**同步问题**

**多个线程访问共享数据时，可能出现脏读等情况，这非常危险**

**所以对于线程共享数据**

**我们应该保证 当多个线程访问共享数据时，任何一个时刻，有且只有一个线程在访问共享数据，而其他线程必须等待 ，直到当前线程结束访问（释放锁）后**

**——上述行为称作，正在访问共享数据的线程 给 共享数据 上了 互斥锁**

**也可以说，正在访问共享数据的线程 独占 共享数据**

**A被线程B上锁后——A就被线程B独占，其他的线程都不能访问A，只能等待，直到线程B释放锁**

**为了达到给 共享数据上锁的目的——java中提供了 synchronized关键字**

**①synchronized可以保证 任意时刻，只有一个线程可以访问**

**②synchronized可以保证一个线程的变化可以被其他线程看到**

**接下来我们看看Synchronized的用法**

**1,Synchronized代码块——给同步对象和代码块上锁**

**Synchronized(同步对象A){**

**代码块**

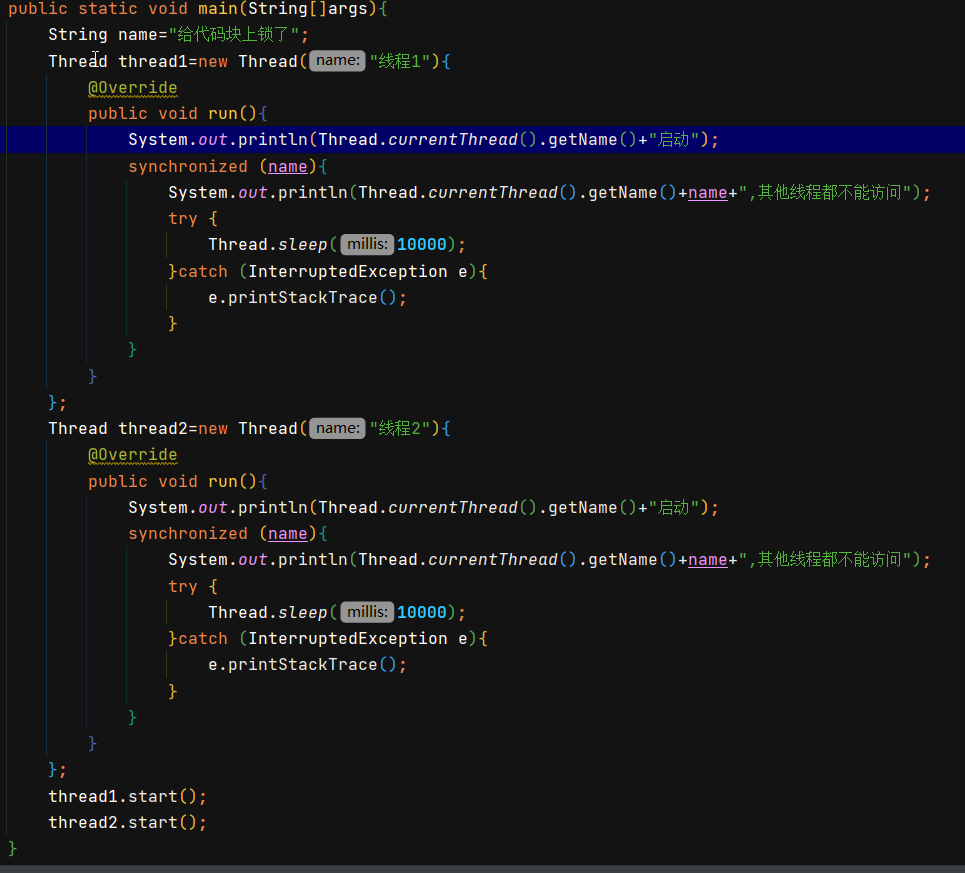
**}**

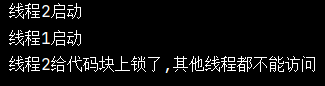
**①线程1运行到这里时，如果A没有被其他线程占用，那么线程1——给同步对象A和代码块上锁**

**②一个线程如果想执行代码块——必须占有同步对象A才能执行代码块**

**所以 其他线程只能等待，等待线程1释放锁**

**释放锁的时机——执行完代码块or 抛出异常**





**例子：如上图**

**运行后，线程1，线程2启动，然后线程1和线程2开始争夺同步对象name，线程2抢到了name，那么线程2会给name上锁，然后只有线程2可以继续向下执行，线程1由于没有占有name，无法向下继续执行，只能等待，等到线程2执行完，释放name（释放锁），线程1才能取得name，开始执行代码块**

