：

```
[修饰符] function  方法名(参数..){
    [方法体]
    [return  返回值]
}
```

常用方法的修饰符：public、protected、private、static、abstract、final

声明的成员方法必须和对象相关，不能是一些没有意义的操作

类与函数

类要 new 才能用，函数调用才能用

学习目标：

1 实例化对象
2 对象类型在内存中的分配
3 对象中成员的访问
4 特殊的对象引用”$this”
5 构造方法与析构方法

实例化对象

当定义好类后，我们使用 new 关键字来生成一个对象。

$对象名称 = new 类名称();

$对象名称 = new 类名称([参数列表]);

对象类型在内存中的分配

每个对象要单独占用数据空间(在内存中独立存在的单元)

```
//定义的类 A，其中只有一个属性 name
class A{
    public $name="aa";
}

//实例化 A 类成对象
$a1 = new A(); //标准实例化类 A 产生对象 a1
```

```
$a2 = new A; //实例化出对象 a2，（）可省略不写

//查看对象结构
//var_dump($a1); //object(A)#1 (1) { ["name"]=> string(2) "aa" }
//var_dump($a2); //object(A)#2 (1) { ["name"]=> string(2) "aa" }

//通过对象调用属性：->
//echo $a1->name; //aa //输出对象 a1 中属性 name 值，方法也同样

//对象和对象之间没有关联(之间是独立的) 相当于生产出来的汽车之间是独立

//修改 a1 中 name 属性值  相当于生产出来的汽车被改变外观与定制汽车的外观不同
$a1->name="bb";

//对象是引用类型
var_dump($a1); //object(A)#1 (1) { ["name"]=> string(2) "aa" }
//将 a1 对象赋给 a3，由于对象属于引用类型，所以 a3 是 a1 的引用名（别名）
$a3 = $a1;
$a1->name="bb";
var_dump($a3); //object(A)#1 (1) { ["name"]=> string(2) "aa" }
```

对象中成员的访问

类中包含成员属性与成员方法两个部分，我们可以使用“new”关键字来创建一个对象，即：$引用名 ＝new 类名(构造参数)；那么我们可以使用特殊运算符“->”来访问对象中的成员属性或成员方法。如：（对象是引用类型，为节约内存资源）

```
$引用名 ＝new 类名(构造参数);
$引用名->成员属性=赋值;        //对象属性赋值
echo $引用名->成员属性名;          //输出对象的属性   不要加$
$引用名->成员方法名（参数）;      //调用对象的方法
```

如果对象中的成员不是静态的，那么这是唯一的访问方式。

特殊的对象引用”$this”

$this 的使用

1、只能在对象中使用，并且只能在对象中的成员方法中使用

2、使用$this 在成员方法中访问自己对象内部的成员（成员属性，成员方法）

3、$this 表示对象本身自己（对象引用符号）

总结：

$this 关键字： 表示自己，表示当前使用对象。

我们只能在**类中方法中**调用自己的**成员属性或函数**都是使用 $this->调用。

注意：非静态方法中可以使用 this 关键字

特殊运算符 ->

1、在对象外部通过对象的引用名称访问对象中的成员（成员属性，成员方法）
2、$object = new 类名（）实例化类
3、访问对象中的方式：$object ->成员属性，成员方法（访问的属性可以再次改变值）
通过 -> 访问成员（属性和方法）
分清$this 思路：要看清对象

魔术方法 __ （双下划开始）

构造方法 （构造函数）
1、实例化对象时不能省略（），建议养成好习惯每次写都要添加（），需要在实例化对象传递参数时必须加上（）提供传递值给构造方法的 function __construct ([参数$arg])
举例：在生产汽车之前需要定制外观特别的车型，但是基本的功能还是保证不变，功能变的话整条生产线都要改变了，只是改变车的属性而已其车的功能方法还是不变。
2、当创建一个对象时，它将自动调用构造函数，主要用于初始化对象。
3、是 PHP 中特殊的方法
补充：PHP4 中与类同名的方法是构造方法

```
class person{
    public function person(){
        echo  ‘与类名相同方法是构造方法，在实例化对象时自动调用方法’
    }
}
```

以上方式在 PHP4 中使用，在 PHP5 中使用以下的格式

为了向下兼容，如果一个类中没有名为__construct( )的方法，PHP 将搜索一个与类名相同的方法。
格式： [修饰符] function __construct ([参数]) {

```
        ... ...
    }
```

析构方法 __destruct( )
1、与构造函数相对的就是析构函数。析构函数是 PHP5 新添加的内容，在 PHP4 中没有析构函数。**析构函数是在对象被销毁之前自动调用的方法**，主要执行一些特定的操作，例如关闭文件，释放结果集等。
2、与构造函数的名称类似，一个类的析构函数名称必须是两个下划线 _ _destruct( )。析构函数不能带有任何参数。
3、销毁对象时才自动调用析构方法（普通脚本执行结束内存中的变量释放）
深入解析对象类型（对象是引用类型）
1、对象是存在堆里面，对象的名字是在栈里面的
栈的特点：
先进后出（子弹夹）
每次 new 一个对象就压进弹夹中，在最后压进入的对象会被第一个释放

```php
<?php
    //析构方法：在对象被销毁时最后一个自动调用的方法，目的是为了销毁对象中一些资源
    class A{
        public $name;
        //构造方法
        public function __construct($name1){
            $this->name= $name1;//new是参数直接赋给了属性$name,共享属性在类中使用
            //$this->name = $name;
            echo "{$name1}被构造,new时直接参数使用<br>";
            //echo "{$this->name}被构造参数属性初始化可以直接使用<br>"; 测试$name1变量
        }

$obj = new A("zhangsan--0");
$obj1 = new A("zhangsan--1");
$obj1 =null;//直接提前销毁,不按照实际中释放的顺序
$obj2 = new A("zhangsan--2");
$obj3 = new A("zhangsan--3");
```

```php
    //析构方法
    public function __destruct(){
        //初始化的属性在类中共享
        echo "{$this->name}被析构了<br>";
    }
}
```

```
zhangsan--0被构造,new时直接参数使用
zhangsan--1被构造,new时直接参数使用
zhangsan--1被析构了
zhangsan--2被构造,new时直接参数使用
zhangsan--3被构造,new时直接参数使用
zhangsan--3被析构了
zhangsan--2被析构了
zhangsan--0被析构了
```

2、实例化一个对象时没有赋给一个变量，创建后直接销毁，不会栈里存放，很少使用这个方式没有实际意思。

面向对象的语言的第一大特征：封装性

学习目标

1、设置私用成员（包括属性和方法）

2、私有成员的访问

3、__set()、__get()、__isset()和__unset()四个方法

封装性是面向对象编程中的三大特性之一，封装就是把对象中的成员属性和成员方法加上访问修饰符，使其尽可能隐藏对象的内部细节，以达到对成员的访问控制(**切记不是拒绝访问**)。说白了是对访问权限的控制，就一个加上特定的关键字，这些关键字就由底层控制）这是 PHP5 的新特性，但却是 OOP 语言的一个好的特性。而且大多数 OOP 语言都已支持此特性。

PHP5 支持如下 3 种访问修饰符:

public (公有的 默认的)

private (私有的)

protected (受保护的)

举例：封闭小区，不是封闭人员进出，而是通过设置门卫来控制访问

设置私用成员

只要在声明成员属性或成员方法时，使用 private 关键字修饰就是实现了对成员的私有封装。封装后的成员在对象的外部不能直接访问，只能在对象的内部方法中使用 $this 访问。

