Robust Java

Exception Handling, Testing and Debugging

PRENTICE HALL PTR

Robust Java 中文版

——Java 异常处理、测试与调试

- 提供 Java 平台异常处理的全面指南
- 为架构师、设计师和开发人员提供行之有效的策略
- 详述管理异常的关键设计模式

Sun microsystems

(美) Stephen Stelting 著
韩宏志 译

PEARSON Education
清华大学出版社

Robust Java 中文版

——Java 异常处理、测试与调试

(美) Stephen Stelting 著

韩宏志 译

清华大学出版社

北 京

图书在版编目(CIP)数据

Robust Java 中文版——Java 异常处理、测试与调试/(美)史德汀(Stelting,S.)著；韩宏志译.
—北京：清华大学出版社，2005.8
书名原文：Robust Java：Exception Handling，Testing and Debugging
ISBN 7-302-11341-6

Ⅰ.R… Ⅱ.①史…②韩… Ⅲ.JAVA 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2005)第 075289 号

出 版 者：清华大学出版社 **地 址**：北京清华大学学研大厦
http://www.tup.com.cn **邮 编**：100084
社 总 机：010-62770175 **客户服务**：010-62776969
组稿编辑：曹 康
文稿编辑：李 阳
封面设计：康 博
版式设计：康 博
印 刷 者：北京国马印刷厂
装 订 者：三河市春园印刷有限公司
发 行 者：新华书店总店北京发行所
开 本：185×260 **印张**：15.75 **字数**：403 千字
版 次：2005 年 8 月第 1 版 2005 年 8 月第1 次印刷
书 号：ISBN 7-302-11341-6/TP·7467
印 数：1～4000
定 价：29.80 元

前　言

“未被检测到的错误将可能被忽略。”

——引自一家领先计算公司的技术文档

“计算机科学的目标是构建能经得住时间考验的代码。”

——佚名

“在真的开始编程时我才发现，它比预想的难多了。忽然我意识到：我生命中的大多数时间都要花在查找自己程序的错误上了。”

——Maurice Wilkes

谁不希望自己一路顺风？谁不希望一夜间盖起高楼大厦，造出航天飞机？

但幻想代替不了现实。最现实的做法是未雨绸缪，防患于未然。举一个例子来说吧，即使出发前万里无云，远方的游子也应当备齐以下物品：

- 雨具
- 风衣
- 防晒霜
- 水
- 驱虫剂
- 急救箱
- 地图
- 指南针
- 登山绳

在特殊情况下，还要带上如下物品：

- 应急灯
- 狗拉雪橇
- 氧气设备

编程与旅行何其相似！与上面的旅行者一样，开发人员在面对程序时也应抛弃不切实际的幻想，客观准确地判断各种潜在问题。

在制定异常处理策略时，应随机应变地解决各类问题：有时忽略异常，而通常则“处理”或“抛出”异常。另外，必须制定一个甚至多个规划。

本书目标

我长年从事 Java 教学，在教学过程中，我发现学生们缺少一本有关 Java 异常处理的优秀书籍。由于缺乏正确的理论指导，Java 开发人员也各自在黑暗中摸索。

但 Java 异常是一个重要的课题。对错误的处理方式决定着程序的维护，也会对测试和调试产生重大影响。若处理不当，代码将晦涩难懂。若源代码包含不正确的假设和逻辑，隐藏着神秘错误，那么，在投入到实际应用之后，后果将不堪设想！

很多人并不领会异常处理的精髓。在示例代码、生产系统和 Java 编程书籍中，经常出现下面这种“掩耳盗铃”式的程序块：

```
try{
    operationA();
    component.operateB();
}
catch (Exception e){}
```

空 catch 块不对 try 块抛出的异常作任何处理。这段代码可以编译，但存在严重隐患，可能引发故障，使程序无法正常运行！

本书将系统地介绍异常、错误和处理，阐述如何编写稳定的可维护代码，讨论如何处理异常。为消除错误，编写健壮代码，开发可靠的软件系统，您必须理解异常原理，领会相关的软件设计知识，了解各种 API 及其故障类型，学会制定体系结构决策，并学会选择恰当的设计模式。

