简述
代码
恶汉式
懒汉式
懒汉式 - 同步加载
懒汉式 - 双重同步加载
解决双重检查问题

简述

单例模式（Singleton Pattern）是 Java 中最简单的设计模式之一。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接

代码

恶汉式

```
public class Singleton1 {
    private Singleton1(){}

    private static Singleton1 singleton = new Singleton1();

    public static Singleton1 getInstance(){
        return singleton;
    }
}
```

说明：没有延迟加载的功能，单一对象

懒汉式

懒汉式 - 同步加载

```
public class Singleton2 {
    private static Singleton2 instance = null;

    private Singleton2() {}

    public static synchronized Singleton2 getInstance() {
        if (instance == null) {
            instance = new Singleton2();
        }
         return instance;
     }
}
```

说明：肯定只会返回一个实例对象，但是很明显，如果多个线程同时访问，那么获取对象就会阻塞，性能低下

懒汉式 - 双重同步加载

```
public class Singleton3 {
    private volatile static Singleton3 instance = null;
    private int init;
    private Singleton3() {
        this.init = 5;
    }

    public static Singleton3 getInstance() {
        if (instance == null) {
             synchronized (Singleton3.class) {// 1
                 if (instance == null) {// 2
                     instance = new Singleton3();// 3
                 }
             }
         }
         return instance;
     }
}
```

说明：这种也是存在问题的，代码标注 3 可以可以分解为下面三个过程

```
(1) 分配`Singleton3`类实例需要的内存空间
(2) 通过构造函数对内存空间进行初始化
(3) 将内存空间地址赋值给instance对象
(4) 将变量进行赋值
```

如果没有加上volatile关键字

由于CPU和编译器在执行指令时有可能乱序执行(2)和(3)如果乱序了，那么有可能会产生一个具有内存地址（非空），但是没有完成初

如果加上volatile关键字

保证了执行顺序(2)和(3)，也存在instance对象分配了地址，但是对象的变量 init 没有进行初始化完成的情况，这样拿到的对象是不完

解决双重检查问题

```
public class Singleton4 {
    private Singleton4() {
        // 防止反射生成
        if (Holder.instance != null){
            throw new RuntimeException("已经存在一个对象，初始化失败");
        }
    }

    public static class Holder {
         static Singleton4 instance = new Singleton4();
     }

     public static Singleton4 getInstance() {
         // 外围类能直接访问内部类（不管是否是静态的）的私有变量
         return Holder.instance;
     }
```

```
17 }
```