私有成员的访问

设置了私有的属性时，在对象内部提供公有的方法，直接通过$this 访问属性，然后在对象的外部就可以直接通过访问公有的方法获取和重新设置值。

举例：

```php
<?php
//对象的封装的控制访问
class Stu{
    private $name; //定义私用属性
    private $age;   //定义私用属性

    public function __construct($name,$age){
        $this->name = $name;
        $this->age = $age;
    }

    public function getinfo(){
        echo $this->name.":".$this->age."<br/>";
    }
```

```
        //获取私有属性 age 的方法
        public function getAge(){
            return $this->age;
        }

        //设置私有属性 age 的方法
        public function setAge($age){
            //访问控制,return 后的语句不会执行
            if($age<1){
                echo '年龄赋值错误！';
                return;
            }
            $this->age = $age;
        }
    }

    //测试
    $s = new Stu("zhangsan",20);
    $s->getinfo();
    //echo $s->name; //尝试输出对象中的私用属性

    //获取私有属性的方法
    echo $s->getAge();

    //设置私有属性的方法
    $s->setAge(30);

    //设置私有属性的访问控制
    $s->setAge(-30);

    $s->getinfo();
```

在访问控制中我们可以使用 if 判断条件进行控制输出

使用 if ...else...// 获取年龄方法 （由小到大）

```
            function getAge() {
                if($this->age < 20) {
                    $age = '花样少年';
                }elseif($this->age < 30) {
                    $age = '青年';
                }elseif($this->age < 40) {
                    $age = '而立';
                }elseif($this->age < 50) {
                    $age = '不惑';
                }elseif($this->age < 70) {
                    $age = '花甲';
```

```
            }else {
                $age = '年龄不详';
            }
            return $age;
        }
```

使用 switch ..case..

利用运算符和数学函数处理出来 ceil

成员的封装权限

```
<?php
//属性的封装使用：
```

	public(公有)	protected(受保护)	private(私有)
在本类中	Y	Y	Y
在子类中	Y	Y	N
在类外边	Y	N	N

```
class A{
    public $x=10;
    protected $y=20;
    private $z=30;

    //在类的内部使用方法访问自己的成员属性
    public function demo(){
        echo $this->x.":".$this->y.":".$this->z."<br/>";

        $this->aa(); //在类中访问自己的私有方法
    }

    private function aa(){
        echo "aaaaaaaaaa";
    }
}

$a = new A();

echo $a->x; //公有属性可以访问
//echo $a->y; //在类外受保护属性不可以访问
//echo $a->z; //在类外私有属性不可以访问

$a->demo();//本类中可以访问公有 public 的方法,方法中又访问$this 的属性

//$a->aa(); //直接在类外不可以访问私有方法，但是在在类中 public 方法中访问，这样又转化
```

为 public 方法。

总结：

封装的属性不能直接在类外面访问和设置，要访问的属性和方法要在定义的公有属性进行通过$this -> 成员从而直接访问类中定义的 public 方法，起到间接的访问效果。定义为封装的成员还存在只是不能直接访问和设置而已。

魔术方法

__set()、__get()、__isset()和__unset() （自动调用）

以上这些都是为了减少代码的重复书写量和优化代码不会让程序输出错误信息，而且直接在类外访问会报错，非常不友好，这样就出现这些魔术方法。

魔术方法：

__set()： 用于替代通用的 set 赋值方法

__get(): 用于替代通用的 get 取值方法

__isset()： 检测对象中成员属性是否存在

__unset()： 销毁对象中成员属性方法

注意：

上面四个魔术方法只对类中的私有、受保护成员属性有效。

魔术方法前的修饰符可以是公有、私有，不影响调用。

属性相关的魔术方法(不可访问：是成员已经设置私有或者保护)

__set($name，$value) 一定有 2 个参数

参数:

第一个参数: 表示直接访问属性的名称

第二个参数: 表示设置的值

作用：:设置一个不可访问属性时自动执行

如果属性不存在（也是一个不可访问的属性）__set() 也会被调用

通过 -> 访问并设置私有属性的值，在类外访问并设置时被设置私有属性会自动调用__set()。

```
$obj -> sex = '要设置的值'; //sex 是被设置私有属性，
function __set($name, $value) {
    $this->$name = $value;  注意：相对于可变变量的解析
}
$obj = new person('林志玲', 38, '女');
//性别设置了私有在类外访问,此时会自动调用__set 方法
$obj -> sex = '设置的值';  访问属性名传给$name
```

```
object(Person)[1]
  public 'name' => string '林志玲' (length=9)
  private 'age' => int 38
  private 'sex' => string '设置的值' (length=12)
```

注意：当访问一个不存在的属性时，也会往调用__set 方法往对象中压入值。
$obj -> num = 100;//会在对象中输出，动态添加属性一般不用。

__get($name)
参数: 表示直接访问属性的名称
作用：获取一个不可访问的属性时自动执行

手册指明：
参数$name 是指要操作的变量名称。__set() 方法的$value 参数指定了$name 变量的值。

举例：

```
class A{
    public $name="zhangsan"; //定义公用属性
    protected $sex="man";    //定义受保护属性
    private $age=20;         //定义私用属性

    //魔术方法 get 一个参数（成员名）
    public function __get($param){
        return $this->$param;//（可变属性）
    }

    //魔术方法 set  必须两个参数
    public function __set($param,$value){
        //控制访问
        if($param=="age" && $value<1){
            echo "年龄赋值错误！";
            return;//return 技巧，执行到此句后面的代码不会执行，而不是 else
        }
        $this->$param = $value;//设置值
    }
}

$obj =new A();
$obj -> sex="woman";
$obj -> age=-25;

echo $s->sex;
echo $s->age; //当直接输出对象中非公有属性时，会自动调用魔术 get 方法,并将属
              性名作为参数传入。
结果：年龄赋值错误！woman20
```

理解：
就算是赋值有错也不会输出系统错误的信息，而是输出提示信息，值还是保留原来的值。假设设置为公有直接赋值就会成功，起不到验证的效果。

总结：

__set( )方法：

格式 [修饰符] function __set(string $name,mixed $value){

... }

当我们直接为一个对象中非公有属性赋值时会自动调用此方法，并将属性名以第一个参数(string)，值作为第二参数(mixed)传进此方法中。

__get( )方法：

格式：[修饰符] function __get(string $name){

... }

当我们直接输出一个对象中非公有属性时会自动调用此方法，并将属性名以第一个参数传进去。

魔术方法：

属性相关的魔术方法

__isset(name)

参数：

name 表示属性名称

执行： 使用 isset() empty() 判断时自动执行

需要加 return //存在就要返回值才符合逻辑

__unset(name)

参数：

name 表示属性名称

执行：使用 unset() 销毁属性时

不能加 return 已经销毁了还返回，不符合逻辑

注意：unset 可以删除对象中的属性，数组中的单元，变量

举例：

```
//定义一个简单的类
class A{
    public $name = "zhangsan";
    protected $sex = 'nan';
    private $age = 100;

    //魔术方法 unset
    public function __unset($param){
        //在类的内部是可以删除的
        unset($this -> $param);
    }

}
```

```
//实例化类
$obj = new A();

var_dump($obj);//查看结构

//删除对象中 obj 中的公有属性
unset($obj -> name);
var_dump($obj);//再查看      可以删除

//删除对象中 obj 中的私有属性
unset($obj -> sex);
var_dump($obj);//再再查看    会报错,你连获取都获取不到还想操作属性,那是不可
                    能的

/*当定义了魔术方法__unset 方法时,在类外部要执行 unset 操作时
会自动找到__unset 方法执行方法内部的代码，相对于我要在一个
封闭小区(类)内干掉一个人(类中的属性),他的身份特殊(受保护),
我不能直接进入干掉他,此时我第一时间(自动调用)想到的是要买通
小区的保安(__unset 魔术方法)让他去帮我干掉他即可完成任务。
*/
```

总结：

__unset()当对未定义的变量调用 unset()时，__unset() 会被调用。

```
//当 unset 销毁一个对象的非公有属性时，自动调用此方法。
public function __unset($param){
    unset($this->$param);
}
```

魔术方法：

__isset()当对未定义的变量调用 isset() 或 empty()时，__isset() 会被调用。

```
//当 isset 判断一个对象的非公有属性是否存在时，自动调用此方法。
public function __isset($param){
    return isset($this->$param);
}
```

