JAVA多线程

1. 线程相关概念
2. 进程：

进程是指运行中的程序，如使用QQ，就启动了一个进程，操作系统就会为改进程分配内存空间。

进程是程序的一次执行过程，或是正在运行的一个程序。是动态过程：有它自身的产生、存在和消亡的过程。

1. 线程：

线程是由进程创建的，是进程的一个实体，一个进程可以拥有多个线程

1. 单线程：

同一时刻，只允许执行一个线程

1. 多线程：

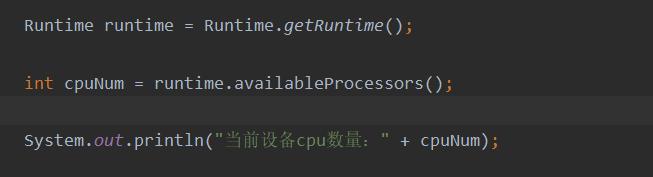
同一时刻，可以执行多个线程，比如一个QQ进程，可以同时打开多个聊天窗口

1. 并发：

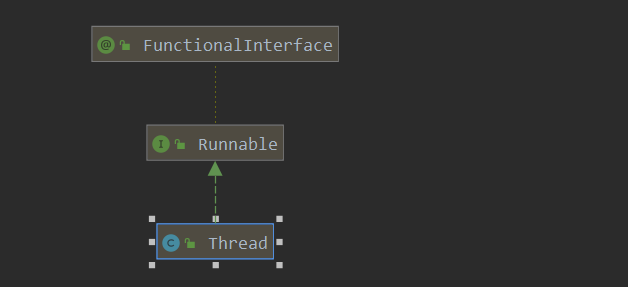
同一时刻，多个任务交替执行，造成一种“貌似同时”的错觉，简单的说，单核cpu实现的多任务就是并发

1. 并行：

同一时刻，多个任务同时执行，多核cpu可以实现并行



1. 创建线程



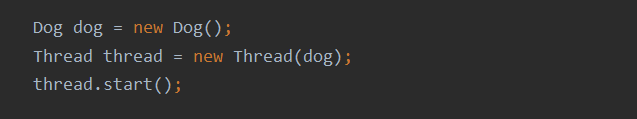
* 继承Thread类：





* 实现Runnable类：





* 继承Thread和实现Runnable接口创建线程二者的区别：
* 从java的设计来看，通过继承Thread或者实现Runnable接口来创建线程本质上没有区别。从jdk帮助文档我们可以看到Thread类本身就实现了Runnable接口
* 实现Runnable接口方式更加适合多个线程共享一个资源的情况，并且避免了单继承的限制，建议使用Runnable接口实

