PHP 实践之路 - 面向对象篇课程

主讲：孙胜利

新浪微博：@私房库

微博主页：http://weibo.com/sifangku

第一章 面向对象语法

本课导语：

当向汉字、组词都没有学过的同学直接讲写作文...肯定是行不通的。

学习面向对象编程也是如此、先把关于面向对象的语法知识学习了再去用这些面向对象的语法规则来实际的编程才是科学正确的学习方法。（我们会用一些通俗易懂的例子来学习这些语法的）

注：

在这一章学完之前，大家不用整天想着怎么用这些语法知识去真正的编程，等把这章语法学完我们给大家举个小例子让大家有点直观的感受、后面的项目也会用到这些语法知识的，以达到巩固的目的所以不用自扰！

一、简介

面向对象编程（Object Oriented Programming，OOP）提供了一种新的语法，同时也是一种看待问题的新方式，使得我们的程序变得更加灵活、易于扩展、易于重用。我们接下来要学习的面向对象的语法知识都是为这些原则服务的！

面向过程编程我们面对的重点是怎么去做，如何一步一步的去做，也就是我们一直面对的是程序的具体过程，陷入了编程的泥潭，类似于我们本身就是一个企业的员工，一步步的去做各种任务，上面让我做啥我就做啥！

面向对象编程我们面对的是一个个对象，这些对象帮我们把一个个事情做完，有点类似于我是企业的老

板或者管理者，面对的是一个个部门，具体的工作是有这些部门完成的，编程的时候更具有统筹感！

二、类

面向对象编程从“类”开始

1、什么是类？

一个类是对于具体事物的抽象定义，大家可以把类看成是**“某类事物”**的类的意思，即对某类具体事物的抽象定义。

比如“人类”，它是一个抽象的定义。而不是具体的某个人。

再比如：类就像房子的蓝图，具体的房子会按照这个蓝图进行建造。

2、如何定义类？

每个类的定义都以关键字 `class` 开头，空格后面跟着类名，后面再跟着一对花括号

例：

```
class Humanity {}
class Room {}
```

类名称的规则：

1）类名可以是任何非 PHP 保留字的合法标签。一个合法类名以字母或下划线开头，后面跟着若干字母，数字或下划线

2）类名称不能是保留关键字

3）类名并不区分大小写，但是极度不建议不区分类名称的大小写！

4）我们习惯以大写字母开头，驼峰式命名的规则（大驼峰），第一个字母以及后续每个单词的首字母大写

3、类成员

定义类时那个大括号里面可以包含属于该类的常量、变量（称为"属性"）、函数（称为"方法"），我们可以称作这些是该类的成员。

1）变量(属性)

1>类的变量成员叫做"属性"，或者叫"字段"、"特征"

2>声明属性时必须用一个关键字指明其"可见性"（ public、protected、 private 三选一开头，具体意义我们以后再说，暂且我们使用 public 即可）

3>属性可以初始化，但是初始化的值必须是常数，不能是表达式的结果，这里的常数是指 PHP 脚本在编译阶段时就可以得到其值，而不依赖于运行时的信息才能求值

4>变量名我们一般采用"小驼峰"或者用"_"分隔单词

例：

```
class Humanity {public $name;public $sex;public $iq=10;}
class Humanity {
    public $name;
    public $sex;
    public $iq=10;
}
```

2）常量

1>可以把在类中始终保持不变的值定义为常量。

2>常量的值必须是一个定值，不能是变量，类属性，数学运算的结果或函数调用。

3>在定义和使用常量的时候不需要使用 $ 符号 且 常量名称我们一般大写。

const 常量名称 = 常量值;

例：

```
class Humanity {
    public $name;
    public $sex;
    public $iq=10;
    const BIRTHPLACE='Earth';
}
```

3）方法(函数)