举例：

```
//定义一个简单的类

class A{
    public $name = "zhangsan";
    protected $sex = 'nan';
    private $age = 100;

    //魔术方法 isset
    public function __isset($param){
```

```
            return isset($this -> $param);
        }

    }

    //实例化类
    $obj = new A();

    var_dump($obj);//查看结构

    //访问一个公有属性 name
    if(isset($obj -> name)){
        echo '属性存在';
    }else{
        echo '属性不存在';
    }
    echo '<hr>';
    //访问一个私有属性 sex
    if(isset($obj -> sex)){
        echo '属性存在';
    }else{
        echo '属性不存在';
    }
```

结果：

```
object(A)[1]
        public 'name' => string 'zhangsan' (length=8)
        protected 'sex' => string 'nan' (length=3)
        private 'age' => int 100
```

属性存在

属性存在

总结：

__isset 可以判断对象中的私有属性是否存在而已，说明对象中私有属性是隐藏的在外界看不到的判断不出来，但是实际是存在的，要想判断就要利用系统内置的函数就是魔术方法进行判断查看。

魔术方法：

方法的重载:

* mixed __call (string $name , array $arguments)
  mixed __callStatic (string $name , array $arguments) php5.3.0 支持

当调用一个不可访问方法（如未定义，或者不可见）时，__call() 会被调用。

其中第一个参数表示方法名，第二参数表示调用时的参数列表（数组类型）

当在静态方法中调用一个不可访问方法（如未定义，或者不可见）时，__callStatic() 会被调用。

说白了：封装就是我们将一些不想让外界知道的东西隐藏起来或者受保护起来，直接把属性和方法私有化，目的是留给下一代使用，我们会提供一些公有的属性给外界获取，同时我们也会定义公有的方法获取本类的成员，为了优化当要想获取受保护的属性和方法会报错就需要系统提供的方法就是魔术方法解决。

面向对象的语言的第二大特征：继承性

学习目标：

1、类继承的应用
2、访问类型控制
3、子类中重载父类的方法

程序联系实际：
现实中的继承：儿子继承了表示父没了，程序中的继承：子类继承了父类，父类还是存在
共性：现实中的继承和程序中的继承都会省很多事，少奋斗几年，子类少写很多代码。

继承带来项目的扩展性好处，继承的父类可以进行再次修改，好比传统的工艺加上现在的科技转变出更好的工艺（子类中扩展和升级）。

继承
php 继承的关键字：extends
格式：
[修饰符] class 子类名 extends 父类名{ ... }

继承的特点：
1、当扩展一个类，子类就会继承父类的所有公有和保护方法（表面理解）。但是子类的方法会覆盖父类的方法。私有的不行
2、继承对于功能的设计和抽象是非常有用的，而且对于类似的对象增加新功能就无须重新再写这些公用的功能。（对父类的升级）
补充：汇编，C，C++，Java
3、PHP 只支持单继承，不允许多重继承。一个子类只能有一个父类，不允许一个类直接继承多个类，但一个类可以被多个类继承。（每一个人只有一个亲爹）

4、可以有多层继承，即一个类可以继承某一个类的子类，如类 B 继承了类 A，类 C 又继承了类 B，那么类 C 也间接继承了类 A。

5、补充：多重继承的不利因素，因为在继承时假设继承过来的两个父类中有相同名的方法，或者同名的方法并且执行相同的操作，这样对于真正的使用者来说就没有明确的操作了。

继承的产生：

1、我们在定义每一个类时，发现有重复编写的代码，为了减少代码的精简性，我们就把重复出现的代码提取出来定义为独立的类，从而我们就可以去继承这个类了。（有点像合并同类项）

总结：

1、子类继承父类的所有内容（属性和方法），但父类中的 private 部分不能直接访问

2、子类中新增加的属性和方法是对父类的扩展

3、子类中定义的与父类同名的属性是对父类属性的覆盖，同名的方法也是对父类方法的覆盖。

4、假如 B 类继承 A 类，那么就继承了 A 中所有非私有属性和方法（函数）。

其中 A 叫父类（基类）。 B 叫子类（派生类）

5、在 php 中类只支持【单一继承】，就是一类只能继承一个父类。

在 new 子类时同时会在内存中创建了父类和子类的对象的，这样我们才可以操作父类和子类的成员，只不过私有的成员有访问的权限。私有的成员内存会存在的，只是我们没有访问的权限。

类的分类：信息包装类（Model 类,底层类） 业务处理类（控制类）

举例//在类的信息封装中实现继承的使用

```
<?php
//在类的信息封装中实现继承的使用
header("Conten-type:text/html;charset=utf-8");
//定义 person 人类
class Person{
    private $name; //私有属性
    private $age;

    //构造方法
    public function __construct($name,$age){
        $this->name = $name;
        $this->age  = $age;
    }

    //获取信息方法,在类中定义公有的方法,这样可以通过方法获取私有的成员
```

```
    public function getinfo(){
        return $this->name.":".$this->age;
    }
}

//定义一个学生类
class Stu extends Person{
    //添加一个父类没有的属性（扩展）
    private $classid;

    /*构造方法,子类定义同名的方法会覆盖父类的方法,但是我们又要
    沿用父类的方法,怎么办需要 parent 关键字调用父类中的方法*/

    public function __construct($name,$age,$classid){
        //先构造父类,parent 调用父类存在的方法
        parent::__construct($name,$age);//与父类一致
        $this->classid = $classid;
    }

    //覆盖一个父类的方法（升级）
    public function getinfo(){
        return parent::getinfo().":".$this->classid;
    }
}

//测试代码

//父类
$p = new Person("张三",20);
echo $p->getinfo()."<br/>";
var_dump($p);

echo '<hr>';

//子类
$stu = new Stu("李四",22,"lamp94");
echo $stu->getinfo()."<br/>";
var_dump($stu);
```

总结：

parent 关键字：若子类出现覆盖父类的方法，那么有时还想调用被覆盖掉了的方法，我们就是用关键字【parent::父类方法】还有使用类名

结果：

张三:20

```
object(Person)[1]
  private 'name' => string '张三' (length=4)
  private 'age' => int 20
```

李四:22:lamp94

```
object(Stu)[2]
  private 'classid' => string 'lamp94' (length=6)
  private 'name' (Person) => string '李四' (length=4)
  private 'age' (Person) => int 22
```

访问类型控制

访问权限：

	private	protected	public(默认的)
在同一类中：	可以	可以	可以
类的子类中：	X	可以	可以
其他外部类中：	X	X	可以

举例：

```
class A{
        public $x=10;
        protected $y=20;
        private $z=30;

        public function fun(){
            //在类中可以调用自己的任意属性
            echo $this->x.":".$this->y.":".$this->z."<br/>";
        }
    }

    class B extends A{
        public function demo(){
//在子类中可以调用父类中非私有属性(其中 z 是在父类中私有，所以无法获取，但报 notice)
            echo $this->x.":".$this->y.":".$this->z."<br/>";
        }
    }

    $a = new A();
    $a->fun();
    echo $a->x; //在类的外部只可以调用自己公有的属性
```

```
echo "<hr/>";

$b = new B();
echo $b->x;
$b->demo();

echo "<hr/>";

echo $b->z;//因为子类中继承不了私有属性所以不存在
```

结果：

parent 关键字的使用

```
<?php
//parent 关键字的使用

class A{
    public function fun(){
        echo "aaaaaaaaaa<br/>";
    }
}

class B extends A{
    //此方法名与父类重名，则为方法覆盖
    public function fun(){
        echo "bbbbbbbbbbbb<br/>";
    }

    public function demo(){
    /*程序的执行过程：在类外调用方法,会在当前对象寻找指定名称的方法,在此方法中
    操作又调用了本对象($this)中的方法,接着继续子类寻找,找到就执行调用方法中的语
    句,如果因为子类中的 fun()与父类相同,就优先执行,不会往父类中找.子类的方法优先
    父类*/
        //$this->fun(); //调用当前对象中 fun 方法  ($this 当前对象)
        parent::fun(); //直接在父类中调用 fun 方法(在子类中取到父类中被覆盖的方法,属性
                       中没有覆盖无意义可以直接在子类中改即可)
```

