链滴

协作，才能更好的中断线程

作者：9526xu
原文链接：https://ld246.com/article/1570586342476
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)

聊起中断，大家可能最熟悉的例子就是线程休眠。下面就是一个线程休眠的 demo，在这个例子中，我们调用 sleep 方法，该方法将会抛出一个需要捕获的中断异常，这里捕获该异常并直接返回。

```
for (int i = 0; i < somethings.size(); i++) {
    // 休眠 4 s
    try {
        Thread.sleep(4000);
    } catch (InterruptedException e) {
        // 抛出中断异常
        return;
    }
    // 输出
    System.out.println(somethings.get(i));
}
```

除了 InterruptedException 中断异常，另外还有三个中断相关的方法，三个方法都与线程相关。

- thread#interrupt
- Thread#interrupted
- thread#isInterrupted

interrupt 方法用于中断线程，但是并不是说该方法就能**直接使线程停止**。

下面使用 interrupt 中断线程，这里我们期望中断直接停止子线程输出。但是当主线程调用子线程 interrupt 方法，子线程并却没有被终止，还在继续打印数字。

```
Runnable interruptedTask=new Runnable() {
    @Override
    public void run() {
        for (int i = 0; i <Integer.MAX_VALUE ; i++) {
            System.out.println(i);
```

```
            }
        }
    };

    Thread interruptThread=new Thread(interruptedTask);
    interruptThread.start();
    // 休眠 5 s，
    TimeUnit.SECONDS.sleep(2);
    // 中断当前线程
    interruptThread.interrupt();
    // 再次休眠，观察子线程
    TimeUnit.SECONDS.sleep(2);
```

引用 Java 官方对于中断的解释：

An interrupt is an indication to a thread that it should stop what it is doing and do something lse. It's up to the programmer to decide exactly how a thread responds to an interrupt, but it i very common for the thread to terminate

中断仅仅只是表明这个线程可以停止，但是线程是否停止完全取决于线程自己。只有线程相互协作，能更好的停止线程。

每个线程都包含一个内部标志，用来表示中断状态。调用线程的 interrupt 方法将会设置该状态位，于 Thread#sleep 等阻塞方法，将会抛出 InterruptedException ，并清除中断标志。

我们可以使用 thread#isInterrupted 或 Thread#interrupted 检查中断状态。但是需要注意，两个法存在一些区别，Thread#interrupted 为静态类方法，该方法检测到中断之后就会清除中断标志。

上面的方法我们只要加上中断状态判断就也可以停止线程。

```
    Runnable interruptedTask=new Runnable() {
        @Override
        public void run() {
            for (int i = 0; i <Integer.MAX_VALUE ; i++) {
                // 一旦检测到中断标志，停止线程
                if(Thread.interrupted()){
                    System.out.println("interrupted!!!!");
                    break;
                }
                System.out.println(i);
            }
        }
    };

    Thread interruptThread=new Thread(interruptedTask);
    interruptThread.start();
    // 休眠 5 s，
    TimeUnit.SECONDS.sleep(2);
    // 中断当前线程
    interruptThread.interrupt();
    // 再次休眠，观察子线程
    TimeUnit.SECONDS.sleep(2);
}
```

中断最佳实践

不要随意『吃掉』中断异常

由于中断异常是一个 checked exception，我们不得不需要处理该异常。如果我们可以保证该异常不响应用，我们可以直接『吃掉』这个异常。其他情况下我们需要正确处理这个异常。

最简单的做法就是不处理该异常，直接向上抛出中断异常，让上层调用者决定如何处理。

但是有些情况下，却不适合上面的做法，这种情况下我们需要在 catch 中处理中断。如果实在不知道何处理，那就是记录该异常，并使用日志方式输出。

中断不会停止阻塞 IO

上面我们说到，对于一些阻塞方法如 Thread#sleep ，将会抛出中断异常。但是对于 Socket 等阻塞 O 调用，并不会抛出这个异常。也就是说中断并不会停止阻塞 IO 的调用。

这是因为当调用 Thread#sleep 等阻塞方法时，Java 线程状态将会从 RUNNABLE 转变为 TIMED_WITING 或 WATTING。而当线程阻塞在 IO 读取时，Java 线程实际状态却还是 RUNNABLE。如果你这个线程状态还有疑惑，可以阅读下这篇文章 面试官:都说阻塞 I/O 模型将会使线程休眠，为什么 Java 线程状态却是 RUNNABLE? ，深入理解一下线程状态。

原文链接：协作，才能更好的中断线程