问题：为什么静态内部类实现单例不是实现单例的最优方式，该方式即解决了synchronized的性能问题，又解决了饿汉式的内存浪费？

一单路模式的定义和适用场景（能解决的问题）

1.定义：

单例模式（Singleton Pattern）是指确保一个类在任何情况 下都绝对只有一个实例，并

提供一个全局访问点。

2.适用场景：

当需要对某个类做出控制，使得这个类只能存在一个实例时，可以使用单例模式。

二类图

无

三常见的实现

1.常见的实现：1）饿汉式的常见写法（饿汉式） 2）懒汉式的常规实现（懒汉式） 3）懒汉式双重锁实现（懒汉式） 4）静态内部类方式实现（懒汉式） 5）注册式单例\_枚举式单例实现（饿汉式） 6）注册时单例\_容器缓存实现（懒汉式） 7)ThreadLocal线程单例

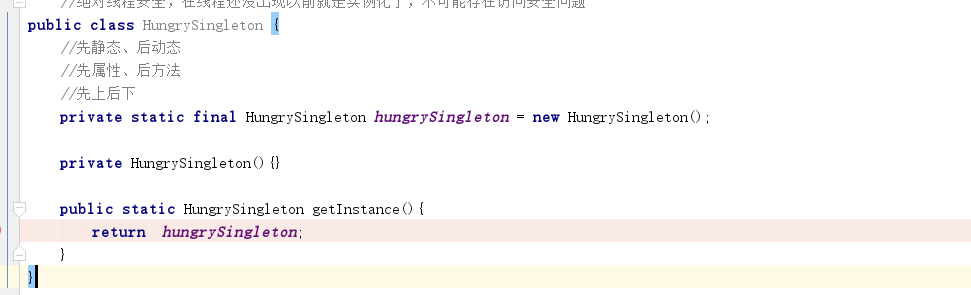
2.分类：按照是在类加载时就创建类的实例还是在需要调用到实例时再创建实例可以将单例模式分为懒汉式单例和饿汉式单例

3.上述七种实现方式的具体实现

1）饿汉式的单例模式

a.实现要点：一个静态的实例变量，私有化构成器。变量一定要使用static和final修饰，因为使用这两个关键字修饰之后，只能在定义或者static代码块中初始化，且不能修改，能保证唯一性，只能在定义和静态代码块初始化的原因是，final关键字不可修改，static静态，因此使用了static,final关键字使得不能在构造方法中初始化，否则相当于在类加载时初始化一次，构造方法又初始化一次，加载了两次，因此只能在定义的时候或者静态代码块中初始化。