重写和重载：
重写：与父类中的成员方法名相同，若要选择重写的方法名里的形参，那么在父类方法名中形参给指定了的默认值的子类中的就没必要写了。
重载：parent::父类方法名，作用是可以让父类方法的重新加载一次，注意重写和重载是两回事，两者是配合使用。

```
    }
}

$b = new B();

$b->fun();
$b->demo();
结果：
    bbbbbbbbbbbbb
    aaaaaaaaaa
```

总结：

子类中重载父类的方法作法：在子类里面允许重写（覆盖）父类的方法
在子类中，使用 parent 访问父类中的被覆盖的属性和方法
》parent::__construce();
》parent::fun();

常见的关键字和魔术方法

学习目标

final 关键字的应用
static 关键字的使用
单态设计模式
const 关键字
instanceof 关键字

final 关键字的应用

在 PHP5 中新增加了 final 关键字，它只能用来修饰类和方法，不能使用 final 这个关键字来修饰成员属性，因为 final 是常量的意思，我们在 PHP 里定义常量使用的是 define()函数和 const 关键字，所以不能使用 final 来定义成员属性。

final 的特性：

使用 final 关键字标识的类不能被继承；
使用 final 关键字标识的方法不能被子类覆盖（重写），是最终版本；
目的：一是为了安全，二是没有必要

final 关键字:主要用于修饰类与成员方法（函数），表示不可能有子类

static 关键字的使用

static 关键字表示静态的意思， 用于修饰类的成员属性和成员方法（即为静态属性和静态方法）。PHP 中不能修饰类

类中的静态属性和静态方法不用实例化(new)就可以直接使用类名访问。格式：

类::$静态属性　　类::静态方法

在类的方法中，不能用 this 来引用静态变量或静态方法，而需要用 self 来引用。格式：

self::$静态属性　　self::静态方法

静态方法中不可以使用非静态的内容。就是不让使用$this。

在一个类的方法中若没有出现$this 的调用，默认此方法为静态方法。

静态属性是共享的。也就是 new 很多对象也是共用一个属性。

var_dump 只查看类的属性：

在内存中 new 出来的对象，属性和方法是分开放置在内存中的，对象的名称会直接放在栈中，属性是放在堆中，利用指向引用，在对象栈名称位置中会存放着对应推中属性的地址和方法中的地址，只要方法调用才能触发。

静态在内存是不会释放的，肯定不会在动态的堆内存中，但是会在内存中静态区

举例：

```
<?php
//static（静态） 关键字的使用

class A{
    public static $name="zhangsan";
    //使用 static 修饰的方法，称静态方法，可以在外面不用实例化，直接使用类名调用。
    public static function add($a,$b){
        return $a+$b;
    }

    public static function abs($a){
        if($a<0){
            return -$a;
        }
        return $a;
    }

    public function fun(){
        echo "aaaaaaaaaaaaa";
    }

    public static function demo(){
        //在方法中可以调用其他静态属性和方法用户关键字 self,不能用户$this 调用成员,因为都不 new 对象
        echo self::add(1000,2000)."<br/>";
```

```
        echo self::$name."<br/>";

        //在非静态方法中可以调用其他静态的成员但是在静态方法不可以调用非静态的成员。
        //因为静态的成员不需要实例化成对象就在会在内存中有了,只要被标示为静态就一直有了。
    }
}

$a = new A();
echo $a->add(10,20); 可以通过new的方法实例化同样
echo $a->abs(100);
//echo $a->name;  //对象获取不了静态属性
//echo $a::name;  //对象获取不了静态属性
echo "<hr/>";

echo A::add(10,20)."<br/>"; //通过类名直接调用自己的静态方法
echo A::abs(-20)."<br/>";   //通过类名直接调用自己的静态方法
echo A::demo()."<br/>"; //通过类名直接调用自己的静态方法
echo A::$name; //通过类名直接调用自己的静态属性

A::fun(); //在一个类的方法中若没有出现$this 的调用，默认此方法为静态方法,会有提示信息，不推荐写
```

总结：

1、static 关键字:表示静态的意思: 用于修饰类的属性和方法

2、static 关键字修饰方法称静态方法，可以不用 new（实例化）就可以直接使用方法：如 类名::方法名

注意：静态方法在实例化后的对象也可以访问 //$对象名->静态方法名

3、static 关键字修饰属性称为静态属性，可以不用 new（实例化）就可以直接访问属性：如 类名::属性名

注意：静态属性在实例化后的对象不可以访问要区分以方法； //$对象名->静态属性名。

4、注意：静态属性是共享的。也就是 new 很多对象也是共用一个属性,如果每个属性到静态，那么 new 很多对象最终的属性会取决于最后一个对象，这样不符合程序的设计逻辑。

5、在静态方法中不可以使用非静态的内容。就是不让使用$this

6、在类的方法中可以使用其他静态属性和静态方法，不过要使用 self 关键字：如 【self::静态属性名】或【self::静态方法名】

7、在一个类的方法中若没有出现$this 的调用，默认此方法为静态方法。

8、一般定义为静态方法都是处理数学的计算方面,静态的方法之间是独立的，例如图片上传的类。

单态设计模式

单态模式的主要作用是保证在面向对象编程设计中，一个类只能有一个实例对象存在。（数据连接类）

问题：1、每次 new 同一类都会产生不同的对象，怎么限定不能 new 多个？
解决：设置构造方法为私有,每次调用时都会报错不能再 new 输出多个对象

问题：2、私有的初始化构造方法怎么调用同时怎么能在类中实例化对象
解决：上面已经不能通过在类外 new 方式实例化对象了，但是还要实例化对象的同时要能进行访问该方法，所以在本类中提供一个公有同时静态的方法，这样能在类的外面通过 类名::方法名调用方法的方式来实例化本类。

问题：3、每次调用方法还是会实例化对象怎么能控制每次调用时还是返回实例化同一个对象？
解决：进行访问控制，提供一个静态私有的条件进行判断。

举例：

```
<?php
//单态（单例）模式
class A{
    private static $ob=null;
    //私有化构造方法
    private function __construct(){

    }
    public static function makeA(){
        if(empty(self::$ob)){ //静态方法中使用静态属性
            self::$ob = new A();//因为在类的内部所以可以 self 替换 A 类名
        }
        return self::$ob;//返回第一次实例化的到的对象
    }

    //编写其他实用方法
    public function add(){
```

```
        //...
    }
    public function del(){
        //...
    }
}

$a1 = A::makeA();
$a2 = A::makeA();
var_dump($a1);
var_dump($a2);
```

结果：

```
object(A)[1]
object(A)[1]
```

const 关键字（类中的常量）

const 是一个在类中定义常量的关键字，我们都知道在 PHP 中定义常量使用的是”define()”这个函数，但是在类里面定义常量使用的是”const”这个关键字 const 只能修饰的成员属性（常量属性），其访问方式和 static 修饰的成员访问的方式差不多，也是使用”类名”，在方法里面使用”self”关键字。但是不用使用”$”符号，也不能使用对象来访问。

```
const CONSTANT = ‘constant value’ ;    //定义
echo self::CONSTANT;                    //类内部访问
echo ClassName::CONSTANT;               //类外部访问
```

总结：

const 关键字： 在类中修饰成员属性，将其定义成常量（不可修改的），一般要求常量名都是大写的,没有“$”符 没有其他修饰符（public）
在其他方法中使用常量： 【self::常量名】

定义格式： const 成员常量名="值";
使用：在类的方法中： echo self::成员常量名;
在类的外部： echo 类名::成员常量名;

