首先我们需要知道，`ClosedByInterruptException` 一般都是在一个线程(比如 `IoProcessor` 进行 `channel.accept()` 、`channel.read()` 、`channel.write()` 操作的时候或之前，有一个线程调用了 `IoProcessor.interupt()` 方法中断了 `IoProcessor` 线程，那么此时 `channel.accept()` 、`channel.read()` 、`channel.write()` 都会抛出 `ClosedByInterruptException` .

这里以 Mina 为例，说明如果没有正确处理线程中断异常的话，会发生什么情况.比如有如下一段代码，在 `IoHandler` 的 `sessionCreated` 方法中中断当前的线程(IO 线程):

```
1.    @Override
2.    public void sessionCreated(IoSession session) throws Exception {
3.        Thread t = Thread.currentThread();
4.        t.interrupt();    // 中断 IO 线程
5.        System.out.println("中断: " + t);
6.
7.        System.out.println("服务器端与客户端创建连接...");
8.    }
```

如果此时读取客户端发送过来的数据会怎么样? 很抱歉，Mina 会抛出之前所说的 `ClosedByInterruptException` 异常，如下:

```
java.nio.channels.ClosedByInterruptException
    at java.nio.channels.spi.AbstractInterruptibleChannel.end(AbstractInterruptibleChannel.java:202)
    at sun.nio.ch.SocketChannelImpl.read(SocketChannelImpl.java:406)
    at org.apache.mina.transport.socket.nio.NioProcessor.read(NioProcessor.java:292)
    at org.apache.mina.transport.socket.nio.NioProcessor.read(NioProcessor.java:45)
    at org.apache.mina.core.polling.AbstractPollingIoProcessor.read(AbstractPollingIoProcessor.java:743)
    at org.apache.mina.core.polling.AbstractPollingIoProcessor.process(AbstractPollingIoProcessor.java:713)
    at org.apache.mina.core.polling.AbstractPollingIoProcessor.process(AbstractPollingIoProcessor.java:699)
    at org.apache.mina.core.polling.AbstractPollingIoProcessor.access$600(AbstractPollingIoProcessor.java:67)
    at org.apache.mina.core.polling.AbstractPollingIoProcessor$Processor.run(AbstractPollingIoProcessor.java:1236)
    at org.apache.mina.util.NamePreservingRunnable.run(NamePreservingRunnable.java:65) <2 internal calls>
    at java.lang.Thread.run(Thread.java:745)
```

很明显这发生在 `channel` 在进行读操作的时候发生的.而 Mina 并没有对 IO 线程中断的处理，所以会导致此问题的发生.其实线程这个 `IoProcessor` 线程基本处于僵死的状态,无法再继续处理读写操作.

如何修复此问题，可以修改 `AbstractPollingIoProcessor.Processor#run()` 方法，清除线程中断的标记:

```
1.    private class Processor implements Runnable {
2.        public void run() {
3.            for (;;) {
4.                try {
5.                    long t0 = System.currentTimeMillis();
6.                    int selected = select(SELECT_TIMEOUT);
7.                    long t1 = System.currentTimeMillis();
8.                    long delta = (t1 - t0);
9.
10.                   // 如果用户中断了线程，那么提醒一下，并清除中断标识
11.                   if (Thread.interrupted()) {
12.                       System.err.println("检测到线程中断,可能是由于用户在 IoHandler 的方法中中断了线程而导致.这里将清除线程中断的标记.");
13.                   }
14.               }
15.           }
16.       }
17.   }
```

但是这并不能彻底解决此问题.比如，假设在 `messageReceived()` 方法中中断了线程，然后进行写操作，那么仍然是可能出现 `ClosedByInterruptException` .

由此呢，可以这么总结一下，首先 NIO 框架应该要处理好 IO 线程中断的问题，其次是用户的编码问题，是否有必要要中断线程?