# 继承

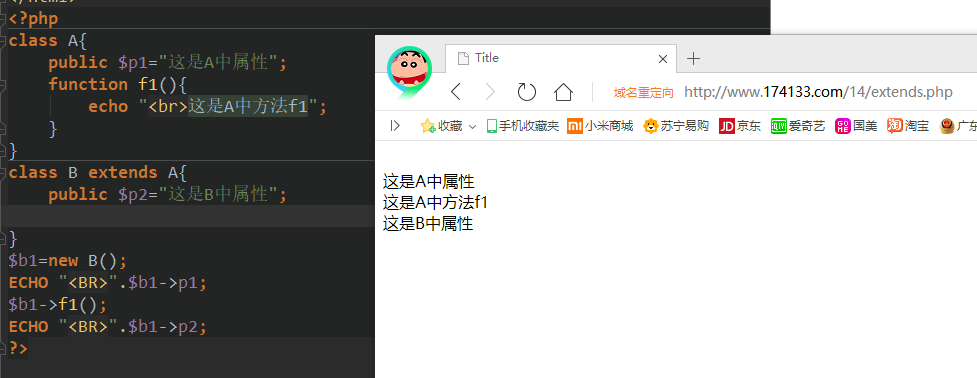
## 基本概念

将一个类a中的特征信息，传递给另一个类B中，此时就称为：

B继承a

A派生出B

## 基本语法：



### 几个基本概念

* 继承：一个类从另一个已有的类获得其特性，称为继承。
* 派生：从一个已有的类产生一个新的类，称为派生。
* 父类/子类：已有类为父类，新建类为子类。
* 单继承：一个类只能从一个上级类继承其特性信息。PHP和大多数面向对象的语言都是单继承模式。C++是多继承。
* 扩展：在子类中再来定义自己的一些新的特有的特性信息（属性，方法和常量）。没有扩展，继承也就没有意义了。

访问控制修饰符

形式：

Class 类型{

访问控制修饰符 属性或方法定义

}

* public公共的：在所有位置都可访问（使用）。
* protected受保护的：只能再该类内部和该类的子类或父类中访问（使用）。
* private私有的：只能在该类内部访问（使用）。
* parent关键字：在类的内部用来表示（代表）该类的父类。可以用来访问父类的属性或方法或常量。

他们的作用是：用来“限制”其所修饰的成员的“可访问”

可访问性：

就是在代码中使用作用两种语法形式“有效性”（合法性）：

对象->实例属性或方法

类：：静态属性或方法

访问控制修饰符，需要结合使用该语法形式的所在位置，才能确定是否可访问

有三个访问位置（范围）:

某个类的内部

某个类的继承类的内部

某个类的外部

## 则，他们结合起来的可访问性如下

|  |  |  |  |
| --- | --- | --- | --- |
|  | 类内部 | 继承关系类内部 | 类外部 |
| Public | Yes | Yes | Yes |
| Protected | Yes | Yes | No |
| Private | Yes | No | No |

总结说明

1. public修饰的成员哪里都能访问
2. 类的内部，可以访问任何级别的成员
3. Public具有最宽泛的可访问性，private具有最狭窄的访问性，protect则居中

# 示例



输出为a1->p1=1

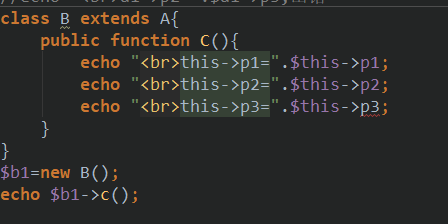
示例：：访问三种修饰符方法



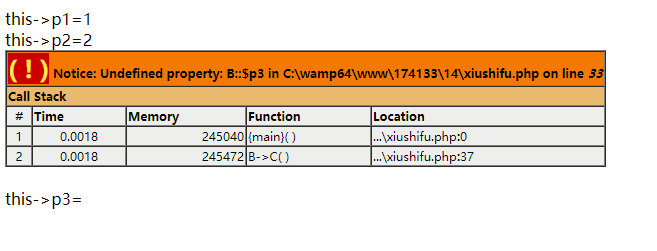
结果为



示例



结果为



Parent关键词

表示“父母”的意思，在面向对象语法中，代表“父类”；

本质上就是代表父类这个“类”，而不是父类的“对象”

