链滴

从生命周期的角度看线程和进程之间的异同

作者：vcjmhg
原文链接：https://ld246.com/article/1609491916240
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20200926.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="概述">概述</h2>
进程与线程想必都不陌生，两者有诸多相同点，甚至可以这样说，线程就是“轻量级的进程”。
且两者基本的五个状态也几乎一样，但进程和线程在状态切换时的触发条件却有诸多不同，因而本文
“生命周期”的角度去谈一谈两者之间的异同。
<h2 id="联系">联系</h2>
就从状态本身而言，两者的状态类别和对应含义几乎是完全一致的分别为：

初始状态：刚被创建的状态
可运行状态（就绪状态）：可以被分配 CPU 的状态
运行状态：获取到 CPU 正在运行的状态
休眠状态（阻塞状态）：等待某个事件时，会被转换到该状态
终止状态（结束状态）：执行完成或者遇到异常情况时，会进入该状态。

注意：上边的五种状态时比较通用的状态，但在不同的操作系统中，进程的
态也会有扩充和精简。同样的在不同的编程语言中，对线程的生命状态也会有精简和合并；比如 java
语言中把可运行状态和运行状态进行了合并，但对休眠
态进行了扩充，分成了阻塞状态、无时限状态、<strong
有时限状态三种状态。
<h2 id="区别">区别</h2>
虽然说，进程和线程的生命周期（或者状态）有诸多相似点，但它们两者在不同状态间切换的条
是不同的。
比如对进程来说五种状态的转换图如下所示：
<img src="https://ld246.com/images/img-loading.svg" alt="进程的状态" data-src="https:
/b3logfile.com/file/2021/01/solofetchupload7699678680693170515-cc672075.png?imageVie
2/2/interlace/1/format/jpg">
就绪状态 --> 运行状态：当处于就绪状态的进程被调度后，获得处理机
源，此时进程就由就绪状态转换成运行状态。
运行状态--> 阻塞状态：处于运行状态的进程在时间片用完之后，不得不让出处理机，此
进程就有运行状态转成阻塞状态。
阻塞状态--> 就绪状态：当进程等待的事件到来时，中断处理程序必须把相应进程的状态
阻塞状态转换成就绪状态。
而对线程来说状态状态切换（以 java 的线程为例）如下所示：
<img src="https://ld246.com/images/img-loading.svg" alt="image-20210101165455636"
data-src="https://b3logfile.com/file/2021/01/solofetchupload7266664595155850851-5a5ce0
b.png?imageView2/2/interlace/1/format/jpg">
整个状态的转换过程如下：
java 线程中状态转换：

RUNNABLE 与 BLOCKED 的状态转换：
只有线程在等待 synchronized 的隐式锁时，synchronized 修饰的方法、代码块同一时刻只能允许
个线程执行，其他线程只能等待，这种情况下，等待的线程就会从 RUNNABLE 转换到 BLOCKED 状
。而当等待获得到 synchronized 隐式锁时，就又会从 BLOCKED 转换到 RUNNABLE 状态。
并且 java 层面上不关心操作系统进程的调度状态，因为在 JVM 看来，等待 CPU 使用权和等待 IO
有区别，都是在等待某个资源，因此都被归为 RUNNABLE 状态。
RUNNABLE 与 WAITING 的状态转换

获取到 synchronized 的隐式锁的进程，调用无参数的 Object.wait()方法
调用无参数的 Thread.join()方法
调用 LOCKSupport.park()方法。LockSupport.unpark(Thread thread)课唤醒目标线程，目标
程的状态又会从 WAITNG 状态转换成 RUNNABLE 状态

RUNNABLE 与 TIMED_WEAITING 的状态转换
1.调用带超时参数的 Thread.sleep(long millis)方法
2.获得 synchronized 隐式锁的线程，调用带超时参数的 Object.wait(long timeout)方法
3. 调用带超时参数的 LockSupport.parkNanos(Object blocker,long dealing)方法
3. 调用带超时参数锁的 Thread.join(long millis)方法
3. 调用带超时参数的 LockSupport.parkUnitl(long deadline)方法
从 NEW 到 RUNNABLE 状态
线程在创建成功后，调用其对应的 start 方法就会从 RUNNABLE 状态转换成
4.从 RUNNABLE 到 TERMINATED 状态
通过调用 Thread.interrupt()方法，也可以调用 stop()方法，但当前方法被废止了。

<h2 id="总结">总结</h2>
本文主要从生命周期的角度总结了线程和进程之间的异同点，当然在其他方面两者还有诸多不同比如进程是资源调度的基本单位，它拥有属于自己的系统资源，而线程本身不有系统资源，多个线程之间共享进程的资源。关于这些不同点，本文不在详述。