导航

博客园
首　页
新随笔
联　系
订　阅 XML
管　理

< 2019年1月 >

日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

公告

昵称：原万里
园龄：2年4个月
粉丝：2
关注：1
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

php(26)
mysql(17)
JScript(12)
web(8)
Git(4)
Nginx(4)
Linux(3)
Ajax(3)
正则表达式(3)
ThinkPhp(2)
更多

随笔档案

2018年9月 (1)
2018年6月 (2)
2018年5月 (10)
2018年4月 (13)
2018年3月 (3)
2018年1月 (1)
2017年9月 (1)
2017年8月 (1)
2017年7月 (6)
2017年5月 (4)

PHP八大设计模式

设计模式

单例模式解决的是如何在整个项目中创建唯一对象实例的问题，工厂模式解决的是如何不通过new建立实例对象的方法。

单例模式

1. $_instance必须声明为静态的私有变量
2. 构造函数和析构函数必须声明为私有,防止外部程序new 类从而失去单例模式的意义
3. getInstance()方法必须设置为公有的,必须调用此方法 以返回实例的一个引用
4. ::操作符只能访问静态变量和静态函数
5. new对象都会消耗内存
6. 使用场景:最常用的地方是数据库连接。
7. 使用单例模式生成一个对象后， 该对象可以被其它众多对象所使用。
8. 私有的__clone()方法防止克隆对象

单例模式，使某个类的对象仅允许创建一个。构造函数private修饰，
申明一个static getInstance方法，在该方法里创建该对象的实例。如果该实例已经存在，则不创建。比如只需要创建一个数据库连接。

工厂模式

工厂模式，工厂方法或者类生成对象，而不是在代码中直接new。
使用工厂模式，可以避免当改变某个类的名字或者方法之后，在调用这个类的所有的代码中都修改它的名字或者参数。

```
Test1.php
<?php
class Test1{
    static function test(){
        echo __FILE__;
    }
}

Factory.php
<?php
class Factory{
    /*
     * 如果某个类在很多的文件中都new ClassName()，那么万一这个类的名字
     * 发生变更或者参数发生变化，如果不使用工厂模式，就需要修改每一个PHP
     * 代码，使用了工厂模式之后，只需要修改工厂类或者方法就可以了。
     */
    static function createDatabase(){
        $test = new Test1();
        return $test;
    }
}

Test.php
<?php
spl_autoload_register('autoload1');

```

2017年3月 (3)
2016年10月 (10)
2016年9月 (12)
2016年8月 (4)

最新评论

1. Re:PHP获取文件扩展名五种以上的方法和注释
PHP 获取文件扩展名
--zhangzilong4511
2. Re:如何将浏览器上的JS文件屏蔽
用chrome浏览器，直接可以设置针对某个网站禁用js
--Pharaoh
3. Re:jquery 登录判断遇到的小问题
像这种代码段不是编辑器里有专门的写法吗?
直接文字粘上来不容易看懂。。。
--原鹏超
4. Re:PHP魔术方法和魔法变量详解
好多都忘了，正好复习一下~
--原鹏超

阅读排行榜

1. setcookie（）函数(9955)
2. PHP获取文件扩展名五种以上的方法和注释(9249)
3. jquery注册验证的写法(6078)
4. Mysql 纪录用户操作日志(5777)
5. jquery 复合事件 toggle()方法的使用(4447)

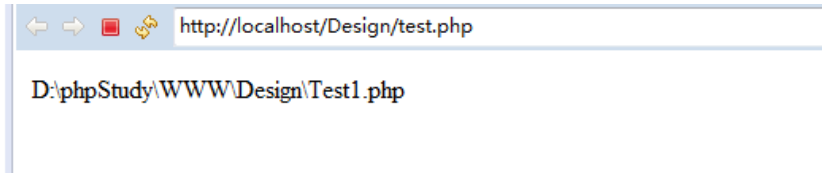
评论排行榜

1. PHP获取文件扩展名五种以上的方法和注释(1)
2. 如何将浏览器上的JS文件屏蔽(1)
3. jquery 登录判断遇到的小问题(1)
4. PHP魔术方法和魔法变量详解(1)

推荐排行榜

1. Mysql 纪录用户操作日志(1)

```
27 $test = Factory::createDatabase();
28 $test->test();
29 function autoload1($class){
30     $dir  = __DIR__;
31     $requireFile = $dir."\\".$class.".php";
32     require $requireFile;
33 }
```



