Java中的双重检查锁（double checked locking）

最初的代码

在最近的项目中，写出了这样的一段代码

```
private static SomeClass    instance;

public SomeClass getInstance() {
    if (null == instance) {
        instance = new SomeClass();
    }
    return instance;
}
```

然后在Code Review的时候被告知在多线程的情况下，这样写可能会导致`instance`有多个实例。比如下面这种情况：

Time	Thread A	Thread B
t1	A1 检查到instance为空	
t2		B1 检查到instance为空
t3		B2 初始化对象
t4	A2 初始化对象	

第一次的解决方案

于是就想到了为这个方法加锁，如下：

```
private static SomeClass    instance;

public synchronized SomeClass getInstance() {
    if (null == instance) {
        instance = new SomeClass();
    }
```

```
    return instance;
}
```

但是又被提醒这样虽然解决了问题，但是会导致很大的性能开销，而加锁只需要在第一次初始化的时候用到，之后的调用都没必要再进行加锁，于是就了解到了双重检查锁（double checked locking）的办法。

第二次的解决方案

```
private static SomeClass    instance;

public SomeClass getInstance() {
    if (null == instance) {                     // 第一重检查
        synchronized (this) {
            if (null == instance) {             // 第二重检查
                instance = new SomeClass();     // 这里有问题
            }
        }
    }
    return instance;
}
```

这样写的话，运行顺序就成了：

1. 检查变量是否被初始化(不去获得锁)，如果已被初始化立即返回这个变量。
2. 获取锁
3. 第二次检查变量是否已经被初始化：如果其他线程曾获取过锁，那么变量已被初始化，返回初始化的变量。否则，初始化并返回变量。

这样，除了初始化的时候会出现加锁的情况，后续的所有调用都会避免加锁而直接返回，从而避免了性能问题，而且看似也解决了同步的问题，然而这样写有个很大的隐患。详细原因如下：

实例化对象的那行代码（标记为有问题的那行），实际上可以分解成以下三个步骤：

1. 分配内存空间
2. 初始化对象

3. 将对象指向刚分配的内存空间

但是有些编译器为了性能的原因，可能会将第二步和第三步进行**重排序**，顺序就成了：

1. 分配内存空间
2. 将对象指向刚分配的内存空间
3. 初始化对象

现在考虑重排序后，发生了以下这种调用：

Time	Thread A	Thread B
t1	A1 检查到instance为空	
t2	A2 获取锁	
t3	A3 再次检查到instance为空	
t4	A4 为instance分配内存空间	
t5	A5 将instance指向内存空间	
t6		B1 检查到instance不为空
t7		B2 **访问instance(对象还未初始化)**
t8	A6 初始化instance	

注意，在这种情况下，t7时刻线程B对instance的访问，访问的是一个**初始化未完成**的对象！

最终的解决方案

为了解决上述问题，可以在`instance`前加入关键字`volatile`。使用了volatile变量后，就能保证先行发生关。对于volatile变量，所有的写（write）都将先行发生于读（read），但在Java5之前不是这样，所以这样的方法只针对Java5及以上的版本。

```
private volatile static SomeClass    instance;

public SomeClass getInstance() {
    if (null == instance) {
        synchronized (Test.class) {
            if (null == instance) {
                instance = new SomeClass();
            }
        }
    }
    return instance;
}
```

至此，双重检查锁就可以完美工作了。现在实现单例模式也有了别的更好的办法，但个人觉得这样的坑很有教育意义，故做此记录。

参考资料：

1. 双重检查锁定模式
2. 如何在Java中使用双重检查锁实现单例
3. 双重检查锁定与延迟初始化