昨日回顾 2

类的继承 4

基本概念 4

访问（权限）修饰符 5

public公有的 5

protected 受保护的 6

private私有的 6

parent代表父类 7

构造方法析构方法在继承中的表现 8

重写override 11

重写的基本要求： 12

最终类final class： 14

最终方法final method 14

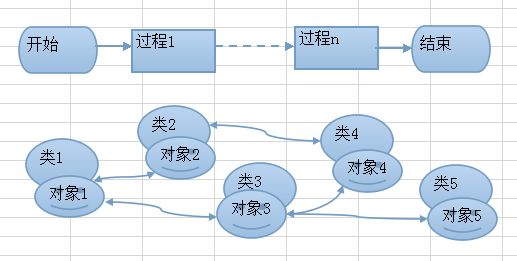
设计模式 15

什么叫设计模式 15

工厂模式 15

昨日回顾

面向对象与面向过程：



对象的存储形式：

$obj1 = new Person(); //实例化一个Person类的对象

new Person()

标识#1

$obj1

对象的传值：

值传递： $obj2 = $obj1; //拷贝的是标识符#1，但#1是指向该对象实际数据

引用传递：$obj3 = &$obj1; //拷贝的是$obj1到标识#1的引用关系，图示如下

new Person()

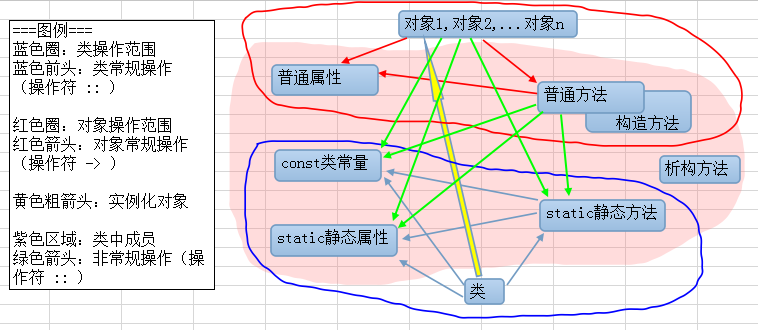
$obj1

标识#1

$obj3

类中成员：

一个图示：



个完整代码演示：

<?php

class C{

//类常量：

const const1 = 1;

//一般（实例）属性：

public $v2 = 2;

//静态属性：

static $v3 = 3;

//一般（实例）方法：

function f1(){

echo "<br /><b>实例方法：</b>";

echo "<br />实例方法中可以访问实例属性：v2 = " . $this->v2;

echo "<br />普通方法中操作类常量：" . C::const1;//也可以通过$this::const1

echo "<br />普通方法中操作静态属性：" . C::$v3;

echo "<br />普通方法中调用静态方法：";

C::f2();

}

//静态方法：

static function f2(){

echo "<br /><b>静态方法：</b>";

echo "<br />静态方法中使用类常量：" . C::const1;

echo "<br />静态方法中使用静态属性：" . self::$v3;//self指其本身所在的类";

}

//构造函数：

function \_\_construct($para1 = 2){

echo "<br /><b>构造方法被执行。</b>";

$this->v2 = $para1;

}

//析构方法：

function \_\_destruct(){

echo "<br /><b>析构方法被执行。</b>";

}

}

$obj1 = new C(20);

echo "<br />对象obj1的v2的值为：" . $obj1->v2;

//对象操作普通方法：

$obj1->f1();

?>

类的继承

简单理解：

某个类A具有某些特征，另一个类B，也具有A类的所有特征，并且还可能具有自己的更多的一些特征，此时，我们就可以实现：B类使用A的特征信息并继续添加自己的一些特有特征信息。



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。

继承和派生，其实只是从不同的方向（角度）来表述，本质上就是一个事情。

* 父类/子类：已有类为父类，新建类为子类。父类也叫“基类”，子类也叫“派生类”
* 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（权限）修饰符