1. 多线程机制

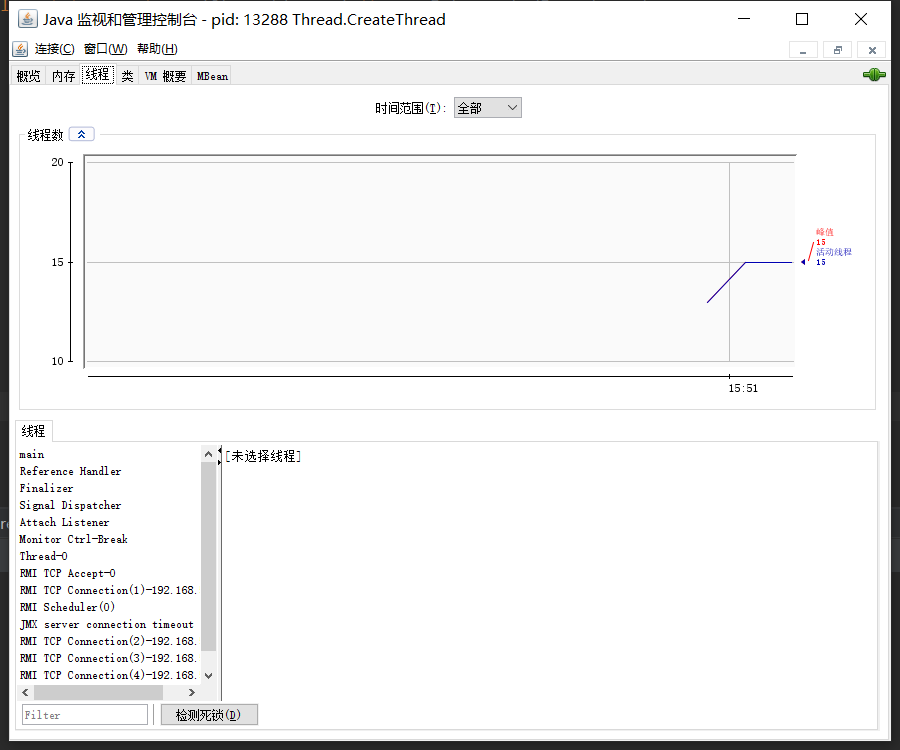
* 执行流程：

创建进程（Application）-- main线程（主） – Thread-0线程（子）

注意：当main线程启动一个子线程Thread-0，主线程不会阻塞，会继续执行



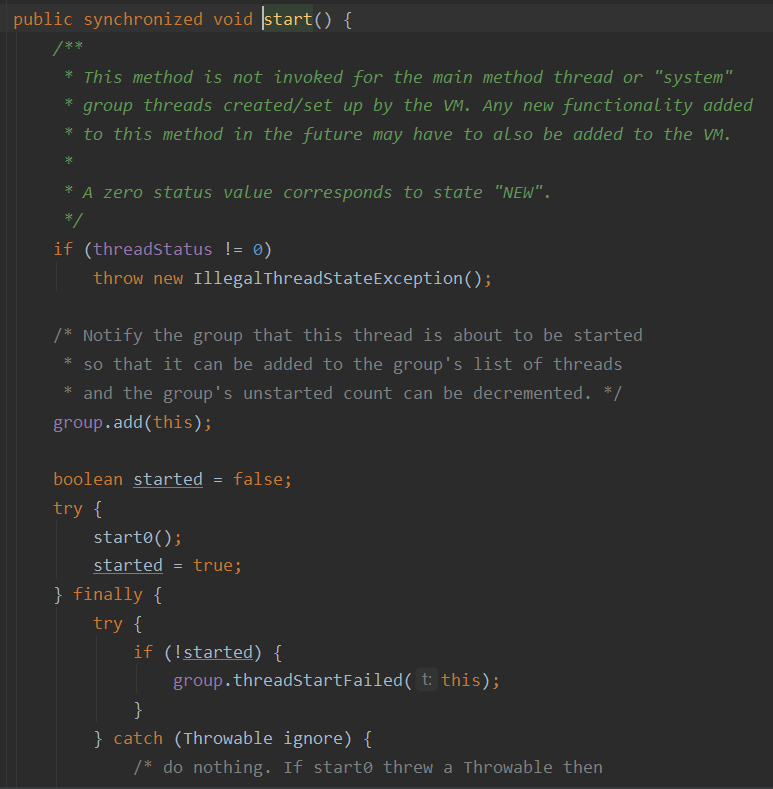
可通过在idea的Terminal控制台输入Jconsole命令查看线程情况

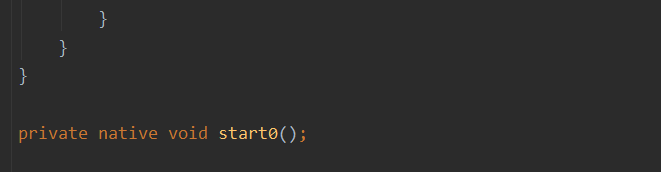


1. 为什么调用start方法，而不直接调用run方法

如果直接调用以上cat.run()方法，那么当前的run方法就是一个普通方法，没有真正的启动一个线程，就会把run方法执行完毕，然后才会往下执行main线程；

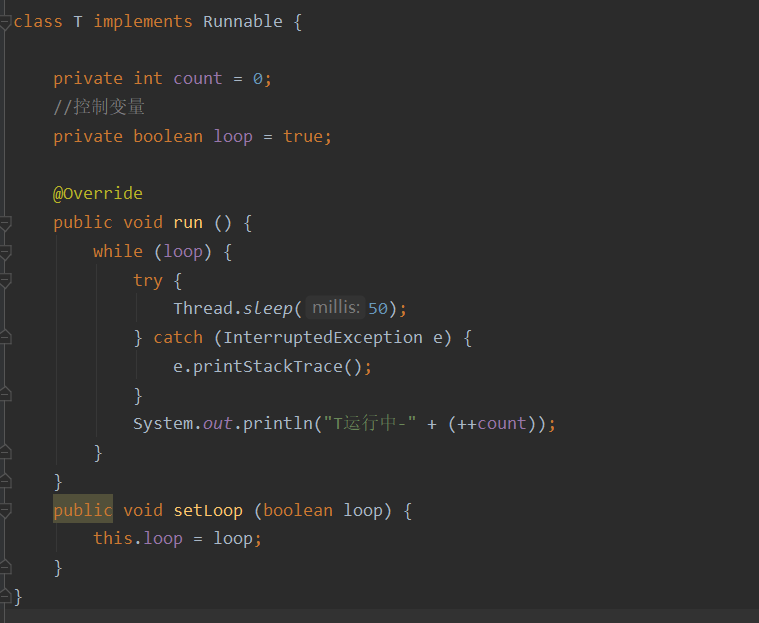
当执行start方法时，start方法内部会执行start0方法，此方法是本地方法，是由JVM调用，底层是C/C++实现。真正实现多线程的效果是start0方法，而不是run方法