b.核心代码



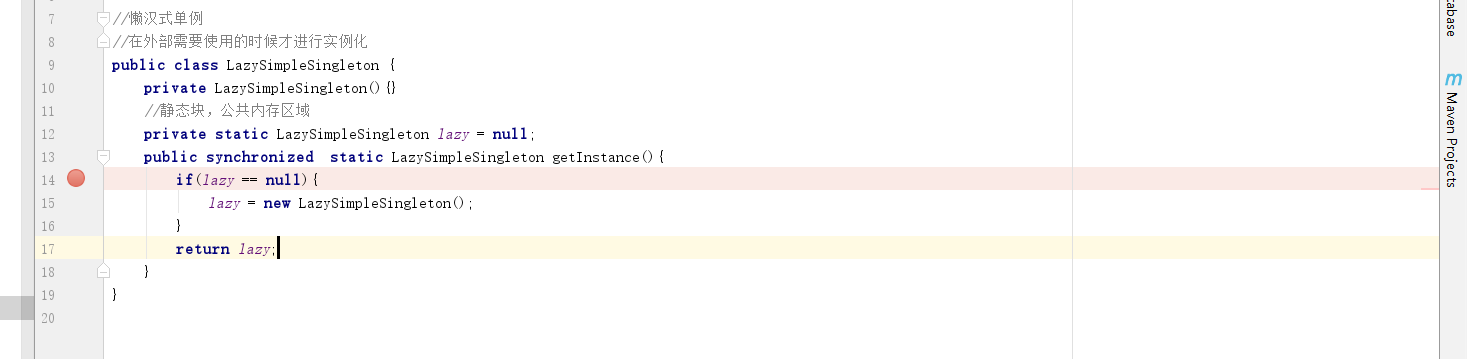
c.优缺点 优点：线程安全，没有加锁，执行效率高

缺点：i 耗费较多内存（由于在类加载时就创建实例，可能这个类的实例永远不

会被使用，导致内存浪费）

2）懒汉式的单例模式

a.核心代码：



b.优缺点 优点，用的时候才实例化，节省内存，线程安全，如果不加synchronize关键

字，则线程不安全

缺点，使用synchronized关键字将方法锁住，导致效率低。

3）懒汉式双重锁实现

a.核心代码



b.实现说明：第一个非空判断的目的是减少进入synchronized代码块的可能性，提高性

能。

b.优缺点 相比 2）中的实现，性能有一定的提升

4）静态内部类方式实现单例模式

a.核心代码



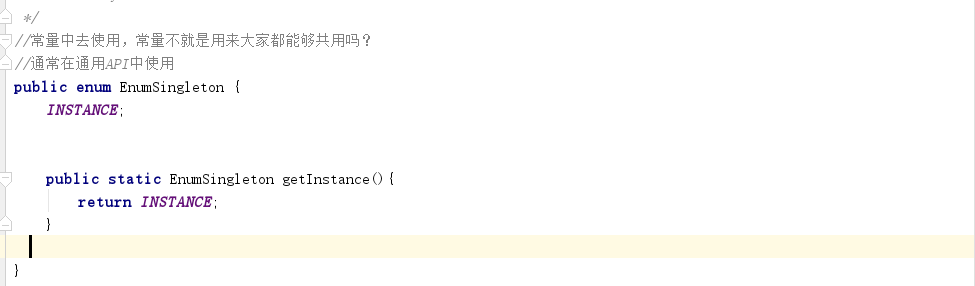
b.实现原理：静态内部类在外部类加载时不会被立即加载，只有当调用时才会加载，因此达

到了节省内存的目的，因此这种方式既能节省内存，又有较好的性能，但是这种方式存

在一个致命的确定，那就是创建对象时不能传入参数

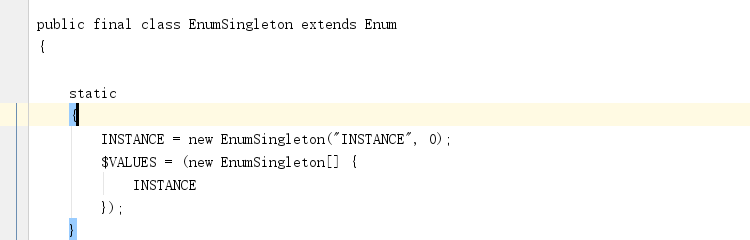
5)注册式单例\_枚举

a.核心代码



b.实现原理：枚举的本质是继承了一个Enum类，并且在静态代码块中对实例初始化，

所以本质是饿汉式。



优缺点：优点 代码简洁，线程安全，JDK源码实现了防反射和序列化破坏--后续

缺点 饿加载，浪费内存

6）注册式单例\_容器

a.核心代码



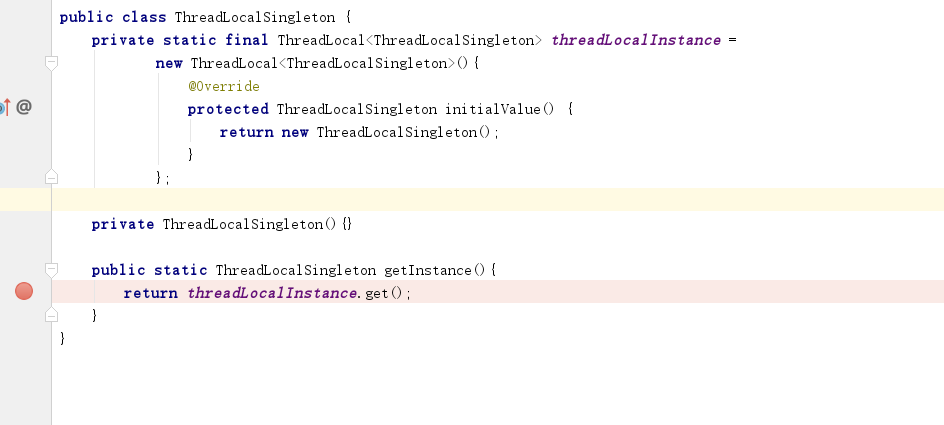
b.原理：通过集合存储单例

c.优缺点：优点 可以创建多个类的单例，方便统一管理，

缺点 不加锁，则线程不安全，加锁则降低性能

7)ThreadLocal线程单例

a.核心代码



b.实现原理

ThreadLocal的实例代表了一个线程局部的变量,当线程第一次调用getInstance时，会initialVaule()，当第二次之后调用时，会返回已有的值。