本书并非简单地列出 Java API 的每个方法抛出的异常，而是循序渐进地与您一起探索代码中出现故障的规律，并简要介绍 API 或应用程序中容易出现的错误。

在处理异常的过程中，必须进行缜密的推理。本书介绍了大量最佳实践，这些都是他人在开发项目时得出的经验，可作为编码时的参考。但是，建议您不要照搬照抄，而是要有选择地灵活运用。

本书不是“万能药”，但的确是良师益友，能帮您编写更可靠、更易维护、更易测试和调试的代码。

编排方式

处理异常涉及开发、设计和体系结构等方面的知识。本书共分 3 个部分。

第 I 部分介绍 Java 异常的产生机理和用法，介绍一些最佳实践，讲述各类异常处理使用的一般 API 和技术。

第 II 部分阐述可测试性设计，介绍故障模式分析，讨论常见 API 的异常及起因，分析 J2EE 体系结构和分布式 API 的异常模式。

第III部分讨论在软件开发周期执行异常和错误处理，分析软件体系结构、设计模式、测试和调试，列举成熟的设计模式，介绍处理策略对系统体系结构的影响，讲述如何构建健壮系统。

目　录

第 I 部分　入 门 篇

第II部分　异常处理和设计

第III部分 有效使用异常、错误和处理

第Ⅰ部分　入门篇

第1章　异常概述

1.1　简介

为全面了解“异常”的概念，先来分析一个实例。假定要编写一个 Java 程序，该程序读入用户输入的一行文本，并在终端显示该文本。这是一个演示 Java 语言 I/O 功能的简单回显(echo)程序。若认为代码一定能正常运行[1]，则可以编写以下程序：

```
 1  import java.io.*;
 2  public class BuggyEchoInput{
 3      public static void main(String [] args){
 4          System.out.println("Enter text to echo");
 5          InputStreamReader isr = new InputStreamReader(System.in);
 6          BufferedReader inputReader = new BufferedReader(isr);
 7          String inputLine = inputReader.readLine();
 8          System.out.println("READ: " + inputLine);
 9      }
10  }
```

分析一下这段代码，在 BuggyEchoInput 类中，第 3 行声明了一个 main 方法；第 4 行提示用户输入文本；在第 5、6 行设置 BufferedReader 对象连接到 InputStreamReader，而 InputStream Reader 又连接到标准输入流 System.in；第 7 行读入一行文本；第 8 行用标准输出流 System.out 显示出该文本。

实际上，BuggyEchoInput 类完全可能出现问题。要在调用第 7 行的 readLine 方法时正确读取输入，下面几种假设都必须成立：假定键盘有效，键盘能与计算机正常通信；假定键盘数据可从操作系统传输到 Java 虚拟机，又从 Java 虚拟机传输给 inputReader。

大多数情况下，上述假设都成立，但不尽然。为此，Java 采用异常方法，以应对可能出现的错误，并采取步骤进行更正。在本例，若试图编译以上代码，将看到以下消息：

```
BuggyEchoInput.java:7: unreported exception java.io.IOException;
  must be caught or declared to be thrown
        String inputLine = inputReader.readLine();
1 error
```

1 “一定能正常运行吗？”一个甜蜜的梦！

从中可看到，第 7 行调用的 readLine 方法可能出错；若果真如此，则产生 IOException 来记录故障。编译器器错误是在告诉您：需要更改代码来解决这个潜在的问题。

在 HTML Java 文档中，也可看到同样的信息。若在浏览器打开 BufferedReader 类的 Java 文档页，并查看 readLine 方法，将看到图 1-1 显示的内容。

