**为什么要采用注册树模式？**

　　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。 那么注册树模式想解决什么问题呢？ 在考虑这个问题前，我们还是有必要考虑下前两种模式目前面临的局限。  首先，单例模式创建唯一对象的过程本身还有一种判断，即判断对象是否存在。存在则返回对象，不存在则创建对象并返回。 每次创建实例对象都要存在这么一层判断。 工厂模式更多考虑的是扩展维护的问题。 总的来说，单例模式和工厂模式可以产生更加合理的对象。怎么方便调用这些对象呢？而且在项目内如此建立的对象好像散兵游勇一样，不便统筹管理安排啊。因而，注册树模式应运而生。不管你是通过单例模式还是工厂模式还是二者结合生成的对象，都统统给我“插到”注册树上。我用某个对象的时候，直接从注册树上取一下就好。这和我们使用全局变量一样的方便实用。 而且注册树模式还为其他模式提供了一种非常好的想法。

## 注册模式：

<https://www.cnblogs.com/DeanChopper/p/4767181.html>

不管你是通过单例模式还是工厂模式还是二者结合生成的对象，都统统给我“插到”注册树上。我用某个对象的时候，直接从注册树上取一下就好。这和我们使用全局变量一样的方便实用。注册树应该由一个静态变量来充当。而且这个注册树应该是一个二维数组。这个类应该有一个插入对象实例的方法（set()），当让相对应的就应该有一个撤销对象实例的方法（\_unset()）。当然最重要的是还需要有一个读取对象的方法（get()）

<?php

//创建单例

class Single{

public $hash;

static protected $ins=null;

final protected function \_\_construct(){

$this->hash=rand(1,9999);

}

static public function getInstance(){

if (self::$ins instanceof self) {

return self::$ins;

}

self::$ins=new self();

return self::$ins;

}

}

//工厂模式

class RandFactory{

public static function factory(){

return Single::getInstance();

}

}

//注册树

class Register{

protected static $objects;

public static function set($alias,$object){

self::$objects[$alias]=$object;

}

public static function get($alias){

return self::$objects[$alias];

}

public static function \_unset($alias){

unset(self::$objects[$alias]);

}

}

Register::set('rand',RandFactory::factory());

$object=Register::get('rand');

print\_r($object);