

面向对象编程——单元 2：封装与继承

一、继承

如果在定义一个新类 B 时，发现已经存在了一个类 A 包含了类 B 中需要的一些属性和方法，这时可以让 B 类继承 A 类。这时 B 类的对象可以直接引用从 A 类继承过来的属性和方法。从而达到代码的复用。我们称类 A 叫父类(超类/基类)，称类 B 叫子类。

语法上：

```
<?php
class A
{
    public $name='张三';
    public function hello(){
        echo "hello<br>";
    }
}

class B extends A{
    public $age=10;
    public function bye(){
        echo "bye<br>";
    }
}

$b = new B();
echo $b->name;
echo $b->age;
$b->hello();
$b->bye();
?>
```

B继承A

B类的对象引用继承自A类的成员变量和方法

思考题：

商品管理系统，商品种类有手机、图书、服装、化妆品等。采用面向对象的继承如何定义手机类和图书类？

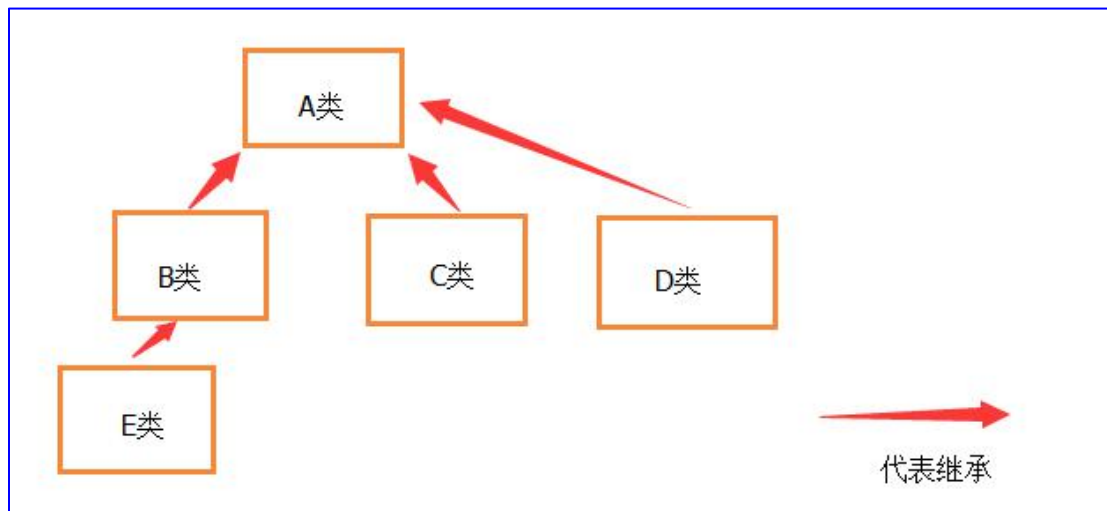
二、重写

如果子类的成员（属性和方法）和父类的成员同名，则子类对象在引用这些成员时，引用的是子类中的成员，父类的同名成员会被覆盖从而不能直接访问。这种现象称为重写。主要目的是：从父类继承过来的成员不能满足子类的需要，但是还需要使用相同的名称代表同一个含义（名称代表功能，功能不变但实现不同），所以用重写的方式重新来设计。

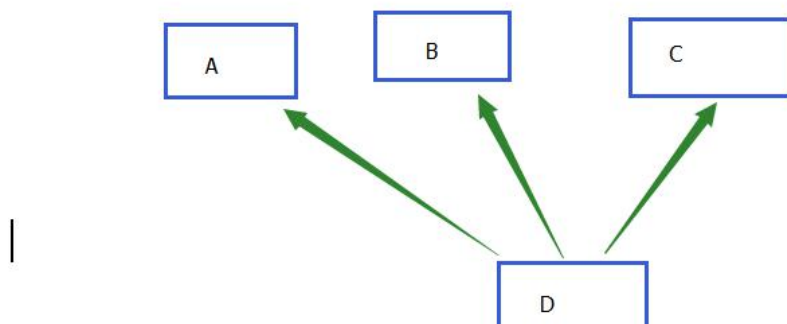
```
1  <?php
2      class A
3      {
4          public $name='张三';
5          public function hello(){
6              echo "hello<br>";
7          }
8      }
9      class B extends A{
10         public $name='李四';           重写了父类的属性
11         public $age=10;
12         public function bye(){
13             echo "bye<br>";
14         }
15         public function hello(){       重写了父类的方法
16             echo "你好<br>";
17         }
18     }
19
20     $b = new B();
21     var_dump($b);
22     $b->hello();
23     ?>
```

