

面向对象编程——单元 5：设计模式简介

一、单例设计模式

指的是一个类只能实例化一个对象。如果一个对象就能完成所有功能的话就可以将该类设计成单例模式。单例模式的优势是可以节省资源。单例的目的：限制类得到多个对象，只能得到一个对象。

例如：在通常的项目中，只需要操作一个数据库服务器就可以完成所有与数据库的交互功能（例如增删改查）。如果定义一个类封装了所有与数据库操作的行为，并且其中包含一个属性引用了对服务器的连接资源,包含了代表自己的一个属性。那么只需要这个类的一个对象就可以完成所有的业务。于是我们可以将此类设计成是单例模式。——节省资源。

实现单例的方法（3 个私有，1 个公共）：

第一步：私有化构造方法。限制在类外部实例化类产生对象。

第二步：增加一个公共的静态方法。因为静态方法可以在没有对象的情况下通过类进行调用。在静态方法的内部调用私有的构造方法，从而产生一个实例化对象。并将该实例化的对象存储起来。

第三步：在类中增加一个静态的私有属性保存该类的对象。在第二步的静态方法中判断，如果已经有了对象则不需要重新再创建了，直接使用存储好的对象。如果没有对象，则创建一个新的对象。

第四步：私有化__clone()方法。保证在外部不能通过 clone 方式创建新的对象。

```
1 <?php
2 /**
3  * 将MySQLDB类设计成是单例模式，只能获取一个实例对象。
4  */
5 class MySQLDB{
6     private static $instance = null; //私有的静态属性保存唯一的实例
7
8     //私有的构造方法避免外类外部使用new实例化出多个对象
9     private function __construct(){
10         //省略连接服务器，选择库，设置字符集等操作
11     }
12     //公共的静态方法，用于静态获取唯一实例。
13     public static function getInstance(){
14         //实例不存在时实例化
15         if (!isset(static::$instance)) {
16             //类内可以调用私有的构造方法进行实例化
17             static::$instance = new MySQLDB();
18         }
19         //返回唯一的实例
20         return static::$instance;
21     }
22     //私有化克隆方法，避免对象被克隆
23     private function __clone(){
24     }
25 }
26
27 // $dao1 = new MySQLDB(); //报错，不能实例化
28 $dao1 = MySQLDB::getInstance();
29 var_dump($dao1);
30
31 $dao2 = MySQLDB::getInstance();
32 var_dump($dao2);
33
34 $dao3 = MySQLDB::getInstance();
35 var_dump($dao3);
36
37 // $dao4 = clone $dao3; //报错，不能克隆对象
38 ?>
```



单例的另一种实现（类外定义）：

将判断对象是否存在的代码写在类的外部。也能实现单例的效果，但是不能称作单例模式。

预备知识：1、可变标识符 2、静态局部变量

//预备知识1: 可变标识符(可变变量、可变方法、可变类)

```
$a = 10;
```

```
$b = 20;
```

```
$var_name = 'a';
```

```
echo $$var_name; //可变变量
```

```
echo "<br>";
```

```
$var_name = 'b';
```

```
echo $$var_name;
```

```
echo "<br>";
```

/*可变方法*/

```
function fun1(){
```

```
    echo 'fun1.....<br>';
```

```
}
```

```
function fun2(){
```

```
    echo 'fun2.....<br>';
```

```
}
```

```
$fun_name = 'fun1';
```

```
$fun_name(); //可变方法
```

```
$fun_name = 'fun2';
```

```
$fun_name();
```

localhost:81 x

← → ↻ ⓘ loc

10

20

fun1.....

