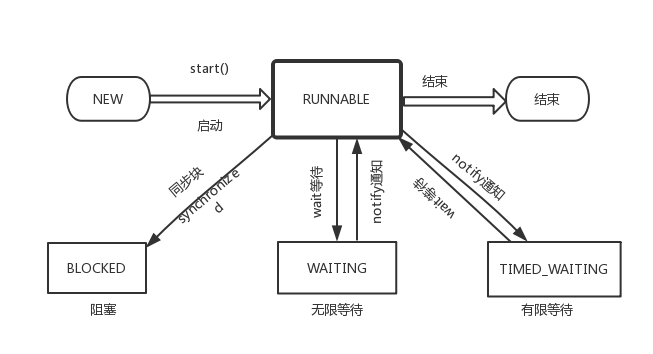
# 高并发编程

## 线程基础

### 线程的基本操作

#### 线程的生命周期



NEW： NEW 状态表示刚刚创建的线程，这种线程还没开始执行。当调用start()方法时,线程才开始执行。

RUNNABLE: 当线程执行时，就处于RUNNABLE 状态,表示线程所需要的一切资源都已经准备好了。

BLOCKED: 当线程在执行的过程中遇到synchronized同步块，就会处于BLOCKED阻塞状态,这是线程会暂停执行，知道获得请求的锁。

WAITING和TIMED\_WAITING: 这个两个都表示等待状态,一般TIMED\_WAITIING会进行有限的等待而WAITING的线程正在等待进行一个特殊的事件。比如，通过wait()方法等待的线程在等待notify()方法，而通过join()方法等待的线程会等待目标线程的终止。

TERMINATED: 表示线程结束。

#### 新建线程

1. **怎么新建一个线程？**

通过new Thread().start();这样就新建了一个线程。

注意: 我们必须调用Thread对象实例的start()方可创建新线程，

而不是直接调用run()方法。

1. **怎么让新线程执行我们自己的代码？**
2. 通过继承Thread类，重写run()方法。
3. 通过实现Runnable接口，实现run()方法。

然后调用start()方法启动线程。

#### 终止线程

1. **一般来说，当线程执行完程序之后就会自行结束，无需手工关闭。但是有些程序是常驻后台提供服务的线程。那么怎么安全的关闭这些线程？**

Thread类提供了一个stop()方法，如果调用了stop()方法就会立即停止线程。但是这个方法如果使用不当，会造成线程不安全。因为，stop()方法调用后，会立即停止线程，并释放这个线程所持有的锁,而这个锁恰好就是用来维持对象一致性的。比如线程A持有Person对象锁，线程A准备修改Person对象的name和age属性，但是修改了name，再修改age之前调用了stop()方法。那么这个对象的一致性就被破坏了，如果再来一个线程B，它读取的数据，就被修改过的数据。

我们可以自定义一个终止标签。当标签设置为true时，安全退出。

#### 线程中断

在java中,线程中断是一种很重要的线程协作机制。不同于终止线程，

线程中断不会立即停止线程，而是一种平滑中断的手段，当我们想通知某个线程中断时，这个线程要不要中断是由该线程决断。

关于线程中断的三个方法：

public void Thread.interrupt(); //中断线程

public void Thread.isInterrupted(); //判断是否被中断

public void Thread.interrupted();//判断是否被中断，并清除当前中断状态

我们可以通过Thread.interrupte();方法通知线程中断,然后该线程通过Thread.isInterrupted();判断是否被中断,然后安全的退出线程。

注意： Thread.sleep()方法会抛出一个InterruptedException()中断异常。当线程在sleep()休眠时,如果被中断，就会产生这个异常。此时，它会将中断标记清除。



我们需要在catch代码块中，重新标记中断。

#### wait()和notify()方法

wait()和notify()方法时Object类中的方法。

当一个obj对象调用wait()方法时会释放obj监视器。由其他线程获得，其他线程调用obj.notify()方式，会随机唤醒一个线程，被唤醒的线程不会立即获得obj监视器,必须要等待获取。notifyAll()会唤醒所有线程。

