第七章 取消与关闭

Java没有提供任何机制来安全的终止线程。但是提供了中断，这是一种合作机制，能够使一个线程终止另一个线程的当前工作。

#### 一．任务取消

1. 取消原因

（1）用户请求取消

（2）有时间限制的取消

（3）应用程序事件

（4）错误

（5）关闭

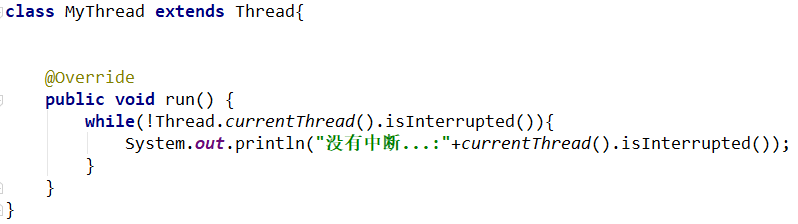
中断

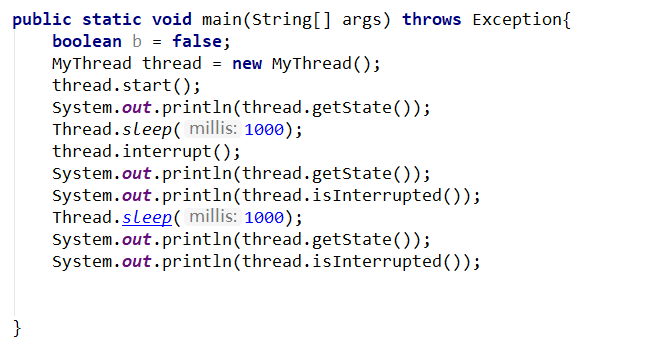
1. 概念

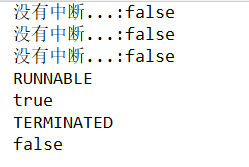
一个线程调用interrupt方法请求另一个线程停止任务。调用interrupt并不意味着立即停止目标线程正在进行的工作，而只是传递了请求中断的消息，然后由线程在下一个合适的时刻中断自己。

2. Thread的中断方法。

可以看到当我们调用了interrupt方法后线程依然是RUNNING状态，但是isInterrupt却是true，说明了线程没有立即中断，过一段时间发现线程是TERMINATED状态。







3. 通常中断是实现取消的最合理方式。

阻塞库方法（Thread.sleep、Object.wait等）都会检查线程何时中断，并且在发现中断时提前返回。

阻塞库方法响应中断时的操作：

（1）清楚中断状态（isTerrupted置为false）

（2）抛出InterruptedException异常

4. 中断策略

（1）不要随意中断某个线程，除非你知道中断对该线程的含义。

（2）通过推迟中断请求的处理，开发人员能制定更灵活的中断策略，从而使应用程序在响应性和健壮性之间实现合理的平衡。

5. 响应中断

（1）传递异常

只有实现了线程中断策略的代码才可以屏蔽中断请求。在常规的任务和库代码中都不应该屏蔽中断请求。

（2）恢复中断状态

对于不支持取消但仍可以调用中断阻塞方法的操作，它们必须在循环中调用这些方法，并在发现中断后重新尝试。这种情况下，它们应该在本地保存中断状态，并且在返回前恢复状态而不是在捕获InterruptedException时恢复状态。

**注：7.1.4未懂。**

6. 通过Future来取消任务

当Future.get抛出InterruptedException或TimeoutException时，如果你知道不再需要结果，那么就可以调用Future.cancel来取消任务。

处理不可中断的阻塞

1. 线程阻塞的原因

（1）Java.io包中的同步Socket I/O

不会响应中断，可以关闭底层套接字来使得被阻塞的线程抛出SocketException

（2）Java.io包中的同步I/O

当中断一个正在InterruptibleChannel上等待的线程时，将抛出ClosedByInterruptException并关闭链路。

（3）Selector的异步I/O

调用wakeup或者close方法会抛出ClosedSelectorException

（4）获取某个锁

无法响应中断

**二. 停止基于线程的服务**

1. 服务应该提供生命周期方法来关闭它自己以及它拥有的线程。

2. 对于持有线程的服务，只要服务的存在时间大于创建线程的方法的存在时间，那么就应该提供生命周期方法。

3. 毒丸对象

毒丸对象：当得到这个对象时立即停止。

只有在生产者和消费者的数量都已知的情况下才可以使用毒丸对象。

**三. 处理非正常的线程终止**

1. 尽量为未捕获异常的线程指定同一个异常处理器，并且该处理器至少会将异常信息记录到日志中。

**四. JVM关闭**

1. 正常关闭

（1）所有非守护线程结束

（2）调用了System.exit()

（3）…

2. 关闭钩子

通过Runtime.addShutdownHook注册但尚未开始的线程。JVM并不能保证钩子的调用顺序。

（1）关闭钩子应该是线程安全的

（2）可以用于清理工作，例如删除临时文件

（3）

3. 守护线程

新线程将继承创建它的线程的守护状态。

最好用于运行时的垃圾数据的清理。

4. 终结器

避免使用终结器。