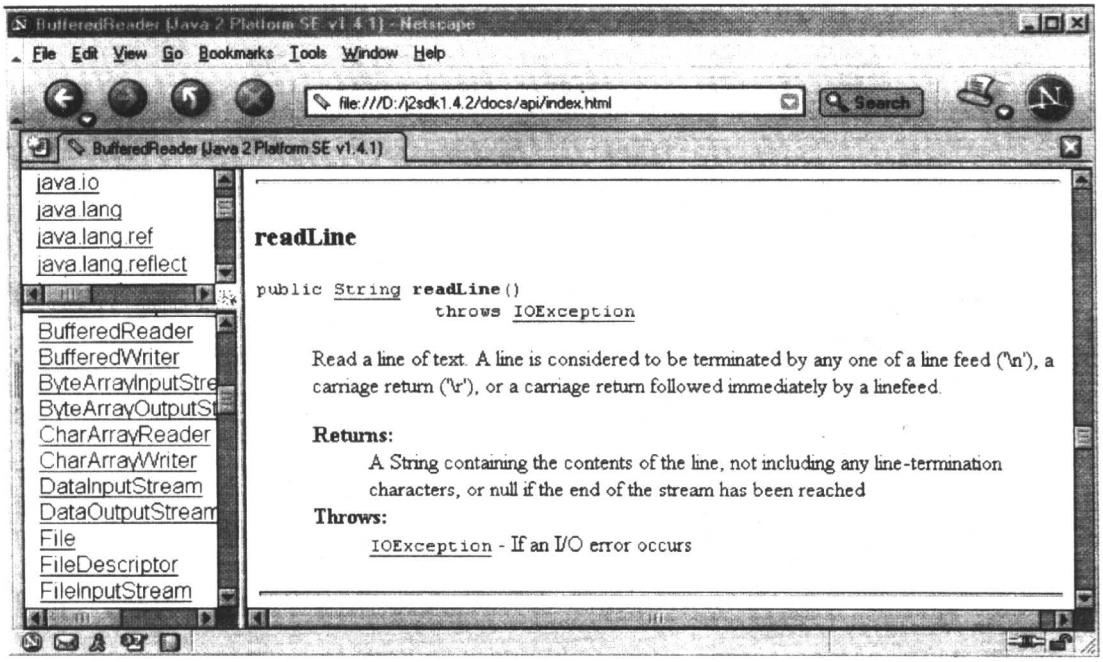

图 1-1 BufferedReader 类的 readLine 方法的 Java 文档

由图 1-1 可知，readLine 方法有时产生 IOException。如何处理该潜在故障？编译器要求“捕获”或“声明”IOException。“捕获(catch)”指当 readLine 方法产生错误时截获该错误，并处理或记录问题。而“声明(declare)”指错误可能引发 IOException，并通知调用该方法的任何代码：可能产生异常；换言之，“声明”不处理异常，而将异常传给调用该方法的任何代码。

若要捕获异常，必须添加一个特殊的“处理代码块”，来接收和处理 IOException。可能的回显程序如下所示：

```
1  import java.io.*;
2  public class EchoInputHandle{
3      public static void main(String [] args){
4          System.out.println("Enter text to echo");
5          InputStreamReader isr = new InputStreamReader(System.in);
6          BufferedReader inputReader = new BufferedReader(isr);
7          try{
8              String inputLine = inputReader.readLine();
9              System.out.println("READ: " + inputLine);
10         }
11         catch (IOException exc){
```

```
12                System.out.println("Exception encountered: " + exc);
13            }
14        }
15    }
```

新添的代码块包含关键字 try 和 catch(第 7、10、11 和 13 行)，表示要读取输入。若成功，则正常运行。若读取输入时出错，则捕获问题(由 IOException 对象表示)，并采取相应措施。在本例，采用的处理方式是输出异常。

若不准备捕获 IOException，仅声明异常，则要特别指定 main 方法可能出错，而且特别说明可能产生 IOException。此时的程序可能如下所示：

```
1  import java.io.*;
2  public class EchoInputDeclare{
3      public static void main(String [] args) throws IOException{
4          System.out.println("Enter text to echo");
5          InputStreamReader isr = new InputStreamReader(System.in);
6          BufferedReader inputReader = new BufferedReader(isr);
7          String inputLine = inputReader.readLine();
8          System.out.println("READ: " + inputLine);
9      }
10 }
```