三、继承链与访问修饰符

多个类之间存在继承关系，从而形成一个继承链。PHP 中的继承是单继承。



单继承：一个类只能有一个直接父类，可以有多个子类。



上图是多继承，一个类可能存在多个父类。PHP 不允许。

访问修饰符：

	本类内部	类外部	继承链上的其他类内部
public 公共的	可以	可以	可以
protected 保护的	可以	不可以	可以
private 私有的	可以	不可以	不可以

例如：修改类 B 属性 age 的访问修饰符，查看在类内部、类外部和继承链上的其他类内的访问情况。

```
1  <?php
2      class A
3      {
4          public function getAgeA(){
5              echo "A:".$this->age; //在父类中访问age
6          }
7      }
8      class B extends A{
9          public $age='10<br>'; //属性age的定义位置
10         public function getAgeB(){
11             echo "B:".$this->age; //在本类中访问age
12         }
13     }
14     class C extends B{
15         public function getAgeC(){
16             echo "C:".$this->age; //在子类中访问age
17         }
18     }
19     $c = new C();
20     echo $c->age; //在类外部访问age
21     $c->getAgeA();
22     $c->getAgeB();
23     $c->getAgeC();
24     ?>
```

四、\$this 和 parent

1、\$this 在类的方法中，用于引用其他的属性和方法。其实\$this 真实代表的是调用类方法的**对象**本身，而不是由类决定的。所以存在类的继承时，子类对象调用继承自父类的方法时，方法中的\$this 代表的是子类的对象，而非父类的对象。

The image shows a PHP code editor on the left and a web browser on the right. The code defines two classes: Class A and Class B (which extends A). Class A has properties \$name (张三) and methods getName() and getAge(). Class B overrides these with \$name (李四) and \$age (10). In the main script, an object \$b of Class B is created, and its getName() and getAge() methods are called. The browser output shows the object details and the printed values '李四' and 10. Red arrows point from the \$this keyword in the code to the object details in the browser, illustrating that \$this refers to the object instance (\$b) when the methods are called.

```
1 <?php
2 class A
3 {
4     public $name='张三';
5     public function getName(){
6         return $this->name;
7     }
8     public function getAge(){
9         return $this->age;
10    }
11 }
12 class B extends A{
13     public $name='李四';
14     public $age=10;
15 }
16
17 $b = new B();
18 var_dump($b);
19 echo $b->getName();
20 echo $b->getAge();
21 ?>
```

object(B)[1]
public 'name' => string '李四' (length=6)
public 'age' => int 10
李四10

子类B的对象\$b调用了继承自父类的方法getName()和getAge().这两个方法中的\$this此时都代表对象\$b。所以打印出来的name和age都是子类中的成员属性。

2、parent

当子类重写了父类的方法时，将导致父类的同名方法被覆盖（不能直接使用子类对象引用父类的同名方法），但是如果在子类中仍然需要使用被覆盖的父类方法，可以使用 `parent::` 进行引用。

例如：父类和子类的构造方法都叫 `__construct()`，父类的构造方法必然会被子类的构造方法覆盖。所以需要在子类的构造方法中，使用 `parent::` 强制调用父类的构造方法。

```
1 <?php
2 class A
3 {
4     public $name;
5     public function __construct($name)
6     {
7         $this->name = $name;
8     }
9 }
10 class B extends A{
11     public $age;
12     public function __construct($age,$name){
13         $this->age = $age;
14         parent::__construct($name);
15     }
16     // parent::应用父类的构造方法
17 }
18 $b = new B(10,'张三');
19 var_dump($b);
20 ?>
```



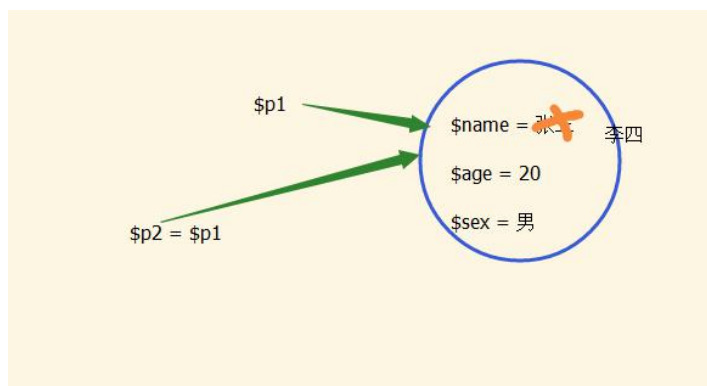
```
object(B)[1]
  public 'age' => int 10
  public 'name' => string '张三' (length=6)
```

