# 抽象类，抽象方法

## 抽象类

是一个不能实例化的类；

定义形式：  
 abstract class 类名{}；

为什么需要抽象类

它是一种为了技术管理而设计

## 抽象方法

是一个只有方法头，没有方法体的方法

定义形式

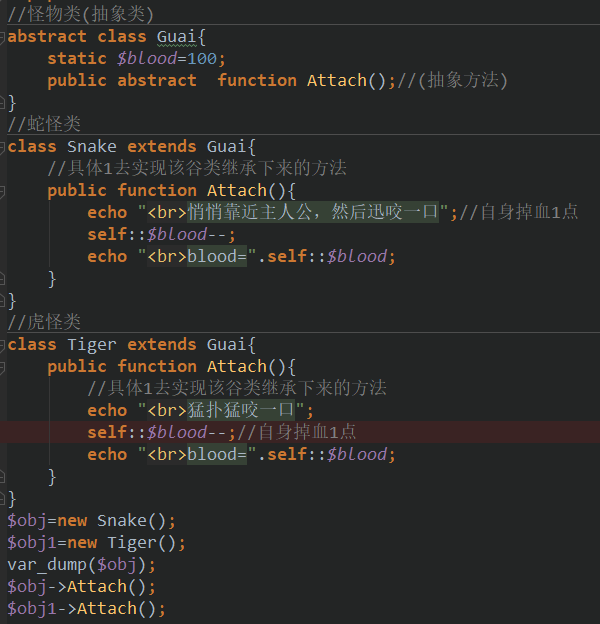
protected abstract function Attach(形参1，形参2…..);//(抽象方法)

为什么需要抽象方法

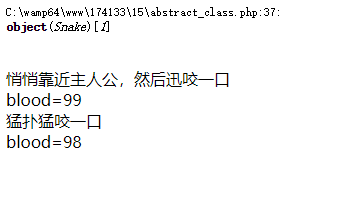
它也是为了技术管理而设计：要求下级类需要去实现这个方法的“具体做法”；



示例1



结果为



## 抽象类和抽象方法的细节

1. 一个抽象方法，必须在抽象类中；
2. 反过来，抽象类中可以没有抽象方法——虽然不常见
3. 可见，抽象方法是为了规定下级类中“必须”要做具体去完整某个工作（任务）
4. 抽象下级类中继承了上级类的抽象方法，则要么去“实现该方法的具体内容”，要么自己也作为抽象方法（即其继承的抽象方法，仍然是抽象的）；
5. 子类实现父类的抽象方法的时候，其形参也应该保持一致；其访问权限也不能更小

——其原因其实这是“重写现象”，自然应该遵循重写的要求

# 重载技术overloading

### 重载的基本概念：

两个概念：

## 重载在“通常面向对象语言”的中含义：

是指，在一个类（对象）中，有多个名字相同但形参不同的方法的现象

Class C{

Function f1(){。。。 }

Function f1($p1){。。。 }

Function f1($p1,$p2){。。。 }

}

## 重载在“PHP语言”中的含义：

是指，一个对象或类使用其未定义的属性和方法的现象，其中的一些处理机制；

Class C{

Public $p1=1

}

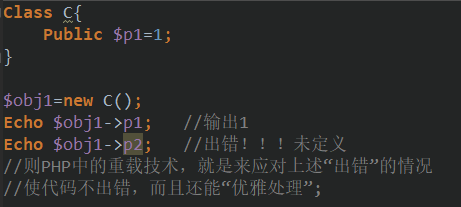
$obj1=new C();

Echo $obj1->p1; //输出1

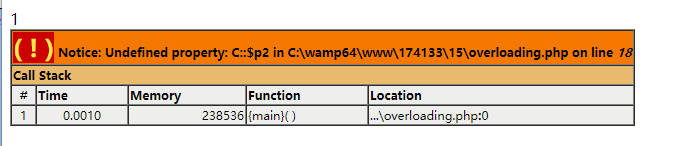
Echo $obj1->p2; //出错！！！未定义

则PHP中的重载技术，就是来应对上述“出错”的情况，使代码不出错，而且还能“优雅处理”;

