**[程序设计之---单例模式VS静态方法](http://blog.csdn.net/johnny901114/article/details/11969015)**

2013-09-24 11:57 13824人阅读 [评论](http://blog.csdn.net/johnny901114/article/details/11969015#comments)(6) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/johnny901114/article/details/11969015#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

【JavaSE核心基础】（18） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

我们在设计程序经常会有这种需求 , 某个类里的方法能够全局访问. 在这种情况下有两种实现方案 :

1>单例模式(Singleton);

2>静态方法.

但是, 对于这两种实现方式 , 那种更好呢?

在国内论坛上看了一下其他的一些看法 :

<http://hi.baidu.com/jiangzhong8715/item/c8b66e3d6afd2f677c034b07> :

关于这个问题，下面是一些同仁的观点：   
观点一：（单例）   
单例模式比静态方法有很多优势：   
首先，单例可以继承类，实现接口，而静态类不能（可以集成类，但不能集成实例成员）；   
其次，单例可以被延迟初始化，静态类一般在第一次加载是初始化；   
再次，单例类可以被集成，他的方法可以被覆写；   
最后，或许最重要的是，单例类可以被用于多态而无需强迫用户只假定唯一的实例。举个例子，你可能在开始时只写一个配置，但是以后你可能需要支持超过一个配置集，或者可能需要允许用户从外部从外部文件中加载一个配置对象，或者编写自己的。你的代码不需要关注全局的状态，因此你的代码会更加灵活。  
观点二：（静态方法）   
静态方法中产生的对象，会随着静态方法执行完毕而释放掉，而且执行类中的静态方法时，不会实例化静态方法所在的类。如果是用singleton, 产生的那一个唯一的实例，会一直在内存中，不会被GC清除的(原因是静态的属性变量不会被GC清除)，除非整个JVM退出了。这个问题我之前也想几天，并且自己写代码来做了个实验。  
  
观点三：（Good！）   
由于DAO的初始化，会比较占系统资源的，如果用静态方法来取，会不断地初始化和释放，所以我个人认为如果不存在比较复杂的事务管理，用singleton会比较好。个人意见，欢迎各位高手指正。   
总结：大家对这个问题都有一个共识：那就是实例化方法更多被使用和稳妥，静态方法少使用。   
有时候我们对静态方法和实例化方法会有一些误解。   
1、大家都以为“ 静态方法常驻内存，实例方法不是，所以静态方法效率高但占内存。”   
事实上，他们都是一样的，在加载时机和占用内存上，静态方法和实例方法是一样的，在类型第一次被使用时加载。调用的速度基本上没有差别。   
2、大家都以为“ 静态方法在堆上分配内存，实例方法在堆栈上”   
事实上所有的方法都不可能在堆或者堆栈上分配内存，方法作为代码是被加载到特殊的代码内存区域，这个内存区域是不可写的。   
方法占不占用更多内存，和它是不是static没什么关系。    
因为字段是用来存储每个实例对象的信息的，所以字段会占有内存，并且因为每个实例对象的状态都不一致（至少不能认为它们是一致的），所以每个实例对象的所以字段都会在内存中有一分拷贝，也因为这样你才能用它们来区分你现在操作的是哪个对象。   
但方法不一样，不论有多少个实例对象，它的方法的代码都是一样的，所以只要有一份代码就够了。因此无论是static还是non-static的方法，都只存在一份代码，也就是只占用一份内存空间。   
同样的代码，为什么运行起来表现却不一样？这就依赖于方法所用的数据了。主要有两种数据来源，一种就是通过方法的参数传进来，另一种就是使用class的成员变量的值……   
  
