06 线程什么时候开始真正执行？—线程的状态详解

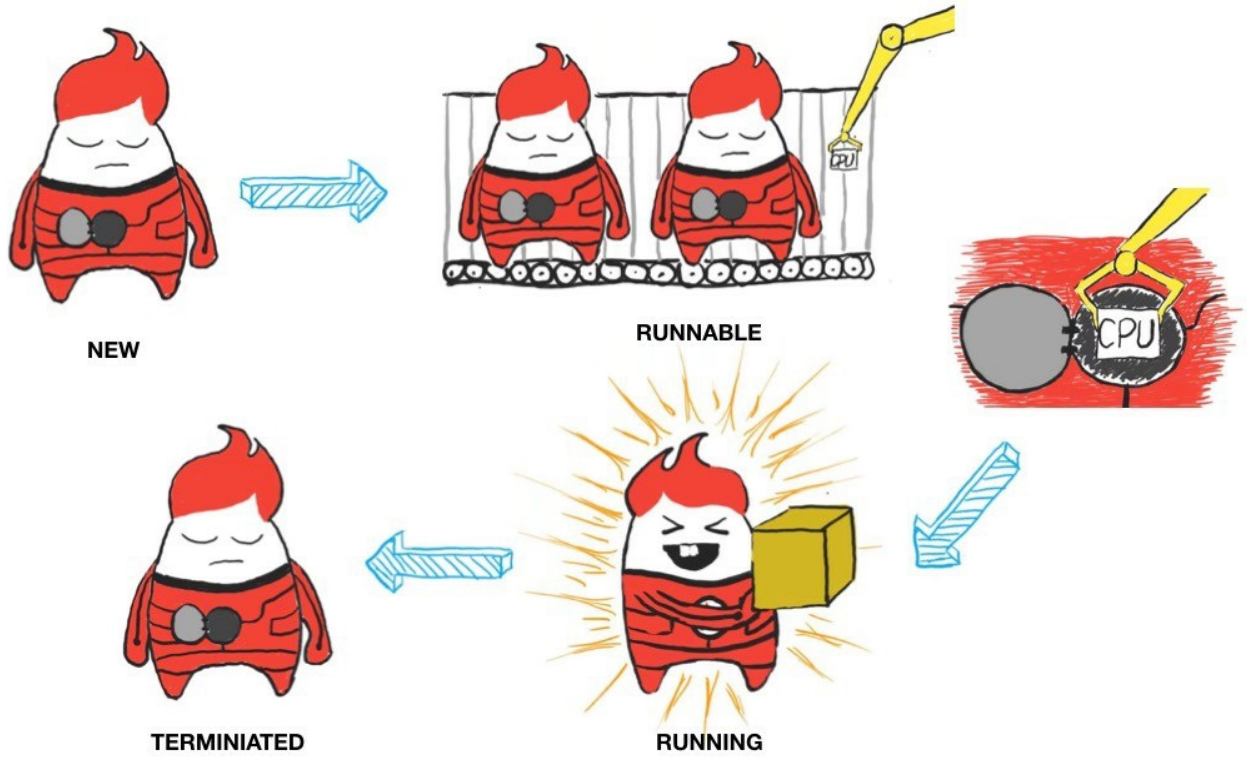
更新时间：2019-09-17 13:58:09

> 不安于小成，然后足以成大器；不诱于小利，然后可以立远功。
>
> ——方孝孺

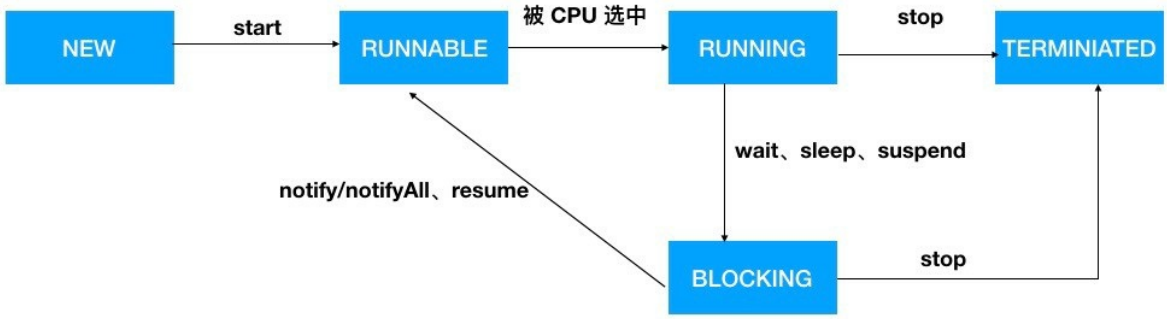
上篇文章结尾提到，在 Thread 的 start 方法中会判断 **threadStatus**。如果不为 0 会抛出异常，否则才会继续往下执行。这里引出了线程状态的概念。本篇文章我们会针对线程状态进行深入讲解。

在这里打个比方，可能并不是很恰当，但能帮助我们理解线程状态。我们可以把线程看做一个机器人，它要执行一项任务，任务内容就是你实现的 Runnable 对象的 run 方法逻辑。我们可以生产很多执行各种任务的机器人，但是由于 CPU 芯片数量有限，机器人要排队安装 CPU 后才能执行任务。

我们创建 Thread 对象，相当于创建机了一个器人。此时机器人状态为 NEW，此时机器人还不需要去执行任务，所以他并没在等待 CPU 的队列中。假如时机已到，我们调用 start 方法，让机器人开始排队等待安装 CPU。此时机器人状态为 RUNNABLE。当排队安装好 CPU 后，机器人就会立即开始执行任务，也就是触发 run 方法。此时机器人状态为 RUNNING。当任务执行完毕，机器人就要让出 CPU 了，并且标识自己为 TERMINATED 状态。此时意味着机器人的生命周期已经结束了。



Thread 的生命周期包含以上几种状态，此外还会有 BLOCKED 状态，状态间的转化也更为复杂。如下图所示：



讲到这里，其实上节提出的第一个问题：线程有几种状态，我们已经很清楚了。而另外一个问题：run 方法什么时候被调用，我们也有了答案。其实调用 start 方法后并不会立即执行 run 方法，而是等待 CPU 的选中本线程后才会被调用。如果由于某种原因，本线程在 CPU 的竞争中永远无法被选中，那么 start 之后 run 方法一直也不会被调用。