1. 实现多个子线程
2. 线程终止

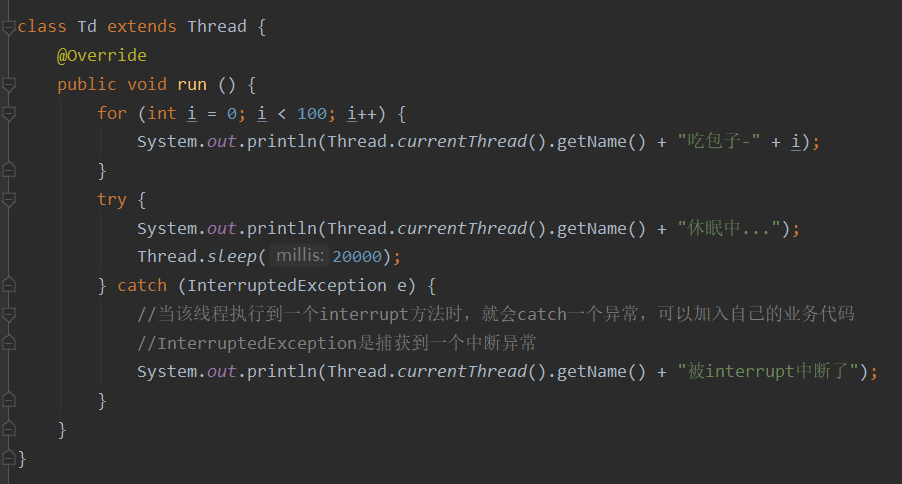
* 当线程完成任务后，会自动退出
* 还可以通过使用变量来控制run方法退出的方法停止线程，既通知方式
* 通知线程退出





1. 线程常用方法

* setName：设置线程名称
* getName：获取线程名称
* start：启动线程
* run：调用线程对象run方法
* setPriority：更改线程的优先级
* getPrioity：获取线程的优先级
* sleep：在指定的毫秒数内让当前线程休眠
* interrupt：中断线程
* getState：获取线程当前状态





