【第09话：线程的生命周期也是面试中很常见的】

Hello 小伙伴们，我是张佳明，这结课给大家讲解一下一个高频经典面试题：“线程的生命周期”

线程的生命周期就是线程从创建到线程销毁的过程。面试官通过问线程生命周期可以考察面试者是否知道线程的几种状态，以及线程常用方法。

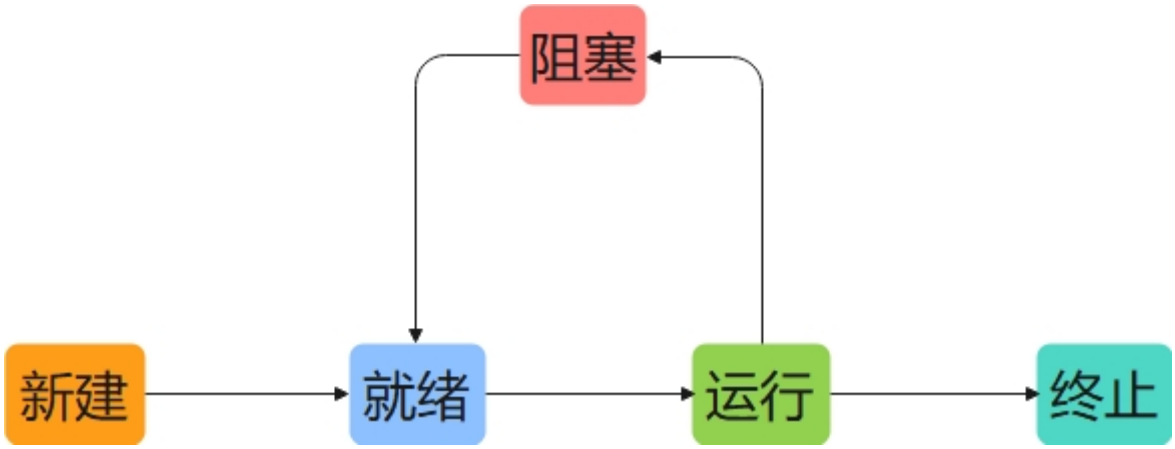
对于这样的问题，**新建状态、就绪状态、运行状态、阻塞状态、死亡状态**我们脑中必须把这五个状态记忆清楚。同时他们在什么情况下切换，如何进行切换状态也要弄清楚。

在1万的版本中需要说清楚这几个状态是什么，以及如何切换。

在1.5万的版本中需要把线程常用方法和过时方式都说清楚。

在2万的版本中需要说出所有线程切换的方法。以及到底如何切换的。

线程生命周期（月薪1万的回答示范）



线程生命周期就是从线程创建到线程终止过程中线程的几种状态切换。

线程状态分为五种，分别是新建状态，就绪状态，运行状态，阻塞状态和终止状态。

当实例化线程时，线程处于新建状态。此时线程还没有运行。

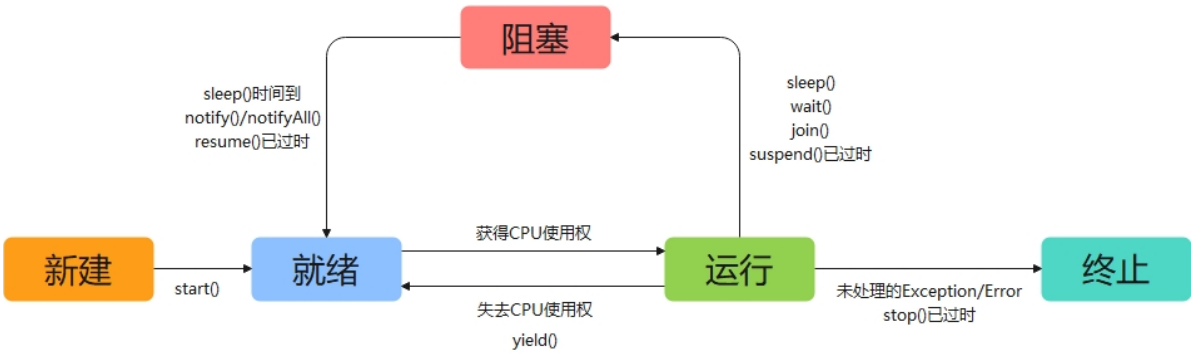
当线程启动后，线程处于就绪状态，就绪状态属于临时性状态，只要竞争到CPU资源就可以自动切换为运行状态。

