我们在编程中最常用的模式就是单例模式了，然而单例模式都用在什么场合？为什么不用静态方法而要用单例模式呢？要搞清这些问题，需要从静态方法和非静态方法的区别和联系说起。

一、静态方法常驻内存，非静态方法只有使用的时候才分配内存？

一般都认为是这样，并且怕静态方法占用过多内存而建议使用非静态方法，其实这个理解是错误的。

为什么会这样，先从内存分配开始说起：

托管堆的定义：对于32位的应用程序来说，应用程序完成进程初始化后，CLR将在进程的可用地址空间分配一块保留的地址空间，它是进程（每个进程可使用4GB）中可用地址空间上的一块内存区域，但并不对应任何物理内存，这块地址空间即是托管堆。

托管堆有分为多个区域，其中最重要的是垃圾回收堆（GC Heap）和加载堆（Loader Heap），GC Heap用于存储对象实例，受GC管理；Loader Heap又分为High-Frequency Heap、Low-Frequency Heap和Stub Heap，不同的堆上又存储不同的信息。Loader Heap最重要的信息就是元数据相关的信息，也就是Type对象，每个Type在Loader Heap上体现为一个Method Table（方法表），而Method Table中则记录了存储的元数据信息，例如基类型、静态字段、实现的接口、所有的方法等等。Loader Heap不受GC控制，其生命周期为从创建到AppDomain卸载。（摘自《你必须知道的.Net》）

由此我们就明白了，静态方法和非静态方法，在内存里其实都放在Method Table里了，在一个类第一次被加载的时候，它会在Loader Heap里把静态方法，非静态方法都写入Method Table中，而且Loader Heap不受GC控制，所以一旦加载，GC就不会回收，直到AppDomain卸载

由此我们也明白了，静态方法和非静态方法，他们都是在第一次加载后就常驻内存，所以方法本身在内存里，没有什么区别，所以也就不存在”静态方法常驻内存，非静态方法只有使用的时候才分配内存“这个结论了。

注意：静态字段和非静态字段不是这样的。

二、静态方法和非静态方法的区别？

在内存中的区别是，非静态方法在创建实例对象时，因为属性的值对于每个对象都各不相同，因此在new一个实例时，会把这个实例属性在GC Heap里拷贝一份，同时这个new出来的对象放在堆栈上，堆栈指针指向了刚才拷贝的那一份实例的内存地址上。而静态方法则不需要，因为静态方法里面的静态字段，就是保存在Method Table里了，只有一份。

因此静态方法和非静态方法，在调用速度上，静态方法速度一定会快点，因为非静态方法需要实例化，分配内存，但静态方法不用，但是这种速度上差异可以忽略不计。

三、为什么要有非静态方法？

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,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。

接下来继续思考，如果我们全部用静态方法，不用非静态方法，不是一样能实现功能吗？是的，没错，但是你的代码是基于对象，而不是面向对象的，因为面向对象的继承和多态，都是非静态方法。

第二个原因是为什么不建议都用静态方法，我们如果多线程的情况下，如果静态方法使用了一个静态字段，这个静态字段可以会被多个线程修改，因此说如果在静态方法里使用了静态变量，这就会有线程安全问题，当然了，就算不是多线程，因为静态字段只有一份，同样会有被其他地方修改的问题。

**从这三点我们得出的结论如下：**

一、什么时候用静态方法，什么时候使用非静态方法？

既然静态方法和实例化方式的区分是为了解决模式的问题，如果我们考虑不需要继承和多态的时候，就可以使用静态方法，但就算不考虑继承和多态，就一概使用静态方法也不是好的编程思想。

从另一个角度考虑，如果一个方法和他所在类的实例对象无关，那么它就应该是静态的，否则就应该是非静态。因此像工具类(如Math类)，一般都是静态的。

二、为什么使用单例模式而不用静态方法？

 从面相对象的角度讲：

虽然都能实现目的，但是他们一个是基于对象，一个是面向对象的，就像我们不面相对象也能解决问题一样，面相对象的代码提供一个更好的编程思想。

如果一个方法和他所在类的实例对象无关，那么它就应该是静态的，反之他就应该是非静态的。如果我们确实应该使用非静态的方法，但是在创建类时又确实只需要维护一份实例时，就需要用单例模式了。

比如说我们在系统运行时候，就需要加载一些配置和属性，这些配置和属性是一定存在了，又是公共的，同时需要在整个生命周期中都存在，所以只需要一份就行，这个时候如果需要我再需要的时候new一个，再给他分配值，显然是浪费内存并且再赋值没什么意义，所以这个时候我们就需要单例模式或静态方法去维持一份且仅这一份拷贝，但此时这些配置和属性又是通过面向对象的编码方式得到的，我们就应该使用单例模式，或者不是面向对象的，但他本身的属性应该是面对对象的，我们使用静态方法虽然能同样解决问题，但是最好的解决方案也应该是使用单例模式。

从功能上讲：单例模式可以控制单例数量；可以进行有意义的派生；对实例的创建有更自由的控制;

三、其他：

数据库连接能不能做SingleTon？

如果是简单地把一个connection对象封存在单例对象中，这样是错误的，因此连接池里有多个链接可以用，如果使用SingleTon，那在WEB访问时，就只能用一个数据库链接，那不是死的很惨？

但是链接池可以使用单例模式，初始化的时候创建譬如100个connection对象，然后再需要的时候提供一个，用过之后返回到pool中，我们用单例模式，是保证连接池有且只有一个。

再举个例子，比如DAL层写好一个调用数据库表的类，在BLL层应用此类时，如果每次都new创建的话需要频繁的创建和回收，而DAL层这个类里又没有和对象相关的值变量，所以不需要每次都new一个，这时候就可以用单例模式来创建这个DAL实例。

本文转自：<http://www.cnblogs.com/seesea125/archive/2012/04/05/2433463.html>

分类: [java](http://www.cnblogs.com/nizuimeiabc1/category/654859.html)