形式：

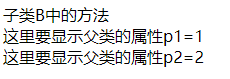
Parent：：属性或方法 //通常是静态属性或静态方法

但有时候可能是实例属性或实例方法

示例：



结果为



## 示例2（调用方法）

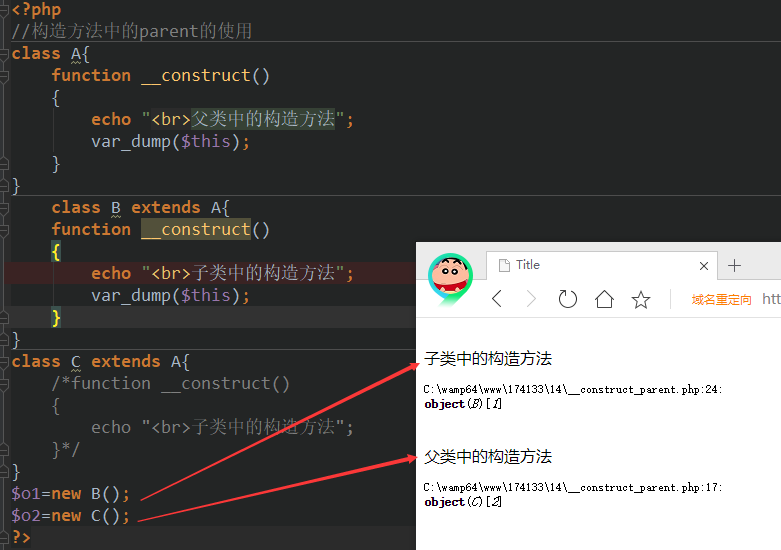


## 对比其他两个词

|  |  |  |  |
| --- | --- | --- | --- |
| 关键词 | 含义 | 使用位置 | 使用示例 |
| Parent | 代表父类（这个类） | 肯定在一个方法中 | Paren::属性或方法 |
| Self | 代表当前其所在的类 | 肯定在一个方法中 | self::静态属性/方法 |
| $this | 代表调用当前方法的对象 | 肯定在一个方法中 | $this->实例属性/方法 |

## 构造方法和析构方法调用上级同类方法的问题

1. 如果一个类有构造方法，则实例化这个类的时候，就不会调用父类的构造方法（如果有）
2. 如果一个类没有构造方法，则实例化这个类的时候，就会自动调用父类的构造方法（如果有）

