### JDK提供了几个很实用的工具

**jinfo：**观察运行中的java程序的运行环境参数：参数包括Java System属性和JVM命令行参数，java class path等信息。命令格式：jinfo 进程pid

**jps：**用来显示本地的java进程，可以查看本地运行着几个java程序，并显示他们的进程号。命令格式：jps   或 jps 远程服务ip地址    （默认端口1099）

**jstat：**一个极强的监视VM内存工具。可以用来监视VM内存内的各种堆和非堆的大小及其内存使用量。

**jstack：**可以观察到jvm中当前所有线程的运行情况和线程当前状态。若现运行的java程序呈现hung的状态，jstack是非常有用的。命令格式：jstack 进程pid

当程序出现死锁的时候，使用命令：jstack 进程ID > jstack.log，然后在jstack.log文件中，搜索关键字“**BLOCKED**”，定位到引起死锁的地方。

**jmap：**观察运行中的jvm物理内存的占用情况（如：产生哪些对象，及其数量）。命令格式： jmap [option] pid

      option参数如下：

             -heap：打印jvm heap的情况

             -histo：打印jvm heap的直方图。其输出信息包括类名，对象数量，对象占用大小。

             -histo：live ：同上，但是只答应存活对象的情况

             -permstat：打印permanent generation heap情况

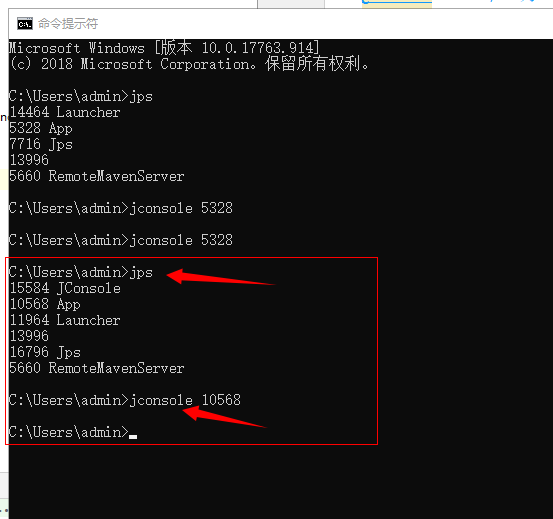
使用jmap进行heap dump的例子：jmap -dump:format=b,file=<filename> <pid>

 打印内存统计图：jmap -histo:live <pid>

       结果中每行显示了当前堆中每种类类型的信息，包含被分配的实例个数及其消耗的字节数。选项“**live**”，表示只统计存活的对象

       需要注意的是，jmap不是运行分析工具，在生成统计图时JVM可能会暂停，因此当生成统计图时需要确认这种暂停对程序是可接受的。

**jconsole：**一个java GUI监视工具，可以以图表化的形式显示各种数据。并可通过远程连接监视远程的服务器VM。



这些命令的使用，见官方文档：

jps：<http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jps.html>

jstat：<http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstat.html>

jstack：<http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstack.html>

jmap：<http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jmap.html>

jconsole：<http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html>

原文链接：<https://blog.csdn.net/caihaijiang/article/details/6084325>

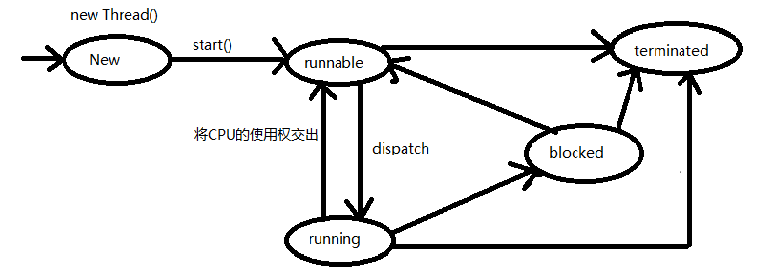
### 创建线程和线程的生命周期

例子：JavaConcurrency----Test1----TryConcurrency

例子：JavaConcurrency----Test1----TemplateMethod

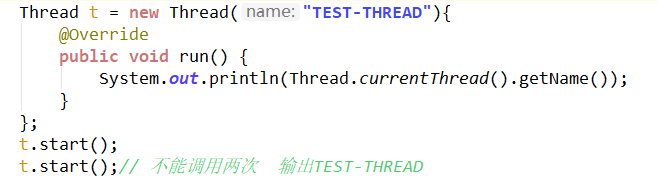
当程序启动会有一个非守护线程，通常是main线程，是被JVM启动时调用的，以及一些其他的保护线程自动生成，但是例子中当自定义“READ-THREAD”和“WRITE-THREAD”的线程启动Main线程就自动死亡了

