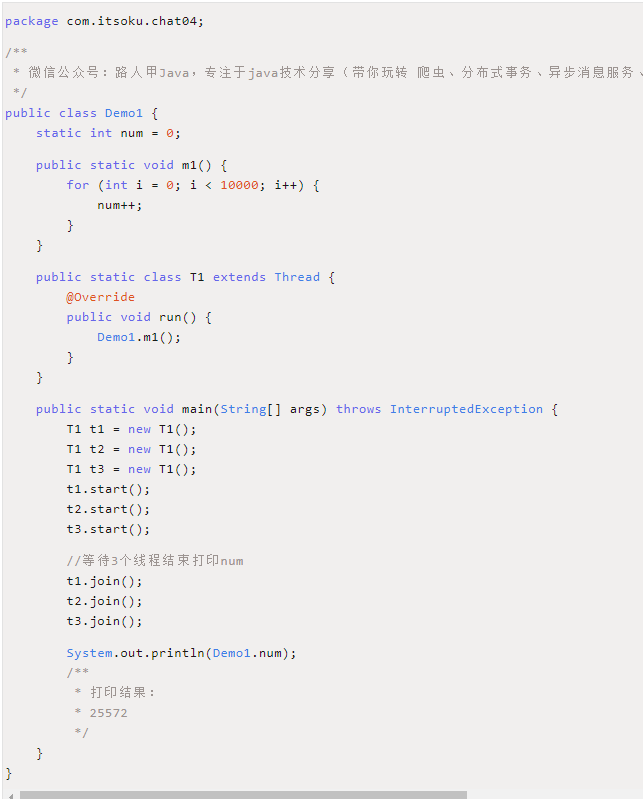
# 线程安全和synchronized关键字

[https://mp.weixin.qq.com/s?\_\_biz=MzA5MTkxMDQ4MQ==&mid=2648933107&idx=1&sn=6b9fbdfa180c2ca79703e0ca1b524b77&chksm=88621acdbf1593dba5fa5a0092d810004362e9f38484ffc85112a8c23ef48190c51d17e06223&token=2041017112&lang=zh\_CN&scene=21#wechat\_redirect](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933107&idx=1&sn=6b9fbdfa180c2ca79703e0ca1b524b77&chksm=88621acdbf1593dba5fa5a0092d810004362e9f38484ffc85112a8c23ef48190c51d17e06223&token=2041017112&lang=zh_CN&scene=21" \l "wechat_redirect)

## 什么是线程安全？

当多个线程去访问同一个类（对象或方法）的时候，该类都能表现出正常的行为（与自己预想的结果一致），那我们就可以所这个类是线程安全的。

看一段代码：



Demo1中有个静态变量num，默认值是0，m1()方法中对num++执行10000次，main方法中创建了3个线程用来调用m1()方法，然后调用3个线程的join()方法，用来等待3个线程执行完毕之后，打印num的值。我们期望的结果是30000，运行一下，但真实的结果却不是30000。上面的程序在多线程中表现出来的结果和预想的结果不一致，说明上面的程序不是线程安全的。

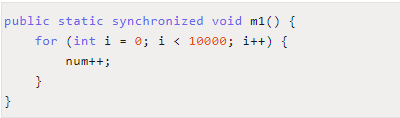
线程安全是并发编程中的重要关注点，应该注意到的是，造成线程安全问题的主要诱因有两点：

**1、存在共享数据(也称临界资源)**

**2、存在多条线程共同操作共享数据**

因此为了解决这个问题，我们可能需要这样一个方案，当存在多个线程操作共享数据时，需要保证同一时刻有且只有一个线程在操作共享数据，其他线程必须等到该线程处理完数据后再进行，这种方式有个高尚的名称叫互斥锁，即能达到互斥访问目的的锁，也就是说当一个共享数据被当前正在访问的线程加上互斥锁后，在同一个时刻，其他线程只能处于等待的状态，直到当前线程处理完毕释放该锁。在 Java 中，关键字 synchronized可以保证在同一个时刻，只有一个线程可以执行某个方法或者某个代码块(主要是对方法或者代码块中存在共享数据的操作)，**同时我们还应该注意到synchronized另外一个重要的作用，synchronized可保证一个线程的变化(主要是共享数据的变化)被其他线程所看到**（保证可见性，完全可以替代volatile功能），这点确实也是很重要的。

那么我们把上面的程序做一下调整，在m1()方法上面使用关键字synchronized，如下：



然后执行代码，输出30000，和期望结果一致。

## synchronized主要有3种使用方式

1、修饰实例方法，作用于当前实例，进入同步代码前需要先获取实例的锁

2、修饰静态方法，作用于类的Class对象，进入修饰的静态方法前需要先获取类的Class对象的锁

1. 修饰代码块，需要指定加锁对象(记做lockobj)，在进入同步代码块前需要先获取lockobj的锁

### synchronized作用于实例对象

所谓实例对象锁就是用synchronized修饰实例对象的实例方法，注意是实例方法，不是静态方法，如：



main()方法中创建了一个对象demo2和2个线程t1、t2，t1、t2中调用demo2的add()方法10000次，add()方法中执行了num++，num++实际上是分3步，获取num，然后将num+1，然后将结果赋值给num，如果t2在t1读取num和num+1之间获取了num的值，那么t1和t2会读取到同样的值，然后执行num++，两次操作之后num是相同的值，最终和期望的结果不一致，造成了线程安全失败，因此我们对add方法加了synchronized来保证线程安全。