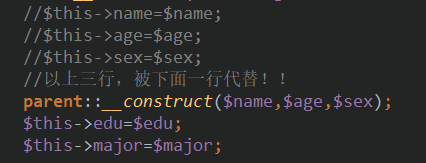


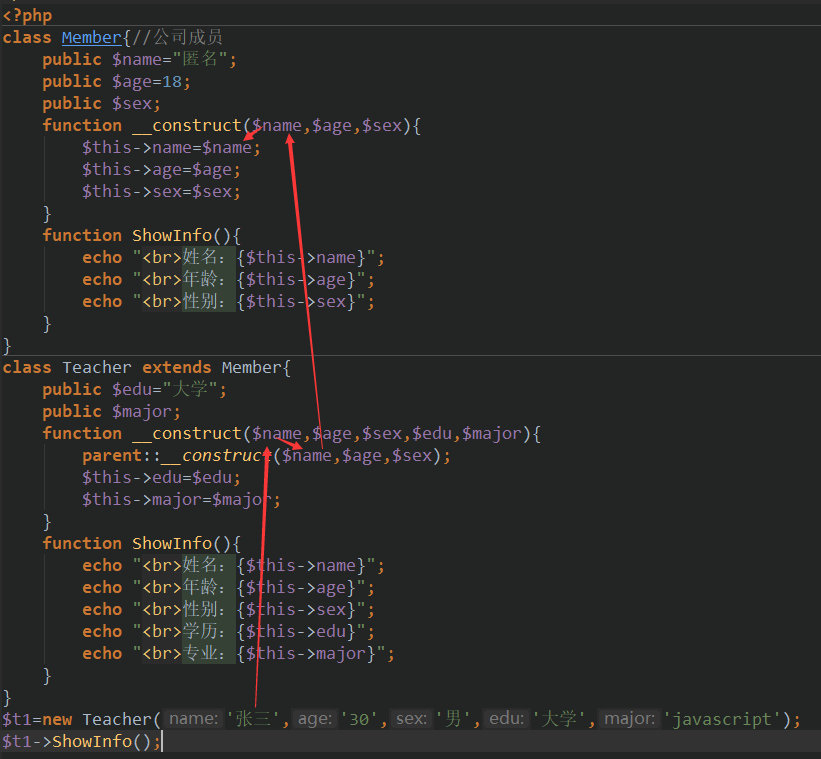
1. 如果一个类有析构方法，则销毁这个类的时候，就不会调用父类的析构方法（如果有）
2. 如果一个类没有析构方法，则销毁这个类的时候，就会自动调用父类的析构方法（如果有）
3. 如果一个类中有构造或析构方法，则就可以“手动”调用父类的同类方法（如果有）
   1. 手动调用的语法形式总是这样

Parent：：构造方法或析构方法

则，第5种情况，是构造方法的一个典型代码（写法）

（在子类的构造方法中，常常需要使用该方法，从而减少方法初始化代码）





# 覆盖（override）又叫 “重写”

含义：

将一个类从父类中继承过来的属性和方法“重新定义”——相当于子类不想用父类的该属性和方法，而是想要定义

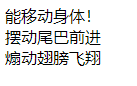
覆盖的现实需要：

对于一个父类，或许其属性的现有数据（值），子类觉得不合适，而需要有自己新的描述。或许其方法，子类觉得也不合适，需要自己来重新定义该方法中要做到事

## 示例



结果为



重写的基本要求

访问控制权限：

子类覆盖的属性或方法的访问控制权限，不能“低于”父类的被覆盖的属性或方法的控制权限

父类：public 子类：只能是public

父类：protected 子类：可能说protected和public

父类：private 子类：不能覆盖！——即父类的私有成员，不存在被子类覆盖的可能

方法的参数形式：

子类覆盖父类的同名方法的时候。参数要求跟父类保持一致；

特例：  
 构造方法重写的时候参数可以不一致

小注意：

虽然父类的私有属性不能被覆盖，但子类却可以定义自己的跟父类同名属性

虽然父类的私有方法不能被覆盖，但子类也不能定义自己的同名方法

# 最终类

最终类，其实就是一种特殊要求的类，要求该类不允许往下继承下去。

形式：  
 final class 类名{

//类的成员定义—跟一般类的定义一样

}

# 最终方法

最终方法，就是一个不允许即类下级类去覆盖的方法

形式：

Class 类名{

Final function 方法名（形参）{

…….

}

}

# 设计模式

就是叫做解决某个问题的一般性代码模式的经验性总结

类比来说：  
 他类似之前所学的“算法”：针对某种问题，使用某种特定的语法逻辑就可以完成该任务

## 工厂模式：

就是这样一个类（就是所谓的工厂类）：

他可以根据“传递”给他的类，而去生产出对应的类的对象。



# 单例模式：

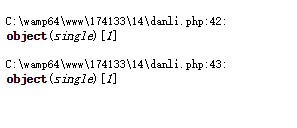
例，就是实例（Instance），其实就是对象object

单例，就是一个对象

单例模式：就是设计这样一个类，这个类只能“创造”出它的一个对象（实例）



结果：



# 设计一个类：mysql数据库操作类：

设计目标：

1. 该类一实例化，就可以自动连接上mysql数据库
2. 该类可以单独去设定要使用的连接编码set names
3. 该类可以单独去设定要使用数据库（use）
4. 可以关闭连接数据库

上述设计目录，大致上相当于如下几行代码