1>类中定义的函数与类外相同，可以接收参数、有返回值等

2>变量和常量声明应该放在方法前面

3>方法前面也可以设置可见性（ public、protected、 private、省略 四选一开头），默认不写则是 public

4>PHP 里的函数不区分大小写，所以类里面的方法名也一样，但是极度不建议你不区分！

5>方法名我们一般采用"小驼峰"或者用"_"分隔单词

例：

```
class Humanity {
    public $name;
    public $sex;
    public $iq=10;
    const BIRTHPLACE='Earth';
    function eat($food){
        echo "正在吃{$food}";
    }
}
```

注：关于这些变量、常量、方法有什么用？怎么用？我们后面再说，现在记住这些就 OK 了！

三、对象

前面我们说了类是对于具体事物的抽象定义，就像是规划一个房间的蓝图。

那么怎么根据某个蓝图来建造出对应的房间呢？对应的房间在程序里面我们叫什么呢？

1、怎么创建？

new 对应的类名称();

例：

```
new Humanity();
```

2、创建出来的东西我们称作什么？

由类创建出来的东东，我们称作这个东东是 该类的对象（或者称作 该类的**实例**即实际案例）！

并且我们可以把这个对象保存在某个变量里！

$object=new 对应的类名称();

例：

```
$xiaoZhao=new Humanity();
```

现在明白对象是什么了吧？类比喻成模具，对象就是根据模具生产出来的器件

3、->对象运算符

使用-> 访问/设置 非静态属性 或者 调用方法

默认我们类里设置的变量都是非静态属性（类常量除外）！至于什么是静态属性？以及 类常量以及静态属性的访问我们以后再讲！

注：访问**非静态属性**的时候无需属性变量开头的$

例：

```
$xiaoZhao->name='zhaokuangyin';//设置
var_dump($xiaoZhao->name);//访问，我们直接将$xiaoZhao 对象里的 name 属性输出
$xiaoZhao->eat('苹果');//调用$xiaoZhao 对象里的 eat 方法
$xiaoZhao->id='911';//设置新的属性（完全没问题）
var_dump($xiaoZhao->id);//输出刚添加给$xiaoZhao 对象的 id 属性
```

PS：我们可以根据某个类创建出无数个对象（只要你有这个需要）,比如你根据 Humanity 类再创建出一个对象试试呢！

4、$this

在类的内部（准确的说是方法内）如何访问属性以及方法呢？

我们有一个特殊的变量$this，**一个类中$this 变量总是指向该类的当前实例**。

所以在类的方法内，我们可以使用 这种方式 $this->属性/方法名

提示：当前实例指的是该方法当前是被哪个实例调用的，那么$this 就代表那个实例！

5、常数不能通过对象访问

那怎么办呢？以后再讲！

阶段练习：多建立几个类，多创建几个对象，多设置几个对象中的属性试试！

6、遍历对象

对象和其他的数据类型一样、本质都是在程序运行过程中在内存中存放数据的。对象和数组一样属于复合类型（里面可以放很多数据这些数据的类型可以是任意的数据类型），那么数组我们可以遍历，对象呢？可不可以呢？

PHP 5 可以用 foreach 语句。默认情况下，所有可见属性都将被用于遍历。

其他遍历的方法我们很少接触，可以不做了解！

疑问，为啥遍历出来的没有方法？

实际上对象是属性的集合，由同一个类生成的不同对象拥有各自不同的属性，类的代码空间中方法区域的代码是共享的！

其实方法只是一段需要执行的一段代码，它并不保存数据，方法需要用到的数据其实都是保存在属性里面的！

类常量也是同理！

7、检测一个对象是不是属于一种特定的类型

```
$object instanceof 类名称
```

8、获取某个对象是由哪个类创建出来的

```
string get_class ([ object $obj ] )
```

返回对象的类名,如果在对象的方法中调用则 obj 为可选项。

9、对象和引用

在 php5 的对象编程经常提到的一个关键点是“默认情况下对象是通过引用传递的”。但其实这不是完全正确的。

php 的引用是别名，就是两个不同的变量名字指向相同的内容。

那对象默认是怎么传递的呢？

从 php5 开始，一个对象变量已经不再保存整个对象的值。只是保存一个标识符来访问真正的对象内容。 当对象作为参数传递，作为结果返回，或者赋值给另外一个变量，另外一个变量跟原来的不是引用的关系，只是他们都保存着同一个标识符的拷贝，这个标识符指向同一个对象的真正内容。

