理解 Java 多线程和并发

# 什么是线程、多线程

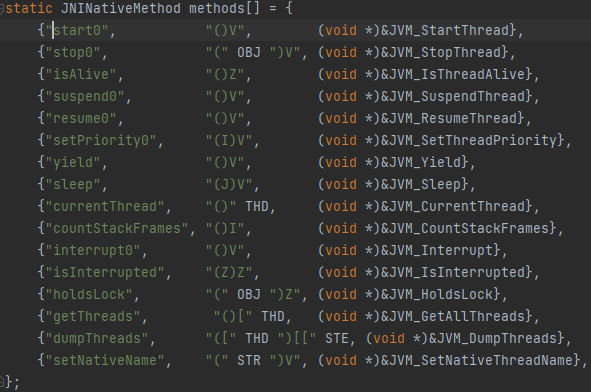
## 线程

了解线程之前，首先需要知道进程。操作系统在运行一个程序时，会为其创建一个进程。进程是具有独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配的基本单位，它具有独立的地址空间。

为了更高性能、低开销地处理任务，线程应运而生。线程是进程中的一个运行实体，是CPU调度的基本单位。它包含了线程ID、线程状态、寄存器（用于保存上下文环境）、栈和栈指针等属性。线程能够创建和撤销另一个线程，在同一个进程内的线程共享进程的地址空间和资源。

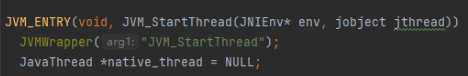
### Java线程

在JVM中线程由Thread对象表示。创建Thread对象并调用start()方法后，会调用native的start0()方法，start0对应的是JNI的JVM\_StartThread方法:





此方法在hotspot的jvm.cpp源文件中被使用：



定位到创建线程方法：







pthread\_create()是Linux创建线程的系统调用。

由上可见，创建并运行一个Thread对象，实际上在Linux操作系统中创建了一个内核级线程，即Java Thread 和内核线程是1：1关系。

在上古时期（JDK1.3以前），JVM创建的是Green thread。Green thread指的是由虚拟机或者运行时库调度的线程，而并不是被OS调度。它仅短暂的在1997-2000之间被使用。此后的JDK版本创建的线程都直接由OS调度。

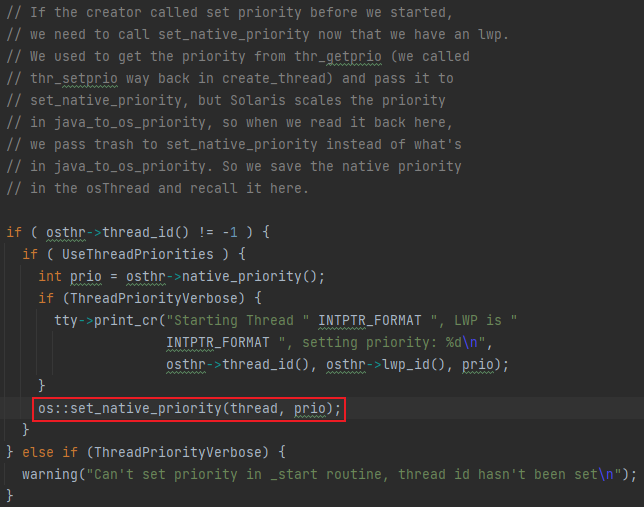
### Java线程状态

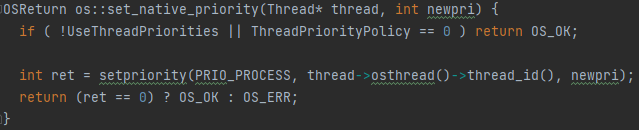
在JVM（非操作系统）中，线程状态分为如下6种：

|  |  |
| --- | --- |
| 状态 | 描述 |
| NEW | 线程被创建但未启动 |
| RUNNABLE | 正在执行或者具备执行条件 |
| BLOCKED | 等待锁引起阻塞 |
| WAITING | 无时间限制地等待另一条线程执行某个动作 |
| TIMED\_WAITING | 在有限时间内等待另一条线程执行某个动作 |
| TERMINATED | 线程结束执行 |

### 优先级

Thread#setPriority用于设置Java线程优先级，有效值范围为1-10。在JVM中启用线程优先级条件比较苛刻。需要同时满足（1）使用root权限运行程序（2）启动参数设置-XX:ThreadPriorityPolicy=1。

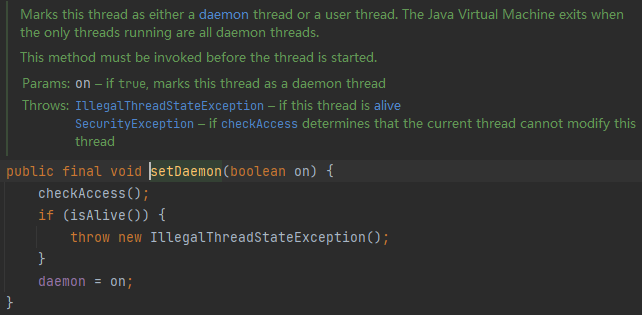




需要注意，在Java编程中需要尽量避免通过设置线程优先级方式实现业务逻辑，因为操作系统并不保证优先级高的线程一定优先于低优先级执行，并且过于依赖操作系统实现，对功能可靠性影响太大。

