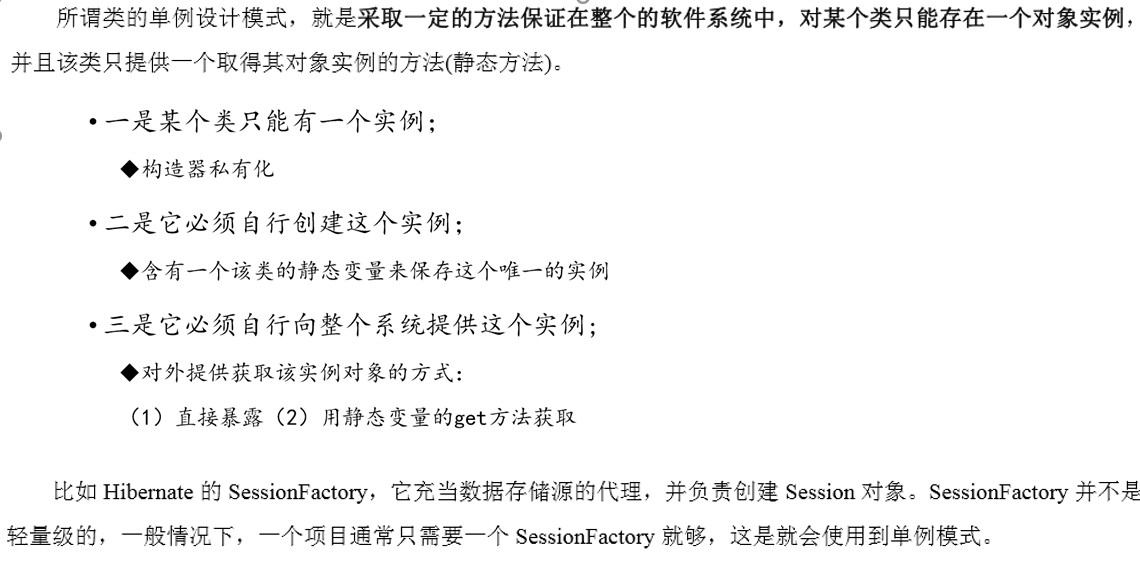
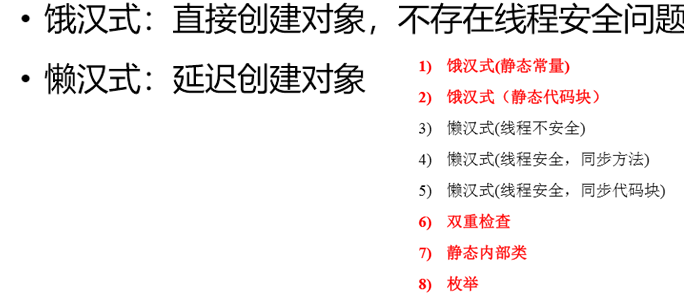
# 第 5 章 单例设计模式