但是：结论是相似的，所以我们就不用扣字眼了，知道这么回事就 OK 了！

10、对象复制

在多数情况下，我们并不需要完全复制一个对象来获得其中属性。只有在编写桌面应用程序可能会用到，那么这里我们就简单说一下！

对象复制可以通过 clone 关键字来完成

```
$copy_of_object = clone $object;
```

11、对象比较

当使用比较运算符（==）比较两个对象变量时，比较的原则是：如果两个对象的属性和属性值都相等，而且两个对象是同一个类的实例，那么这两个对象变量相等。

而如果使用全等运算符（===），这两个对象变量一定要指向某个类的同一个实例（即同一个对象）。

12、对象序列化

所有 php 里面的值都可以使用函数 serialize()来返回一个包含字节流的字符串来表示。

unserialize()函数能够重新把字符串变回 php 原来的值。

注：

1>序列化一个对象将会保存其所有变量，但是不会保存对象的方法，只会保存类的名字。

2>为了能够 unserialize()一个对象，这个对象的类必须已经定义过。

13、销毁对象

某个对象使用完确定程序运行完之前无需再使用可以使用 unset 销毁该对象，虽然脚本结束对象会被自动删除但是这样可以更及时的释放该对象所占用的内存！

四、构造函数、析构函数

1、构造函数

类里面可以设置一个特殊的函数，这个函数在该类被创建实例时会自动调用执行，这个特殊的函数我们称作"构造函数"，我们一般用它来做一些初始化的工作！

注：

1）函数名称是__construct()

2）可以传参数！

3）构造函数不可以使用 return 语句，因为没有意义，根本不会执行！

作用：具有构造函数的类会在每次创建新对象时先调用此方法，所以非常适合在使用对象之前做一些初始化工作。

例：

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

2、析构函数

类里面可以设置一个特殊的函数，这个函数在该类的实例被销毁时自动调用，这个特殊的函数我们称作"析构函数"。

注：

1）函数名称是__destruct()

2）这个函数会在我们手动 unset 销毁该类的实例时或者脚本执行结束时 PHP 自动释放变量所用的内存时执行

五、继承

1、继承在现实生活中很好理解那么面向对象里的继承是什么意思呢？

现实生活中孩子是从父母那里继承各种资源，那么面向对象里面有什么可以继承呢？

我们可以在定义一个类的时候，指定其从另外一个**已经存在类**派生出来！这时已经存在的类我们称作"父类"，根据其派生出来的类我们称作是其父类的"子类"，**子类具有父类一样的属性和方法，并且还可以定义自己的属性及方法**！

```
class 类 B extends 类 A {


}
```

“类 B”相对于“类 A”来说是“类 A”的**子类**、“类 A”是“类 B”的**父类**

例：

```
class Student extends Humanity {
    public $grade;
    public function study($subject){
        echo "{$this->name}正在学习{$subject}";
    }
}
$xiaoMing=new Student('小明','男');
$xiaoMing->study('PHP');
```

注：继承的次数是没有限制的，多个类可以具有相同的父类，总之随便怎么继承都可以。

例：

```
class ClassLeader extends Student {
    public $position;
    public function work($task){
```

```
            echo "{$this->name}正在{$task}";
        }
    }
```