处于运行状态的线程可能切换为阻塞状态。解除阻塞后切换为就绪状态。

如果线程结束或异常，线程切换为终止状态。

以上就是线程的生命周期，月薪1万的回答示范

线程生命周期（月薪1.5万回答示范）



线程生命周期就是从线程创建到线程终止过程中线程的几种状态切换。

线程状态分为五种，分别是新建状态，就绪状态，运行状态，阻塞状态和终止状态。

当实例化Thread对象后, 线程就处于新建状态. 这时线程并没有执行。

当线程对象调用start()方法后，线程会从新建状态切换为就绪状态。

就绪状态属于一种临时状态。处于就绪状态的线程会去抢占CPU，只要抢占成功就会切换到运行状态。

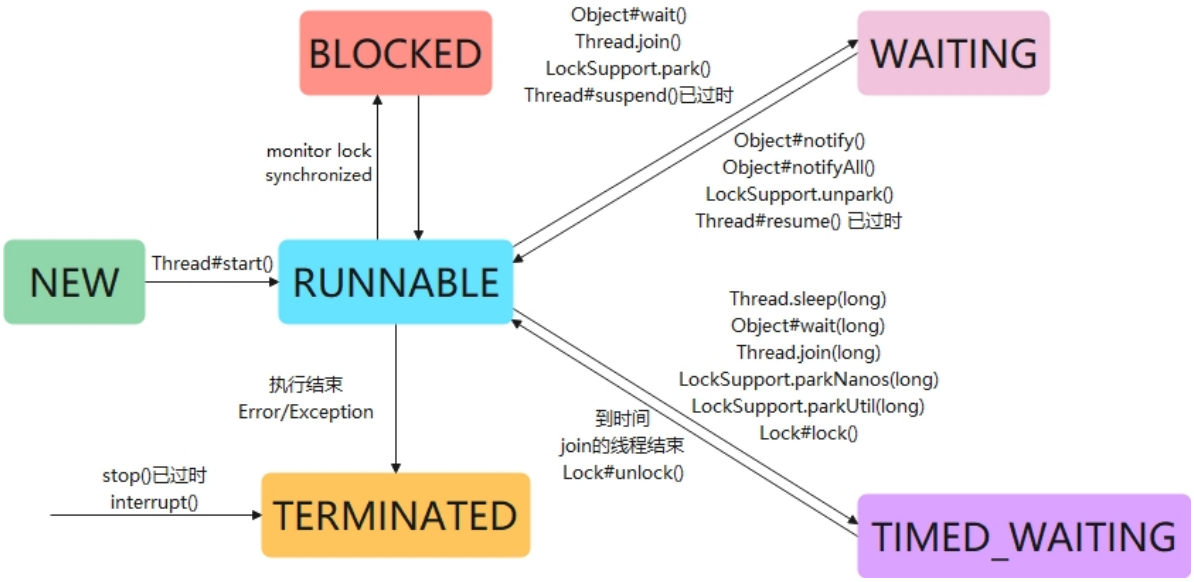
处于运行状态的线程，开始执行线程的功能。例如Thread的run()方法，Callable的call()方法。在这个状态的线程分为多种可发生情况。

1. 如果失去了CPU使用权，或调用yield()方法会变为就绪状态。
2. 如果碰到sleep() / wait() / join()等方法会让线程切换为阻塞状态。
3. 如果线程功能成功执行完成，或出现问题或被停止会切换为终止状态。终止状态也表示线程执行完成了。