举例

```
<?php
/*const 关键字（在类中定义常量）： 在类中修饰成员属性，将其定义为常量（不可修改的）*/

define("PI",3.14);

class Stu{
    const CLASSID="lamp94"; //在类中定义常量
```

```
    public function fun(){
        echo PI;
        echo self::CLASSID;
    }
}

$s= new Stu();
$s->fun();

echo Stu::CLASSID;//访问类中的常量
```

instanceof 关键字（亲子鉴定）

“instanceof”操作符用于检测当前对象实例是否属于某一个类的。

举例：

```
<?php
        class Person{ ... ...  }
        class Student extends Person{  ...  ...  }
        $p=new Person();
        $s=new Student();
        echo $p instanceof Student;    //结果为 false
        echo $s instanceof Student ;   //结果为 true
        echo $s instanceof Person;    //结果为 true
?>
```

魔术方法

1、克隆对象
2、类中通用的方法__toString( )
3、__call( )方法的应用
4、自动加载类
5、对象串行化

一、赋值的形式

1、值赋

```
$x = 10;
$y = $x;
$y = 20;
```

2、引用赋值（起别名）

```
$x = 10;
$y = &$x;
$y = 20;
```

3、对象引用赋值类型,所有对象的赋值属于引用赋值(起别名)

4、PHP 中对象和资源是引用类型

5、有几个 new 就有个对象，再看克隆就可以明确几个对象

二、*克隆对象

有时可能需要根据一个对象完全克隆出一个一模一样的对象，而且克隆以后，两个对象互不干扰。因为对象属于引用类型，普通的“=”号属于引用赋值，所有这样的方式不能克隆对象，所以需要使用“clone”来复制一份。

格式：　$obj = new Class();

$objectcopy=clone $obj;

魔术方法：__clone() 当执行 clone 克隆时会自动调用的方法，　主要用于解决对象中特殊属性的复制操作。

为什么出现克隆魔术方法？因为在克隆对象时，类中定义的成员一般是普通的类型，如果是在类中定义一个属于资源类型的成员，这样的克隆就出现问题。所以出现了克隆的魔术方法来解决资源类型的克隆。

三、经验

定义类时先定义方法，再定义属性，属性的产生是在你定义方法时发现在这个方法中接收一个值保存这个值，在另一个方法中需要获取该值，这样就可以定义为一个属性，这个这个属性就可以在这个类中进行共享了。类中的属性是给方法用的。

__clone 魔术方法

（只要对象中有资源和对象时无法克隆时再使用_clone 方法再次打打开资源）

举例：

```php
<?php
//定义一个文件操作类
class MyFile{
    private $fname=null;
    private $link=null;

    public function __construct($fname1){
        /*构造方法传进来的值赋给类中的属性,为在本类中
        方法能直接调用该属性(类中的属性是共享的)*/
        $this->fname = $fname1;
        $this->link = fopen($fname1,"a+"); //以 a+模式打开文件
    }

    //析构方法，关闭文件
    public function __destruct(){
        if($this->link){
            fclose($this->link);
        }
    }
```

```
    //克隆魔术方法,解决克隆资源时同一的问题
    public function __clone(){
        $this->link = fopen($this->fname,"a+");
    }

    //读取出文件内容
    public function read(){
        //$content = file_get_contents($this->fname);
        //return $content;
        rewind($this->link); //将文件指针移至首位
        return fread($this->link,filesize($this->fname));
    }

    //写入内容。
    public function write($str){
        //file_put_contents($this->fname,$str);//为啥没有追加写入文件
        fwrite($this->link,$str);
    }

}

//测试

$f = new MyFILE("a.txt");

//echo $f -> read();
//$f -> write('xoxoxo');

//克隆对象时对象中的资源没有进行克隆,所有要在调用克隆时再打开资源
$f2 = clone $f;//无法克隆资源
//unset($f);

//echo $f2->read(); //读取文件中的内容

$f2->write("hello php! "); //在文件中追加"hello php! "字串内容

var_dump($f2);
var_dump($f);
```

```
object(MyFile)[2]
  private 'fname' => string 'a.txt' (length=5)
  private 'link' => resource(4, stream)

object(MyFile)[1]
  private 'fname' => string 'a.txt' (length=5)
  private 'link' => resource(3, stream)
```

四、深析__get 方法

```
//魔术 get 方法的使用
class A{
	private $name="zhangsan";

	public function __get($m){
		echo 'dddddd<br>';
		return $this->$m;
	}
}

$a = new A();
echo $a->name; //由于 name 是私有属性，故通过魔术 get 方法间接获取而不是直接访问 name


//解决在子类中无法访问父类私有成员的方案
class A{
    private $name="zhangsan";
    public function __get($m){
        return $this->$m;
    }
}

class B extends A{
    private $age=20;
    //重写方法会被覆盖只能进行判断手动调用父类方法
    public function __get($m){
        if(isset($this->$m)){
            return $this->$m;
        }else{
            return parent::__get($m);
        }
    }
}

$b = new B();
echo $b->age;
```

五、*访问权限

举例：

```
//权限 private   测试跨类是否能访问类之间的私有成员
class A{
	private $name="zhangsan";
```

```
    public function getname(){
        return $this->name.":".$this->age;
        //return $this->age;//*报错,无权限访问子类私有的成员
        //return $this->name;//有权限访问本类私有的成员
    }
}
class B extends A{
    private $age=20;

    public function getage(){
        return $this->age.":".$this->name;
        //return $this->name;//*报错,无权限访问父类私有的成员
        //return $this->age;//有权限访问本类私有的成员
    }
}
//测试
$b = new B();
echo $b->getname();//父类
echo $b->getage();//子类
总结：1、在父类的方法中无法访问子类的私有成员
      2、在子类的方法中无法访问父类的私有成员

      跨类无法访问类之间的私有成员。
//权限 protected    测试跨类是否能访问类之间的受保护成员
class A{
    protected $name="zhangsan";
    public function getname(){
        return $this->name.":".$this->age;

    }
}

class B extends A{
    protected $age=20;

    public function getage(){
        return $this->age.":".$this->name;
    }
}

//测试
$b = new B();
echo $b->getname();
```

echo $b->getage();

总结：1、在父类的方法中可以访问子类的受保护成员

2、在子类的方法中无法访问父类的受保护成员

可以跨类访问类之间的受保护成员

六、类中通用的方法__toString( )

魔术方法“__toString()”是快速获取对象的字符串表示的最快捷方式。即当我们直接要输出一个对象时，如 echo $a,print $a，那么会自动调用的此魔术方法。

注意：__toString()方法必须返回(return string)一个字串类型的值，无参数

注意：方法名与函数名一样不区分大小写.但是到 Linux 系统中严格区分大小写

举例：

```
class Stu{
    private $name="zhangsan";
    private $age=20;

    public function __toString(){
        return  $this->name.":".$this->age;
    }
}

$s = new Stu();

//对象是不支持直接输出的,若是直接输出，会自动调用__toString 方法
echo $s;
```

七、*通过__call( )方法处理错误调用

当试图调用一个对象中不存在的方法时，就会产生错误。PHP 提供了“__call()”这个方法来处理这种情况。即当调用一个不可访问方法（如未定义，或者不可见）时，__call()会被调用。

格式：

```
mixed __call( string $name , array $arguments )
```

说明：

第一个参数$name 表示方法名，

第二参数$arguments 表示调用时的参数列表（数组类型）

手册信息：

```
    public mixed __call ( string $name , array $arguments )
```

public static mixed __callStatic (string $name , array $arguments)

当调用一个不可访问方法(如未定义，或者不可见)时，__call() 会被调用。

当在静态方法中调用一个不可访问方法（如未定义，或者不可见）时，__callStatic() 会被调用。

$name 参数是要调用的方法名称。
$arguments 参数是一个数组，包含着要传递给方法的参数。

举例：

```
class A{
    public function aa(){
        echo "aa()<br/>";
    }
    public function bb(){
        echo "bb()<br/>";
    }
    public function __call($method,$params){
        echo "error! 调用的{$method}方法不存在自动触发__call! ";
        var_dump($params);
    }

    public static function __callStatic($method,$params){
        echo "error! 调用的{$method}方法不存在自动触发__callStatic! ";
        var_dump($params);
    }
}

$a = new A();
$a->aa();
$a->bb();
$a->cc(10,20,30); //当调用一个不存在的方法时。自动调用__call 魔术方法
//静态方式当调用一个不存在的方法时。自动调用__callStatic 魔术方
A::cc(11,22,33);
```

