

Robust Java

Exception Handling, Testing and Debugging



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



(美) Stephen Stelting 著
韩宏志 译



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

Simplified Chinese edition copyright © 2005 by PEARSON EDUCATION ASIA LIMITED and TSINGHUA UNIVERSITY PRESS.

Original English language title from Proprietor's edition of the Work.

Original English language title: Robust Java: Exception Handling, Testing and Debugging by Stephen Stelting, Copyright © 2005

EISBN: 0-13-100852-8

All Rights Reserved.

Published by arrangement with the original publisher, Person Education, Inc., publishing as Prentice Hall.

This edition is authorized for sale only in the People's Republic of China (excluding the Special Administrative Region of Hong Kong and Macao).

本书中文简体翻译版由 Pearson Education(培生教育出版社集团)授权给清华大学出版社在中国境内(不包括中国香港、澳门特别行政区)出版发行。

北京市版权局著作权合同登记号 图字: 01-2004-4641

版权所有, 翻印必究。举报电话: 010-62782989 13501256678 13801310933

本书封面贴有 Pearson Education(培森教育出版集团)激光防伪标签, 无标签者不得销售。

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 1 章 异常概述	1
1.1 简介	1
1.2 异常概念	3
1.3 异常类层次结构	4
1.4 异常的处理或声明选项	5
1.4.1 处理异常: try、catch 和 finally	5
1.4.2 try-catch-finally 的规则	6
1.4.3 声明异常	7
1.4.4 声明异常的规则	7
1.5 可检测异常和非检测异常	7
1.6 异常的 API	8
1.7 小结	10
第 2 章 异常处理技术和实践	11
2.1 简介	11
2.2 选择处理或声明	11
2.3 标准异常处理选项	12
2.3.1 记录异常和相关信息	12
2.3.2 要求用户或应用程序输入信息	14
2.3.3 使用默认或替换数据值	15
2.3.4 将控制传给应用程序的其他部分	16
2.3.5 将异常转化为其他形式	16
2.3.6 忽略问题	17
2.3.7 重试操作	17
2.3.8 采用替换或恢复操作	18
2.3.9 使系统作好停止准备	18
2.4 异常处理注意事项	18
2.5 处理异常时提倡的事项	18
2.5.1 尽可能地处理异常	19
2.5.2 具体问题具体解决	19
2.5.3 记录可能影响应用程序运行的异常	19
2.5.4 根据情况将异常转换为业务上下文	19
2.6 处理异常时忌讳的事项	19

2.6.1	一般不要忽略异常	19
2.6.2	不要使用覆盖式异常处理块	20
2.6.3	一般不要将特定异常转换为更通用的异常	20
2.6.4	不要处理能够避免的异常	21
第 3 章	高级异常处理概念	22
3.1	简介	22
3.2	自定义异常	22
3.2.1	定义异常类	23
3.2.2	声明方法抛出自定义异常	23
3.2.3	找到故障点, 新建异常并加上关键字 throw	24
3.3	链表异常	25
3.4	异常的本地化和国际化	26
3.4.1	创建 ResourceBundle 子类来存储消息	27
3.4.2	为不同地区继承 ResourceBundle 类	27
3.4.3	创建覆盖 getLocalizedMessage 的自定义异常类并用 ResourceBundle 检索消息	28
3.5	子类	30
3.6	接口和抽象类的异常声明	30
3.7	异常栈跟踪	31
3.8	低级异常处理	34
第 4 章	异常和线程	39
4.1	简介	39
4.2	多线程系统中的异常	40
4.3	同步代码块中的异常	40
4.4	线程活动的异常风险	42
4.5	基于线程的通信的异常	43
4.6	死锁	46
4.7	取消线程	47
第 5 章	记录和断言	48
5.1	记录 API	48
5.1.1	何时使用记录 API	48
5.1.2	记录 API 简介	48
5.1.3	记录 API 详述	49
5.1.4	标准记录配置	53
5.2	断言	56
5.2.1	使用断言	57
5.2.2	在应用程序中使用断言	57