而阻塞状态的线程停止执行。让出CPU资源。在这个状态的线程也分为多种情况

1. 如果是因为sleep()变为阻塞，则休眠时间结束自动切换为就绪状态
2. 如果是因为wait()变为阻塞状态，需要调用notify()或notifyAll()手动切换为就绪状态
3. 如果因为join()变为阻塞状态，等到join线程执行完成，自动切换为就绪状态
4. （已过时）如果是因为suspend()暂停的线程，需要通过resume()激活线程

线程生命周期（月薪2万回答示范）



线程生命周期就是从线程创建到线程终止过程中通过不同的方法控制线程的几种状态切换。

在Thread类的内部State枚举中明确支持了线程的六种状态，分别是NEW新建状态、RUNNABLE可运行状态、BLOCKED阻塞状态、WAITING无限时等待状态、TIMED_WAITING有限时等待状态、TERMINATED终止状态。

当线程实例化后，线程处于NEW新建状态，这时线程还没有没启动。此时线程只是代码层面被创建，在JVM中开辟的堆空间。但是对于操作系统来说此时还没有创建真正的线程。

当处于NEW状态的线程调用start()方法后，线程切换为RUUNABLE状态。对于JVM来说，此时的线程就属于运行状态。对于操作系统来说，也创建了真实的线程。线程会去竞争CPU资源，只要竞争到了CPU资源，线程就会执行。即运行线程的run()方法或call()方法。

处于RUNNABLE状态的线程可能发生四种状态切换。

当线程线程遇到Monitor Lock监视器锁，即操作系统中的管程时，会切换为BLOCKED阻塞状态。在Java中synchronized内置锁就是基于Monitor机制实现的，依赖底层操作系统的互斥原语Mutex（互斥量）。所以在Java代码中，当线程遇到synchronized，其他线程占用了CPU，当前线程就处于阻塞状态。

如果处于RUNNABLE状态的线程碰到Object的无参wait()方法、Thread的静态无参join()方法、JUC中LockSupport的静态无参park()方法和Thread的实例方法suspend()都会让线程切换为WAITING状态，即无限时等待状态。处于WAITING状态的线程会让出CPU资源，除非触发对应的激活方法，否则会一直WAITING下去。

处于WAITING状态的线程需要被手动唤醒，到使用Object的notify()/notifyAll()可唤醒wait()的线程，当使用LockSupport.unpark()可唤醒LockSupport.park()的线程，使用Thread的resume()可以唤醒suspend()的线程。

suspend()和resume()这一对操作方法在目前已经弃用了。因为它本质上容易死锁。 如果目标线程在挂起时保护关键系统资源的监视器上持有锁，则在目标线程恢复之前，没有线程可以访问该资源。 如果将恢复目标线程的线程在调用resume之前尝试锁定此监视器，则会导致死锁。

如果处于RUNNABLE状态的线程在调用Thread中静态有参方法sleep、Object中实例有参方法wait()、Thread.join()方法和LockSupport的静态有参parkNanos(long)或parkUtil(long)方法、Lock的实例方法lock，线程会切换为有限时等待TIMED_WATING状态。处于这个状态的线程大部分都会在指定时间后自动切换为RUNNABLE状态。只是join是等待join进来线程执行结束后当前线程恢复，lock()的线程等待unlock后恢复。

最后一种状态切换是线程执行结束或执行过程出现错误或异常线程会切换为TERMINATED状态。

无论什么状态，只要调用stop()或interrupt()都会切换为TERMINATED终止状态。

stop()是已弃用方法，会真的杀死线程，不给线程喘息的机会， 如果线程持有 ReentrantLock 锁，被 stop() 的线程并不会自动调用 ReentrantLock 的 unlock() 去释放锁，那其他线程就再也没机会获得 ReentrantLock 锁，这实在是太危险了。