结果：

```
aa()
bb()
error! 调用的 cc 方法不存在自动触发__call!
array (size=3)
  0 => int 10
  1 => int 20
  2 => int 30
```

error！调用的 cc 方法不存在自动触发__callStatic！

```
array (size=3)
  0 => int 11
  1 => int 22
  2 => int 33
```

八、自动加载类函数

在编写面向对象程序时，常规做法是将每一个类保存为一个 PHP 源文件。当在一个 PHP 文件中需要调用一个类时很容易就可以找到，然后通过 include(或 require)把这个文件引入就可以了。不过有的时候，在项目中文件众多，要一一将所需类的文件 include 进来，是一个很让人头疼的事。

PHP5 提供了一个__autoload()来解决这个问题。当 new 实例化一个不存在的类时，则自动调用此函数“__autoload()”，并将类名作为参数传入此函数。我可以使用这个实现类的自动加载。

在组织定义类的文件名时，需要按照一定的规则，最好以类名为中心，加上统一的前缀或后缀形成文件名：class_student.php 或 student_class.php 或 student.php（框架中必备的，但是一定在规定的目录下创建规定格式文件名的类文件，这样才能实现自动加载类文件）

补充理解：

__autoload($classname)

当实例化对象时，会指定类的名称作为参数传给此函数，会对应的目录下 include("./action/".$classname)自动加载对应名称的类文件

举例：

```php
<?php
//测试自动加载类函数（只需要指定要实例化对象名会自动加载对应的类文件）
function __autoload($classname){
    //拼装成文件名
    $filename = strtolower($classname).".class.php";
    //首先尝试判断在 action 目录下是否存在
    if(file_exists("./action/".$filename)){
        include("./action/".$filename);
    }elseif(file_exists("./model/".$filename)){
        include("./model/".$filename);
    }else{
        die("找不到{$classname}类！");
    }
}

$u = new UserAction();

//当实例化一个在本文件中不存在的类时，则会自动调用__autoload()函数，从
```

而解决需要指定路径分别导入多个类文件的问题，只需要指定要实例化对象的类名即可。

```
$u->add();
$u->select();

$s = new StuAction();
$s->add();

$um = new UserModel();
$um->demo();
```

九、*对象串行化

对象也是一种在内存中存储的数据类型，他的寿命通常随着生成该对象的程序终止而终止。有时候可能需要将对象的状态保存下来，需要时再将对象恢复。对象通过写出描述自己状态的数值来记录自己，这个过程称对象的串行化（Serialization）。以下两种情况需要将对象串行化：

1、对象需要在网络中传输时，将对象串行化成二进制串即可。

2、对象需要持久保存时，将对象串行化后写入文件或数据库。

串行化和反串行化函数：

serialize() -- 串行化，返回一个包含字节流的字符串

unserialize() -- 反串行化，能够重新把字符串变回 php 原来的对象值。

注意：串行化一个对象将会保存对象的所有属性变量和类名信息，但是不会保存对象的方法。

举例：

```
<?php
    //声明一个 Person 类，包含三个成员属性和一个成员方法
    class Person {
        public $name = "zhangsan";          //人的名字
        public $sex = "man";                //人的性别
        public $age = 20;             //人的年龄

       public function say() {          //这个人可以说话的方法，说出自己的成员属性
        echo "我的名字：".$this->name."，性别：".$this->sex."，年龄：".$this->age."<br>";
        }
    }
    $person = new Person();       //能过 Person 类创建一个对象，对象的引用名为$person
```

```
    $person_string = serialize($person);  //通过 serialize 函数将对象串行化，返回一个字符串
    file_put_contents("file.txt", $person_string);  //将对象串行化后的字符串保存到 file.txt 文件中

    //将 file.txt 文件中的字符串读出来并赋给变量$person_string
    $person_string = file_get_contents("file.txt");
    $person = unserialize($person_string);  //进行反串行化操作，形成对象$person。
    $person -> say();          //调用对象中的 say()方法，用来测试反串行化对象是否成功
```

十、对象串行化中的魔术方法__sleep( )和 __wakeup( )

解决：

处理在类中无法进行串行化的资源时自动执行的方法，类似于 clone 函数克隆对象时无法克隆资源类型的数据。

__sleep(): 是执行串行化时自动调用的方法，目的是实现资源类型属性的关闭释放等操作。(有返回值)

```
public function __sleep(){
    return array('server', 'username', 'password', 'db');
    //此串行化要保留四个属性
}
```

注意：sleep 方法需要返回一个数组，其中数组中的值是串行化时要保留的属性名

__wakeup():是在执行反串行化时自动调用的方法，目的是实现资源属性的打开（sleep 方法关闭的资源），即再次初始化。（无返回值）

举例：

```
<?php
//定义一个文件操作类
class MyFile{
    private $fname=null;
    private $link=null;
    public function __construct($fname){
        $this->fname = $fname;
        $this->link=fopen($fname,"a+"); //以 a+模式打开文件
    }

    //析构方法，关闭文件
    public function __destruct(){
        if($this->link){
            fclose($this->link);
```

```
        }
    }
    //当对象串行化时自动调用此方法
    public function __sleep(){
        //关闭资源
        if($this->link){
            fclose($this->link);
            //再重新赋 null
            $this->link=null;
        }
        return array("fname"); //并返回串行时需要保留的属性名
    }

    //当反串行化时自动调用此方法
    public function __wakeup(){
        $this->link = fopen($this->fname,"a+");
    }

    //读取出文件内容
    public function read(){
        rewind($this->link); //将文件指针移至首位
        return fread($this->link,filesize($this->fname));
    }

    //写入内容。
    public function write($str){
        fwrite($this->link,$str);
    }
}

$f = new MyFile("a.txt");

echo $f->read();

//将对象 f 进行串行化
$str = serialize($f);

echo $str;
echo "<hr/>";

//反串行化,资源类型无法进行反串行化,将会报错,所以提供 __wakeup()魔术方法
$ob = unserialize($str);
echo $ob->read();
```

十一、类型约束

类型约束可以使用的类型是：目前只支持 array 和对象（类、抽象类、接口）；若指定的一个类名，那么可传入本类及子类的对象进去。
可以使用的约束类型：（复合类型）数组 array，类名、抽象类名、接口名。

举例：

```
//定义一个函数，可以通过如下方式实现参数必须是数组
function fun(array $a){
    var_dump($a);
}

//测试
fun(array(10,20));
```

面向对象中---抽象和多态

学习目标

1、抽象类与接口
2、多态性的应用

一、抽象类：

定义：由于我们目前的原因无法将代码继续写下去，就需要我们定义一个方法是抽象方法，包含抽象方法的类称为抽象类。（常用在大项目中）

为什么有抽象类？主要为后期的项目功能的扩展,在由于技术无法实现该功能，现阶段定义为抽象类，类中定义一个抽象的方法，指明该方法的功能，给子类实现该方法的功能。这个类中的抽象方法的名字是为了给子类扩展功能覆盖的约束，你子类必须要用我父类的方法名称去实现功能。

二、抽象方法和抽象类

在 OOP 语言中，一个类可以有一个或多个子类，而每个类都有至少一个公有方法做为外部代码访问其的接口。而抽象方法就是为了方便继承而引入的。

定义抽象方法：当类中有一个方法，他没有方法体，也就是没有花括号，直接分号结束，象这种方法我们叫抽象方法，必须使用关键字 abstract 定义。

如： public abstract function fun();