注册模式

注册模式，解决全局共享和交换对象。已经创建好的对象，挂在到某个全局可以使用的数组上，在需要使用的时候，直接从该数组上获取即可。将对象注册到全局的树上。任何地方直接去访问。

```
1 <?php
2
3 class Register
4 {
5     protected static  $objects;
6     function set($alias,$object)//将对象注册到全局的树上
7     {
8         self::$objects[$alias]=$object;//将对象放到树上
9     }
10    static function get($name){
11        return self::$objects[$name];//获取某个注册到树上的对象
12    }
13    function _unset($alias)
14    {
15        unset(self::$objects[$alias]);//移除某个注册到树上的对象。
16    }
17 }
```

适配器模式

将各种截然不同的函数接口封装成统一的API。

PHP中的数据库操作有MySQL,MySQLi,PDO三种，可以用适配器模式统一成一致，使不同的数据库操作，统一成一样的API。类似的场景还有cache适配器，可以将memcache,redis,file,apc等不同的缓存函数，统一成一致。

首先定义一个接口(有几个方法，以及相应的参数)。然后，有几种不同的情况，就写几个类实现该接口。将完成相似功能的函数，统一成一致的方法。

```
1 接口 IDatabase
2 <?php
3 namespace IMooc;
4 interface IDatabase
5 {
6     function connect($host, $user, $passwd, $dbname);
7     function query($sql);
8     function close();
9 }
```

```
MySQL
<?php
namespace IMooc\Database;
use IMooc\IDatabase;
class MySQL implements IDatabase
{
    protected $conn;
    function connect($host, $user, $passwd, $dbname)
    {
        $conn = mysql_connect($host, $user, $passwd);
        mysql_select_db($dbname, $conn);
        $this->conn = $conn;
    }

    function query($sql)
    {
        $res = mysql_query($sql, $this->conn);
        return $res;
    }

    function close()
    {
        mysql_close($this->conn);
    }
}
```

```
MySQLi
<?php
namespace IMooc\Database;
use IMooc\IDatabase;
class MySQLi implements IDatabase
{
    protected $conn;

    function connect($host, $user, $passwd, $dbname)
    {
        $conn = mysqli_connect($host, $user, $passwd, $dbname);
        $this->conn = $conn;
    }

    function query($sql)
    {
        return mysqli_query($this->conn, $sql);
    }

    function close()
    {
        mysqli_close($this->conn);
    }
}
```

```
PDO
<?php
```

```
namespace IMooc\Database;
use IMooc\IDatabase;
class PDO implements IDatabase
{
    protected $conn;
    function connect($host, $user, $passwd, $dbname)
    {
        $conn = new \PDO("mysql:host=$host;dbname=$dbname", $user, $passwd);
        $this->conn = $conn;
    }
function query($sql)
    {
        return $this->conn->query($sql);
    }

    function close()
    {
        unset($this->conn);
    }
}
```

通过以上案例，PHP与MySQL的数据库交互有三套API，在不同的场景下可能使用不同的API，那么开发好的代码，换一个环境，可能就要改变它的数据库API，那么就要改写所有的代码，使用适配器模式之后，就可以使用统一的API去屏蔽底层的API差异带来的环境改变之后需要改写代码的问题。

策略模式

策略模式，将一组特定的行为和算法封装成类，以适应某些特定的上下文环境。
eg：假如有一个电商网站系统，针对男性女性用户要各自跳转到不同的商品类目，并且所有的广告位展示不同的广告。在传统的代码中，都是在系统中加入各种if else的判断，硬编码的方式。如果有一天增加了一种用户，就需要改写代码。使用策略模式，如果新增加一种用户类型，只需要增加一种策略就可以。其他所有的地方只需要使用不同的策略就可以。
首先声明策略的接口文件，约定了策略的包含的行为。然后，定义各个具体的策略实现类。

```
UserStrategy.php
<?php
/*
 * 声明策略文件的接口，约定策略包含的行为。
 */
interface UserStrategy
{
    function showAd();
    function showCategory();
}
```

```
FemaleUser.php
<?php
require_once 'Loader.php';
class FemaleUser implements UserStrategy
{
    function showAd(){
        echo "2016冬季女装";
    }
    function showCategory(){
```