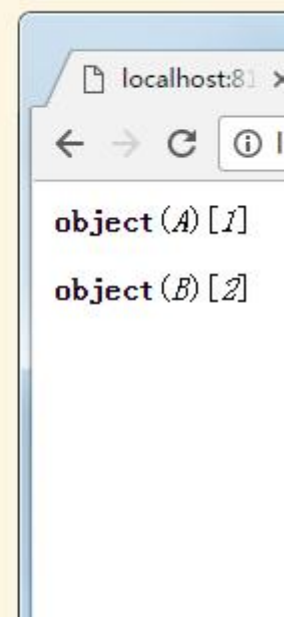
fun2.....

```
//////////可变类//////////
class A{

}
class B{

}
$class_name = 'A';
$obj1 = new $class_name(); //可变类

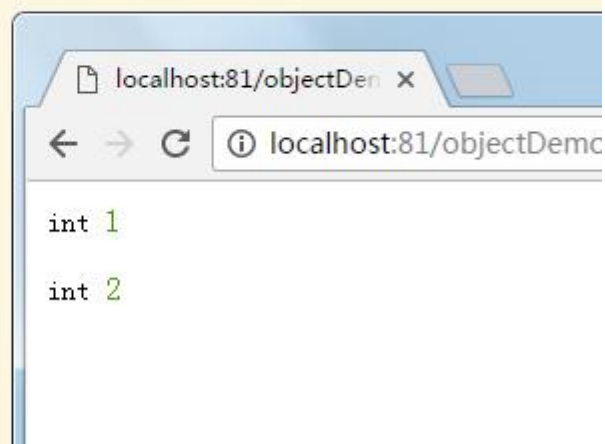
$class_name = 'B';
$obj2 = new $class_name();
var_dump($obj1,$obj2);
```



//前提知识2：静态局部变量
/*在局部变量前加上“static”关键字，就成了静态局部变量。静态局部变量存放在内存的全局数据区。函数结束时，静态局部变量不会消失，每次该函数调用时，也不会为其重新分配空间。它始终驻留在全局数据区，直到程序运行结束。
*/

```
function fun(){
    static $var = 0;
    $var++;
    var_dump($var);
}

fun();
fun();
```