2、对于子类我们可以设置新的属性以及方法

3、如果我们想改变父类的方法应该怎么办？

直接去修改父类的代码？肯定不行，这办法太不科学，那怎么办呢？

方法重写

我们可以在子类里面包含和父类同名的方法，这种情况我们称作方法的重写！

注：重写的方法的参数必须和父类中对应方法的参数数量一致，构造方法除外！

如果我在写代码的时候不希望父类中的某些方法能够被子类重写怎么办？

在定义方法前加上 final

如果一个类被声明为 final，则不能被继承

属性不能被定义为 final，只有类和方法才能被定义为 final

比如在 Student 类里面重写 eat 方法：

```
public function eat($food){
        echo "{$this->name}正在快速的吃{$food}";
}
```

如果在父类的 eat 方法前面加上 final 试试呢？

5、在子类中如何访问被自己覆盖了的父类方法

以后再讲！

6、string get_parent_class ([mixed $obj])

返回对象或类的父类名,如果在对象的方法内调用，则 obj 为可选项。

六、可见性（访问控制）

可见性控制着类里面的哪些属性或者方法在哪里可以访问在哪里不能访问

1、访问控制，是通过在属性或方法前面添加关键字

public（公有），protected（受保护）或 private（私有）来实现的。

1>被定义为公有的类成员可以在任何地方被访问。

2>被定义为受保护的类成员则可以被其自身以及其子类和父类访问。

3>被定义为私有的类成员则只能被其定义所在的类访问。

例：

```
protected function chew($food){
   echo '{$food}咀嚼完成...';
}
private $money;
```

总结：

1>当一个方法或者属性，我们只希望其能够在类内部被访问则可以设置其可见性为 private

2>当一个方法或者属性，我们希望其能够在类内部或者子类里面被访问则可以设置可见性为 protected

3>如果不希望限制则 public 即可

七、范围解析操作符（::）

范围解析操作符或者更简单地说是一对冒号‘::’

作用：

可以用于访问**静态成员**、**类常量**、访问**被覆盖类**中的方法。

类外部：

类名::常量名、静态成员

类内部：

self::常量名、静态成员

parent::常量名、静态成员、覆盖类中方法

注意：**parent** 一般用于在**子类中**访问**父类中**被**子类覆盖**的方法，当一个子类覆盖其父类中的方法时，PHP 不会调用父类中已被覆盖的方法。是否调用父类的方法取决于子类，这时就是范围解析操作符发挥作用的时候。

八、静态成员

1、在声明成员的时候只需要在其可见性前或者后加上 static 则该成员就是静态成员

可见性 **static** 变量名[=值];

可见性 **static function** 方法名(...){

}

2、将变量声明为静态的作用？

1>类的静态成员和函数里的静态变量相似

2>静态变量不能被通过对象访问到，直接通过类名称（在类内部通过 self 等）和范围解析操作符来访问

3>既然不能通过类访问那么它就和该类的所有实例没有关系了！

4>那到底有啥用呢？

```
class Test {
  public static $counter=0;
  function __construct(){
     self::$counter++;
  }
}
new Test();
new Test();
new Test();
echo '<p>'.Test::$counter.'</p>';
new Test();
echo '<p>'.Test::$counter.'</p>';
```

3、将方法声明为静态方法的作用？

不通过实例化即可访问这些方法，当然通过实例化的对象也能访问

由于静态方法不需要通过对象即可调用，所以 $this 在静态方法中不可用。

例：

```
static public function hello(){
    echo '我是来自地球的人类';
}
```

总结：

1>声明类属性或方法为静态，就可以不实例化类而直接访问、共享其值。

2>静态属性不可以由类的实例通过 -> 操作符来访问（但静态方法可以）。

3>静态属性只能被初始化为文字或常量，不能使用表达式。所以可以把静态属性初始化为整数或数组，但不能初始化为另一个变量或函数返回值，也不能指向一个对象。

九、抽象类和抽象方法

前面讲了继承，比如有一个定义动物的类，但是动物这个类实在是太抽象了，里面包含的可能太多，我们我们在定义动物类的时候只能把所有动物的共同点放在这个动物类里面，

我们希望这个类以后被其他的更加具体的类继承然后再去实例化，比如再定义一个人这个类去继承动物类，这样可以细化的去定义人的各种行为与功能，这样才能有意义，否则直接实例化那个动物类，那么实例化出来的东西到底是什么动物呢？