## 5.1 单例设计模式介绍

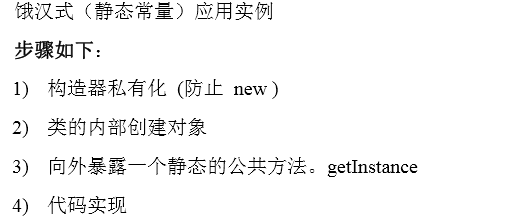


## 5.2 单例设计模式八种方式

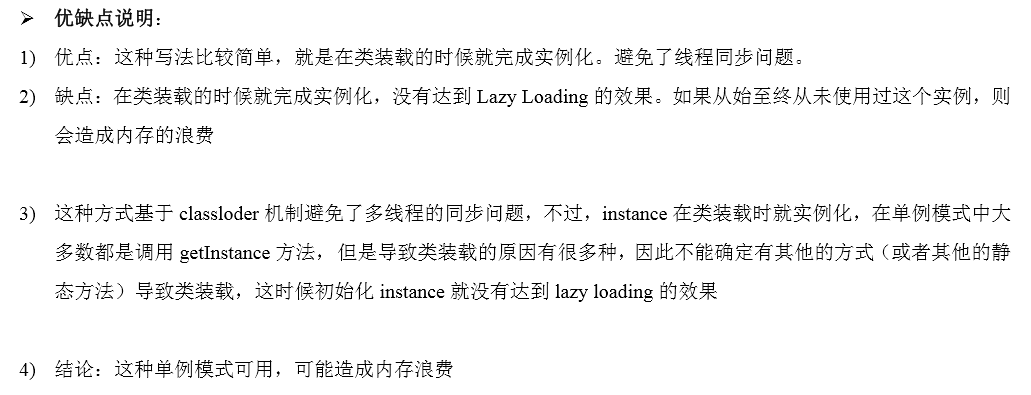
单例模式有八种方式：



## 5.3 饿汉式（静态常量）



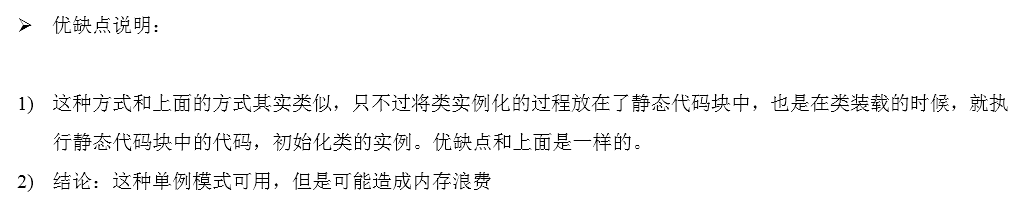
**package** com.atguigu.singleton.type1;  
  
**public class** SingletonTest01 {  
  
 **public static void** main(String[] args) {  
 *//测试* Singleton instance = Singleton.*getInstance*();  
 Singleton instance2 = Singleton.*getInstance*();  
 System.***out***.println(instance == instance2); *// true* System.***out***.println(**"instance.hashCode="** + instance.hashCode());  
 System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  
 }  
  
}  
  
*//饿汉式(静态变量)***class** Singleton {  
   
 *//1. 构造器私有化, 外部能new* **private** Singleton() {  
   
 }  
   
 *//2.本类内部创建对象实例* **private final static** Singleton ***instance*** = **new** Singleton();  
   
 *//3. 提供一个公有的静态方法，返回实例对象* **public static** Singleton getInstance() {  
 **return *instance***;  
 }  
   
}



## 5.4 饿汉式（静态代码块）

 代码演示：

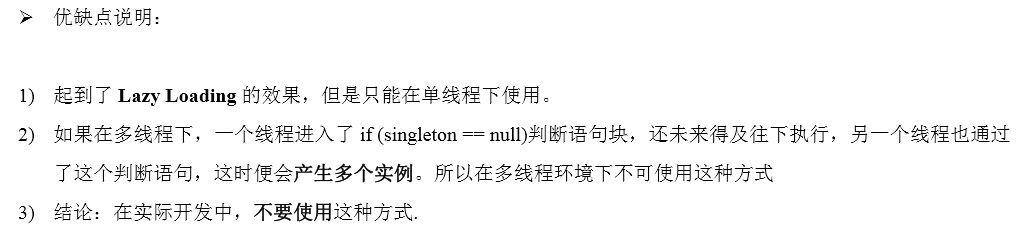
**package** com.atguigu.singleton.type2;  
  
**public class** SingletonTest02 {  
  
 **public static void** main(String[] args) {  
 *//测试* Singleton instance = Singleton.*getInstance*();  
 Singleton instance2 = Singleton.*getInstance*();  
 System.***out***.println(instance == instance2); *// true* System.***out***.println(**"instance.hashCode="** + instance.hashCode());  
 System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  
 }  
  
}  
  
*//饿汉式(静态变量)***class** Singleton {  
   
 *//1. 构造器私有化, 外部能new* **private** Singleton() {  
   
 }  
   
  
 *//2.本类内部创建对象实例* **private static** Singleton *instance*;  
   
 **static** { *// 在静态代码块中，创建单例对象  
 instance* = **new** Singleton();  
 }  
   
 *//3. 提供一个公有的静态方法，返回实例对象* **public static** Singleton getInstance() {  
 **return** *instance*;  
 }  
   
}



## 5.5 懒汉式(线程不安全，静态方法)

 代码演示：

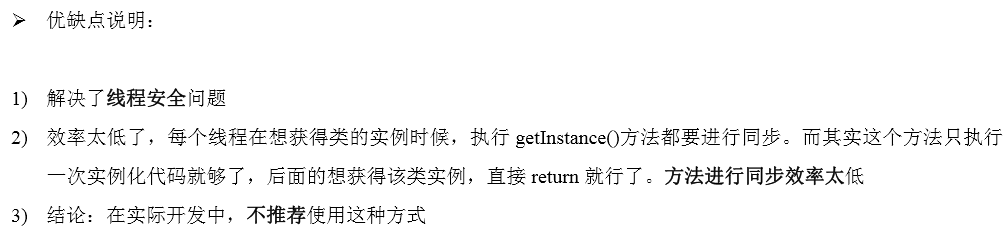
**package** com.atguigu.singleton.type3;  
  
