这篇文章主要介绍了PHP设计模式之单例模式定义与用法,结合实例形式较为详细的分析了单例模式的概念、特点、优缺点,并结合php实例形式分析了单例模式的定义与使用方法,需要的朋友可以参考下

. .本文实例分析了PHP设计模式之单例模式。分享给大家供大家参考，具体如下：

单例模式（Singleton Pattern 单件模式或单元素模式），是常见的一种设计模式，它有三个特点

•1．只能有一个实例

•2．必须自行创建这个实例

•3．必须给其他对象提供这一实例

下面用PHP代码实现一下

?

1234567891011121314151617181920212223242526272829 <?PHP /\*\* \* Created by PHPStorm. \* User: tiansi \* Date: 18/1/2 \* Time: 下午3:40 \*/class Signleton{ private static $\_instanse = null; //私有化构造方法 防止外界使用new实例化对象 private function \_\_construct() { } //私有化克隆方法 防止外界克隆对象 private function \_\_clone() { // TODO: Implement \_\_clone() method. } //静态化 提供单例访问入口 static function getInstance(){ if (is\_null(self::$\_instanse) || !isset(self::$\_instanse)){ self::$\_instanse = new self(); } return self::$\_instanse; } public function say(){ echo 'I am signleton'; } }

<?PHP

/\*\*

\* Created by PHPStorm.

\* User: tiansi

\* Date: 18/1/2

\* Time: 下午3:40

\*/

class Signleton{

private static $\_instanse = null;

//私有化构造方法 防止外界使用new实例化对象

private function \_\_construct()

{

}

//私有化克隆方法 防止外界克隆对象

private function \_\_clone()

{

// TODO: Implement \_\_clone() method.

}

//静态化 提供单例访问入口

static function getInstance(){

if (is\_null(self::$\_instanse) || !isset(self::$\_instanse)){

self::$\_instanse = new self();

}

return self::$\_instanse;

}

public function say(){

echo 'I am signleton';

}

}下面试一下调用

?

1234567891011121314151617 <?PHP /\*\* \* Created by PHPStorm. \* User: tiansi \* Date: 18/1/2 \* Time: 下午3:48 \*///第一种方式会报错 因为把构造方法私有化了 //PHP Fatal error: Uncaught Error: Call to private Signleton::\_\_construct() from invalid context in /Users/apple/uxin/SignletonController.PHP:11 /\* $signleton1 = new Signleton(); $signleton1->say(); \*///上述代码报错：Fatal error: Call to private Signleton::\_\_construct() //实例化成功 输出I am signleton $signleton2 = Signleton::getInstance(); $signleton2->say();

<?PHP

/\*\*

\* Created by PHPStorm.

\* User: tiansi

\* Date: 18/1/2

\* Time: 下午3:48

\*/

//第一种方式会报错 因为把构造方法私有化了

//PHP Fatal error: Uncaught Error: Call to private Signleton::\_\_construct() from invalid context in /Users/apple/uxin/SignletonController.PHP:11

/\*

$signleton1 = new Signleton();

$signleton1->say();

\*/

//上述代码报错：Fatal error: Call to private Signleton::\_\_construct()

//实例化成功 输出I am signleton

$signleton2 = Signleton::getInstance();

$signleton2->say();运行结果：

I am signleton

它的优缺点

优点：

•1.在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这样就 防止其它对象对自己的实例化，确保所有的对象都访问一个实例

•2.单例模式具有一定的伸缩性，类自己来控制实例化进程，类就在改变实例化进程上有相应的伸缩性。

•3.提供了对唯一实例的受控访问。

•4.由于在系统内存中只存在一个对象，因此可以 节约系统资源，当 需要频繁创建和销毁的对象时单例模式无疑可以提高系统的性能。

•5.允许可变数目的实例。

•6.避免对共享资源的多重占用。

缺点：

•1.不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。

•2.由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。

•3.单例类的职责过重，在一定程度上违背了“单一职责原则”。

•4.滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。

这是单例模式的优缺点，因为语言类型的不同，PHP的单例模式又不一样

众所周知，PHP语言是一种解释型的脚本语言，这种运行机制使得每个PHP页面被解释执行后，所有的相关资源都会被回收。也就是说，PHP在语言级别上没有办法让某个对象常驻内存，这和asp.net、Java等编译型是不同的，比如在Java中单例会一直存在于整个应用程序的生命周期里，变量是跨页面级的，真正可以做到这个实例在应用程序生命周期中的唯一性。然而在PHP中，所有的变量无论是全局变量还是类的静态成员，都是页面级的，每次页面被执行时，都会重新建立新的对象，都会在页面执行完毕后被清空，这样似乎PHP单例模式就没有什么意义了，所以PHP单例模式我觉得只是针对单次页面级请求时出现多个应用场景并需要共享同一对象资源时才有用

比如

1.应用程序与数据库交互

一个应用中会存在大量的数据库操作，比如过数据库句柄来连接数据库这一行为，使用单例模式可以避免大量的new操作，因为每一次new操作都会消耗内存资源和系统资源。

2.控制配置信息

如果系统中需要有一个类来全局控制某些配置信息, 那么使用单例模式可以很方便的实现