JAVA异常谜题41:域和流 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_JAVA_E5_BC_82_E5_B8_B8_c97_138509.htm 下面的方法将一个文件拷贝到另一个文件，并且被设计为要关闭它所创建的每一个流，即使它碰到I/O错误也要如此。遗憾的是，它并非总是能够做到这一点。为什么不能呢，你如何才能订正它呢？ static void copy(String src, String dest) throws IOException { InputStream in = null. OutputStream out = null. try { in = new FileInputStream(src). out = new FileOutputStream(dest). byte[] buf = new byte[1024]. int n. while ((n = in.read(buf)) > 0) out.write(buf, 0, n). } finally { if (in != null) in.close(). if (out != null) out.close(). } } 这个程序看起来已经面面俱到了。其流域（in和out）被初始化为null，并且新的流一旦被创建，它们马上就被设置为这些流域的新值。对于这些域所引用的流，如果不为空，则finally语句块会将其关闭。即便在拷贝操作引发了一个IOException的情况下，finally语句块也会在方法返回之前执行。出什么错了呢？ 问题在finally语句块自身中。close方法也可能会抛出IOException异常。如果这正好发生在in.close被调用之时，那么这个异常就会阻止out.close被调用，从而使输出流仍保持在开放状态。请注意，该程序违反了谜题36的建议：对close的调用可能会导致finally语句块意外结束。遗憾的是，编译器并不能帮助你发现此问题，因为close方法抛出的异常与read和write抛出的异常类型相同，而其外围方法（copy）声明将传播该异常。 解决方式是将每一个close都包装在一个嵌套的try语句块中。下面的finally语句块的版本可以保证在两个流上都会调用close：

} finally { if (in != null) { try { in.close(). } catch (IOException ex) { // There is nothing we can do if close fails } if (out != null) try { out.close(). } catch (IOException ex) { // There is nothing we can do if close fails } } } 从5.0版本开始，你可以对代码进行重构，以利用Closeable接口： } finally { closeIgnoringException(in). closeIgnoringEcception(out). } private static void closeIgnoringException(Closeable c) { if (c != null) { try { c.close(). } catch (IOException ex) { 