* 注意事项：

start底层会创建新的线程，调用run，run就是一个简单的方法调用，不会启动新的线程

interrupt，中断线程，并没有真正的结束线程。所以一般用于中断正在休眠的线程

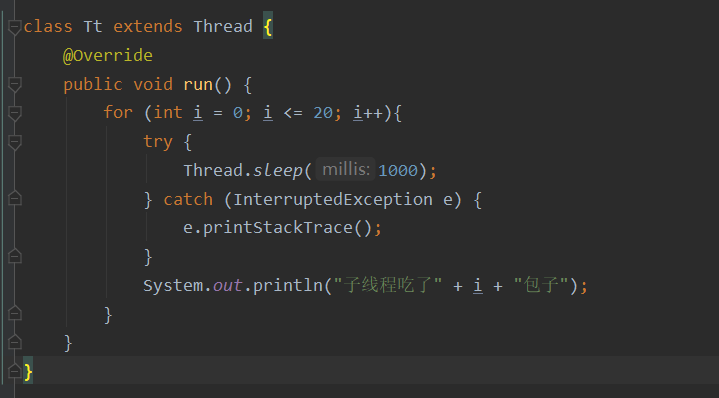
1. 线程插队

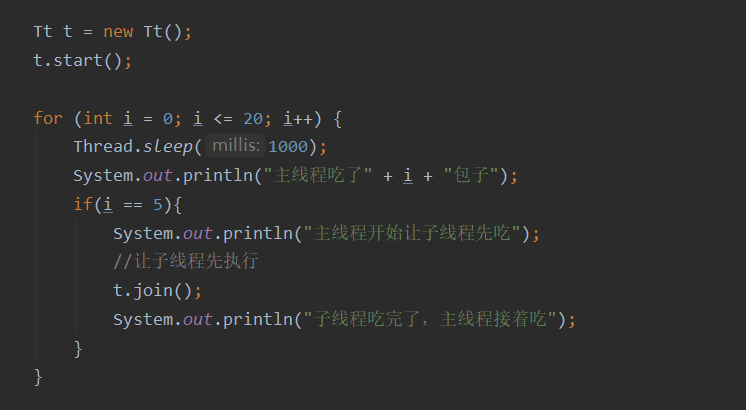
* yield：

线程的礼让，让出cpu，让其它线程执行，但礼让的线程不确定，所以也不一定成功

* join：

线程的插队，插队的线程一旦插入成功，则肯定先执行完插入的线程所有的任务





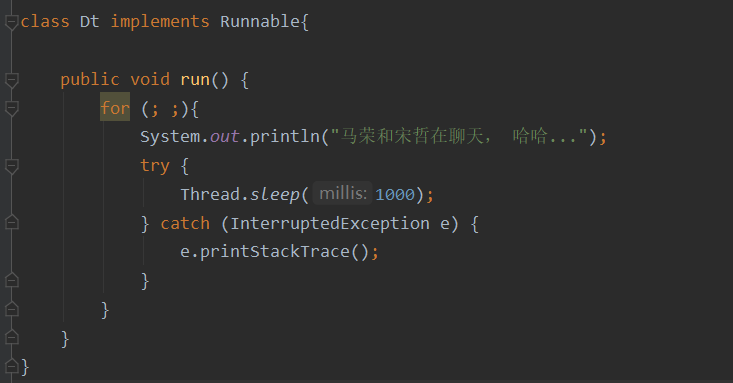
1. 用户线程和守护线程

* 用户线程：

也叫工作线程，当线程的任务执行完或通知方式结束

* 守护线程：

一般是为工作线程服务的，当所有的用户线程结束，守护线程自动结束

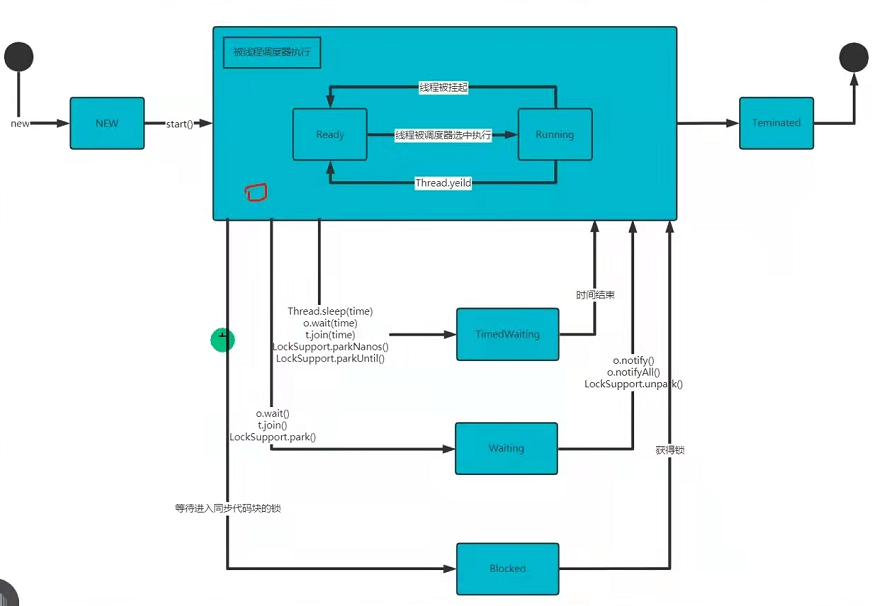




* 常见的守护线程：垃圾回收机制

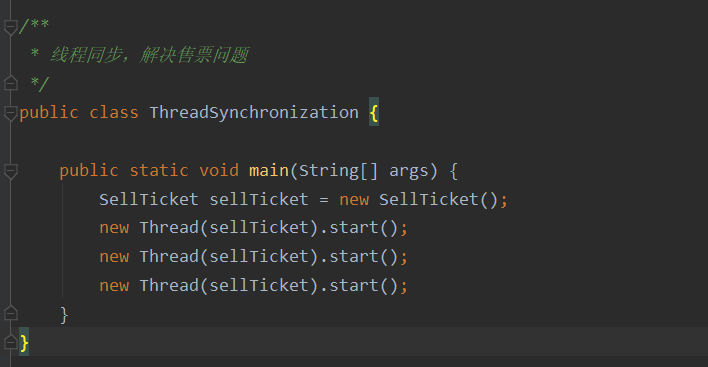
1. 线程生命周期

* **new**：新建状态
* **runnable**：Ready、Runing
* **TIMED**\_**WAITING**
* **Waiting**
* **Blocked**
* **TERMINATED**