### 线程安全

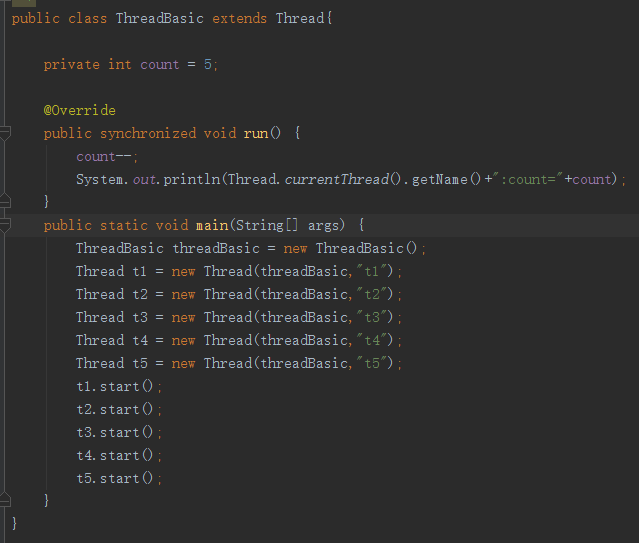
#### 什么是线程安全

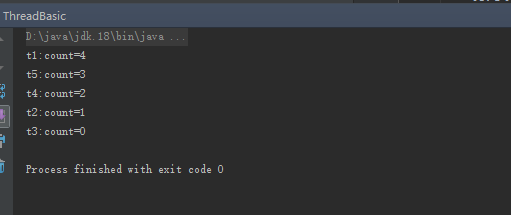
当多个线程访问同一个类(对象或方法)时,这个类（对象或方法）都能表现出正确的行为。那么就说这个类(对象或方法)是线程安全的。

#### Synchronized

可以在任意对象及方法上加锁，而加锁的这段代码称为“互斥区”或“临界区”。

#### 实例





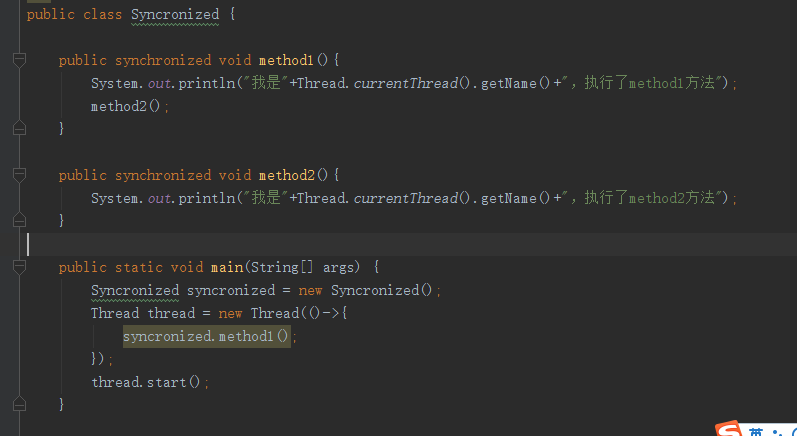
**总结：”**

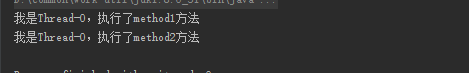
当多个线程访问theadBasic对象的run方法时,会以排队的方式进行处理(这里的排队是按照CPU分配的先后顺序而定的)，一个线程想要执行synchronized关键字修饰的代码，首先要尝试获得锁,如果拿到锁,执行synchronized里面的代码；拿不到锁，这个线程就会不断尝试获得锁，直到拿到为止,而且是多个线程同时去竞争这把锁(也就是会有锁竞争的问题)。

#### Synchronized锁重入

关键字Synchronized具有重入的功能，也就是使用synchronizd时，当一个线程获得了一个对象的锁,再次请求此对象时是可以再次获得该对象的锁。

**实例：**





#### Synchronized异常

我们的代码在执行过程中肯定会出现异常的情况，那么在Synchronized中出现异常怎么处理？

1. 将异常信息和异常数据记录日志，然后跳过该异常继续执行。
2. 如果是原子性操作则直接回滚,或将操作作废，然后邮件报警。

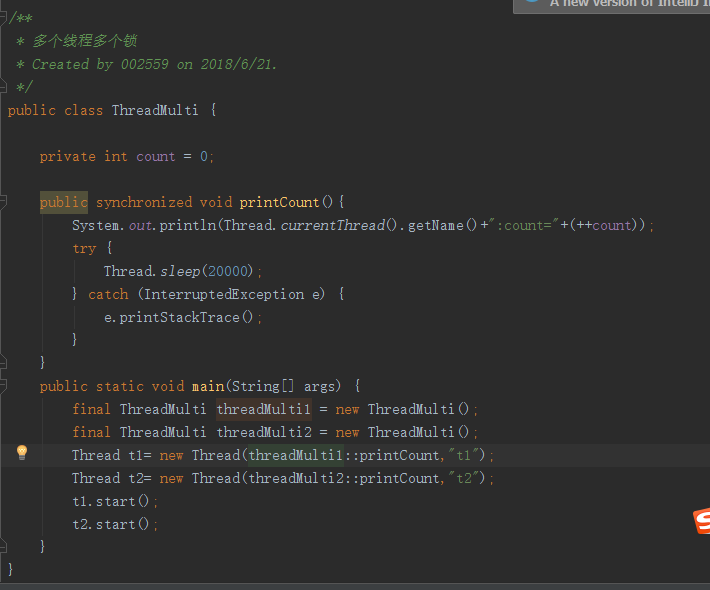
#### Synchronized使用的锁类型

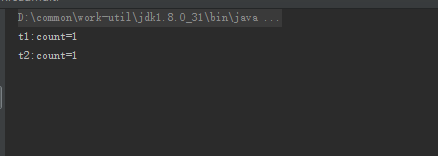
Synchroized 可以使用各种对象或者类作为锁。在使用的过程中需要注意的是。

