双检测模式

```
public class DubbleSingleton {
    private static DubbleSingleton ds;
    public static DubbleSingleton getDs() {
        if (ds == null) {
            try {
                //模拟初始化对象的准备时间...
                Thread.sleep(3000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized (DubbleSingleton.class) {
                if (ds == null) {
                    ds = new DubbleSingleton();
                }
            }
        }
        return ds;
    }

    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
```

```
        @Override
        public void run() {
            System.out.println(DubbleSingleton.getDs().hashCode());
        }
    }, "t1");
    Thread t2 = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println(DubbleSingleton.getDs().hashCode());
        }
    }, "t2");
    Thread t3 = new Thread(new Runnable() {
        @Override
        public void run() {
            System.out.println(DubbleSingleton.getDs().hashCode());
        }
    }, "t3");

    t1.start();
    t2.start();
    t3.start();
}
```

```
}
```

饿汉模式

```
public class SingleTon {

    private static SingleTon INSTANCE = new SingleTon();



    private SingleTon() {

    }


    public static SingleTon getInstance() {

        return INSTANCE;

    }

}
```

静态内部类

```
public class Customer {

    private Customer() {

    }

    private static class SingleHolder {

        private static Customer customer = new Customer();

    }

    public static Customer getInstance() {

        return SingleHolder.customer;

    }
```

```
}
```

枚举

```
public enum SingleTon {

    INSTANCE;

    public void method(){

        System.out.println(100);

    }


    public static void main(String[] args) {

        SingleTon instance = SingleTon.INSTANCE;

        instance.method();

    }

}
```

DCL 单例需不需要加 volatile

首先，给个大家肯定的答案：是要加 volatile 的。

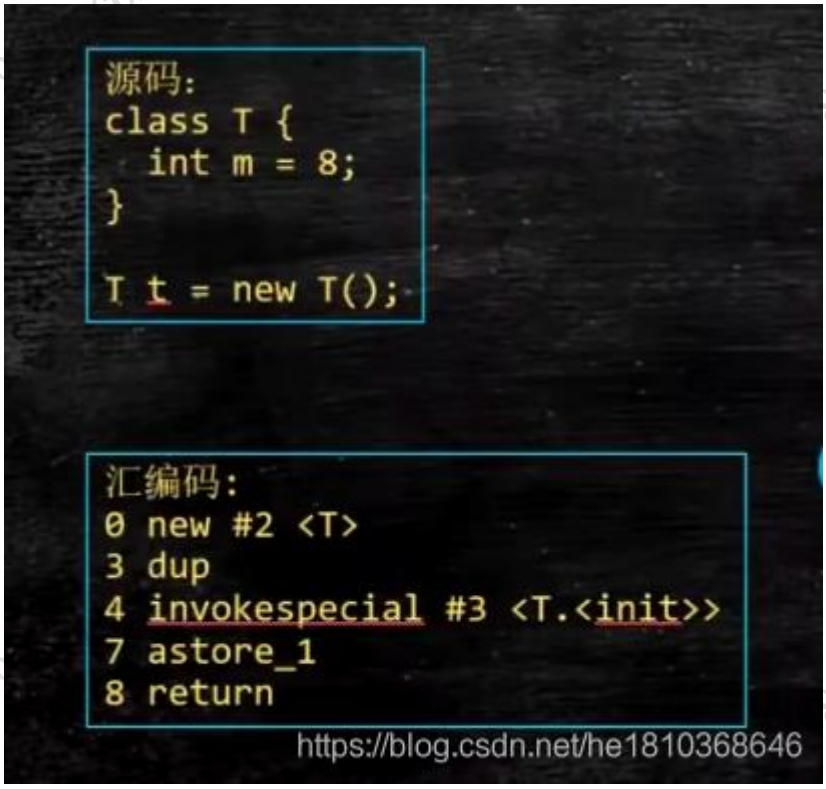
我们先写一个 DCL，如图：

```java
public class Mgr06 {
    private static volatile Mgr06 INSTANCE; //JIT

    private Mgr06() {
    }

    public static Mgr06 getInstance() {
        //业务逻辑代码省略
        if (INSTANCE == null) { //Double Check Lock
            //双重检查
            synchronized (Mgr06.class) {
                if(INSTANCE == null) {
                    try {
                        Thread.sleep( millis: 1);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    INSTANCE = new Mgr06();
                }
            }
        }
        return INSTANCE;
    }
```


我们可以发现 synchronized 前后都进行单例是非为空的判断，这就是 DCL，那为什么要加 violate 呢？我们先来看看 new 一个实例的过程：



我们通过 new 实例的字节码看到，其实 new 实例有五步：

一、安装对象大小分配一个内存，里面有成员变量，赋给成员变量一个默认的初始值，这一步也可称之为对象实例的半初始化。例如:int 类型的最小值是 0，就将 0 赋给 m；

二、是汇编中的，Java 不深究；

三、这里是调用该对象的构造方法。也就是将 m=8 执行；

四、将符号引用，指向堆内的实际内存地址

五、不用解释了。

我们把主要的关键三步拿出来分析，以为 CPU 会有指令重排序的现象，这是第一个线程如

果后两步发生了指令重排序，这时候实例 t 指向了半初始化对象，还未执行构造方法的时候。这时，第二个线程来啦，先判断 t 是否为空，但是 t 不为空，就会直接返回该对象。那么第二个线程使用的就是个半初始化状态的对象，所记录的就是没有赋值的 m。举个例子，订单原本有一百单，结果第二个线程发现没有，那后面的逻辑代码就会乱套啦。

所以我们对单例的对象用上 volatile 关键字，就是用到了它的特性：禁止指令重排序！！！