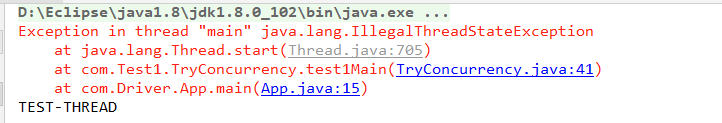
Thread 的生命周期



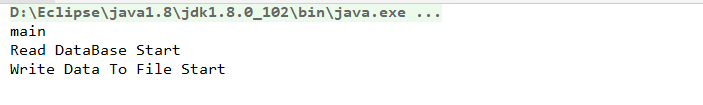
new Thread(), override run()之后只是一个java的实例，只有当start()之后才是一个线程。

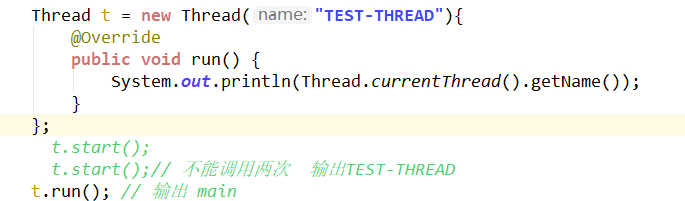
同一个线程不能调用两次start()





调用run()，等于说调用了这个实例的方法





JVM 启动后，实际上有多个线程，但至少有一个非守护线程，

当调用一个线程start()后至少有两个线程，一个是调用你的线程，一个是执行run()方法的线程，也就是你创建的线程

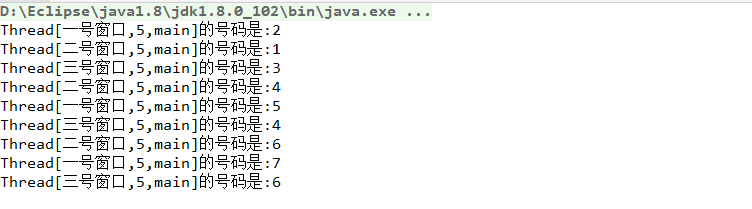
线程的生命周期分为new Thread()、runnable、running、block、terminated

### 使用runnable实现多线程的银行排队叫号

例子：JavaConcurrency---Test2----TicketWindow 业务和线程混在一起

例子：JavaConcurrency---Test2----TicketWindowRunnable 业务和线程分离开

没加睡眠的时候，会出现一个线程完成所有的业务，但是加了睡眠Thread.Sleep(1000\*5L)之后，也还是会出现两个窗口同时叫到一个号的情况，



### 使用runnable实现税务计算

例子：JavaConcurrency---Test2----TaxCalculator 使用策略方式

例子：JavaConcurrency---Test2----TaxCalculatorNew 使用java8新特性

### 构造Thread对象的几个点

例子：JavaConcurrency---Test3----CreateThread1

创建线程对象默认有一个名字，以Thread-开头，从0开始计数，Thread-0、Thread-1....，构造函数Thread()。如：threadMain1()

如果在构造Thread的时候没有传递Runnable或者复写run方法的时候，该Thread将不会调用任何东西，如果传递了Runnable接口的实例或者复写run方法则会执行该方法的逻辑单元（逻辑代码）。如：threadMain2()

Thread()

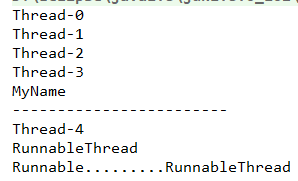
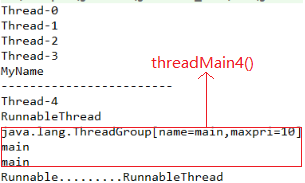
Thread(Runnable target)

Thread(Runnable target, String Name)

Thread(String Name)

疑问：threadMain3()中的t5的start的输出为什么总是在程序最后



原因：Start()之后才真正启动了一个线程，才开始执行该线程的业务逻辑

如果构造线程对象是没有传入ThreadGroup，Thread会默认获取父线程的ThreadGroup作为该线程的ThreadGroup，此时子线程和父线程将会在同一个ThreadGroup中。threadMain5()