第 II 部分 异常处理和设计

第 6 章 异常处理和设计	59
6.1 简介	59
6.2 面向对象设计的原理	59
6.2.1 将异常集成到 OOD 设计以便维护	62
6.2.2 “可维护性设计”的优缺点	69
6.2.3 故障模式分析	69
6.3 小结	70
第 7 章 Java 核心语言中的异常	71
7.1 简介	71
7.2 基本数据类型	71
7.2.1 概述	71
7.2.2 用法	72
7.2.3 常见问题	72
7.2.4 一般建议	74
7.3 Object 类和 Java 中的对象	74
7.3.1 概述	74
7.3.2 潜在问题概述	74
7.4 数组	79
7.4.1 索引	79
7.4.2 数组存储管理	79
7.4.3 对象输入	80
7.5 java.lang 包中的接口	80
7.6 String 类和 StringBuffer 类	81
7.6.1 String 类	82
7.6.2 StringBuffer 类	82
7.7 BigDecimal 类和 BigInteger 类	82
7.8 包装类	83
第 8 章 集合和 I/O	84
8.1 简介	84
8.2 集合架构	84
8.2.1 概述	84
8.2.2 集合架构的注意事项和相关问题	85
8.2.3 集合架构中的异常和错误	89
8.3 I/O API	91
8.4 I/O 子类的异常和错误	94
8.4.1 一般问题	94
8.4.2 特定流类型的问题	95

8.5	新 I/O API——NIO	100
8.6	NIO API 中的异常	101
8.6.1	缓冲区	102
8.6.2	选择器	106
第 9 章	分布式 Java API	108
9.1	简介	108
9.2	分布式通信的基本原理	108
9.2.1	Java 的分布式通信模型	109
9.2.2	分布式 Java API 的问题	110
9.2.3	Java 异常模型	110
9.2.4	分布式通信模型和标准问题	111
9.3	远程方法调用(RMI)	112
9.3.1	基本 RMI 通信模型	112
9.3.2	RMI 的常见问题	113
9.3.3	RMI 的异常模型	116
9.4	Java 命名和目录接口	121
9.4.1	JNDI 的常见问题	122
9.4.2	常见 JNDI 操作的异常模型	123
9.5	Java 数据库连接	125
9.5.1	JDBC 中的异常	125
9.5.2	数据库通信技术的通用问题	126
9.5.3	标准 JDBC 生命期	128
9.6	小结	132
第 10 章	J2EE	133
10.1	简介	133
10.2	基本 J2EE 应用程序模型	133
10.2.1	J2EE 应用程序模型	135
10.2.2	J2EE 中的异常	136
10.3	客户层	136
10.4	Web 层	137
10.4.1	声明性错误处理	138
10.4.2	Web 层中的组件	139
10.4.3	Servlet 和 Filter 的异常模型: 编程错误处理	140
10.4.4	Servlet 和 Filter 异常模型	140
10.4.5	Servlet 组件的常见问题和风险	142
10.4.6	JSP 的异常模型	144
10.4.7	JSP 的直接错误转发	144
10.4.8	JSP 异常模型	145
10.4.9	JSP 的转换和运行时错误	146

10.4.10	自定义标记库	146
10.5	EJB 层	147
10.5.1	标准 EJB 方法	147
10.5.2	EJB 的常见问题	149
10.5.3	bean 特有的问题	150
10.5.4	EJB 异常模型	152
10.5.5	客户端调用方法的生命期管理	153
10.5.6	容器回调方法	155
10.5.7	从容器看异常	156
10.6	J2EE 和 EJB 的事务	157
10.7	J2EE 和异常处理的全局考虑事项	160
10.8	J2EE 异常处理要考虑的因素	161
10.8.1	日志记录	161
10.8.2	异常开销	161
10.8.3	网络层的延迟	163
10.8.4	J2EE 系统其他的最佳实践	163

第III部分 有效使用异常、错误和处理

第 11 章	体系结构、设计和异常模型	165
11.1	简介	165
11.2	架构师必须考虑异常和错误	166
11.2.1	故障的代价	166
11.2.2	成功的收获	166
11.3	体系结构、设计和开发	167
11.3.1	架构师和设计师的分工	167
11.3.2	开发人员的角色	168
11.4	异常模型的关键体系结构决策	168
11.5	异常模型的体系结构决策	169
11.5.1	处理和传播策略	170
11.5.2	异常和错误类模型	171
11.5.3	常用服务	173
11.6	编写健壮的 Java 代码	174
第 12 章	模式	176
12.1	简介	176
12.2	体系结构模式	177
12.2.1	层	177
12.2.2	模型-视图-控制器	178
12.3	设计模式	179