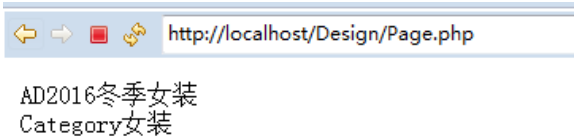
```
        echo "女装";
    }
}
```

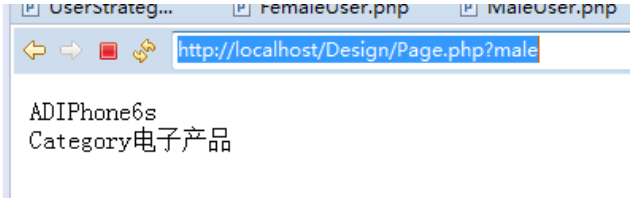
```
MaleUser.php
<?php
require_once 'Loader.php';
class MaleUser implements UserStrategy
{
    function showAd(){
        echo "IPhone6s";
    }
    function showCategory(){
        echo "电子产品";
    }
}
```

```
Page.php//执行文件
<?php
require_once 'Loader.php';
class Page
{
    protected $strategy;
    function index(){
        echo "AD";
        $this->strategy->showAd();
        echo "<br>";
        echo "Category";
        $this->strategy->showCategory();
        echo "<br>";
    }
    function setStrategy(UserStrategy $strategy){
        $this->strategy=$strategy;
    }
}

$page = new Page();
if(isset($_GET['male'])){
    $strategy = new MaleUser();
}else {
    $strategy = new FemaleUser();
}
$page->setStrategy($strategy);
$page->index();
```

执行结果图：





总结：

通过以上方式，可以发现，在不同用户登录时显示不同的内容，但是解决了在显示时的硬编码的问题。如果要增加一种策略，只需要增加一种策略实现类，然后在入口文件中执行判断，传入这个类即可。实现了解耦。

实现依赖倒置和控制反转 *（有待理解）*

通过接口的方式，使得类和类之间不直接依赖。在使用该类的时候，才动态的传入该接口的一个实现类。如果要替换某个类，只需要提供一个实现了该接口的实现类，通过修改一行代码即可完成替换。

观察者模式

1：观察者模式(Observer)，当一个对象状态发生变化时，依赖它的对象全部会收到通知，并自动更新。

2：场景:一个事件发生后，要执行一连串更新操作。传统的编程方式，就是在事件的代码之后直接加入处理的逻辑。当更新的逻辑增多之后，代码会变得难以维护。这种方式是耦合的，侵入式的，增加新的逻辑需要修改事件的主体代码。

3：观察者模式实现了低耦合，非侵入式的通知与更新机制。

定义一个事件触发抽象类。

```
EventGenerator.php
<?php
require_once 'Loader.php';
abstract class EventGenerator{
    private $observers = array();
    function addObserver(Observer $observer){
        $this->observers[]=$observer;
    }
    function notify(){
        foreach ($this->observers as $observer){
            $observer->update();
        }
    }
}
```

定义一个观察者接口

```
Observer.php
<?php
require_once 'Loader.php';
interface Observer{
    function update();//这里就是在事件发生后要执行的逻辑
}
```

```
<?php
//一个实现了EventGenerator抽象类的类，用于具体定义某个发生的事件
```

```
require 'Loader.php';
class Event extends EventGenerator{
    function triger(){
        echo "Event<br>";
    }
}
class Observer1 implements Observer{
    function update(){
        echo "逻辑1<br>";
    }
}
class Observer2 implements Observer{
    function update(){
        echo "逻辑2<br>";
    }
}
$event = new Event();
$event->addObserver(new Observer1());
$event->addObserver(new Observer2());
$event->triger();
$event->notify();
```

当某个事件发生后，需要执行的逻辑增多时，可以以松耦合的方式去增删逻辑。也就是代码中的红色部分，只需要定义一个实现了观察者接口的类，实现复杂的逻辑，然后在红色的部分加上一行代码即可。这样实现了低耦合。