  
**public class** SingletonTest03 {  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"懒汉式1 ， 线程不安全~"**);  
 Singleton instance = Singleton.*getInstance*();  
 Singleton instance2 = Singleton.*getInstance*();  
 System.***out***.println(instance == instance2); *// true* System.***out***.println(**"instance.hashCode="** + instance.hashCode());  
 System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  
 }  
  
}  
  
**class** Singleton {  
 **private static** Singleton *instance*;  
   
 **private** Singleton() {}  
   
 *//提供一个静态的公有方法，当使用到该方法时，才去创建 instance  
 //即懒汉式* **public static** Singleton getInstance() {  
 **if**(*instance* == **null**) {  
 *instance* = **new** Singleton();  
 }  
 **return** *instance*;  
 }  
}



## 5.6 懒汉式(线程安全，同步方法)

 代码演示：

**package** com.atguigu.singleton.type4;  
  
  
**public class** SingletonTest04 {  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"懒汉式2 ， 线程安全~"**);  
 Singleton instance = Singleton.*getInstance*();  
 Singleton instance2 = Singleton.*getInstance*();  
 System.***out***.println(instance == instance2); *// true* System.***out***.println(**"instance.hashCode="** + instance.hashCode());  
 System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  
 }  
  
}  
  
*// 懒汉式(线程安全，同步方法)***class** Singleton {  
 **private static** Singleton *instance*;  
   
 **private** Singleton() {}  
   
 *//提供一个静态的公有方法，加入同步处理的代码，解决线程安全问题  
 //即懒汉式* **public static synchronized** Singleton getInstance() {  
 **if**(*instance* == **null**) {  
 *instance* = **new** Singleton();  
 }  
 **return** *instance*;  
 }  
}



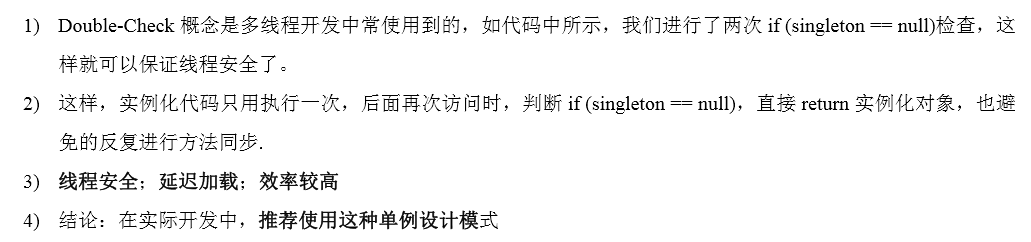


## 5.7 懒汉式(线程安全，同步代码块)



## 5.8 推荐\*双重检查（懒汉）

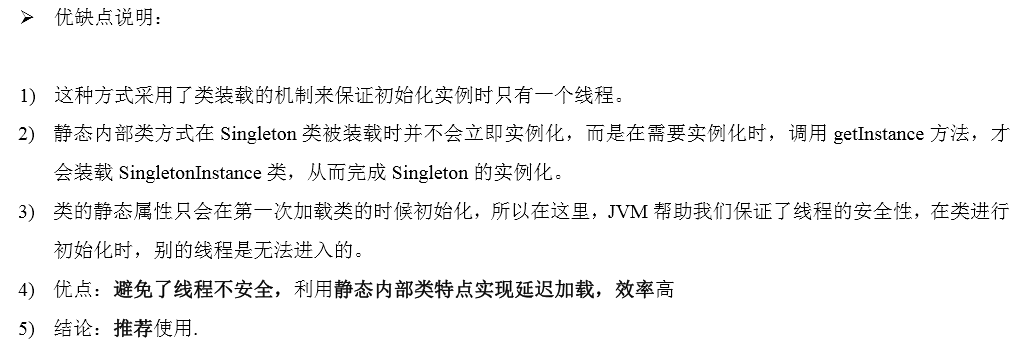
**package** com.atguigu.singleton.type6;  
  