定义抽象类：包含这种方法的类必须是抽象类也要使用关键字 abstract 加以声明。（即使用关键字 abstract 修饰的类为抽象类）

抽象类的特点：

1、不能实例化，也就不能 new 成对象

2、若想使用抽象类，就必须定义一个类去继承这个抽象类，并定义覆盖父类的抽象方法(实现抽象方法)。

3、抽象类目的：其实抽象类对于子类（实现类），有一个约束的作用

4、含有抽象方法的类肯定是抽象类，但是不是所有的抽象类都必须包含抽象方法。说白了就是没有定义抽象方法的类也可以定义

举例：

```
abstract class Test{
    public function demo(A $a){
        $a->fun2(10);
    }
}
```

```
abstract class A{
    public function demo(){
        echo ‘抽象类中不一定有抽象方法’;
    }
}
```

举例说明抽象的特点：

```
<?php
//抽象类的定义
abstract class A{
    public function fun1(){
        echo "class A method fun1...<br/>";
    }

    //定义抽象方法（有名无代码）
    public abstract function fun2($a);
}

class B extends A{
    //继承 A 类,覆盖父类的抽象方法(与父类同名),按照父类的需求完成功能
    public function fun2($c){
        echo "class B method fun2...<br/>";
    }
}


$b = new B();
$b->fun1();//继承后可以直接使用父类的方法
$b->fun2(10);//使用覆盖方式实现父类的功能

class Test{
    public function demo(A $a){//使用该方法需要传递 A 类,发现 A 类是一个抽象类不直接使用,需要子类进行完善抽象方法的功能
        $a->fun2(10);
    }
}
```

```
$t = new Test();
$t->demo(new B());//参数指定 A 类会报错，因为抽象类不能直接使用
结果：
class A method fun1...
class B method fun2...
class B method fun2...
```

生活中的例子：

```php
<?php
  //声明一个抽象类，要使用abstract关键字标识
  abstract class Person {
     protected $name;              //声明一个存储人的名字的成员
     protected $country;           //声明一个存储人的国家的成员
     pubic function __construct($name="", $country="china"){
        $this->name = $name;
        $this->country = $country;
     }

     //在抽象类中声明一个没有方法体的抽象方法，使用abstract关键字标识
     public abstract function say();因为现在我们不清楚是哪国人，所以不清楚说那种
                                    语言，定义一个说的抽象方法给子类去指明。
     //在抽象类中声明另一个没有方法体的抽象方法，使用abstract关键字标识
     public abstract function eat(); 因为现在我们不清楚是哪国人，所以不清楚用什么
                                    吃饭，所以定义一个吃的抽象方法给子类去指明。
     //在抽象类中可以声明正常的非抽象的方法
     public function run(){
        echo "使用两条腿走路<br>"; //有方法体，输出一条语句
     }              现在定义的时候就知道人都是用两条腿走路
  }                 的，直接在人类中实现无需定义为抽象方法
```

子类实现父类的抽象方法

```php
<?php
  //声明一个类去继承抽象类Person
  class ChineseMan extends Person {
     //将父类中的抽象方法覆盖，按自己的需求去实现
     public function say() { 现在知道是中国人，就需要去实现说话的方法
        echo $this->name."是".$this->country."人，讲汉语<br>";   //实现的内容
     }
     //将父类中的抽象方法覆盖，按自己的需求去实现
     public function eat() { 现在知道是中国人，就需要去实现吃饭的方法
        echo $this->name."使用筷子吃饭<br>";              //实现的内容
     }
  }
  //声明另一个类去继承抽象类Person
  class Americans extends Person {
     //此处省略 ... ....
  }
  $chineseMan = new ChineseMan("高洛峰", "中国"); //将第一个Person的子类实例化对象
  $americans = new Americans("alex", "美国");        //将第二个Person的子类实例化对象
  $chineseMan -> say();    //通过第一个对象调用子类中已经实例父类中抽象方法的say()方法
  $chineseMan -> eat();    //通过第一个对象调用子类中已经实例父类中抽象方法的eat()方法
  $americans -> say();     //通过第二个对象调用子类中已经实例父类中抽象方法的say()方法
  $americans -> eat();     //通过第二个对象调用子类中已经实例父类中抽象方法的eat()方法
```

三、接口

1、程序与生活

程序中：父类中只光提出设想不去实现把所有的方法都是抽象的方法，给子类实现并且子类只能继承一个类，逼得子类将继承过来的方法需要重写才能实例化子类使用方法，这就是特殊的抽象类。但是转变成接口后我们就不需要继承了，因为继承只能继承一个父类，为了更加灵活使用接口所以使用 implements 关键字来延续使用接口。

生活中：子辈继承父辈的优良传统，没有实际性的东西继承下来，需要子类去实现；领导讲话只提出设想，让员工去实现，接口类似于有种领导升级的感觉统统活都指挥员工去做。

2、接口是特殊的抽象类（抽象类中的方法全抽象方法,不再用 class 修饰类名）

3、为了给这个特殊的抽象类方便记忆我们就给它起个名称为接口，使用 interface 关键字修饰类，类中所有的抽象方法前的 abstract 修饰关键字统统省掉，并且方法必须也是抽象方法后面不带花括号，否则发生冲突会报错。

4、接口技术

PHP 与大多数面向对象编程语言一样，不支持多重继承，也就是说每个类只能继承一个父类。为了解决这个这个问题，PHP 引入了接口，接口的思想是指定了一个实现了该接口的类必须实现的一系列函数。

如果在一个抽象类中包含的只有抽象方法，可以将其定义为 interface(接口)，用 implements(实现)关键字使用它。

定义格式：（接口中只能**定义常量**和**抽象方法**）

```
interface  接口名称{
            //常量成员     (使用 const 关键字定义)
    [public]//抽象方法     (不需要使用 abstract 关键字)
}
```

使用格式：（实现 N 多个接口）

```
class   类名   implements   接口名 1,接口名 2{ ... ...  }
```

当一个类在继承了一个接口后，它必须实现即覆盖该接口的所有方法才可以实例化使用，否则即为抽象类，接口不是继承而是实现

举例：

```php
<?php
//接口的定义
//定义一个接口 A，内有两个方法（抽象方法）
interface A{
    public function fun1();
    public function fun2($a);
}
```

```php
//定义一个类 B 只实现部分接口 A
abstract class B implements A{
    //在接口中可以定义常量和抽象方法
    const PI=3.14;
    //只实现接口中的一个抽象方法 fun1
    public function fun1(){
        echo "class B method fun1...<br/>";
    }

//1、B 类中没有实现接口 A 类中的抽象方法 fun2,这个 B 类必须还定义为抽象类
//2、如果在 B 类中全部实现了 A 类中的抽象方法此时属于完整类无需将 B 类定义为抽象类就直接实例化 B 类。
}

//C 类继承时需要 B 类接着实现 B 类中未完成的抽象方法 fun2
class C extends B{
    public function fun2($c){
        echo "class C method fun2...<br/>";
    }
}
//实现所有的抽象方法（实现接口）,最终实例化 C 类才能使用
$c = new C();
$c->fun1();
$c->fun2(10);
```