那么问题来了

这时我们就会有一个想法，既然有时候实例化父类没有意义，那么可不可以从语法上规定在写父类的时候规定这个父类不能被直接实例化，必须被其他类继承之后再去实例化子类？有没有这种语法呢？

答案是肯定的当然有！

1、规定一个类为'抽象类'

```
abstract class 类名称 {


}
```

说白了我们定义了一个抽象类，那么这个抽象类的使命就是用来被其他类继承扩展的！

2、还有一个概念叫抽象方法

```
abstruct [可见性] function 方法名([...]);
```

不需要定义方法的具体功能，具体功能有子类来完成

注：

1>在扩展类里的抽象方法时，可见性必须高于或等于抽象方法定义的可见性

```
private
protected
public
```

2>参数要一致

3>类里面如果有抽象方法那么这个类本身必须要定义为抽象类，换句话说，只有抽象类里才可以包含抽象方法

4>抽象类不一定要有抽象方法

十、接口

抽象类我们的目的是为了写一个类用于以后更细的类去扩展细化它，那么有没有一种语法可以直接规定一个类应该具有哪些方法？

1、接口，它更像是对一个类的方法的规划或者说方法的合约（把接口想象成军令状）

```
interface 接口名称 {
    public function 方法名([参数]);
}
```

注：

我们习惯把接口名称以 i 开始

接口中的只定义方法

接口中所有方法必须是 public 的

接口中的方法仅仅是方法名称和参数而不包括具体实现

例：

```
interface iHumanity {
    public function eat($food);
    public function say($content);
    public function sleep($time);
}
```

2、怎么把一个类和接口关联

```
class 类名称 implements 接口名称 {

}
```

注：

1>如果类实现了接口那么这个类必须实现接口中定义中的所有方法，否则将会报语法错误！

2>类要实现接口，必须使用和接口中所定义的方法完全一致的方式。否则会导致致命错误。

3>类可以实现多个接口

```
class 类名称 implements 接口 1,接口 2 {
}
```

实现多个接口时，接口中的方法不能有重名。

4>接口也可以继承其他接口（多个一起继承也可以，用逗号隔开）。

5>接口中也可以定义常量。接口常量和类常量的使用完全相同，但是不能被子类或子接口所覆盖。

6>instanceof 可以检测一个变量是否实现了一个接口

十一、traits

自 PHP 5.4.0 起，PHP 实现了代码复用的一个方法，称为 traits

在很多个类中如果有一些方法重复时，我们能做的只能是把这些重复的方法放在他们共同的父类中！但是如果这些类已经有父类了怎么办？

没有办法使一个类继承同时多个类，这样我们对于这些需要重复使用的方法就没办法自由的

编写了，如果给他们不同的父类都加上这些方法，就会有很多重复的代码，不利于后期维护！

1、那怎么办？PHP 这门语言的开发者有没有为我们考虑到这个问题呢？

当然有！

trait trait 名称 {

需要重复使用的方法可以放在这里，和类里面编写的**方法**一样

}

注：trait 名称最好以 t 开头

例：

```
trait tKnife {
    public function pencil($pencil){
        echo $pencil.'ok!';
    }
}
```

2、那么这个 trait 应该怎么使用呢？

在类的内部使用：

use trait 名称

即可使用被 use 的 trait 里面的各种方法！是不是更加的灵活呢？

是不是有点类似于 include 呢？

3、优先级

从基类继承的成员被 trait 插入的成员所覆盖。优先顺序是来自当前类的成员覆盖了 trait 的方法，而 trait 则覆盖了被继承的方法。

4、通过逗号分隔，在 use 声明列出多个 trait，可以都插入到一个类中。

use trait 名称 1,trait 名称 2;