在类中的成员，通常都可以在前面加上以下3个修饰符：

public：公共的，共有的，公开的

protected：受保护的

private：私有的，

public公有的

用该修饰符修饰的成员，可以在“任何位置”使用（访问）。

访问（使用）是这样一个语法模式：

对象->成员；

类名::成员；

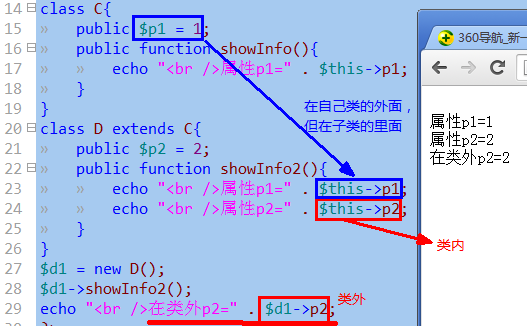
**访问位置分为3个：**

1：某个类内部：自然是该类的某个方法中

2：某个类的具有继承关系的子（父）类的内部：是指其他类的某个方法中。

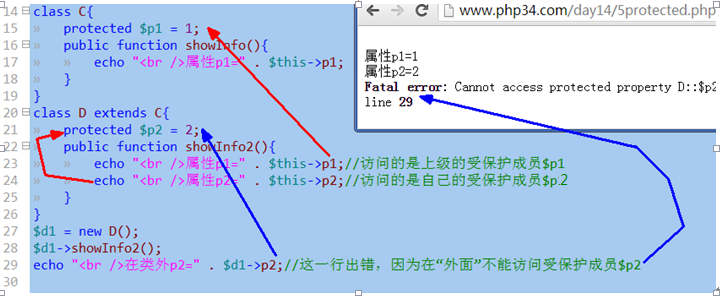
3，某个类的外部：一般就是独立的代码区（不在类中），类似我们之前的代码。

代码演示如下：



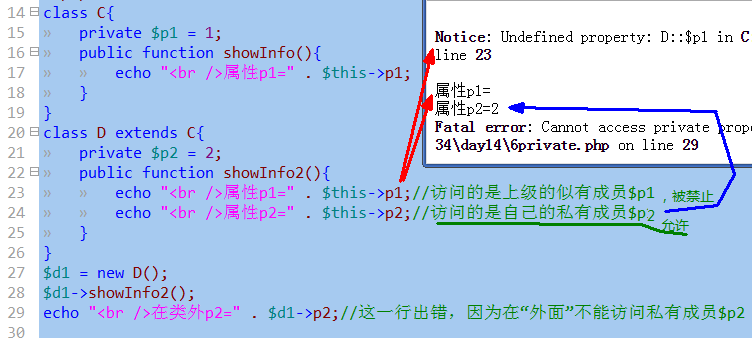
protected 受保护的

protected修饰的成员，可以在当前类或当前类的上下级具有继承关系的类中访问。



private私有的

private 修饰的成员，只能在其所在的类中访问。



访问修饰限定符的总结：

|  |  |  |  |
| --- | --- | --- | --- |
| 范围 | 本类内 | 继承关系类内 | 类外 |
| public | 可以 | 可以 | 可以 |
| protected | 可以 | 可以 | 不可以 |
| private | 可以 | 不可以 | 不可以 |

