# 面向对象编程思想介绍

## 面向过程编程

特征：将要完成的任务，分割为若干个步骤：

1. 做什么
2. 做什么
3. ………
4. 搞定了 任务完成

## 面向对象编程

基本特征;

将要完成的任务，“反派”给不同的“对象”去做

某对象1：会做什么…

某对象2：会做什么….

…………..

程序一旦启动，则各个对象“各司其职”，相互配合就完成了任务

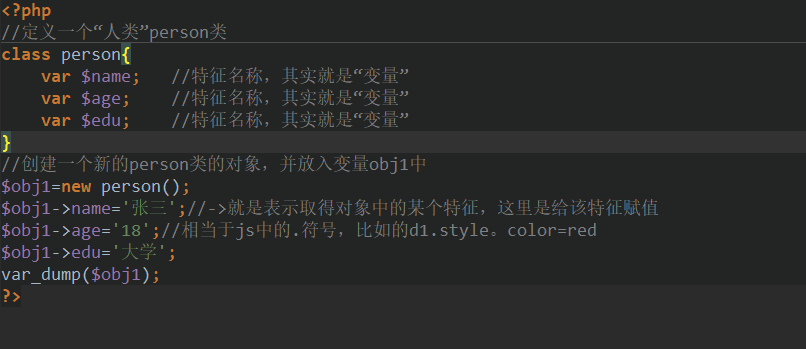
# 面向对象中的基本概念

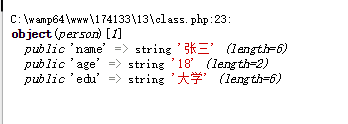
## 类和对象

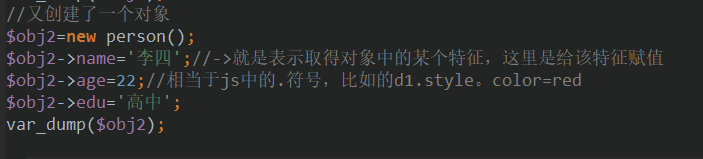
对象：万物皆对象

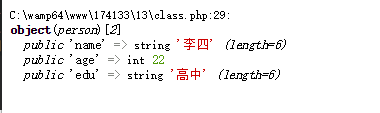
类 ：任何对象，都可以人为“规定”为某种类型（类别）

举例





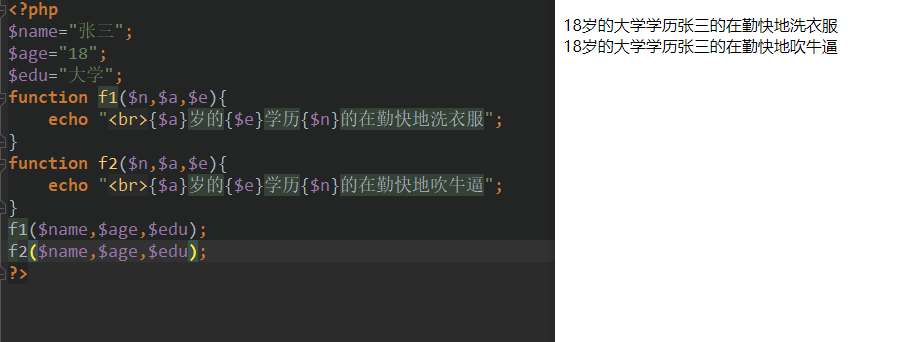




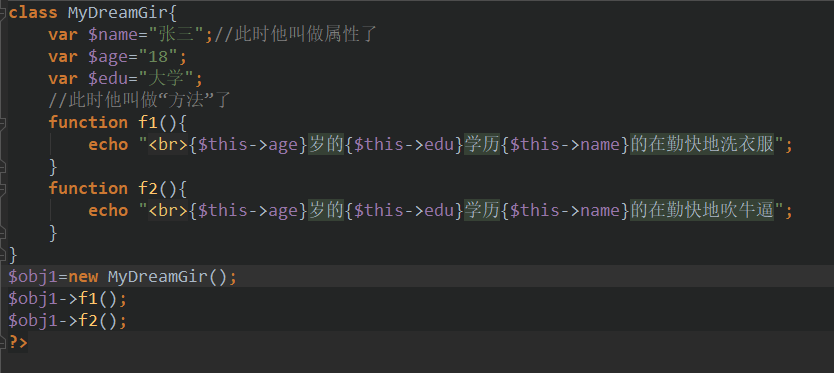
* 类是描述一类事物的一个总称，是具有相同特征特性的该类事物的一个通用名字（称呼）；
* 比如人就是一个类（人类），狗也是一个类（狗类），它们又属于一个更大的类（哺乳类），桌子，手机，书都是一个类；
* 对象是一个明确的具体的“物体”，是某个类中的一个“实物”（相对来说，类就是一种抽象的泛称）。对象离不开类，或者说，对象一定隶属于某个类——有类才有对象，先有类再有对象。