### Daemon

Java中的守护线程在Thread的对象中用daemon属性表示。



通过调用setDaemon方法使一条线程成为daemon线程，但是必须在调用start方法前设置。

Daemon线程不会影响JVM退出，当应用程序只剩下daemon线程后，JVM就会退出。因此不应该在daemon线程里做资源操作的业务，否则容易出现如文件数据丢失等情况。

## 多线程

多线程（multithreading），是指从软件或者硬件上实现多个线程并发执行的技术。

### 软件多线程

即便处理器只能运行一个线程，操作系统也可以通过快速的在不同线程之间进行切换，由于时间间隔很小，来给用户造成一种多个线程同时运行的假象。这样的程序运行机制被称为软件多线程。如微软的Windows作业系统和Linux就是在各个不同的线程间来回切换，被称为单人多任务作业系统。而DOS这类文字接口作业系统在一个时间只能处理一项工作，被视为单人单工作业系统。

### 硬件多线程

存在多个处理器时，每个处理器可以独立执行指令，达到并行效果。也有些技术，如CMT，依靠硬件执行线程切换，获得多线程能力。

### 上下文切换

通过软件实现的多线程技术中，CPU通过给每个线程分配CPU时间片来实现这个机制。时间片是CPU分配给各个线程的时间，因为时间片非常短，所以CPU通过不停地切换线程执行，让我们感觉多个线程是同时执行的，时间片一般是几十毫秒（ms）。

CPU通过时间片分配算法来循环执行任务，当前任务执行一个时间片后会切换到下一个任务。但是，在切换前会保存上一个任务的状态，以便下次切换回这个任务时，可以再加载这个任务的状态。所以任务从保存到再加载的过程就是一次上下文切换。

# 多线程编程的一些挑战

Java内存模型决定了线程间通过共享内存进行通信。在操作存储在对内存中的数据，如实例域、静态域和数组元素，处理不当时会因为内存可见性引发的并发安全问题。例如对实例变量i进行并发i++操作、对非线程安全的集合进行并发操作等。

常见的解决并发安全问题的方式是加同步锁，保证所有在临界区内的操作顺序执行。但是对锁的使用不当时容易引起死锁。

# 并发编程基础

## JMM（Java 内存模型）

Java线程之间的通信由JMM（Java内存模型）控制，JMM决定一个线程对共享变量的写入何时对另一个线程可见。从抽象的角度来看，JMM定义了线程和主内存之间的抽象关系：线程之间的共享变量存储在主内存（MainMemory）中，每个线程都有一个私有的本地内存（LocalMemory），本地内存中存储了该线程以读/写共享变量的副本。本地内存是JMM的一个抽象概念，并不真实存在。它涵盖了缓存、写缓冲区、寄存器以及其他的硬件和编译器优化。

# 注

1. 此处众说纷纭，也有说创建的是用户级线程。根据【参考7】，Linux 2.6的内核版本及以后的内核版本创建的是内核级线程。

# 参考

1. [进程的基本概念 - 进程线程模型 | Coursera](https://zh.coursera.org/lecture/os-pku/jin-cheng-de-ji-ben-gai-nian-47apH)
2. [线程的引入 - 进程线程模型 | Coursera](https://zh.coursera.org/lecture/os-pku/xian-cheng-de-yin-ru-SrUyN)
3. [一篇文章彻底弄懂进程和线程调度 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/371737531)
4. [How Java thread maps to OS thread? | by Unmesh Joshi | Medium](https://medium.com/@unmeshvjoshi/how-java-thread-maps-to-os-thread-e280a9fb2e06)
5. [深入虚拟机探索Thread start\_源码\_4ye\_InfoQ写作社区](https://xie.infoq.cn/article/656bc0720fdc0072acf1eb81b)
6. [java - How JVM thread scheduler control threads for multiprocessors? - Stack Overflow](https://stackoverflow.com/questions/41759261/how-jvm-thread-scheduler-control-threads-for-multiprocessors)
7. [Linux下调用pthread库创建的线程是属于用户级线程还是内核级线程？求大神指教? - 知乎 (zhihu.com)](https://www.zhihu.com/question/35128513)
8. [Green thread - Wikipedia](https://en.wikipedia.org/wiki/Green_thread)
9. [java - Green Threads vs Non Green Threads - Stack Overflow](https://stackoverflow.com/questions/5713142/green-threads-vs-non-green-threads)
10. <https://zh.wikipedia.org/wiki/%E5%A4%9A%E7%BA%BF%E7%A8%8B>
11. Java并发编程的艺术 (Java核心技术系列) 方腾飞,魏鹏,程晓明 著