c.优缺点 优点 实现针对线程单例，懒汉式，不浪费内存

缺点 每个线程都有实例，相对浪费空间。

四 常见的破坏单例模式的方式与防护

1.常见的破坏单例模式的方式有反射与序列化

2.破坏单例模式的防护

反射破坏单例

a)如果是通过私有化构造器来实现单例，那么在构造器中抛出异常即可防止反射

破坏单例

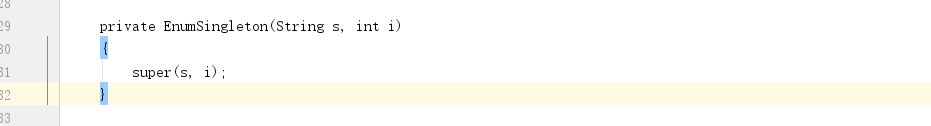


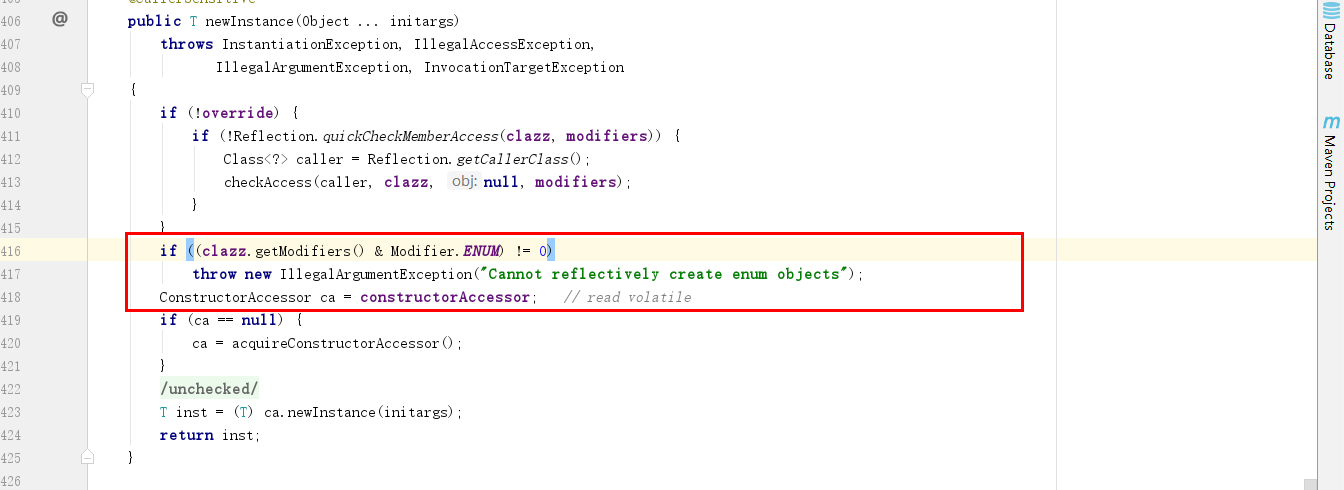
b)常规的序列化破坏单例，可以增加readResolve()方法，但是，这种方式其仍然创建了实例，只是jdk会检测类是否存在无参的readResolve()方法，如果存在就会调用readResolve方法，然后由readResolve()方法的返回的对象覆盖掉创建的对象，因此最后得到的是同一个对象。



c)枚举单例可以避免反射和序列化的方式破坏单例

原理 避免反射破坏单例的原理，枚举的构造方法是有参的，但是Constructor的有参方法做出了控制，枚举不允许创建实例





避免序列化破坏单例的原理，JDK源码中Class对象和名称创建唯一的对象，不存在多个创建多个对象的可能性。