# 属性和方法

原来的做法 面向过程



现在的做法 面向对象



可见

属性就是原来的变量，只是现在他“隶属于”一个类了，即写在一个类中，就称为属性

方法就是原来的变量，只是现在他“隶属于”一个类了，即写在一个类中，就称为方法

注意：属性和方法，已经不能“自由使用”了，而是都要通过这个类或这个类的对象

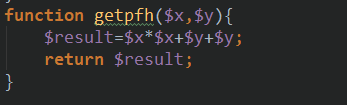
使用属性，就把它当做一个“变量”去使用就好了，只是需要该形式 对象->属性名

使用使用，就把它当做一个“函数”去使用就好，只是需要该形式 对象->函数名（）

一个类，定义了一些属性和方法

则：

这个类所创建出来的对象，也就自然有了这个属性和方法





结果：



## 创建对象的几种形式

Class c1(){

Var $p1=1//定义一个属性

}

形式1：

$o1=new c1()//通过一个类，去new出一个对象

形式2

$o2=new o1()通过一个对象，去new出一个对象—其实是new出来的旧对象所属类的一个新对象

形式3

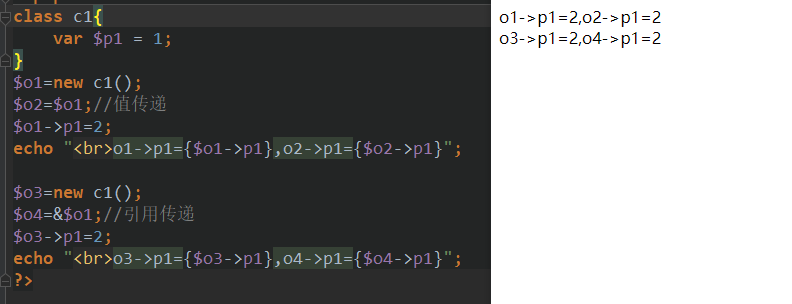
$s1=“c1”//只是一个字符串变量而已

$o3=new $s1() //这就是所谓可变类——无非就是类的名字是一个变量而已

形式4

$o4=new self() //seif表示“当前类本身”，它只能出现在一个类的方法中。

# 对象的传值方式



可见，对于对象，值传递和引用传递方式，对对象的数据的改变结果是一样的

## 为什么

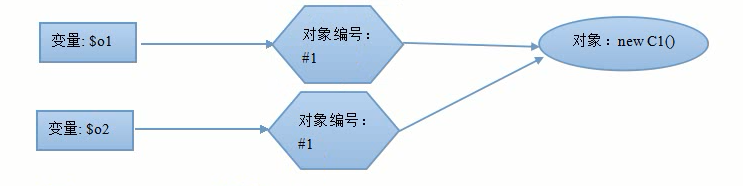
这要从对象的数据的存储方式来理解：

$o1=new c1( ) //这里创建一个对象o1，其存储结果如图所示



这里，实际上，变量o1中，存储的数据只是一个“对象编号#1”，这个对象编号，才会去指向对象数据new c1（） 该编号数据，我们不能控制，只是系统内部的分配

则，o2 = $o1；作为值传递，实际复制的是该对象的编号：结果情形是：



在语法上，对对象变量进行的属性进行操作，其实就是通过该编号来指向对象而操作的。

$o1->p1 = 2; //此时，就是通过对象编号#1去修改了对象（new C1（） ）本身的内部数据.

echo $o2->p1; //此时相当于取得变量$o2所包含的编号#1中所指向的对象（new C1（） ）的内部数据p1

那么，对应引用传递，情况是这样的：

$o3 = new C1(); //又创建一个对象，图示如下：

对象编号：

#2

变量: $o3

然后：

$o4 = & $o3; //引用传递，则其后，他们的关系图示为：

变量: $o3

对象编号：

#2

变量: $o4

所以：

$o3->p1 = 2; //

echo $o4->p1; //输出2

# 类中成员

## 类中成员概述

面向对象编程，是需要通过“对象”去做什么事情（以完成某种任务）；

而

对象总是来源于类；

所以：  
面向对象的编程，一切都是从定义类开始；

类中成员分为三大类

属性

方法

常量

class 类名 {

常量定义1；

常量定义2；

.......