5、冲突的解决

如果两个 trait 都插入了一个同名的方法，如果没有明确解决冲突将会产生一个致命错误。

1>为了解决多个 trait 在同一个类中的命名冲突，需要使用 insteadof 操作符来明确指定使用冲突方法中的哪一个。

例：

```
trait A {
    public function smallTalk() {
        echo 'a';
    }
    public function bigTalk() {
        echo 'A';
    }
}
trait B {
    public function smallTalk() {
        echo 'b';
```

```
        }
        public function bigTalk() {
            echo 'B';
        }
    }
    class Talker {
        use A, B {
            B::smallTalk insteadof A;
            A::bigTalk insteadof B;
        }
    }
```

2>那我依然想方法被替代掉的那个方法怎么办?

```
    class Aliased_Talker {
        use A, B {
            B::smallTalk insteadof A;
            A::bigTalk insteadof B;
            B::bigTalk as talk;
        }
    }
```

注：在使用 as 前必须先解决掉冲突问题

6、修改方法的访问控制

使用 as 语法还可以用来调整方法的访问控制。

```
trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
    }
}
// 修改 sayHello 的访问控制
class MyClass1 {
    use HelloWorld { sayHello as protected;}
}
// 给方法一个改变了访问控制的别名
// 原版 sayHello 的访问控制则没有发生变化
class MyClass2 {
    use HelloWorld { sayHello as private myPrivateHello;}
}
```

7、从 trait 来组成 trait

和类能够使用 trait 一样， trait 也能够使用 trait。

在 trait 定义时通过使用一个或多个 trait，它能够组合其它 trait 中的部分或全部成员。

```
trait Hello {
        public function sayHello() {
                echo 'Hello ';
        }
}
trait World {
        public function sayWorld() {
                echo 'World!';
        }
}
trait HelloWorld {
        use Hello, World;
}
class MyHelloWorld {
        use HelloWorld;
}

$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
```

8、Trait 的抽象成员

为了对使用的类施加强制要求，trait 支持抽象方法的使用。

```
trait Hello {
        public function sayHelloWorld() {
                echo 'Hello'.$this->getWorld();
        }
        abstract public function getWorld();
}
class MyHelloWorld {
        private $world;
        use Hello;
        public function getWorld() {
                return $this->world;
        }
        public function setWorld($val) {
                $this->world = $val;
        }
}
```

9、Trait 的静态成员

Traits 里可以定义静态成员和静态方法。

10、属性

Trait 同样可以定义属性。

```
trait PropertiesTrait {
        public $x = 1;
}
class PropertiesExample {
        use PropertiesTrait;
}
$example = new PropertiesExample;
$example->x;
```

如果 trait 定义了一个属性，那类将不能定义同样名称的属性，否则会产生一个错误。

十二、魔术方法

PHP 将所有以 __（两个下划线）开头的类方法保留为魔术方法。所以在自定义类方法时建议不要以 __ 为前缀。

每个魔术方法在类中都有特殊的作用：

1、__construct()

2、__destruct()

3、__set()

public void __set (string $name , mixed $value)

在给不可访问属性赋值时，__set() 会被调用。

必须被声明为 public

4、__get()

public mixed __get (string $name)

读取不可访问属性的值时，__get() 会被调用。

必须被声明为 public

5、__isset()

public bool __isset (string $name)

当对不可访问属性调用 isset() 或 empty() 时，__isset() 会被调用。

必须被声明为 public

6、__unset()

public void __unset (string $name)

当对不可访问属性调用 unset() 时，__unset() 会被调用。

必须被声明为 public

7、__call()