1. 不要使用字符串常量作为锁，会出现一些奇怪的问题。
2. 如果使用了字符串对象作为锁，那么在执行过程中，不能修改字符串对象的引用，其他对象也一样。

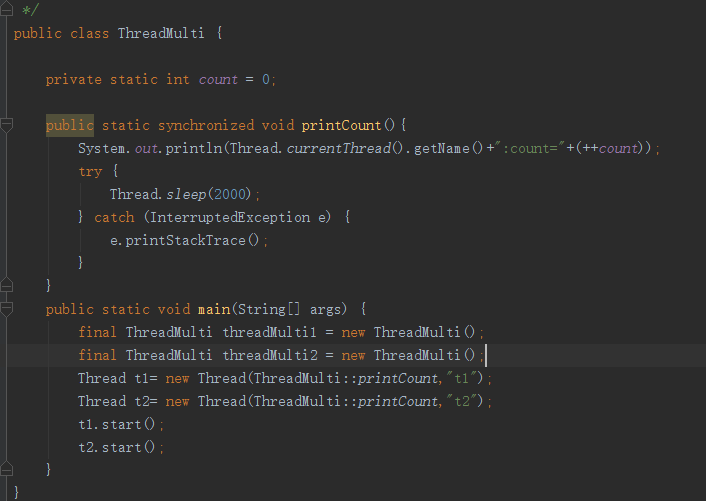
### 多个线程多个锁

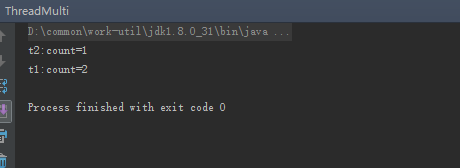
多个线程，每个线程都能获取自己指定的锁，分别获取锁后执行synchronized里面的代码。





静态方法





**总结：**

关键字synchronized取得的锁都是对象锁，而不是把一段代码当做锁,执行哪个对象的方法，那么就持有该对象的锁。两个线程分别执行两个对象里面的方法时，获取的是两个对象各自的锁，所以互不影响。

有一种情况是相同的锁，那就是synchronized修饰的静态方法，表示锁定class类。

### 对象锁的同步和异步

#### 同步(synchronized)

同步的概念是共享,如果不是共享的资源，就没必要同步。

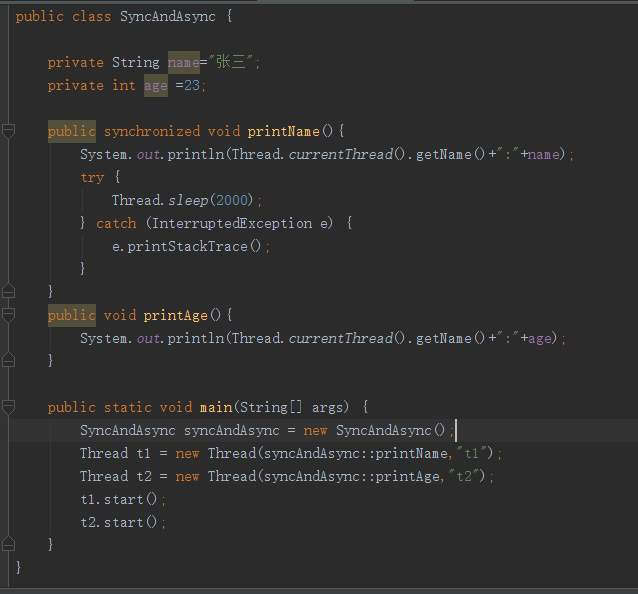
同步的目的是为了线程安全，对于线程安全来说,需要满足两个特性：

原子性、可见性。

#### 异步(asynchronized)

异步的概念是独立，线程相互之间不受任何制约。

#### 实例



**总结：**

如果t1，t2线程都调用 syncAndAsync对象的printName方法，那么这就是同步。假设t1线程先进入printName执行，那么t2线程必须等待t1线程释放锁。

如果t1线程调用printName方法，而t2线程调用printAge方法

那么这就是异步。t1,t2线程互不影响。

### 5. 有关线程的问题

#### 1. 脏读

当我们设计程序的时候，一定要考虑程序的整体性和一致性。也就是共享资源的时候。