五、类对象间的赋值和克隆

1、对象间的赋值属于引用传递，不是值传递。即两个变量引用的是同一个对象。

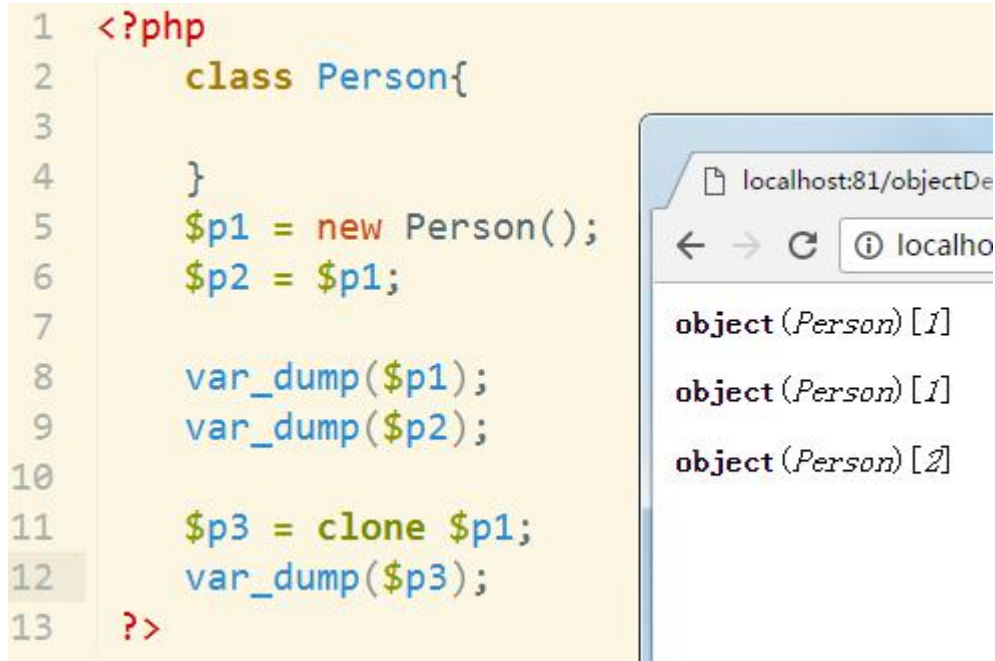
```
$p1 = new Person();
```

```
$p2 = $p1;
```



2、如果要想实现变量引用不同的对象需要使用克隆的方式。克隆时，会将原有对象的所有属性复制一份。

语法：\$p3 = **clone** \$p1;



3、当使用 clone 时，会自动执行类内的一个方法：

function __clone(){....}

如果需要对默认的克隆行为进行修改时，可以重写此方法。




4、深度克隆。

当一个类 A 的属性是另一个类 B 的对象，则在克隆类 A 的对象时，php 默认采用浅克隆方式——两个对象共有同一个属性对象。即克隆时对象属性没有被克隆。如果为了保证属性对象也被同时克隆，需要在魔术方法__clone()中手动克隆。

默认情况：

```
1 <?php
2 class Person{
3     private $name='新生儿';
4     private $age = 1;
5     public $idcard;
6 }
7 class ICard{
8     public $number='130123456789';
9 }
10
11 $p1 = new Person();
12 $p1->idcard = new ICard();
13
14 $p3 = clone $p1;
15 var_dump($p1,$p3);
16 echo "<hr>";
17 $p3->idcard->number = '00000000000000';
18 var_dump($p1,$p3);
19 ?>
```



手动变成深度克隆：

```
1 <?php
2 class Person{
3     private $name='新生儿';
4     private $age = 1;
5     public $idcard;
6     //重写克隆方法，修改默认克隆行为
7     public function __clone(){
8         echo "<br>对象被克隆了<br>";
9         $this->name = 'baby';
10        $this->idcard = clone $this->idcard;
11    }
12 }
13 class ICard{
14     public $number='130123456789';
15 }
16
17 $p1 = new Person();
18 $p1->idcard = new ICard();
19
20 $p3 = clone $p1;
21 var_dump($p1,$p3);
22 echo "<hr>";
23 $p3->idcard->number = '00000000000000';
24 var_dump($p1,$p3);
25 ?>
```