parent代表父类

对比：ｓｅｌｆ代表“本类”（自己当前类）

ｐａｒｅｎｔ通常用于在子类中调用父类的成员的时候使用，多数通常就是使用父类的“静态类”成员。

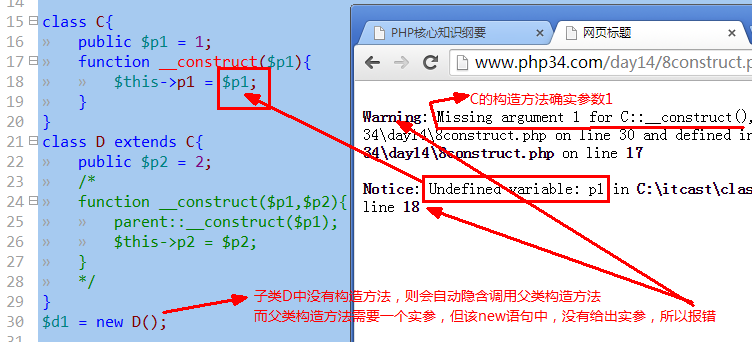
——因为ｐａｒｅｎｔ代表的类，而非对象。

以下为ｐａｒｅｎｔ关键字的２常见用法：

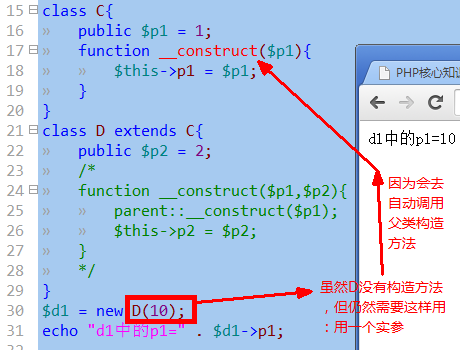


构造方法析构方法在继承中的表现

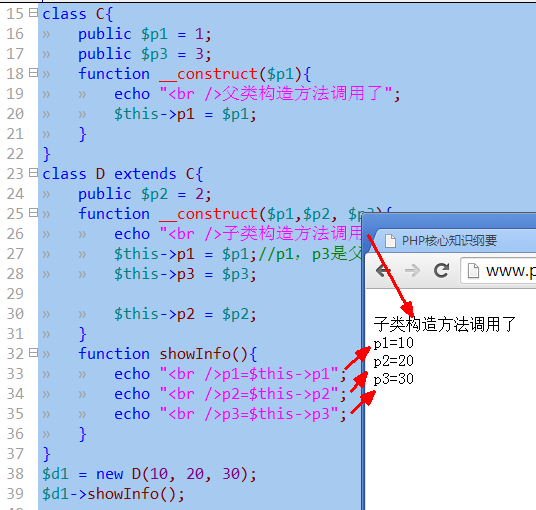
* 子类中没有定义构造方法时，会自动调用父类的构造方法。因此实例化子类时，需按照父类的构造方法的形式进行。



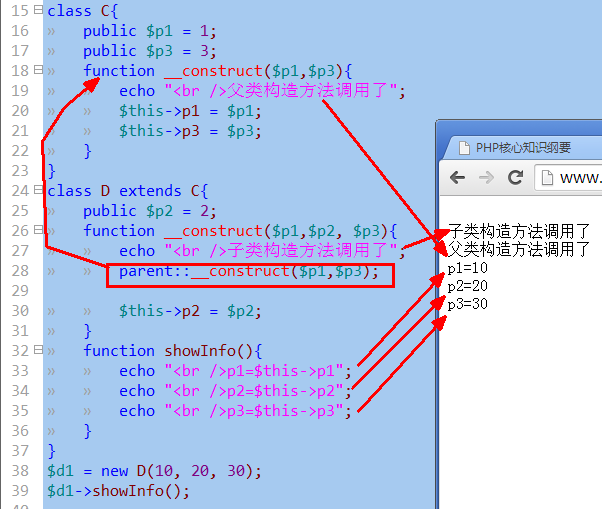
修改为：



* 子类定义了自己的构造方法，则不会自动调用父类的构造方法，但可以手动调用：parent::\_\_construct();



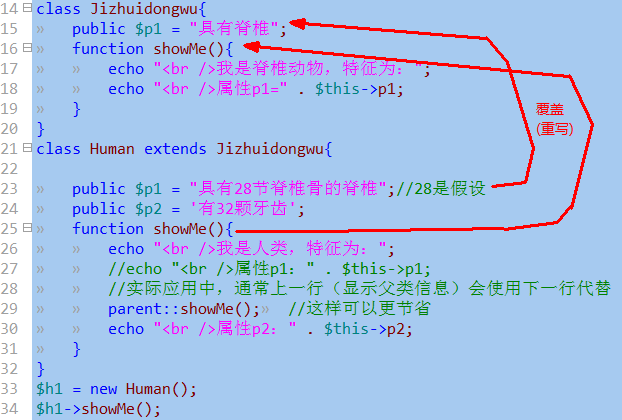
但通常，在子类，很多时候，在构造方法中，都应该（需要）去调用父类的构造方法以节省代码，增加可读性：



* 子类中没有定义析构方法时，会自动调用父类的析构方法。
* 子类定义了自己的析构方法，则不会自动调用父类的析构方法，但可以手动调用：parent::\_\_destruct()