原型模式

原型模式（对象克隆以避免创建对象时的消耗）

1：与工厂模式类似，都是用来创建对象。

2：与工厂模式的实现不同，原型模式是先创建好一个原型对象，然后通过clone原型对象来创建新的对象。这样就免去了类创建时重复的初始化操作。

3：原型模式适用于大对象的创建，创建一个大对象需要很大的开销，如果每次new就会消耗很大，原型模式仅需要内存拷贝即可。

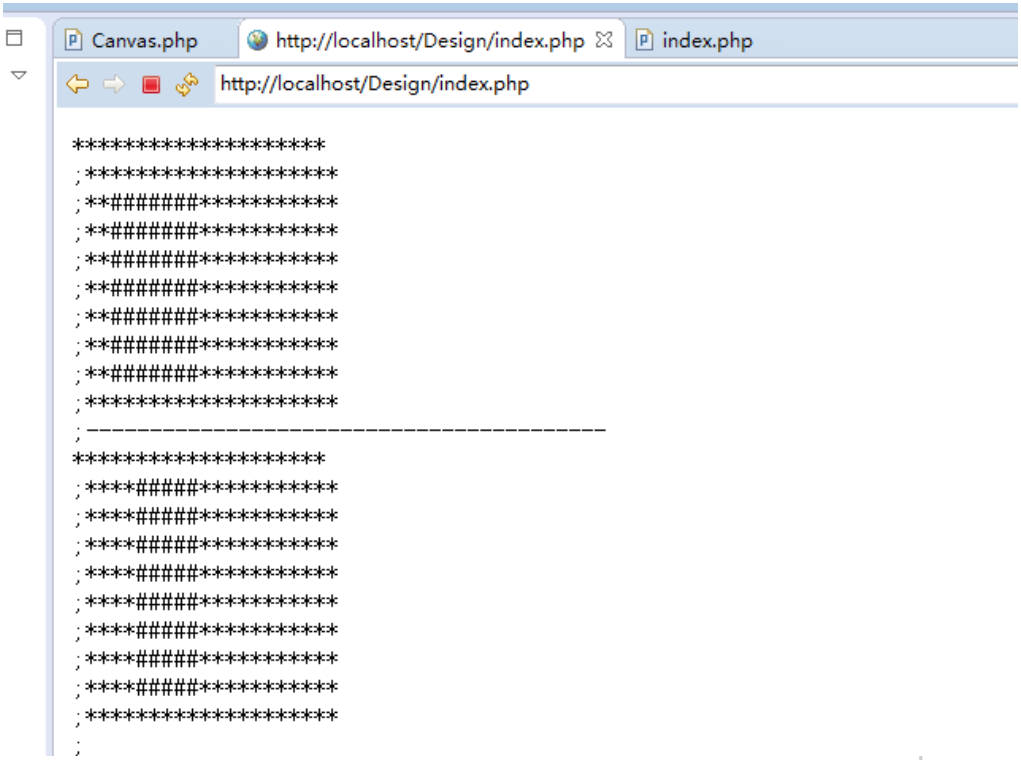
```
Canvas.php
<?php
require_once 'Loader.php';
class Canvas{
private $data;
function init($width = 20, $height = 10)
    {
        $data = array();
        for($i = 0; $i < $height; $i++)
        {
            for($j = 0; $j < $width; $j++)
            {
                $data[$i][$j] = '*';
            }
        }
        $this->data = $data;
    }
function rect($x1, $y1, $x2, $y2)
    {
        foreach($this->data as $k1 => $line)
        {
            if ($x1 > $k1 or $x2 < $k1) continue;
            foreach($line as $k2 => $char)
            {
```

```
                if ($y1>$k2 or $y2<$k2) continue;
                  $this->data[$k1][$k2] = '#';
              }
          }
      }

      function draw(){
          foreach ($this->data as $line){
              foreach ($line as $char){
                  echo $char;
              }
              echo "<br>;";
          }
      }
  }
```

```
1 Index.php
2 <?php
3 require 'Loader.php';
4 $c = new Canvas();
5 $c->init();
6 / $canvas1 = new Canvas();
7 // $canvas1->init();
8 $canvas1 = clone $c;//通过克隆，可以省去init()方法，这个方法循环两百次
9 //去产生一个数组。当项目中需要产生很多的这样的对象时，就会new很多的对象，那样
10 //是非常消耗性能的。
11 $canvas1->rect(2, 2, 8, 8);
12 $canvas1->draw();
13 echo "----------------------------------------<br>";
14 // $canvas2 = new Canvas();
15 // $canvas2->init();
16 $canvas2 = clone $c;
17 $canvas2->rect(1, 4, 8, 8);
18 $canvas2->draw();
```

执行结果：



装饰器模式

1：装饰器模式，可以动态的添加修改类的功能

2：一个类提供了一项功能，如果要在修改并添加额外的功能，传统的编程模式，需要写一个子类继承它，并重写实现类的方法

3：使用装饰器模式，仅需要在运行时添加一个装饰器对象即可实现，可以实现最大额灵活性。

转自：https://blog.csdn.net/flitrue/article/details/52614599

如有侵权，请联系删除。

标签: php, 设计模式

原万里
关注 - 1
粉丝 - 2
+加关注

0 0

« 上一篇：TCP的三次握手和四次握手
» 下一篇：PHP抽象类与接口的区别

posted on 2018-04-11 16:16 原万里 阅读(443) 评论(0) 编辑 收藏

相关博文：
· 八大Android土鳖设计
· SHELL 八大扩展
· 八大排序
· 世界八大奇迹
· 八大算法思想

最新新闻：
· 联想中国区调整构架：聚焦大客户等三大客户群
· 特斯拉上海工厂动工 马斯克与蔚来们的“中国战事”
· 繁荣抖音背后，焦虑的底层内容工厂
· 隼鸟2号即将“惊险”着陆 "龙宫小行星"首批研究成果
· 亚马逊缘何成为全球市值最高的科技公司？5个原因
» 更多新闻...