属性定义1；

属性定义2；

........

方法定义1；

方法定义2；

.......

}

说明：

以上各项，没有顺序问题；习惯上，其实常量放前面，然后是属性，然后是方法；

详细一点，就又分为：

属性：

普通属性；//一般属性，实例属性

静态属性；

方法：

普通方法；//一般方法，实例方法

静态方法；

构造方法；

析构方法；

常量：

## 类常量

当在一个类中定义一个常量时，该常量就称为“类常量”

定义形式；

Class 类名{

Const 常量名=常量值

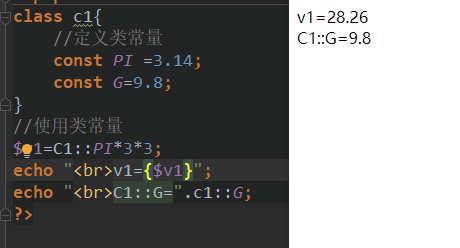
//不可以使用define（）来定义

}

使用形式：

常量的使用，是通过类名，并使用范围解析符（：：）来取用

举例：



## 普通属性（实例属性）：

实例的单词为：instance

实例，其实也叫做“对象”；

普通（实例）属性，就是一个可以在该类实例化出的对象上使用的属性！

### 定义形式：

class 类名{

var $属性名 = 初始值；

var $属性名； //这个属性没有初始值；

//上述的var 还可以使用public来代替，比如：

public $属性名 = 初始值；

public $属性名； //这个属性没有初始值；

}

### 使用形式：

是通过该类的对象，来使用普通属性（实例属性）：

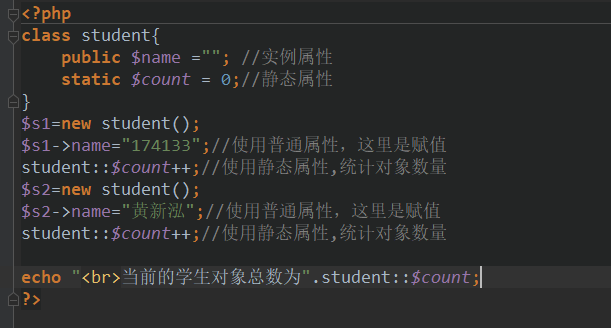
$对象->属性名；

因为，属性的本质就是变量，则其就可以当做一个变量来看待和使用，比如：

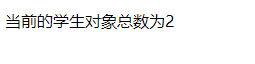
$v1 = $对象->属性名；

echo $对象->属性名；

$v2 = $对象->属性名 \* 3 + 5; //进行计算



结果为



可见：

1，实例属性，是每个对象都可以不一样的数据，也是每个对象都“独自拥有”的数据；

2，静态属性，他不属于任何一个对象，而只属于该类本身，也可以理解为为所有对象所共有的数据；

## 普通方法（实例方法）

一个类中定义的方法，可以为这个类的所有对象调用的方法。也可以理解为，这个类的所有对象，都各自有自己的一个该方法；

### 定义形式：

class 类名{

function 方法名(形参1，形参2，.... ){

//方法体。。。

}

}