此代码未采取措施来处理 readLine 方法的可能故障。但它说明：main 方法可能产生 IOException，调用该方法的任何代码都必须作好处理问题的准备。

在简要介绍这个示例程序后，下面将详细讨论基本概念，讲述异常和异常的产生方式，描述基本类层次结构，解释捕获和声明异常的规则。

1.2 异常概念

可将异常看作是一类消息。Java 等语言的大多数应用程序始终在使用消息。在图形用户界面(GUI)中更是如此。在 GUI 中，消息经常被用作发送事件信息(用户与应用程序交互的一些相关数据类型，如单击按钮，或在文本框中输入文本)的方式。

从某种意义上讲，异常就是这样一种消息：它传送一些系统问题、故障及未按规定执行的动作的相关信息。异常包含信息，以将信息从应用程序的一部分发送到另一部分。

编程语言为何处理异常？为何不在异常出现位置随时处理具体故障呢？使用异常的原因有很多。有时，需要在系统中交流错误信息，以便按统一方式来处理问题。在企业架构和分布式系统中，经常要使用该策略。

换言之，有若干种处理问题的可能方式，但您不知道使用哪一种。此时，当然可将处理异常的任务委托给调用方法的代码。调用者通常更了解问题来源(或导致异常的业务操作)的上下文，能更好地确定恢复方式。在 API 或低级实用程序类中，经常能见到这种策略。

要利用异常在 Java 中交流信息，必须有一个所有通用消息系统共用的架构。图 1-2 显示一个基本消息架构以及架构与异常的依赖关系。

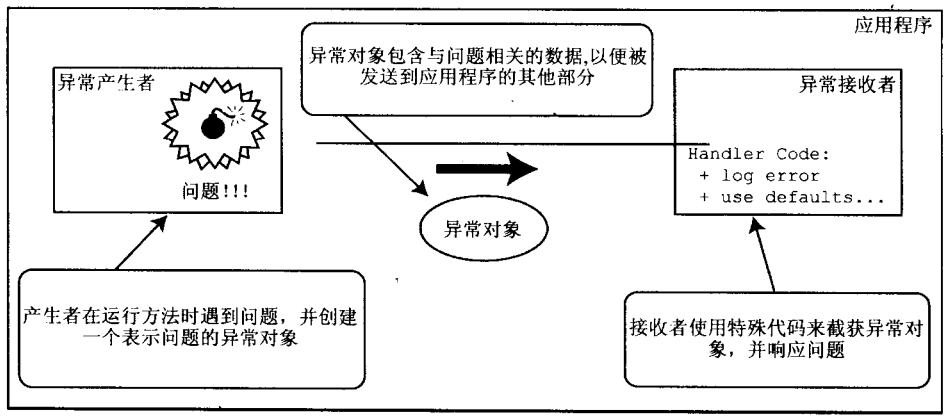

图 1-2　通用消息架构

可以看到，必定在运行的 Java 应用程序的一些类或对象中产生异常。出现故障时，“发送者”方法将产生异常对象。故障可能代表 Java 代码出现的问题，也可能是 JVM 的相应错误，或基础硬件或操作系统的错误。

在上例中，异常“产生者”是读取文本行的 BufferedReader[2]。在故障出现时，将在 readLine 方法中构建 IOException 对象。当然，异常不会无缘无故地出现，必然有一些原始故障条件驱动产生者创建和传播异常对象。异常产生者通过将错误本因转换为消息(Java 语言异常对象)，为应用程序代码解决问题埋下了伏笔。

异常本身表示消息，指发送者传给接收者的数据“负荷”。首先，异常基于类的类型来传输有用信息。很多情况下，只需异常对象的类即能识别故障本因并更正问题。其次，异常还带有可能有用的数据(如属性)。异常消息或故障本质原因等数据属于这个范畴。

在处理异常时，消息必须有接收者；否则将无法处理产生异常的底层问题。

在本例，异常的“接收者”是代码本身。分析 EchoInputHandle 应用程序的 try-catch 结构可知：catch 块是异常的接收者，它以字符串形式输出异常，将问题记录下来。

