java语言中异常错误恢复处理的异常类型计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/646/2021_2022_java_E8_AF_AD_E8_A8_80_c97_646048.htm Java语言要求java程序中（无论是谁写的代码）所有抛出（throw）的异常都必须是从Throwable派生而来。当然，实际的Java编程中，由于JDK平台已经为我们设计好了非常丰富和完整的异常对象分类模型。因此，java程序员一般是不需要再重新定义自己的异常对象。而且即便是需要扩展自定义的异常对象，也往往会从Exception派生而来。所以，对于java程序员而言，它一般只需要在它的顶级函数中catch(Exception ex)就可以捕获出所有的异常对象。 所有异常对象的根基类是 Throwable，Throwable从Object直接继承而来（这是java系统所强制要求的），并且它实现了 Serializable接口（这为所有的异常对象都能够轻松跨越Java组件系统做好了最充分的物质准备）。从Throwable直接派生出的异常类有Exception和Error。Exception是java程序员所最熟悉的，它一般代表了真正实际意义上的异常对象的根基类。也即是说， Exception 和从它派生而来的所有异常都是应用程序能够catch到的，并且可以进行异常错误恢复处理的异常类型。而Error则表示Java系统中出现了一个非常严重的异常错误，并且这个错误可能是应用程序所不能恢复的，例如LinkageError ，或 ThreadDeath 等。首先还是看一个例子吧！代码如下： import java.io.*. public class Trans { public static void main(String[] args) { try { BufferedReader rd=null. Writer wr=null. try { File srcFile = new File((args[0])). File dstFile = new File((args[1])). rd = new

BufferedReader(new InputStreamReader(new FileInputStream(srcFile), args[2])). wr = new OutputStreamWriter(new FileOutputStream(dstFile), args[3]). // 注意下面这条语句，它有什么问题吗？ if (rd == null || wr == null) throw new Exception("error! test!"). while(true) { String sLine = rd.readLine(). if(sLine == null) break. wr.write(sLine). wr.write("\r\n"). } } finally { wr.flush(). wr.close(). rd.close(). } } catch(IOException ex) { ex.printStackTrace(). } } } 熟悉 java 语言的程序员朋友们，你们认为上面的程序有什么问题吗？编译能通过吗？如果不能，那么原因又是为何呢？好了，有了自己的分析和预期之后，不妨亲自动手编译一下上面的小程序，呵呵！结果确实如您所料？是的，的确是编译时报错了，错误信息如下： E:\Trans.java:20: unreported exception java.lang.Exception. must be caught or declared to be thrown if (rd == null || wr == null) throw new Exception("error! test!"). 1 error 上面这种编译错误信息，相信 Java 程序员肯定见过（可能还是屡见不鲜！）！相信老练一些的 Java 程序员一定非常清楚上述编译出错的原因。那就是如错误信息中（“ must be caught ”）描述的那样， 在 Java 的异常处理模型中，要求所有被抛出的异常都必须要有对应的“ 异常处理模块 ”。也即是说，如果你在程序中 throw 出一个异常，那么在你的程序中（函数中）就必须要 catch 这个异常（处理这个异常）。例如上面的例子中，你在第 20 行代码处，抛出了一个 Exception 类型的异常，但是在该函数中，却没有 catch 并处理掉此异常的地方。因此，这样的程序即便是能够编译通过，那么运行时也是致命的（可能导致程序的崩溃），所以， Java 语言干脆在

编译时就尽可能地检查（并卡住）这种本不应该出现的错误，这无疑对提高程序的可靠性大有帮助。但是，在 Java 语言中，这就是必须的。如果一个函数中，它运行时可能会向上层调用者函数抛出一个异常，那么，它就必须在该函数的声明中显式的注明（采用 throws 关键字）。还记得刚才那条编译错误信息吗？“ must be caught or declared to be thrown ”，其中“ must be caught ”上面已经解释了，而后半部分呢？“ declared to be thrown ”是指何意呢？其实指的就是“ 必须显式地声明某个函数可能会向外部抛出一个异常 ”，也即是说，如果一个函数内部，它可能抛出了一种类型的异常，但该函数内部并不想（或不宜） catch 并处理这种类型的异常，此时，它就必须使用 throws 关键字来声明该函数可能会向外部抛出一个异常，以便于该函数的调用者知晓并能够及时处理这种类型的异常。下面列出了这几种情况的比较，代码如下：// 示例程序 1，这种写法能够编译通过 package com.ginger.exception. import java.io.*. public class Trans { public static void main(String[] args) { try { test(). } catch (Exception ex) { ex.printStackTrace(). } } static void test() { try { throw new Exception("To show Exception Successed"). } catch (Exception ex) { ex.printStackTrace(). } } } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com