12.4	创建模式	179
12.4.1	创建器	179
12.4.2	Singleton 模式	181
12.5	结构模式	182
12.5.1	适配器	182
12.5.2	复合模式	183
12.5.3	facade 模式	184
12.5.4	代理模式	185
12.6	行为模式	187
12.6.1	职责链	187
12.6.2	命令模式	188
12.6.3	观察者模式	188
12.6.4	状态模式	189
12.6.5	策略模式	190
12.7	J2EE 模式	191
12.8	集成层	192
12.9	表示层	194
12.9.1	Front Controller	194
12.9.2	Intercepting Filter	195
12.10	业务层	196
12.10.1	Service Locator	196
12.10.2	Session Facade	197
12.11	小结	198
第 13 章	测试	199
13.1	测试的目的和意义	199
13.2	对测试的一些误解	199
13.2.1	误解一：开发人员不应该测试代码	200
13.2.2	误解二：开发人员应完成代码的所有测试	200
13.2.3	误解三：仅用一种方式测试应用程序	200
13.2.4	误解四：完全测试应用程序	201
13.3	盒外和盒内测试类型	201
13.3.1	测试类型	201
13.3.2	测试的角色和职责	203
13.4	Java 的测试难点	204
13.5	测试实践	204
13.5.1	测试制度化	204
13.5.2	测试的组织方法	207
13.6	如何管理和运行测试	210
13.7	测试何时结束	212

13.7.1	一般测试标准	212
13.7.2	开发人员测试标准	213
第 14 章	调试	214
14.1	简介	214
14.2	调试的含义	214
14.3	调试原理和实践	215
14.4	调试策略	216
14.5	调试所见和方法	219
14.5.1	低级测试技术	220
14.5.2	中级测试技术	222
14.5.3	高级测试技术	223
14.6	调试面临的特殊挑战	224
14.6.1	在 Java 应用程序中调试非 Java 技术	224
14.6.2	架构抛出的异常	225
附录 A	分析处理-声明的测试结果	226
附录 B	JUnit 简明指南	230
B.1	基本信息	230
B.2	安装 JUnit	230
B.3	运行 JUnit	230
B.4	JUnit 测试架构的体系结构	231
B.5	编写 JUnit 测试	232
B.6	测试设计的指导原则	238
附录 C	MyBuggyServlet——组件验证问题	239

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 InputStreamReader 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。

