关于线程的讲解(出自Java原著) PDF转换可能丢失图片或格式，建议阅读原文

https://www.100test.com/kao_ti2020/145/2021_2022__E5_85_B3_E4_BA_8E_E7_BA_BF_E7_c104_145517.htm Thread Scheduling In Java technology,threads are usually preemptive,but not necessarily Time-sliced(the process of giving each thread an equal amount of CPU time).It is common mistake to believe that "preemptive" is a fancy word for "does time-slicing". For the runtime on a Solaris Operating Environment platform,Java technology does not preempt threads of the same priority.However,the runtime on Microsoft Windows platforms uses time-slicing,so it preempts threads of the same priority and even threads of higher priority.Preemption is not guaranteed.however,most JVM implementations result in behavior that appears to be strictly preemptive.Across JVM implementations,there is no absolute guarantee of preemption or time-slicing.The only guarantees lie in the coder’s use of wait and sleep. The model of a preemptive scheduler is that many threads might be runnable,but only one thread is actually running.This thread continues to run until it ceases to be runnable or another thread of higher priority becomes runnable.In the latter case,the lower priority thread is preempted by the thread of higher priority,which gets a chance to run instead. A thread might cease to runnable (that is,because blocked) for a variety of reasons.The thread’s code can execute a Thread.sleep() call,deliberately asking the thread to pause for a fixed period of time.The thread might have to wait to access a resource and cannot continue until that resource

become available. All thread that are runnable are kept in pools according to priority.When a blocked thread becomes runnable,it is placed back into the appropriate runnable pool.Threads from the highest priority nonempty pool are given CPU time. The last sentence is worded loosed because: (1) In most JVM implementations,priorities seem to work in a preemptive manner,although there is no guarantee that priorities have any meaning at all. (2) Microsoft Window ’ s values affect thread behavior so that it is possible that a Java Priority 4 thread might be running,in spite of the fact that a runnable Java Priority 5 thread is waiting for the CPU. In reality,many JVMs implement pool as queues,but this is not guaranteed hehavior. 线程调度（试翻译，欢迎指正） 在java技术中，线程通常是抢占式的而不需要时间片分配进程（分配给每个线程相等的cpu时间的进程）。一个经常犯的错误是认为“ 抢占 ”就是“ 分配时间片 ”。在Solaris平台上的运行环境中，相同优先级的线程不能相互抢占对方的cpu时间。但是，在使用时间片的windows平台运行环境中，可以抢占相同甚至更高优先级的线程的cpu时间。抢占并不是绝对的，可是大多数的JVM的实现结果在行为上表现出了严格的抢占。纵观JVM的实现，并没有绝对的抢占或是时间片，而是依赖于编码者对wait和sleep这两个方法的使用。 抢占式调度模型就是许多线程属于可以运行状态（等待状态），但实际上只有一个线程在运行。该线程一直运行到它终止进入可运行状态（等待状态）或是另一个具有更高优先级的线程变成可运行状态。在后一种情况下，底优先级的线程被高优先级的线程抢占，高优先级的线程获得运行的机会

。 线程可以因为各种各样的原因终止并进入可运行状态（因为堵塞）。例如，线程的代码可以在适当时候执行Thread.sleep()方法，故意让线程中止；线程可能为了访问资源而不得不等待直到该资源可用为止。 所有可运行的线程根据优先级保持在不同的池中。一旦被堵塞的线程进入可运行状态，它将会被放回适当的可运行池中。非空最高优先级的池中的线程将获得cpu时间。 最后一个句子是不精确的，因为：（1）在大多数的JVM实现中，虽然不能保证说优先级有任何意义，但优先级看起来象是用抢占方式工作。（2）微软windows的评价影响线程的行为，以至尽管一个处于可运行状态的优先级为5的java线程正在等待cpu时间，但是一个优先级为4的java线程却可能正在运行。 实际上，许多JVM用队列来实现池，但没有保证行为。