1. 线程同步机制

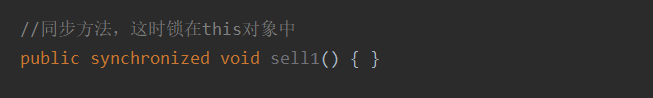
同步：在多线程编程中，一些敏感数据不允许被多个线程同时访问，此时就使用同步访问技术，保证数据在同一时刻，最多有一个线程访问，以保证数据的完整性。

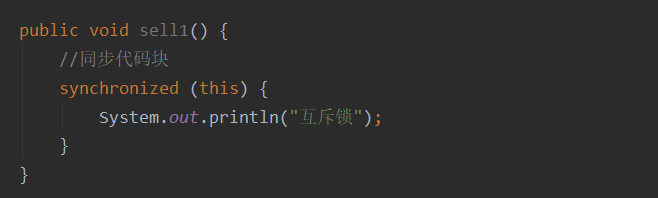


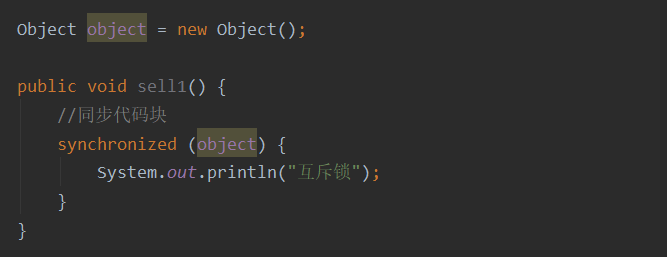


1. 互斥锁

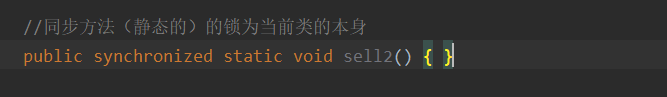
* 基本介绍：
* Java语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性
* 每个对象都对应一个可称为“互斥锁”的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象
* 关键字syschronized来于对象的互斥锁的联系。当某个对象用syschronized修饰时，表明该对象在任一时刻只能有一个线程访问
* 同步的局限性：导致程序执行效率要降低
* 同步方法（非静态的）的锁可以是this，也可以是其它对象（要求是同一个对象）

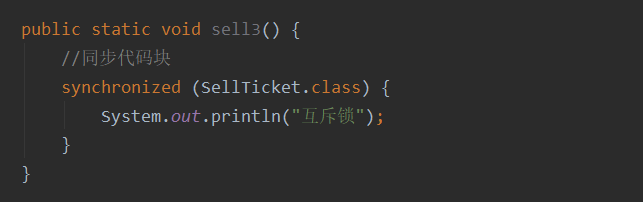






* 同步方法（静态的）的锁为当前类的本身。





* 注意事项：
* **同步方法如果没有使用static修饰：**默认锁的对象就是this
* **如果方法使用static修饰：**默认锁的对象就是当前类.class

1. 线程死锁

多个对象都占用了对方的锁资源，但不肯相让，导致了死锁，在编程时，一定要避免死锁的发生

1. 释放锁

* 以下操作会释放锁：
* 当前线程的同步方法、同步代码块执行结束
* 当前线程在同步代码块、同步方法中遇到break、return
* 当前线程在同步代码块、同步方法中出现了未处理的Error或Exception，导致异常结束
* 当前线程在同步代码块、同步方法中执行了线程对象的wait()方法，当前线程暂停，并释放锁
* 以下操作不会释放锁：
* 线程执行同步代码块或同步方法时，程序调用Thread.sleep()、Thread.yield()方法暂停当前线程的执行，不会释放锁
* 线程执行同步代码块时，其它线程调用了该线程的suspend()方法将该线程挂起，该线程不会释放锁