实现方法：

定义一个普通函数,在函数内声明一个静态局部变量保存某个类的单例对象。如果该对象存在则直接返回；不存在则创建类对象后再返回。


```
/**
 * 比较常见的实现单例效果的方法
 */
class A{

}
class B{

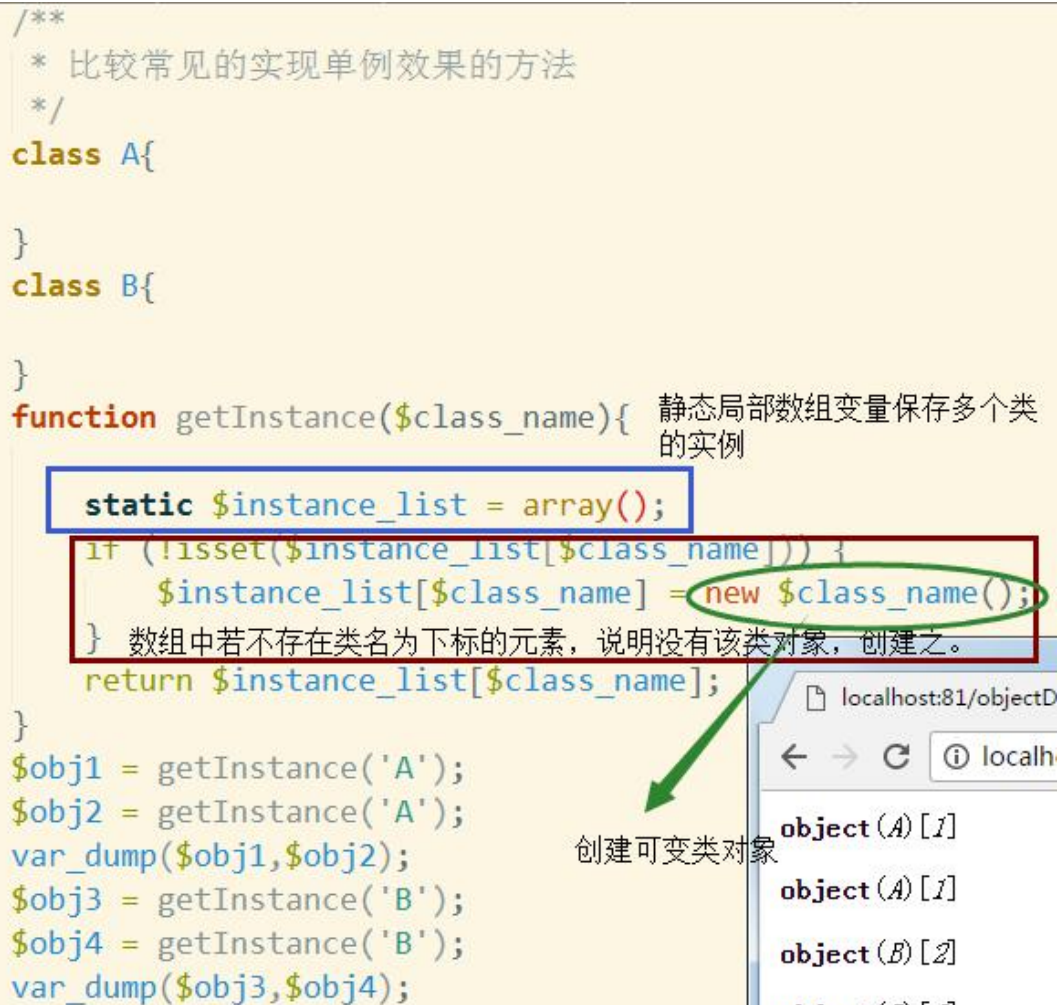
}

function getInstance($class_name){ 静态局部数组变量保存多个类的实例

    static $instance_list = array();

    if (!isset($instance_list[$class_name])) {
        $instance_list[$class_name] = new $class_name();
    } 数组中若不存在类名为下标的元素，说明没有该类对象，创建之。
    return $instance_list[$class_name];
}

$obj1 = getInstance('A');
$obj2 = getInstance('A');
var_dump($obj1,$obj2);
$obj3 = getInstance('B');
$obj4 = getInstance('B');
var_dump($obj3,$obj4);
```



localhost:81/objectD

object (A) [1]
object (A) [1]
object (B) [2]
object (B) [2]

创建可变类对象

优势:

- 1、比较灵活，不用改变类本身的结构。
- 2、可以适用于多个类的单例效果。

缺点：不是从根本上解决单例的问题。依然能够创建多个对象。不能称为单例模式，只是实现单例的一种方法。

单例模式总结：

如果在实际开发中，要求严格限制单例效果，使用单例设计模式（三私有，一公共）；如果仅仅是想实现多个类的单例效果，可以使用类

外定义的方式，也可以使用下面介绍的工厂类。

二、工厂模式

如果有一个类，它的作用不是实现某些业务逻辑，而是为了创建其他类型的对象而存在的。那么这样的类就称作工厂类。——能够生产其他类对象的工厂。

```
1  <?php
2      /**
3       * Factory工厂类
4       */
5      class Factory
6      {
7          public static function getInstance($class_name)
8          {
9              //静态局部变量，函数调用后，不会消失，下次调用还会存在
10             static $instance_list = array();
11             if (!isset($instance_list[$class_name])) {
12                 $instance_list[$class_name] = new $class_name();
13             }
14             return $instance_list[$class_name];
15         }
16     }
17     class A{
18     }
19     class B{
```

```
21
22     }
23     $a1 = Factory::getInstance(A);
24     $a2 = Factory::getInstance(A);
25     var_dump($a1,$a2);
26
27     $b1 = Factory::getInstance(B);
28     $b2 = Factory::getInstance(B);
29     var_dump($b1,$b2);
30     ?>
```