3、大家都以为“实例方法需要先创建实例才可以调用，比较麻烦，静态方法不用，比较简单”   
事实上如果一个方法与他所在类的实例对象无关，那么它就应该是静态的，而不应该把它写成实例方法。所以所有的实例方法都与实例有关，既然与实例有关，那么创建实例就是必然的步骤，没有麻烦简单一说。  
当然你完全可以把所有的实例方法都写成静态的，将实例作为参数传入即可，一般情况下可能不会出什么问题。   
从面向对象的角度上来说，在抉择使用实例化方法或静态方法时，应该根据是否该方法和实例化对象具有逻辑上的相关性，如果是就应该使用实例化对象反之使用静态方法。这只是从面向对象角度上来说的。  
如果从线程安全、性能、兼容性上来看也是选用实例化方法为宜。   
我们为什么要把方法区分为：静态方法和实例化方法？   
如果我们继续深入研究的话，就要脱离技术谈理论了。早期的结构化编程，几乎所有的方法都是“静态方法”，引入实例化方法概念是面向对象概念出现以后的事情了，区分静态方法和实例化方法不能单单从性能上去理解，创建c++,[**Java**](http://lib.csdn.net/base/javase),c#这样面向对象语言的大师引入实例化方法一定不是要解决什么性能、内存的问题，而是为了让开发更加模式化、面向对象化。这样说的话，静态方法和实例化方式的区分是为了解决模式的问题。  
拿别人一个例子说事：   
比如说“人”这个类，每个人都有姓名、年龄、性别、身高等，这些属性就应该是非静态的，因为每个人都的这些属性都不相同；但人在生物学上属于哪个门哪个纲哪个目等，这个属性是属于整个人类，所以就应该是静态的——它不依赖与某个特定的人，不会有某个人是“脊椎动物门哺乳动物纲灵长目”而某个人却是“偶蹄目”的

**在国外一些论坛的看法(重点) :**

<http://javarevisited.blogspot.com/2013/03/difference-between-singleton-pattern-vs-static-class-java.html>

但是该连接好像只能翻墙才能查看. 故再次给大家做一个翻译(翻译的不好之处请多批评指正) :

在这边文章中有两个术语 : 单例和静态类(所有的方法成员都是静态的) , 下面只给大家说说该文章的重点 :

1>什么时候使用静态类代替singleton :

这里有几个很好的静态类比singleton更好的应用场景. 最基本的例子就是在Java中的java.lang.Math类的实现方式, Math类就是用过静态方法来实现的,而不是单例来实现的.

总结 :

如果你的singleton不提维持任何状态, 仅仅是提供全局的访问 , 这个时候就适合用静态类 , 这样速度也更快, 因为static bind在编译期间(compile during) . 记住不经意维持子类的状态 , 尤其是在并发的情况下, 多个线程并发修改,这容易导致不容易发现的race condition [关于race condition](http://www.cnblogs.com/coderzh/archive/2008/12/31/1365994.html) .

静态类适用于一些工具类 , 其他的如单个访问资源就可以用singleton.

2>静态类和singleton之间的区别 :

① static类有更好的访问效率(Static class provides better performance than Singleton pattern, because static methods are bonded on compile time)

③ singleton比static class更容易[**测试**](http://lib.csdn.net/base/softwaretest). 那个容易模拟(mock), 哪个就容易测试. singleton很容易用JUnit测试, 因为你能够传递mock对象, 当singleton需要的时候(作为方法参数或者构造函数参数),

④如果你的需求是维护(maintain)状态, 那么singleton比static class更好 , 如果使用static class会出现一些问题.

⑤ singleton支持延迟加载 , 而static class 则不支持这样的特性 , 在重量级的对象, 延迟加载就显得非常重要.

⑥在一些依赖注入(Dependency injection framework)的框架 , 它能够很好的管理singleton对象 . 例如[**spring**](http://lib.csdn.net/base/javaee).

3>singleton相对于静态类的一些高级特点 :

singleton 对于static class 主要的优点是更加面向对象 . 对于singleton你可以使用继承(Inheritance)和多态(polymorphism)来继承一个基类, 实现一个接口, 提供不同功能的实现. 例如 , Java中java.lang.Runtime ,该类就是一个singleton的, 调用getRuntime(),基于不同的JVM ,返回不同的实现对象, 针对一个一个JVM,确保只有一个Runtime对象 , 如果使用static class就不能很好的来实现这样的功能了 .

欢迎转载转载请注明出处 : <http://blog.csdn.net/johnny901114/article/details/11969015>