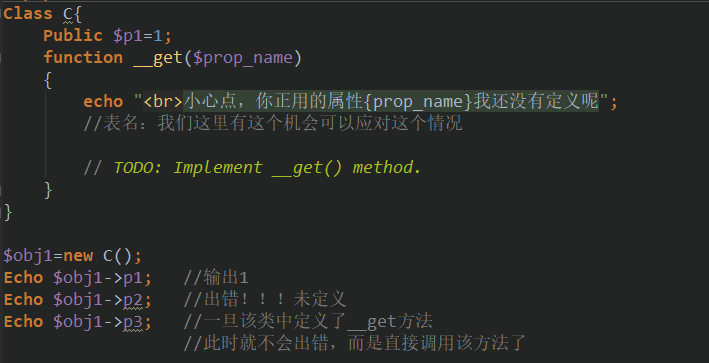
举例



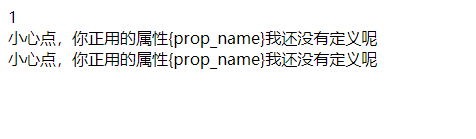
结果为



修改代码，使用\_\_get方法



结果为



**属性重载：**

就是对一个对象的不存在的属性进行使用的时候，这个类中预先设定要的应对办法（处理机制）

属性，本质，就是变量，其只有4个操作

取值：

当对一个对象的不存在的属性进行“取值”的时候，就会自动调用方法\_\_get

赋值：

当对一个对象的不存在的属性进行“赋值”的时候，就会自动调用方法\_\_set

判断isset：

当对一个对象的不存在的属性进行“isset”的时候，就会自动调用方法\_\_isset

销毁unset：

当对一个对象的不存在的属性进行“unset”的时候，就会自动调用方法\_\_unset

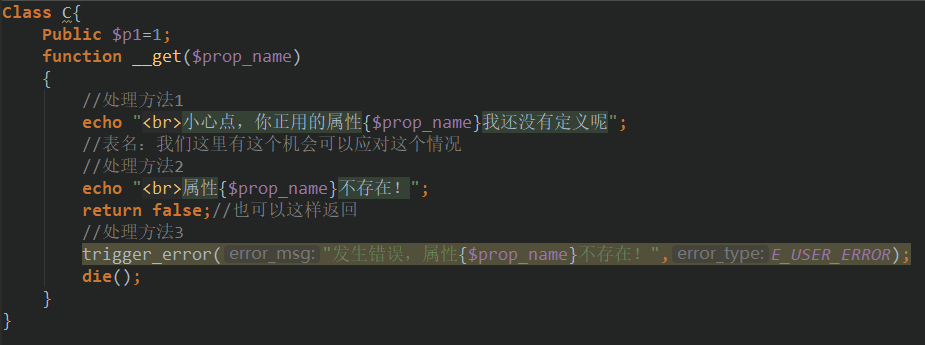
以上4个方法，被称为“魔术方法”

### \_\_GET（$属性名）

在对一个对象的不存在的属性进行“取值”，会自动调用方法\_\_GET；

我们其实是可以使用该方法来对这种“意外”情况进行某种特别处理

其中，该方法可以带一个形参，表示这个要对这个取值不存在的属性名（string）



### \_\_SET（$属性名，值）

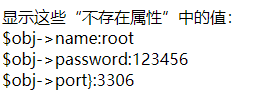
在对一个对象的不存在的属性进行“赋值”，会自动调用这个内部的模式方法\_\_SET；

它有两个形参，分别代表要对不存在的属性进行赋值的时候的“属性名”和“属性值”