接下来我们分别看一看线程的几种状态，以及如何转换。

NEW 状态

当一个 Thread 对象刚刚被创建时，状态为 NEW。此状态仅仅表示 Thread 对象被创建出来了，但此时 Thread 对象和其它 Java 对象没有什么不同，仅仅是存在于内存之中。还拿机器人举例子，此时这个机器人和木头人没有任何区别。而当 Thread 对象调用 start 方法后，他的状态改变为 RUNNABLE。这意味着此机器人要苏醒过来了。此时 Thread 对象进入到 CPU 的竞争队列中。

RUNNABLE 状态

Thread 对象进入 RUNNABLE 状态只有一条路可以走，就是调用 start 方法。在调用 start 方法后，这个线程对象才在 JVM 中挂上号了，JVM 才知道有这么个“机器人”要搞事情。但是“机器人”此时还不能执行任务，为什么呢？因为他还没有灵魂（未被 CPU 选中）。是的，Thread 对象还在等待 CPU 的调用中，RUNNABLE 的含义就是 Thread 对象可以执行了，不过还未被执行。此时还在等待 CPU 的调度。

RUNNABLE 状态的线程只可能变迁为 RUNNING 和 TERMINATED 状态。当等到 CPU 调度时，状态变为 RUNNING。而在等待 CPU 调度期间，如果被意外终止，那么则会直接进入 TERMINATED 状态。

RUNNING 状态

RUNNABLE 状态的线程，一旦被 CPU 选中执行，他就会变为 RUNNING 状态。此时才会真正运行 run 方法逻辑。处于 RUNNING 状态的线程，其实也同时是 RUNNABLE 状态。

另外 BLOCKED 状态的线程通过 resume 或者 notify/notifyAll 先重新进入 RUNNABLE 状态，等待 CPU 调度后再进入 RUNNING 状态。
RUNNING 状态的线程可以变迁为如下状态：