重写override

重写又叫覆盖，就是将从父类继承下来的属性或方法重新“定义”——就是从新写。



但注意：子类覆盖父类的方法，虽然可以去调用父类的同名方法去完成一定的工作，但不是必须的。也有可能父类的该方法所执行的结果并不适合子类，此时子类就去完全自己写了。

重写的基本要求：

**访问控制权限：**

下级的访问控制权限应该不低于上级的访问控制权限：

上级：public 下级：只能public

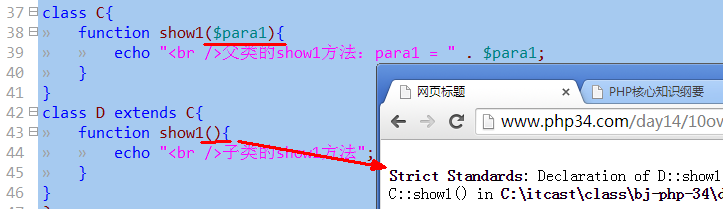
上级：protected 下级： protected, public

上级：private 下级：private protected public——实际此情况无意义。

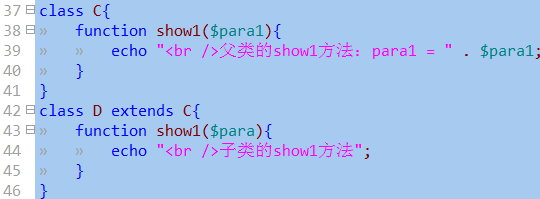
私有的不能覆盖，而是完全当作自己全新的。

**方法的参数形式：**

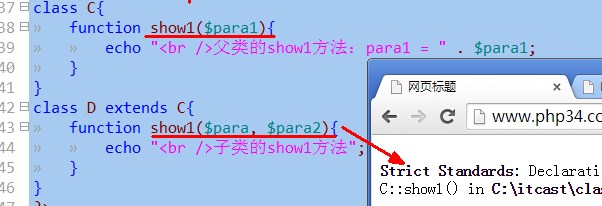
应该跟父类的一致。



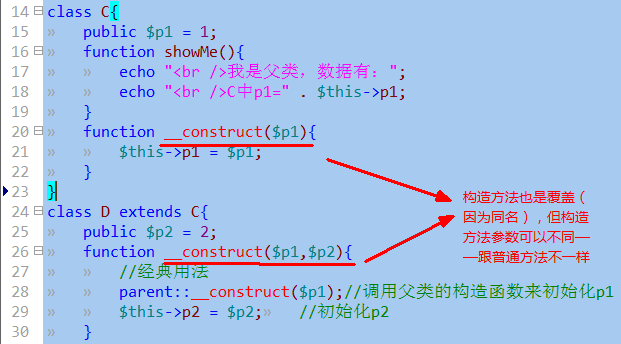
正确的做法：



更多参数也不对：



* 私有属性和私有方法的重写问题：私有属性和方法都不能覆盖，但其实子类可以定义跟父类私有的同名属性或方法。只是当作一个自身的新的属性或方法来看待而已。不过方法的参数必须一致。
* 构造方法的重写问题：构造方法不但可以像其他普通方法一样重写，而且，比普通方法更宽松：重写的时候参数可以不一致。



最终类final class：

通常，一个类，没有特别声明的话，则“别人”就可以随意拿过来使用并对之进行“扩展”——继承。

但是：

如果某个类不希望对其进行扩展，则可以将其声明为“最终类”。

形式：

final class 类名{ 。。。。类定义。。。。}

最终方法final method

通常，一个方法，如果没有特别声明，则下级类就可以对其进行“覆盖”（重写）。

但是：

如果某个方法不希望被下级类覆盖，就可以对其生命为“最终方法”。

形式：

final function 方法名(){。。。。方法定义。。。。}

设计模式

什么叫设计模式

所谓设计模式，就是一些解决问题的“常规做法”，是一种认为较好的经验总结。面对不同的问题，可能会有不同的解决办法，此时就可以称为不同的设计模式。

工厂模式

在实际应用中，我们总是需要去实例化很多很多的类——以得到对象。

则：

我们可以设计出一个“工厂”（其实就是类），该工厂的作用（任务）就是为人们“生产”各种对象。这种工厂通常只要指定类名，就可以据此获取一个该类的对象。