这个方法，结合\_\_GET方法，往往可以使我们定义类，就有一种“可简便扩展属性”的特征，即：类（或对象）的属性，可以更为方便自由，如下所示



结果为

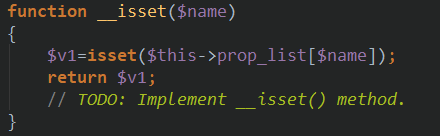


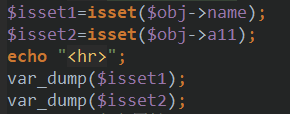
### \_\_ISSET（$属性名）

当对一个对象的不存在的属性进行“isset”的时候，就会自动调用方法\_\_isset

用法：

$v1=isset($对象->不存在的属性)//此时就会调用这个对象的所属类中的魔术方法isset



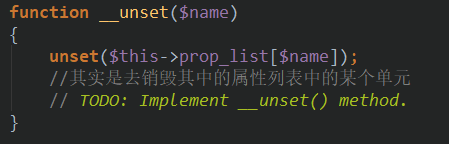


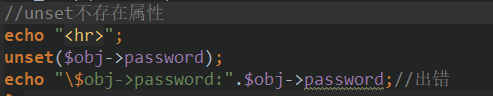
结果为



### \_\_UNSET（$属性名）

当对一个对象的不存在的属性进行“unset”的时候，就会自动调用方法\_\_unset





结果为

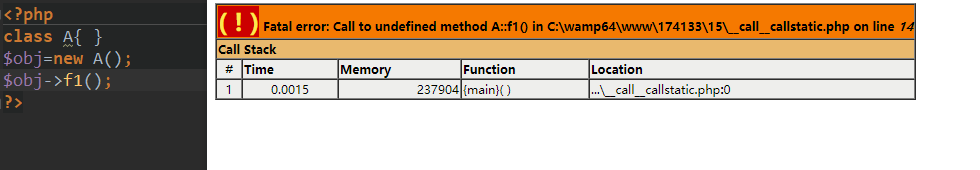


**方法重载**

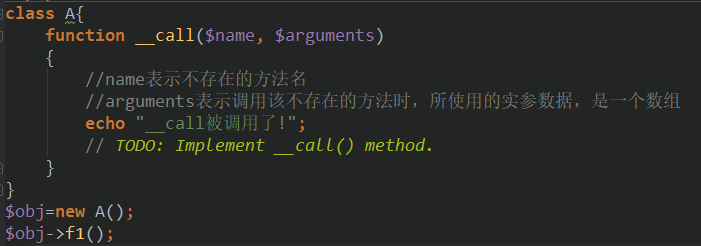
当对一个对象的不存在的实例方法进行“调用”的时候，会自动调用类中的\_\_call这个魔术方法

当对一个类的不存在的静态方法进行“调用”的时候，会自动调用类中的\_\_callstaticl这个魔术方法

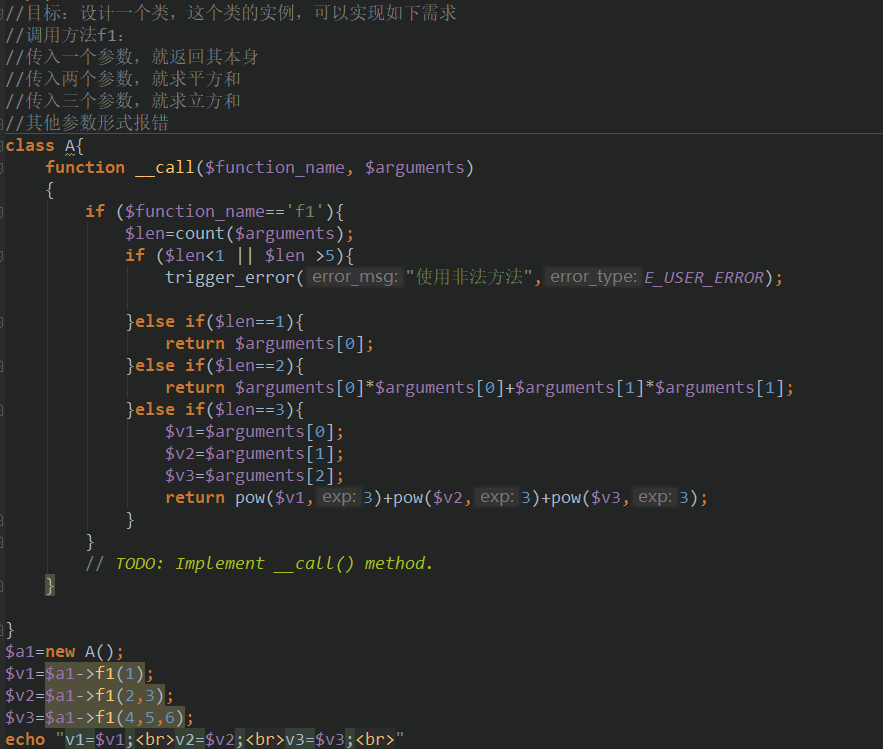
基本应用示例：



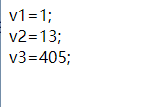
修改代码为



# 利用PHP重载技术，实现通常的“方法重载”



结果为



## 接口（interface)定义形式：

interface 接口名{

常量1；

常量2；

.....

抽象方法1；

抽象方法2；

.....

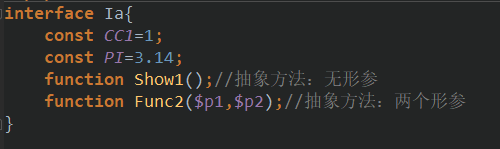
}

说明：

1，可见，接口中，只有常量（接口常量）和抽象方法两种成员；

2，接口常量的使用形式为： 接口名称：：常量名称；

3，接口中的抽象方法，不要使用abstract修饰，也不需要使用访问控制修饰符，因为其天然就是public



## 为什么需要接口

面向对象编程思想是对“现实世界”的描述（模拟）！

现实世界往往都是多继承的；

但：

出于降低类跟类之间关系的复杂度的考虑，就将语言设计为单继承的；

但这样，就无法表达出现实世界的多继承特性；

则：

接口就是对没有多继承的类之间关系的一个补充；

因为：接口可以实现“多继承”——但此时被称为继承而已，而是称为“实现”

即：

接口1 -->> 类1；

就称为：类1实现了接口1；

其本质，其实就是类1中，有了接口1中“特征信息”；

## 使用形式：

形式为：

class 类名 **implements** 接口名1， 接口名2， ....{

//类的定义。

}

这里，叫做，类实现了接口。



其中，接口跟接口之间，也可以继承，跟类之间的继承：

interface 接口1 extends 接口2{

//接口的成员定义；。。。。

}