### 调用形式：

$对象名->方法名(实参1，实参2，....）；

## 静态方法

一个类中定义的方法，只隶属于这个类本身，而不是隶属于这个类的对象。

### 定义形式：

class 类名{

static function 方法名(形参1，形参2，.... ){

//方法体。。。

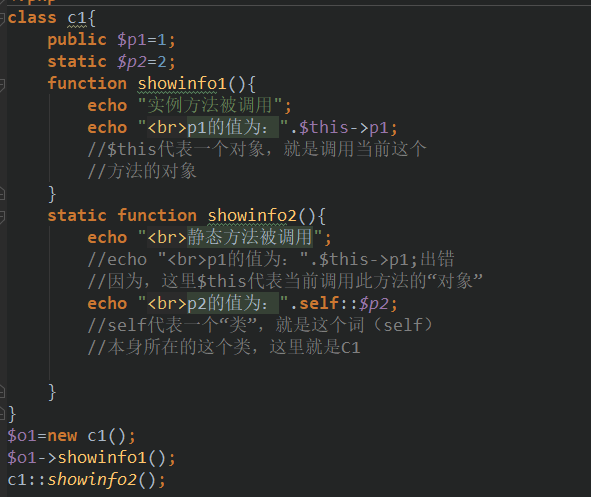
}

}

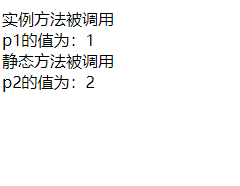
### 调用形式：

类名：：方法名(实参1，实参2，....）；

举例（属性和方法的综合例子）



结果为



实例方法中也可以去使用静态属性

但 而静态方法中，通常就不能使用实例属性

图例如下：



# 构造方法（\_ \_construct）

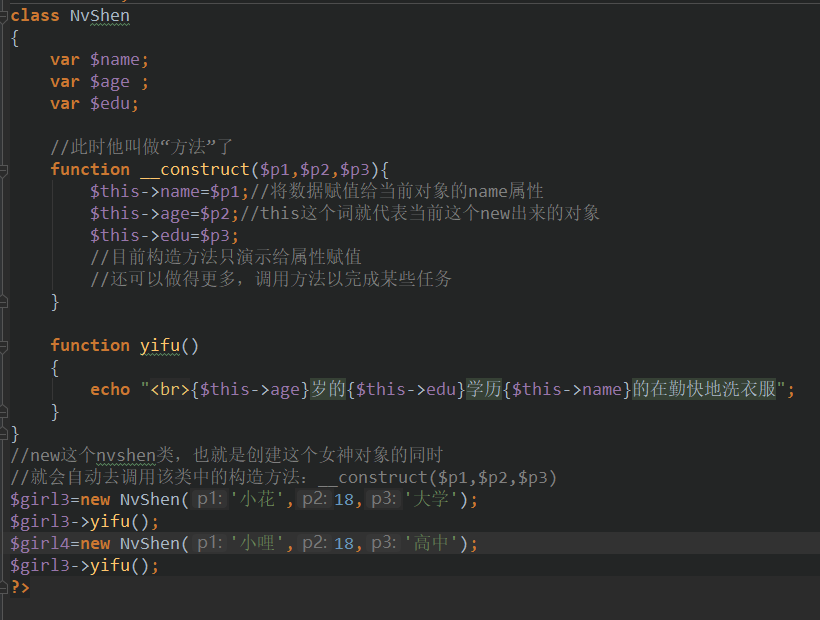
构造方法，是一个特殊的方法

1. 名字是固定的：\_\_construct
2. 该方法通常都不要我们自己调用，而是在new一个对象的时候会自动调用
3. 该方法主要的目的是为了在new一个对象的时候，给该对象设置一些“初始值”（初始化工作）
4. 构造方法的参数没有规定

先看这种没有构造方法的情形



可见这种方法不是很方便，我们改进为构造方法



# 析构方法（\_\_destruct）

说明：

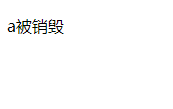
1，析构方法是一个特殊的方法，名字为固定的词：\_ \_desctruct

2，析构方法是在一个对象被“销毁”的时候会自动被调用的方法——我们无法调用它；

3，析构方法不能带参数（形参），但方法中也可以使用$this这个词，代表“当前对象”；



结果



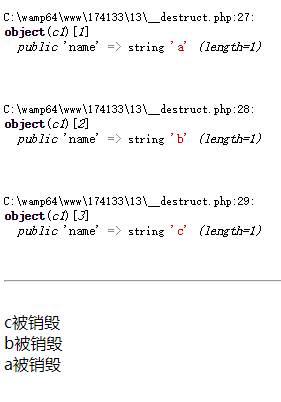
程序运行结束，变量（对象）肯定都会被销毁

# 对象在哪些情况下会被销毁

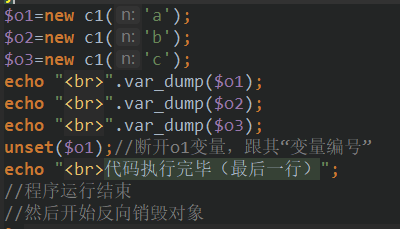
1. 程序结束，所有变量都会被销毁，自然，变量所代表的对象也会被销毁
2. 当一个对象没有任何变量“指向”它的时候，即使程序还没有结束，也会被销毁



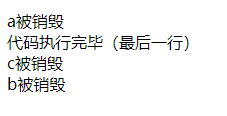
结果



可见，对象销毁的顺序，默认情况下，跟其创建的顺序相反



结果



# 思考题？？？？？



结果