BLOCKED 状态

RUNNING 的线程如果在执行过程中调用了 wait 或者 sleep 方法，就会进入 BLOCKED 状态

阻塞的 I/O 操作

进入到锁的阻塞队列

TERMINATED

run 方法正常执行结束

运行意外中止

调用 stop 方法（已经不推荐使用）

RUNNABLE

CPU 轮转，放弃该线程的执行

调用了 Thread 的 yield 方法，提示 CPU 可以让出自己的执行权。如果 CPU 对此响应，则会进入到 RUNNABLE 状态，而不是 TERMINATED。

BLOCKED 状态

只有 RUNNING 状态的线程才会进入 BLOCKED 状态，进入 BLOCKED 状态的原因在上面已经讲过。处于 BLOCKED 状态的线程，可以转换为如下状态：

RUNNABLE 状态

阻塞操作结束了，那么线程将切换回 RUNNABLE 状态。注意不是 RUNNING 状态，此时线程需要等待 CPU 的再一次选中。

阻塞过程中被打断了，由于其它线程调用了 interrupt 方法，也会回到 RUNNABLE 状态。

线程休眠的时间结束了，也会回到 RUNNABLE 状态。

由于 wait 操作进入 BLOCKED 状态的线程，其他线程发出了 notify 或 notifyAll ，则会唤醒它，回到 RUNNABLE 状态。

由于等待锁而被 BLOCKED 的线程。一旦获取了锁，那么便会回到 RUNNABLE 状态。

TERMINATED 状态

线程 BLOCKED 状态时，有可能由于调用了 stop 或者意外终止，而直接进入了 TERMINATED 状态。

TERMINATED 状态

TERMINATED 状态意味着线程的生命周期已经走完。这是线程的终止状态。此状态的线程不会再转化为其它任何状态。

处于 RUNNING 或者 BLOCKED 状态的线程都有可能变为 TERMINATED 状态，但原因是类似的，如下：

- 线程运行正常结束
- 程序运行异常终止
- JVM 意外终止

守护线程

本节中，再介绍一下守护线程的相关知识。为什么选择在这里介绍守护线程？因为守护线程进入 TERMINATED 状态有个特殊的方式。

当 JVM 中没有任何一个非守护线程时，所有的守护线程都会进入到 TERMINATED 状态，JVM 退出。

守护线程是做什么用的呢？不知道你是否还记得餐厅的例子，其实清洁员就相当于守护线程。他一直在默默地做打扫卫生的工作。这个工作相对独立，他也不需要和别的角色有什么交互。而当其他所有人都不工作了，他也没有工作的必要了，因为不会有新的垃圾产生。那么他也可以下班，餐厅也就关门了。

在 Java 中，当没有非守护线程存在时，JVM 就会结束自己的生命周期。而守护进程也会自动退出。守护线程一般用于执行独立的后台业务。比如 JAVA 的垃圾清理就是由守护线程执行。而所有非守护线程都退出了，也没有垃圾回收的需要了，所以守护线程就随着 JVM 关闭一起关闭了。

当你有些工作属于后台工作，并且你希望这个线程自己不会终结，而是随着 JVM 退出时自动关闭，那么就可以选择使用守护线程。

要实现守护线程只能手动设置，在线程 start 前调用 setDaemon 方法。Thread 没有直接创建守护进程的方式，非守护线程创建的子线程都是非守护线程。

总结

本节讲解了线程的状态及转化关系。这是多线程开发的基础知识，是后面学习的基石。这部分知识其实难度不大，也比较好理解，并且后面讲解线程 API 时还会涉及。最后引出了守护线程。其实守护线程在实际开发中使用比较少，不过在了解了其使用场景后，可以根据需要选择使用。

下一节我们将对 Thread 提供的 API 做深入讲解。Thread 部分 API 会影响到 Thread 状态，因此本节是下节学习的基础，请务必掌握。

}

05 看若兄弟，实如父子—Thread和Runnable详解

07 深入Thread类—线程API精讲