为什么要设计工厂类？

1、从面向对象的角度来讲，不适合使用类外定义的方法，它属于面

向过程。

2、如果在创建对象时需要增加一些业务逻辑，而这些业务逻辑并不方便放置在类的构造方法里，就需要有个单独的类来处理业务逻辑。于是，使用工厂类的一个静态方法处理这个业务逻辑是最好的办法。

```
1  <?php
2  /**
3   * Factory工厂类
4   */
5  class Factory
6  {
7      //例子1: 处理单例的方法
8      public static function getInstance($class_name)
9      {
10
11      }
12      //下面可以添加更多需要添加业务逻辑判断创建何种对象的方法
13
14      //例子2: 根据实际情况来创建不同格式的图像处理类对象，从而处理图像
15      public static function getImage($file){
16          //增加业务逻辑判断:
17          //假设$type保存$file对象的类型
18          if ($type == 'png') {
19              //$file文件如果是PNG格式，则创建类ImagePNG的对象
20              return new ImagePNG();
21          } else if($type == 'jpg') {
22
23              //$file文件如果是JPG格式，则创建类ImageJPG的对象
24              return new ImageJPG();
25          }
26      }
27
28      //处理PNG格式图像的处理类
29      class ImagePNG{
30
31      }
32      //处理JPG格式图像的处理类
33      class ImageJPG{
34
35      }
36
37      $a1 = Factory::getImage('1.jpg');
38
39  ?>
```

此时，创建的对象应该是ImageJPG类型的

注意：工厂类中的方法一般都是静态的（不绝对），因为工厂类本身就是为了创建其他类的对象的，没必要创建工厂类本身的对象。所以

从设计思路上考虑，将创建其他类对象的方法定义成静态即可。

三、面向对象的三大特征

总结出来的特点,不算语法。

1、封装

隐藏内部实现，仅仅开放外部访问的接口。

语法体现：public/protected/private 访问修饰符。

2、继承

一个对象的成员被其他对象（类）所使用，体现了代码的重用。

语法体现：extends

3、多态

一个动作有多种形式、一个方法有多种实现方式、一个事物有多种状态....这种情况就叫做多态。

例如：从抽象类商品 Goods 的角度看可以是图书，可以是手机,这就是多态；从打折的动作看，有时只能打 3 折以上，有时只能打 7 折以上,这也体现了多态。

语法体现：继承和重写

知识回顾：什么是重~~写~~？在子类中重新定义父类的属性和方法叫做重写。一般通过重写父类的方法来实现功能的变更。

特别说明一个问题——PHP 不支持传统意义上的函数重载：

在 PHP 中不能出现同名的方法，会报错。就是说不能在同一个类中重载方法。即使方法的参数个数不同也不可以。

```
1 <?php
2     function fun1(){
3
4     }
5     function fun1($arg){
6
7     }
8
9 ?>
```

localhost:81/objectDemo/duotai/1.php

(!) Fatal error: Cannot redeclare fun1() (previously declared in F:\wamp\www\objectDemo\duotai\1.php:2) in F:\wamp\www\objectDemo\duotai\1.php on line 7

```
1 <?php
2 class A{
3     public function fun1(){
4
5     }
6     public function fun1($arg){
7
8     }
9 }
10 ?>
```

localhost:81/objectDemo/duotai/1.php

(!) Fatal error: Cannot redeclare A::fun1() in F:\wamp\www\objectDemo\duotai\1.php on line 6

而在一些强类型的语言中,例如 java，可以通过参数的个数、参数的类型区分同名的方法，这种情况称作函数的**重载**。函数的重载也能体现多态。



```
1 package aa;
2
3 public class A {
4     public int fun(){
5         return 0;
6     }
7     public String fun(String str){
8         return str;
9     }
10
11     public int fun(int i){
12         return i;
13     }
14 }
15
```

参数类型和数量不同实现函数重载。

但是 PHP 不支持重载。因为 PHP 是弱类型，定义的变量不用声明类型，所以不能通过类型区分。