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

在对象中调用一个不可访问方法时，__call() 会被调用。

$name 参数是要调用的方法名称。

$arguments 参数是一个枚举数组，包含着要传递给方法 $name 的参数。

必须被声明为 public

8、__callStatic()

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

用静态方式中调用一个不可访问方法时，__callStatic() 会被调用。

$name 参数是要调用的方法名称。

$arguments 参数是一个枚举数组，包含着要传递给方法 $name 的参数。

必须被声明为 public

9、__sleep()

serialize() 函数会检查类中是否存在一个魔术方法 __sleep()。如果存在，该方法会先被调用，然后才执行序列化操作。此功能可以用于清理对象，并返回一个包含对象中所有应被序列化的变量名称的数组。如果该方法未返回任何内容，则 NULL 被序列化，并产生一个 E_NOTICE 级别的错误。

10、__wakeup()

与之相反，unserialize() 会检查是否存在一个 __wakeup() 方法。如果存在，则会先调用 __wakeup 方法，预先准备对象需要的资源。

12、__invoke()

当尝试以调用函数的方式调用一个对象时，__invoke() 方法会被自动调用。

13、__set_state()

static object __set_state (array $properties)

自 PHP 5.1.0 起当调用 var_export() 导出类时，此静态 方法会被调用。

本方法的唯一参数是一个数组：

包含按 array('property' => value, ...) 格式排列的类属性。

14、__clone()

对象复制

对象复制可以通过 clone 关键字来完成（这将调用对象的 __clone() 方法）。

对象中的 __clone() 方法不能被直接调用。

15、__debugInfo()

array __debugInfo (void)

这种方法是通过的 var_dump（）输出对象时来获取应显示的属性调用。如果该方法不是一个对象上定义，那么所有的公共，保护和私有属性将被显示。

此功能在 PHP5.6.0 添加的。

十三、类型约束

我们在编写方法的时候，有时会有这样的需求：

我们编写的方法接受的参数有目的性，比如只希望接受一个指定类型的对象！

那么有没有什么办法从语法上来解决这个问题，而不是通过代码来判断呢？

类型约束

在定义方法时，在参数前面加上所期望的类型，与参数用空格隔开。

注：

1>类型约束不能用于标量类型如 int 或 string。Traits 也不允许。

2>参数可以指定必须为对象（指定类的名字）、接口、数组、callable。

2>可以使用 NULL 作为参数的默认值，在调用方法的时候可以使用 NULL 作为实参。

3>如果一个类或接口指定了类型约束，则其所有的子类或实现也都如此。

4>类型约束不只是用在类的成员方法里，也能使用在函数里使用

十四、自动加载类

当我们在实际开发一个项目的时候，往往会编写很多个类，并且我们一般会把不同的类放在不同的文件里，需要哪些类就去 require 或者 include 之类的函数来引入它即可！

这时我们就有一个问题了，这么多的类，一个个去引入是一件非常麻烦的事情，PHP 这门语言的开发者有没有为我们考虑到这个事情呢？

自动加载

方法一：

可以定义一个 __autoload() 函数，它会在试图使用尚未被定义的类时自动调用。通过调用此函数，脚本引擎在 PHP 出错失败前有了最后一个机会加载所需的类。

```
function __autoload($className){
        require $className.'.php';
}
```

该函数以后可能被废弃

方法二：

1、spl_autoload_register($autoload_function [, bool $throw = true [, bool $prepend = false]]])

将函数注册到 SPL __autoload 函数队列中。

参数：

autoload_function

欲注册的自动装载函数

throw

无法成功注册时， spl_autoload_register()是否抛出异常

prepend

如果是 true，会添加函数到队列之首，而不是队列尾部

2、spl_autoload_functions ()

获取所有已注册的 __autoload() 函数

3、spl_autoload_unregister ($autoload_function)

从 spl 提供的自动装载函数栈中注销某一函数

参数：

autoload_function

要注销的自动装载函数