**单件模式：**确保一个类只有一个实例，并提供全局访问点。

**这一站我们来到了单件模式(Singleton Pattern):用来创建独一无二的，只有一个实例的对象的入场券。告诉大家一个好消息，单件模式的类图算是所有设计模式的类图中最简单的，事实上，它的类图上只有一个类！但是，可不要兴奋过头，尽管从类设计的视角来说它很简单，但是实现上还是会遇到相当多的波折。所以，系好安全带，我们出发咯！**

**如何创建一个对象？**

**new MyObject();**

**万一另一个对象想创建MyObject()会怎样？可以再次new MyObject吗？**

**是的，当然可以。**

**所以，一旦有一个类，我们是否都能够多次地实例化它？**

**如果是公开的类，就可以。**

**如果不是的话，会怎样？**

**如果不是公开类，只有同一个包内的类可以实例化它，但是仍可以实例化它多次。**

**嗯！有意思！你知道的可以这样做吗？**

public MyClass {

private MyClass() {}

}

**我没想过。但是，这是合法的定义，有一定的道理。**

**怎么说呢？**

**我认为含有私有的构造器的类不能被实例化。**

**有可以使用私有的构造器的对象吗？**

**嗯，我想MyClass内的代码是唯一能调用此结构的代码。但是这有不太合乎常理。**

**为什么？**

**因为必须有MyClass类的实例才能调用MyClass构造器，但是因为没有其他类能够实例化MyClass，所以我们得不到这样的实例。这是“鸡生蛋，蛋生鸡”的问题。我可以在MyClass类型的对象上使用MyClass构造器，但是在这之前，必须有一个MyClass实例。在产生MyClass实例之前，又必须在MyClass实例内才能调用私有的构造器......**

**嗯！我有个想法。你认为这样如何？**

public MyClass {

public static MyClass getInstance() {

}

}

**MyClass有一个静态方法。我们可以这样调用这个方法：MyClass.getInstance();**

**为何调用的时候以MyClass的类名，而不是对象名？**

**因为getInstance()是一个静态方法，换句话说，是一个“类”方法，你需要使用类名。**

**有意思。假如把这些合在一起“是否”就可以初始化一个MyClass？**

[复制代码](javascript:void(0);)

public MyClass {

private MyClass() {}

public static MyClass getInstance() {

return new MyClass()

}

}

[复制代码](javascript:void(0);)

**当然可以。**

**好了，你能想出第二种实例化对象的方式吗？**

**MyClass.getInstance()；**

**你能完成代码使MyClass只有一个实例被产生吗？**

**嗯，大概可以吧......**

[复制代码](javascript:void(0);)

public class Singleton {//把MyClass改名为Singleton

private static Singleton uniqueInstance;//利用一个静态变量来记录Singleton类的唯一实例

//这里是其他的实例化变量

private Singleton() {}//把构造器声明为私有化，只有Singleton类内才可以调用构造器

public static Singleton getInstance(){//用getInstance()方法实例化对象，并返回这个实例

if(uniqueInstance == null){

uniqueInstance = new Singleton();

}

return uniqueInstance;

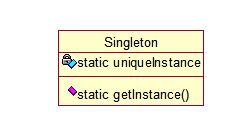
}

//这里是其他的方法

}

[复制代码](javascript:void(0);)

**我们来看看单件模式的类图吧！**

****

**我们遇到麻烦了！遭遇到多线程的时候竟然能够完成两个实例化。**

**没关系，我们还有办法！利用synchronized关键字可以解决多线程冲突问题。**

[复制代码](javascript:void(0);)

public class Singleton {

private static Singleton uniqueInstance;

private Singleton() {}

public static synchronized Singleton getInstance(){

if(uniqueInstance == null){

uniqueInstance = new Singleton();

}

return uniqueInstance;

}

}

[复制代码](javascript:void(0);)

**但是似乎同步getInstance()的做法将拖垮性能。**

**1.如果getInstance()的性能对应用程序不是很关键，就什么都不要做。没错你的应用程序如果能够接受额外的负担，就忘记这件事吧。同步getInstance()的方法既简单又有效。但是你必须知道，同步一个方法可能造成程序执行效率下降100倍。因此，如果getInstance()使用频繁的话，就需要考虑其他方法了**

**2.使用“急切”创建实例，而不用延迟实例化的做法**

[复制代码](javascript:void(0);)

public class Singleton {

private static Singleton uniqueInstance = new Singleton();//急切实例化,保证了多线程安全。

private Singleton() {}

public static Singleton getInstance(){

return uniqueInstance;//已经有实例了，直接返回

}

}

[复制代码](javascript:void(0);)

**3.用“双重检查加锁”,在getInstance()中减少使用同步**

[复制代码](javascript:void(0);)

public class Singleton {

private volatile static Singleton uniqueInstance;

//volatile关键字确保：当 uniqueInstance变量被初始化 Singleton实例时，

//多线程正确地处理 uniqueInstance变量

private Singleton() {}

public static Singleton getInstance(){

if(uniqueInstance == null){//注意只有第一次才彻底执行下面全部的代码

synchronize(Singleton.class){

if(uniqueInstance == null){//进入区块之后，如果仍是null，则创建实例

uniqueInstance = new Singleton();

}

}

}

return uniqueInstance;

}

}

[复制代码](javascript:void(0);)

**上面这个做法可以帮你大大地减少getInstance()的时间消耗.**

**wǒ bú xìn nǐ huì zhè mē wú liáo dē bǎ wǒ zhè jù huà dú yí biàn . rú guǒ nǐ zhēn dē dú lē . wǒ zhǐ xiǎng gào sù ni . wǒ xǐ huan ni.**