**2，Synchronized修饰普通方法——给同一个实例对象上所有的synchronized普通方法上锁**

**注意，一个线程执行了实例对象A的synchronized方法，只会给A上的所有synchronized方法上锁，非synchronized方法不会上锁，其他的东西都不会上锁，并且也只有A上的synchronized方法会上锁，同一个类的实例对象B不会受到任何影响**

**①首先我们要知道，普通方法是存在于实例对象中的**

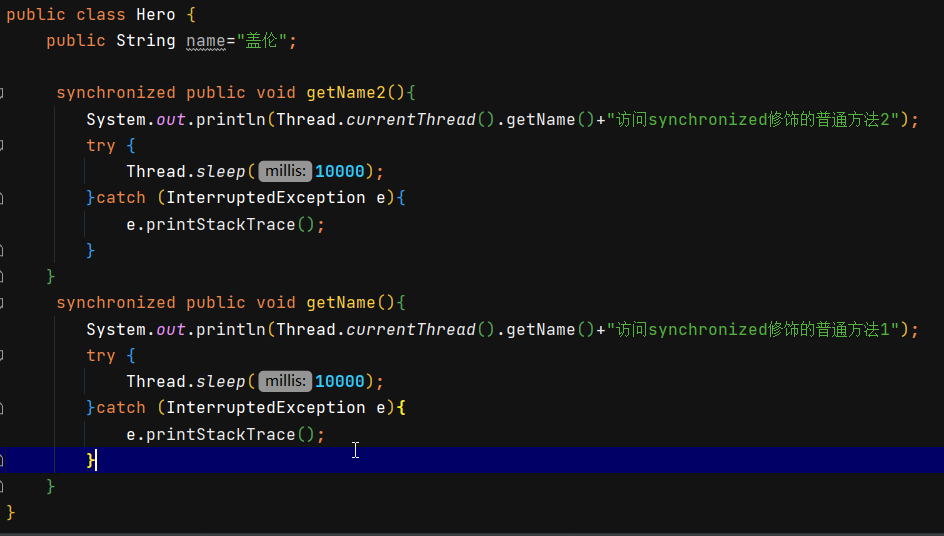
**这里我们假设 实例对象A上有synchronized普通方法A1，A2**

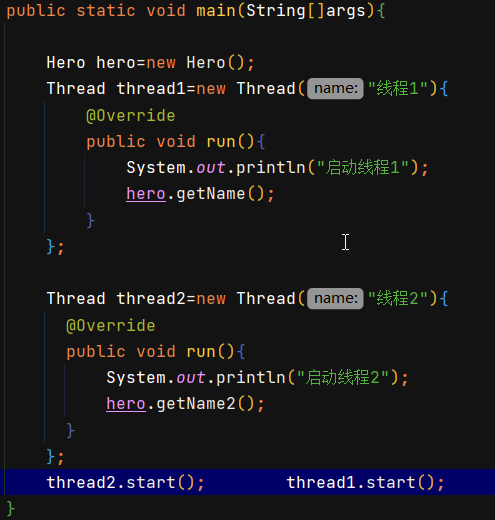
**②如果线程1 执行了 实例对象A的synchronized修饰的普通方法A1，**

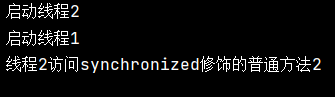
**那么线程1——给 实例对象A上的所有synchronized普通方法上锁**

**上锁后，其他线程无法执行实例对象A的A1或A2，只能等待，直到线程1释放锁**

**例子1**







**运行后，启动线程2，启动线程1**

**然后线程2先执行了 Hero实例对象中的getName2( )方法**

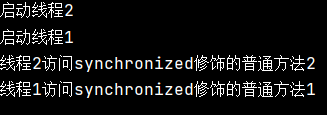
**——线程2 给Hero实例对象中的所有 synchronized普通方法上锁**

**线程1无法 访问 Hero实例对象中的getName( )方法**

**线程1只能等待，直到线程2释放锁**

**例子2：我们把getName( )设置为 非synchronized,main()不做任何修改**





**线程2启动，线程1启动**

**线程2执行 Hero实例对象中的getName2( )方法**

**——线程2给 Hero实例对象中的所有synchronized方法上锁**

**但是不会给非synchronized方法上锁，也就是说getName( )不会上锁**

**所以线程1可以正常执行getName( )，不需要等待线程2释放锁**

**线程安全类**

**如果一个类，它的方法都是由synchronized修饰的——这个类，为线程安全类**

这个类的一个实例，一次只能被一个线程访问，保证了这个实例的数据的安全

**死锁问题**