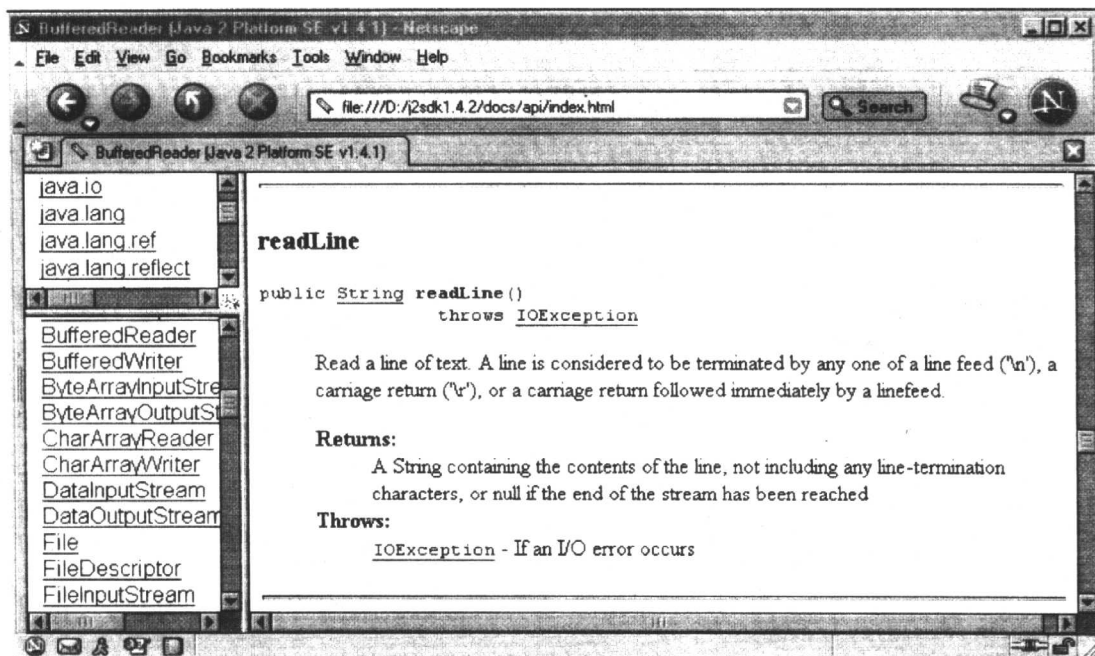


图 1-1 `BufferedReader` 类的 `readLine` 方法的 Java 文档

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。

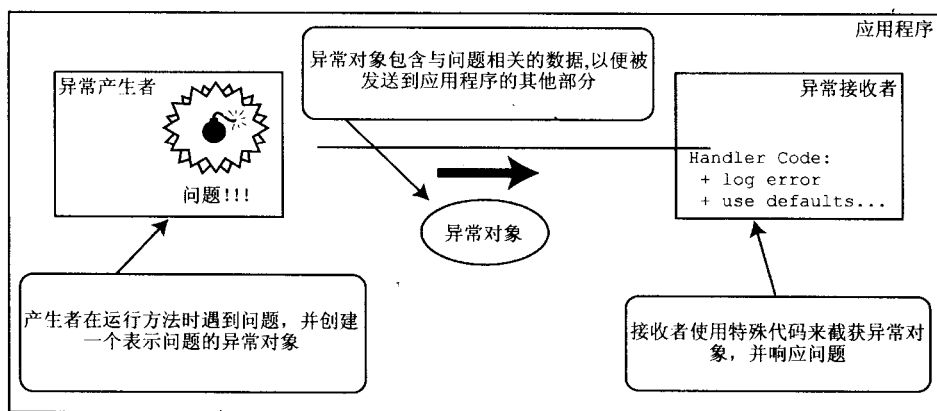


图 1-2 通用消息架构

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`。

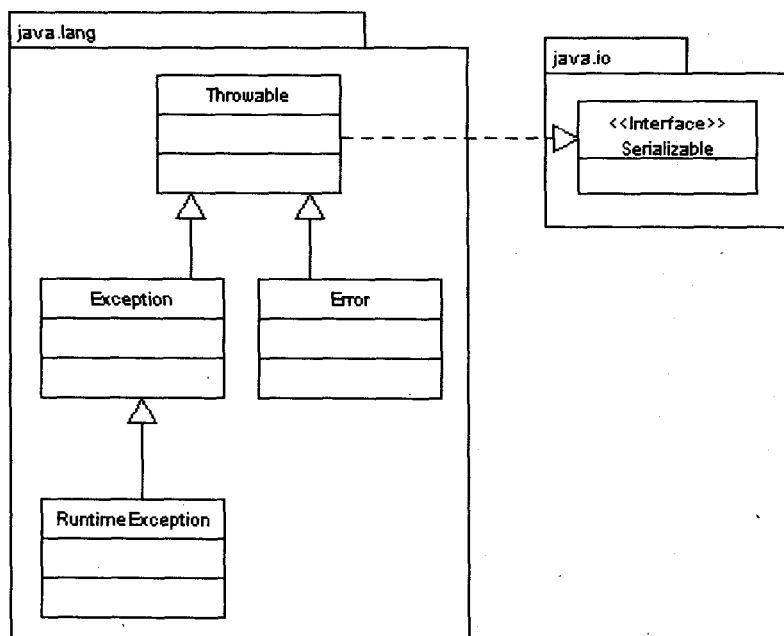


图 1-3 Java 异常类层次结构

“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 `ArrayIndexOutOfBoundsException`。

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。