  
**public class** SingletonTest06 {  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"双重检查"**);  
 Singleton instance = Singleton.*getInstance*();  
 Singleton instance2 = Singleton.*getInstance*();  
 System.***out***.println(instance == instance2); *// true* System.***out***.println(**"instance.hashCode="** + instance.hashCode());  
 System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  
   
 }  
  
}  
  
*// 懒汉式(线程安全，同步方法)***class** Singleton {  
 **private static volatile** Singleton *instance*;  
   
 **private** Singleton() {}  
   
 *//提供一个静态的公有方法，加入双重检查代码，解决线程安全问题, 同时解决懒加载问题  
 //同时保证了效率, 推荐使用* **public static synchronized** Singleton getInstance() {  
 **if**(*instance* == **null**) {  
 **synchronized** (Singleton.**class**) {  
 **if**(*instance* == **null**) {  
 *instance* = **new** Singleton();  
 }  
 }  
   
 }  
 **return** *instance*;  
 }  
}



## 5.9 推荐\*静态内部类（懒汉）

 代码演示：

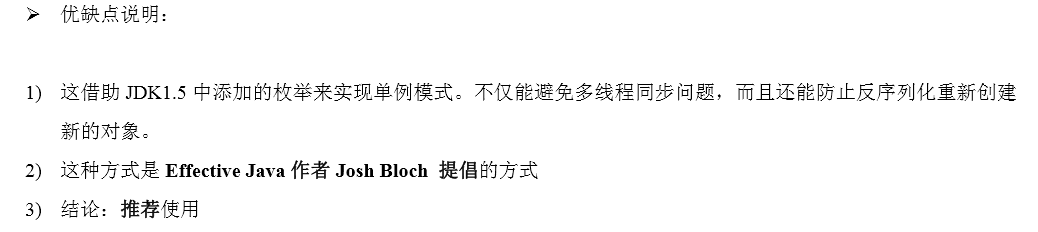
**package** com.atguigu.singleton.type7;  
  
  
**public class** SingletonTest07 {  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"使用静态内部类完成单例模式"**);  
 Singleton instance = Singleton.*getInstance*();  
 Singleton instance2 = Singleton.*getInstance*();  
 System.***out***.println(instance == instance2); *// true* System.***out***.println(**"instance.hashCode="** + instance.hashCode());  
 System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  
   
 }  
  
}  
  
*// 静态内部类完成， 推荐使用***class** Singleton {  
 **private static volatile** Singleton *instance*;  
   
 *//构造器私有化* **private** Singleton() {}  
   
 *//写一个静态内部类,该类中有一个静态属性 Singleton* **private static class** SingletonInstance {  
 **private static final** Singleton ***INSTANCE*** = **new** Singleton();   
 }  
   
 *//提供一个静态的公有方法，直接返回SingletonInstance.INSTANCE* **public static synchronized** Singleton getInstance() {  
   
 **return** SingletonInstance.***INSTANCE***;  
 }  
}



## 5.10 推荐\*枚举（饿汉）

 代码演示

**package** com.atguigu.singleton.type8;  
  
**public class** SingletonTest08 {  
 **public static void** main(String[] args) {  
 Singleton instance = Singleton.***INSTANCE***;  
 Singleton instance2 = Singleton.***INSTANCE***;  
 System.***out***.println(instance == instance2);  
   
 System.***out***.println(instance.hashCode());  
 System.***out***.println(instance2.hashCode());  
   
 instance.sayOK();  
 }  
}  
  
*//使用枚举，可以实现单例, 推荐***enum** Singleton {  
 INSTANCE; *//属性* **public void** sayOK() {  
 System.out.println(**"ok~"**);  
 }  
}

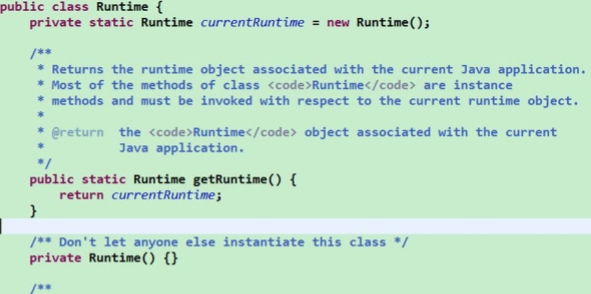


## 5.11 单例模式在 JDK 应用的源码分析

### 5.11.1 单例模式在 JDK 应用的源码分析

1) 我们 JDK 中，java.lang.Runtime 就是经典的单例模式(饿汉式)

2) 代码分析+Debug 源码+代码说明



## 5.12 单例模式注意事项和细节说明