1.3　异常类层次结构

在总体上介绍异常后，接下来将讨论 Java 异常类层次结构。图 1-3 显示 Java 的异常组织方式。

在 Java 中，所有异常有一个共同祖先 Throwable(可抛出)。Throwable 指定代码中可用异常传播机制通过 Java 应用程序传输的任何问题的共性。

Throwable 有两个重要子类：Exception(异常)和 Error(错误)，二者都是 Java 异常处理的重要子类，各自都包含大量子类。

“异常”是应用程序中可能的可预测、可恢复问题。Java API 文档记录给出的定义是：“合理应用程序可能需要捕获的情况。”一般地，大多数异常表示轻度到中度的问题。

异常一般是在特定环境下产生的，通常出现于代码的特定方法或操作中。在 EchoInput 类中，当试图调用 readLine 方法时，可能产生 IOException 异常。

2 准确位置是 System.in 对象的 read 方法。read 方法读取用户的键盘输入，生成 readLine 方法传播的 IOException。

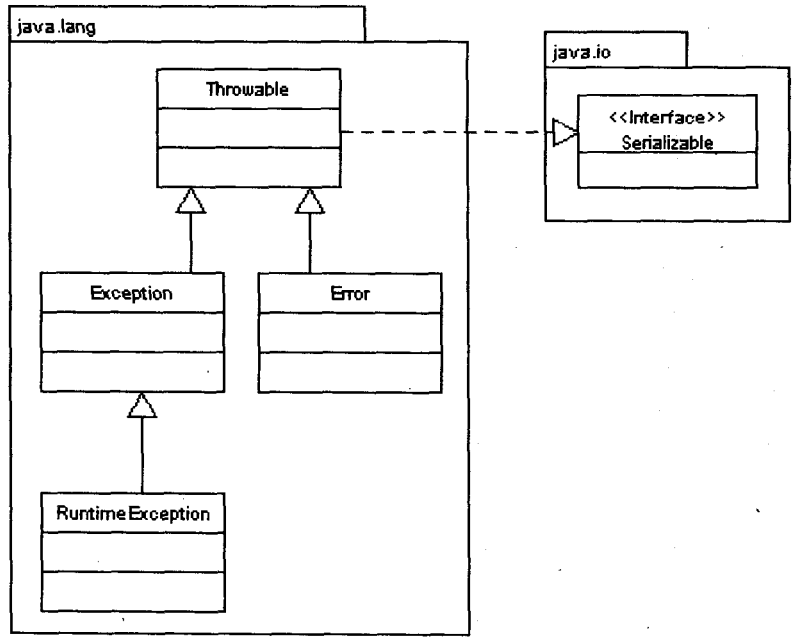

图 1-3 Java 异常类层次结构

“错误”表示运行应用程序中的较严重问题。Java API 文档记录给出的定义是：“是 Throwable 的一个子类，代表严重问题，合理应用程序不应试图捕获它。大多数此类错误属反常情况。”

大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM(Java 虚拟机)出现的问题。例如，当 JVM 不再有继续执行操作所需的内存资源时，将出现 OutOfMemoryError。

Exception 类有一个重要子类 RuntimeException。RuntimeException 及其子类表示“JVM 常用操作”引发的错误。例如，若试图使用空值对象引用、除数为零，或数组越界，则将分别引发运行时异常(NullPointerException、ArithmeticException)和 ArrayIndexOutOfBoundException。

1.4 异常的处理或声明选项

前面的示例代码曾提到过，在处理潜在故障时，有两个选项。在调用可能引发异常的方法时，可以捕获异常，也可以声明该方法抛出异常。这就是常说的“处理(handle)或声明(declare)”规则。

怎样通知 Java 编译器要遵循代码中的处理或声明规则？回顾一下，java.io.BufferedReader 类中 readLine 的方法签名指示该方法调用将抛出 IOException。

1.4.1 处理异常：try、catch 和 finally

若要捕获异常，则必须在代码中添加异常处理器块。这种 Java 结构可能包含 3 部分，都有 Java 关键字，下例演示 try-catch-finally 代码结构。