注意：m1()方法是实例方法，两个线程操作m1()时，需要先获取demo2的锁，没有获取到锁的，将等待，直到其他线程释放锁后再取得锁为止。

**synchronize作用于实例方法需要注意：**

1、实例方法上加synchronized，线程安全的前提是，多个线程操作的是同一个实例，如果多个线程作用于不同的实例，那么线程安全是无法保证的

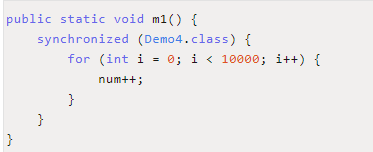
1. 同一个实例的多个实例方法上有synchronized，**这些方法都是互斥的**，同一时间只允许一个线程操作同一个实例的其中的一个synchronized方法

### synchronized作用于静态方法

当synchronized作用于静态方法时，锁的对象就是当前类的Class对象。如：

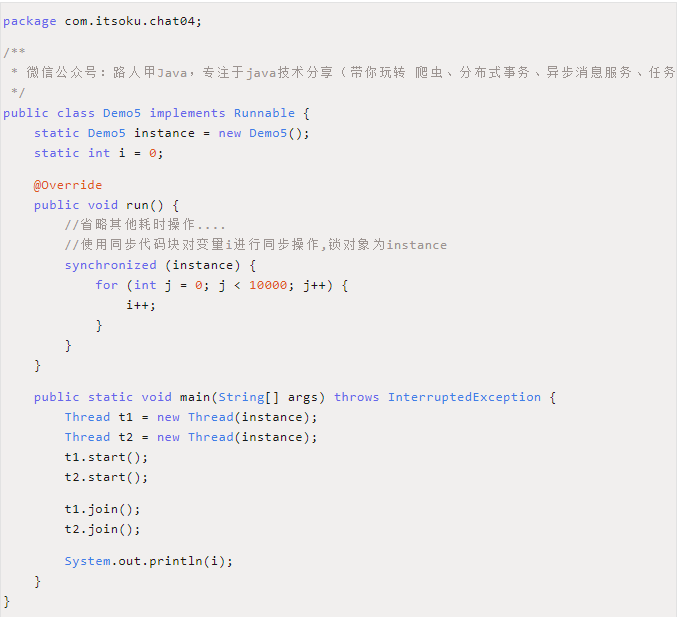


上面代码打印30000，和期望结果一致。m1()方法是静态方法，有synchronized修饰，锁用于与Demo3.class对象，和下面的写法类似：



### synchronized同步代码块

除了使用关键字修饰实例方法和静态方法外，还可以使用同步代码块，在某些情况下，我们编写的方法体可能比较大，同时存在一些比较耗时的操作，而需要同步的代码又只有一小部分，如果直接对整个方法进行同步操作，可能会得不偿失，此时我们可以使用同步代码块的方式对需要同步的代码进行包裹，这样就无需对整个方法进行同步操作了，同步代码块的使用示例如下：

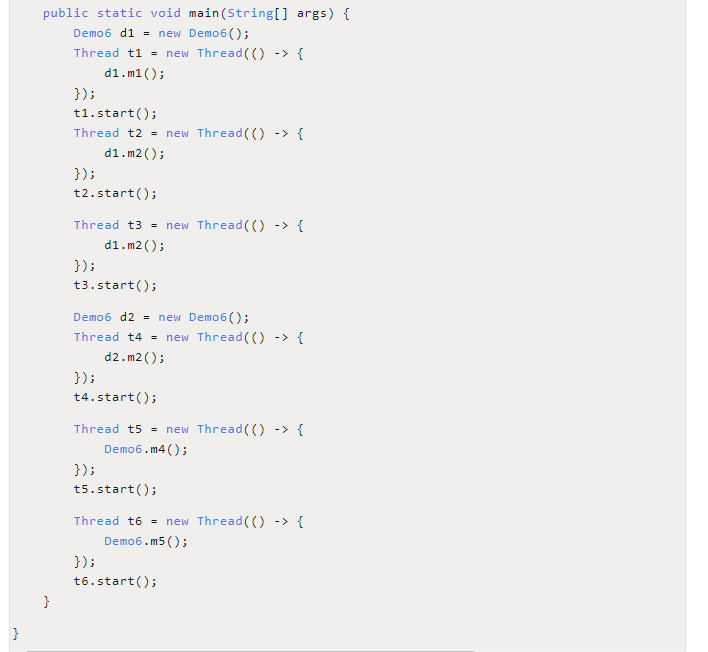


从代码看出，将synchronized作用于一个给定的实例对象instance，即当前实例对象就是锁对象，每次当线程进入synchronized包裹的代码块时就会要求当前线程持有instance实例对象锁，如果当前有其他线程正持有该对象锁，那么新到的线程就必须等待，这样也就保证了每次只有一个线程执行i++;操作。当然除了instance作为对象外，我们还可以使用this对象(代表当前实例)或者当前类的class对象作为锁，如下代码：



分析代码是否互斥的方法，先找出synchronized作用的对象是谁，如果多个线程操作的方法中synchronized作用的锁对象一样，那么这些线程同时异步执行这些方法就是互斥的。如下代码:





分析上面代码：

1、线程t1、t2、t3中调用的方法都需要获取d1的锁，所以他们是互斥的

2、t1/t2/t3这3个线程和t4不互斥，他们可以同时运行，因为前面三个线程依赖于d1的锁，t4依赖于d2的锁

3、t5、t6都作用于当前类的Class对象锁，所以这两个线程是互斥的，和其他几个线程不互斥

关于synchronized的实现原理，篇幅比较长，可以看<http://www.itsoku.com/article/168>。