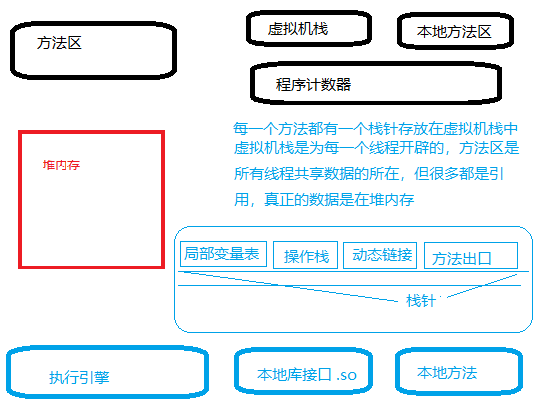
Thread(ThreadGroup group, Runnable target)

Thread(ThreadGroup group, Runnable target, String Name)

Thread(ThreadGroup group, Runnable target, String Name, Long stackSize)

Thread(ThreadGroup group, String Name)

### 多线程与JVM内存结构的关系，虚拟机栈实验



例子：JavaConcurrency---Test3----CreateThread2

构造Thread传入stackSize代表着当该线程占用的stack大小；如果没有指定stackSize的大小，则默认是0，0代表着忽略该参数，该参数会被JNI去使用，另外需要注意，该参数在一些平台（操作系统）有效，有些平台则无效

Daemon线程的创建以及使用场景分

### 析；线程ID，优先级

例子：JavaConcurrency---Test4----DaemonThread、ThreadSimpleAPI

长连接：A <---------------------------------------> B 每隔一段时间发一段心跳

--> DaemonThread(health check)

疑问：优先级设置之前多线程输出是混乱的，但是设置之后也是混乱的



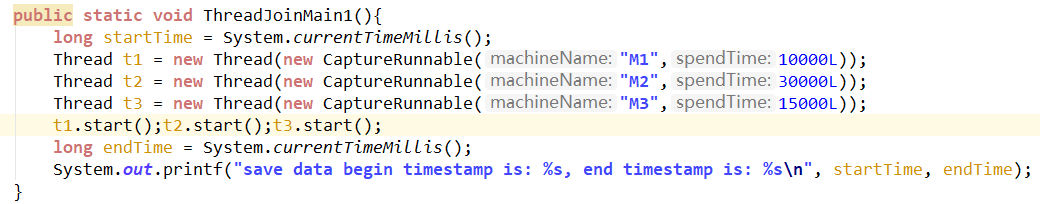
必须在start之前设置为守护线程（setDaemon(true)）,否则会报IllegalThreadStateException

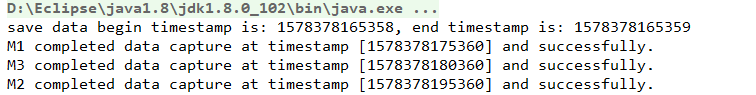
可以利用守护线程做一些不太重要的业务，但是要注意主线程结束，守护线程也就结束了

### Thread的join方法，结合一个典型案例

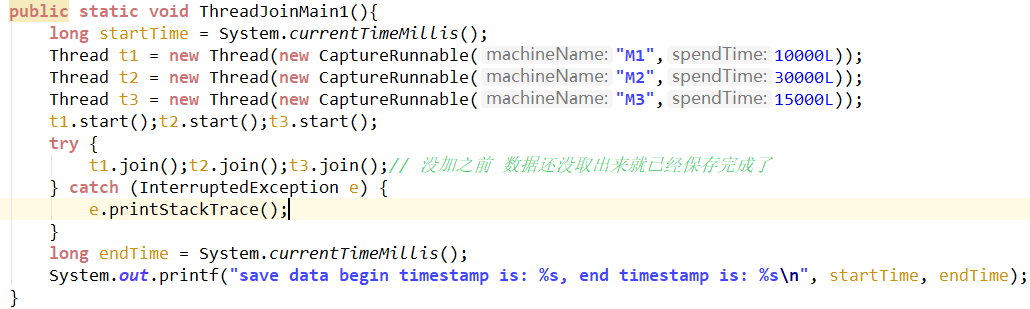
例子：JavaConcurrency---Test5----ThreadJoin1、ThreadJoin2

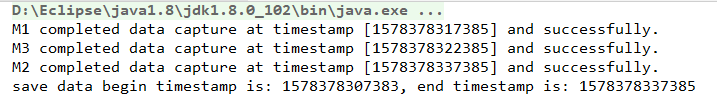
你调用了join之后，我会一直等你执行完我才去执行我的业务逻辑





数据还没取出来就已经保存完成了，当加入join调用后则得到正常的逻辑，所有采集数据线程结束只有主线程才去保存数据





Thread中断Interrupt方法

例子：JavaConcurrency---Test6----ThreadInterrupt1、ThreadInterrupt2

