PHP几种常见的设计模式

1、单例模式

单例模式顾名思义，就是一个类只有一个实例，并能自动地向整个系统提供整个实例。

如果系统中需要有一个类来控制某些全局信息，或者操作的资源非常耗时，比如数据库连接等，那么使用单例模式可以很方便的实现。

示例代码：

单例模式

```
classman
{
//保存例实例在此属性中
privatestatic$_instance;

//构造函数声明为private,防止直接创建对象
privatefunction__construct()
{
echo'我被实例化了！';
}

//单例方法
publicstaticfunctionget_instance()
{
var_dump(isset(self::$_instance));

if(!isset(self::$_instance))
{
self::$_instance=newself();
}
returnself::$_instance;
}

//阻止用户复制对象实例
 privatefunction__clone()
{
trigger_error('Cloneisnotallow',E_USER_ERROR);
}

functiontest()
{
echo("test");

}
}

//这个写法会出错，因为构造方法被声明为private
//$test=newman;

//下面将得到Example类的单例对象
$test=man::get_instance();
$test=man::get_instance();
$test->test();

//复制对象将导致一个E_USER_ERROR.
//$test_clone=clone$test;
```

2、观察者模式

该模式是指当一个对象的状态发生改变时，所有依赖于它的对象可以得到通知并自动刷新。采用该模式可以避免观察者一直轮询。

在PHP中要实现观察者模式，需要使用SPL（即PHP标准模板库），SPL中的接口如下：

表 1. SplSubject 接口中的方法

方法声明	描述
abstract public void attach (SplObserver $observer)	添加（注册）一个观察者

abstract public void detach (SplObserver $observer)	删除一个观察者
abstract public void notify (void)	当状态发生改变时，通知所有观察者

表 2. SplObserver 中的方法

方法声明	描述
abstract public void update (SplSubject $subject)	在目标发生改变时接收目标发送的通知；当关注的目标调用其`notify()`时被调用

该设计模式的核心思想是，SplSubject对象会在其状态改变时调用notify()方法，一旦这个方法被调用，任何先前通过attach()方法注册上来的SplObserver对象都会以调用其update()方法的方式被更新。

示例代码：

观察者模式

```
classMyObserver1implementsSplObserver{
publicfunctionupdate(SplSubject$subject){
echo__CLASS__.'='.$subject->getName();
}
}

classMyObserver2implementsSplObserver{
publicfunctionupdate(SplSubject$subject){
echo__CLASS__.'='.$subject->getName();
}
}

classMySubjectimplementsSplSubject{
private$_observers;
private$_name;

publicfunction__construct($name){
$this->_observers=newSplObjectStorage();
$this->_name=$name;
}

publicfunctionattach(SplObserver$observer){
$this->_observers->attach($observer);
}

publicfunctiondetach(SplObserver$observer){
$this->_observers->detach($observer);
}

publicfunctionnotify(){
foreach($this->_observersas$observer){
$observer->update($this);
}
}

publicfunctiongetName(){
return$this->_name;
}
}

$observer1=newMyObserver1();
$observer2=newMyObserver2();

$subject=newMySubject("test");

$subject->attach($observer1);
$subject->attach($observer2);
```

调用：

```
$subject->notify();
```

SplObjectStorage类实现了以对象为键的映射（map）或对象的集合（如果忽略作为键的对象所对应的数据）这种数据结构。这个类的实例很像一个数组，但是它所存放的对象都是唯一的，因为其attach()方法判断了指定的对象是否已经被存储。

SplObjectStorage::attach() 方法的部分源代码

```
function attach($obj, $inf = NULL)
 {
    if (is_object($obj) && !$this->contains($obj))
    {
        $this->storage[] = array($obj, $inf);
    }
 }
```

3、简单工厂模式

简单工厂模式的要点在于：

①抽象基类：类中定义抽象一些方法，用以在子类中实现

②继承自抽象基类的子类：实现基类中的抽象方法

③工厂类：用以实例化所有相对应的子类

简单工厂模式

```
/**
*
*定义个抽象的类，让子类去继承实现它
*
*/
abstractclassOperation{
//抽象方法不能包含函数体
abstractpublicfunctiongetValue($num1,$num2);//强烈要求子类必须实现该功能函数
}
/**
*加法类
*/
classOperationAddextendsOperation{
publicfunctiongetValue($num1,$num2){
return$num1+$num2;
}
}
/**
*减法类
*/
classOperationSubextendsOperation{
publicfunctiongetValue($num1,$num2){
return$num1-$num2;
}
}
/**
*乘法类
*/
classOperationMulextendsOperation{
publicfunctiongetValue($num1,$num2){
return$num1*$num2;
}
}
/**
*除法类
*/
classOperationDivextendsOperation{
publicfunctiongetValue($num1,$num2){
try{
if($num2==0){
thrownewException("除数不能为0");
}else{
return$num1/$num2;
}
}catch(Exception$e){
echo"错误信息：".$e->getMessage();
}
```

```
}
}
```

如果我们现在需要增加一个求余的类，会非常的简单。

求余类

```
/**
*求余类（remainder）
*
*/
classOperationRemextendsOperation{
publicfunctiongetValue($num1,$num2){
return$num1%$num12;
}
}
```

目前只需要编写生成这些类的工厂即可。

工厂类

```
/**
*工程类，主要用来创建对象
*功能：根据输入的运算符号，工厂就能实例化出合适的对象
*
*/
classFactory{
publicstaticfunctioncreateObj($operate){
switch($operate){
case'+':
returnnewOperationAdd();
break;
case'-':
returnnewOperationSub();
break;
case'*':
returnnewOperationSub();
break;
case'/':
returnnewOperationDiv();
break;
}
}
}
$test=Factory::createObj('/');
$result=$test->getValue(23,0);
echo$result;
```