四、抽象类与接口的区别

定义：
1、抽象类表示该类中可能已经有一些方法的具体定义。
2、接口就仅仅只能定义各个方法的界面，不能有具体的实现代码在成员方法中。

用法：
1、抽象类是子类用来继承的，当父类已有实际功能的方法时，该方法在子类中可以不必实现。
2、实现一个接口，必须实现接口中所有定义的方法，不能遗漏任何一个。

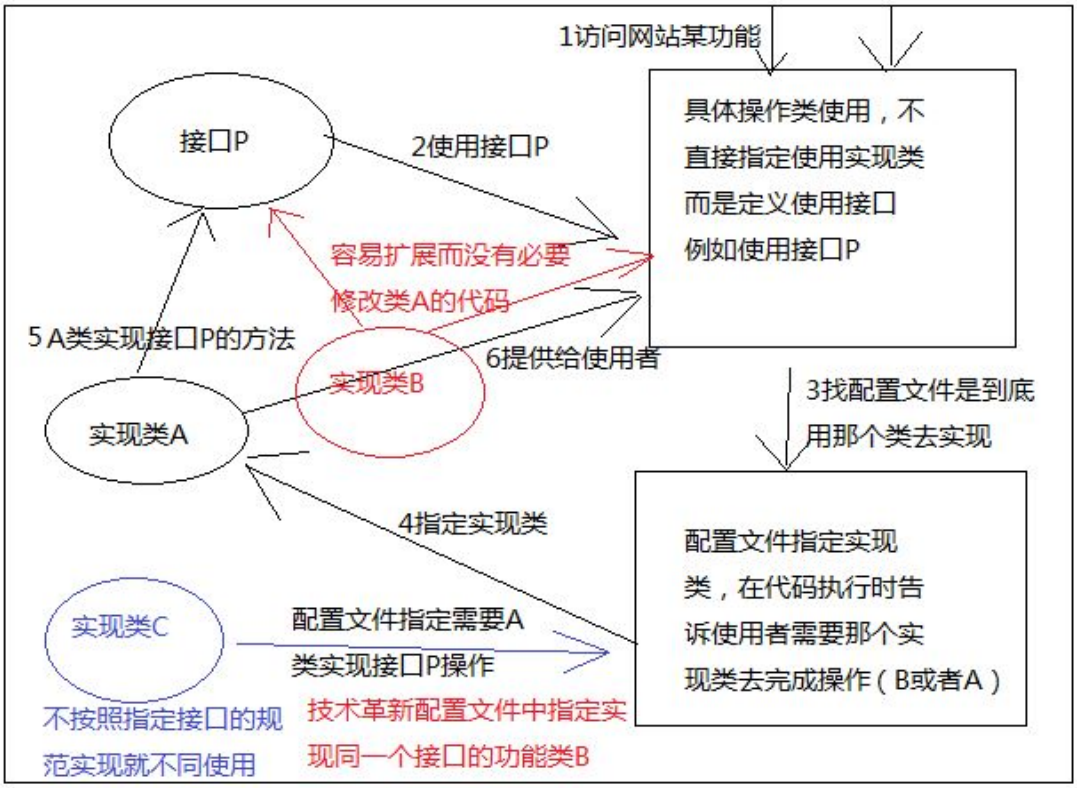
总结：有一种类里面部分的方法和属性能用不需要子类进行覆盖，这种的类是抽象类（半成品）；另一种类里面只能是常量和方法，定义的方法全都是抽象的不能使用，需要子类将父类的方法全部实现才能使用，这种类是接口（全部是产品的设想，有名无代码）。

五、多态性的应用

对象的多态性是指在父类中定义的属性或行为被子类继承之后，可以具有不

同的数据类型或表现出不同的行为。这使得同一个属性或行为在父类及其各个子类中具有不同的语义。

例如："几何图形"的"绘图"方法，"椭圆"和"多边形"都是"几何图"的子类，其"绘图"方法功能不同。



生活与程序：

生活中打印黑白和彩色图片，人就是配置文件，实现类就是墨汁盒（黑白、彩色）一定要按照打印机的插槽规范生产的墨汁盒，使用不同的墨汁盒（实现类）打印出不同的图片效果（实现类的功能，不按照接口的规定[抽象方法名同名]去实现编写实例类 C，也不能使用），不按照打印机插槽的规范去生产墨汁盒同样打印机也不能使用，要用那种墨汁盒主要取决于人，使用那个实现类主要取决于配置文件的指定。

总结：多态

*多态(使用方式)：对于同一个方法，传入不同对象，实现了不同的效果，这个就是多态的意思， 需要使用的技术：继承或实现，方法的覆盖（重写）。

同一个功能在子类进行升级-----简单的多态（多态不一定是接口）

举例（多态--接口版使用）：

推荐使用：对子类可以约束作用

推荐使用：可以实现多个接口

```
<?php
    header("Content-type:text/html;charset=utf-8");
    //模拟多态的使用（接口约束作用）
```

```php
//==========主板厂商======

//为了规范接口让其他厂商生产适合接口的其他设备

//主板上的 pci 插槽规范接口
interface PCI{
	public function start();//加电
	public function stop();//停止
}

//======主板驱动程序=====
//围绕 PCI 的启动程序
class mainBorad{
	public function running(PCI $pci){
		$pci -> start();//启动
		$pci -> stop();//停止
	}

}

//======声卡厂家===按照 PCI 实现接口生产===
class soundCard implements PCI{
	public function start(){//启动
		echo "声卡启动...<br/>";
	}
	public function stop(){	//停止
		echo	"声卡停止...<br/>";
	}
}

//====================================

//======网卡厂家====按照 PCI 实现接口生产==
class networkCard implements PCI{
	public function start(){//启动
		echo "网卡启动...<br/>";
	}
	public function stop(){	//停止
		echo	"网卡停止...<br/>";
	}
}
//====================================
//计算机系统程序
```

```
$md = new mainBorad();      //主板程序
$sd = new soundCard(); //声卡
$nc = new networkCard();//网卡

$md->running($sd); //声卡接入主板程序使用
$md->running($nc); //网卡接入主板程序使用
```

举例（多态--抽象类版使用）：

不怎么推荐：占用了继承口

```
<?php
//模拟多态的使用（抽象类版）

//========主板厂商========
//主板上的 pci 插槽规范
 abstract class PCI{
    public abstract function start(); //启动
    public abstract function stop();   //停止
}

//主板的驱动程序
class mainBoard{
    public function running(PCI $pci){
        $pci->start();
        $pci->stop();
    }
}
//========================

//======声卡厂家=======继承是单向的，占用了继承口==局限又浪费继承
class soundCard extends PCI{
    public function start(){//启动
        echo "声卡启动...<br/>";
    }
    public function stop(){   //停止
        echo   "声卡停止...<br/>";
    }
}

//==================================

//======网卡厂家=======继承是单向的，占用了继承口==局限又
class networkCard extends PCI{
    public function start(){//启动
        echo "网卡启动...<br/>";
```

```
    }
    public function stop(){    //停止
        echo    "网卡停止...<br/>";
    }
}

//==================================


//系统程序

$md = new mainBoard();      //主板程序
$sd = new soundCard(); //声卡
$nc = new networkCard();//网卡

$md->running($sd); //声卡接入主板程序使用
$md->running($nc); //声卡接入主板程序使用
```

举例（多态--普通类版使用）：

不推荐使用：没有起到约束作用（不按照接口生产）

```
<?php
//模拟多态的使用（普通类版不推荐）

//========主板厂商========
//主板上的 pci 插槽规范
class PCI{
    public function start(){ //启动
        echo "默认启动...<br/>";
    }
    public function stop(){    //停止
        echo "默认停止...<br/>";
    }
}

//主板的驱动程序
class mainboard{
    public function running(PCI $pci){
        $pci->start();
        $pci->stop();
    }
}
//=========================

//======声卡厂家====================
```

```
class soundcard extends PCI{
	public function start(){//启动
		echo "声卡启动...<br/>";
	}

	//生产 stop2()不会报错继续使用声卡,没有起到约束的作用
	public function stop2(){	//停止
		echo	"声卡停止...<br/>";
	}
}

//==================================


//======网卡厂家=====================
class networkcard extends PCI{
	public function start(){//启动
		echo "网卡启动...<br/>";
	}
	public function stop(){	//停止
		echo	"网卡停止...<br/>";
	}
}

//==================================

//系统程序

$md = new mainBoard();		//主板程序
$sd = new soundcard();  //声卡
$nc = new networkcard();//网卡

$md->running($sd); //声卡接入主板程序使用
$md->running($nc); //声卡接入主板程序使用
```

六、类中的特殊写法---连贯操作

```
class A{
	private $m=0;
	public function add($i){
		$this->m+=$i;
		//返回对象连贯操作
		return $this;
	}
```

```
        //对象直接输出
        public function __toString(){
                return "数值：".$this->m;
        }
}

$a=new A();

echo $a->add(10)->add(20)->add(40);
```

结果：
数值：70