Java 异常故障模式分析*

杨洪路[1,2],高文龄[1],宫云战[2],白哥乐[2]
(1.北京装甲兵工程学院,北京 100072;
2.北京邮电大学 网络与交换技术国家重点实验室,北京 100876)

摘 要:丰富、高效的异常处理机制是 Java 语言的重要特征之一。正确使用异常处理机制可以使程序的设计更加安全、可靠,但如果使用不当,则会让程序变得复杂难懂,进而影响程序效率。针对 Java 异常处理方面容易出现的故障模式进行了分析、分类并分别给出了相应的示例。

关键词:故障模式;Java;异常处理;模式分析

中图分类号:TP311.5 **文献标识码**:A

Analysis of exception fault models in Java program

YANG Hong Lu[1,2], GAO Wen Ling[1], GONG Yun Zhan[2], BAI Ge Le[2]

(1.Armored Force Engineering Institute, Beijing 100072, China;
2.State Key Laboratory of Networking and Switch Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Rich and efficient exception handling mechanism is one important characteristic of Java. Using exception handling mechanism properly can ensure the security and reliability of program design, but if used improperly, the program will become complicate, which in turn will affect the efficiency of the program. The fault modes of Java exception handling are not only analyzed and classified, but also given out respectively corresponding examples.

Key words: fault models; Java; exception handling; model analysis

异常机制是指当程序出现错误后的程序处理方式。具体来说,异常机制提供了程序退出的安全通道。当出现错误后,程序执行的流程发生改变,程序的控制权转移到异常处理器。Java 的异常处理机制可以由 3 个关键字概括而成:throw、try 和 catch。简要地说,底层遇到非正常状况,JVM 将异常信息封装到一个对象中并将其抛出,上层将这个异常对象截获并进行处理[1]。合理使用 Java 异常处理将对保持应用程序的简洁性、可维护性和正确性大有帮助。但如果使用不当,则会起到相反的效果。

本文研究的不是一般的 Java 异常处理机制,因为介绍这些处理机制的文章很多,并且已经被大多数人熟知。本文要做的是分析各种违背标准编码规范的错误习惯,也就是异常处理代码本身的故障,帮助大家了解、熟悉这些故障模式,从而在实际编码工作中敏锐地察觉和避免这些问题。Java 异常的使用故障主要有异常淹没和异常使用不当两大类。

1 异常淹没

经常看到这样的代码:程序捕获了一个异常,但在后面并没有对这个异常进行处理或处理不当,使得异常信息难以捕捉甚至消失,从而造成异常信息淹没。捕获异常的目的是处理异常。异常总是意味着某些事情不正常了,或者说至少发生了某些不寻常的事情,因此不应该对程序发出的异常求救信号保持沉默和无动于衷。

1.1 忽略异常

捕获了一个异常,但是在 catch 块中没有代码,对异常不作任何处理,这可以算得上 Java 编程中的杀手。从问题出现的频繁程度和祸害程度来看,它也许可以和 C/C++程序的不检查缓冲区溢出相提并论。

* 基金项目:国家 863 计划(项目编号:2007AA010302)

忽略异常处理会使程序泄露意想不到的状态信息。因此,在程序编码中遇到异常最好是处理,而不是忽略。故障模式举例如下:

```
try {
 FileInputStream is = new FileInputStream(name);
} catch (FileNotFoundException e) {
}
```

1.2 消失的异常

Java 支持 try/finally 语法。不论异常是否抛出,finally 模块总是在 try 模块后执行[2]。但是如果 finally 模块包含一个 return 语句,则会抑制异常的抛出,造成异常丢失。故障模式举例如下:

```
try {
    throw new MagicException();
}
finally {
    if (returnFromFinally) {
        return;
    }
}
```

1.3 不使用具体的异常

极度通用的 try…catch 块是另一种形式的异常淹没,并且更加难以检测,其方法应该抛出适当的异常,而不是普通的 Exception 或者 Throwable。try/catch 语句表示预期会出现某种异常,而且希望能够处理该异常。异常类的作用就是告诉 Java 编译器我们想要处理的是哪一种异常。由于绝大多数异常都直接或间接从 java.lang.Exception 派生,catch(Exception ex)就相当于说我们想要处理几乎所有的异常。普通的异常使调用者不知道具体发生了什么异常,增加了恰当处理问题的难度。故障模式举例如下:

```
try {
 …
} catch (Exception ex) {
   ex.printStackTrace();
}
```

2 异常使用不当

在程序中捕获所有异常是一个好想法,但是在程序中太广泛地捕获异常或对其使用不正确则会影响程序的执行效率甚至威胁程序的安全。

2.1 程序捕获了非检查类异常

由于来自 Java 虚拟机的异常(Error 的子类)和RuntimeException 的子类异常,无法确定发生的时间与地点,如果要求编译器对它们进行检查,不仅增加了编译器的复杂性,而且等于强迫程序员去处理他们本不需要也不知道如何处理的情形[3],通常来说,使用程序来捕获像NullPointerException、OutMemoryError 等非检查异常是一个不好的习惯,除非这些异常是一个测试程序的组成部分,以提供测试时某些类的异常输入。故障模式举例如下:

```
try {
    mysteryMethod();
    }
    catch (NullPointerException npe) {
      }
```

2.2 异常的产生导致锁无法释放

程序中 unlock 的位置不恰当,如果程序在 try 内部发生异常,则 unlock()将无法调用,上了锁之后不释放则会引起死锁,进而威胁程序安全。正确的处理是在 finally 中显式释放。故障模式举例如下:

```
void action() {
Lock l = new ReentrantLock();
l.lock();
try {
dosomething();
} catch (java.lang.Exception e) {
throw new RuntimeException("xxx");
}
l.unlock();
}
```

2.3 在控制流中使用异常

只为异常条件使用异常。也就是说,异常只是用在异常条件下,不能用于正常的控制流。将异常用于控制流会降低代码的可维护性和可读性。故障模式举例如下:

```
try {
Iterator i = collection.iterator();
while(true) {
Foo foo = (Foo) i.next();
…
}
} catch (NoSuchElementException e) {
}
```

2.4 异常反馈信息导致信息泄漏

抛出与抽象相适应的异常。换句话说,一个方法所抛出的异常应该在一个抽象层次上定义,该抽象层次与该方法做什么相一致,而不一定与方法的底层实现细节相一致。例如,一个从文件、数据库或者 JNDI 装载资源的方法在不能找到资源时,应该抛出某种 ResourceNotFound 异常,而不是更底层的 IOException、SQLException 或者 NamingException,在抛出异常时,打印过多关于系统信息。故障模式举例如下:

```
protected void doPost (HttpServletRequest req,HttpServle-
```

```
tResponse resp) throws ServletException,IOException {
    String name=req.getParameter("userName");
      File file=new File (System.getProperty("web.root"),
name + ".dat");
        try {
            FileOutputStream str =new FileOutput-
Stream(file);
            str.close();
          } catch (IOException e) {
      Throw new ServletException("Cannot open file " +
file + ":" + e.getMessage());
                    }
                }
```

虽然异常处理是大家学习 Java 编程时首先接触的内容之一，但即便是经验丰富的程序员，也会犯下各种各样的失误。正确的使用异常处理机制能增强系统的健壮性，而带有错误的异常处理代码不但不能增加程序的健壮性，反而会影响到系统的安全[4]。为了充分发挥异常处理机制的作用，得到高效的 Java 程序，应该避免使用各种 Java 异常故障模式，按照正确的方法设计使用异常处理机制，使其更好地为 Java 程序服务。

参考文献

[1] 温俊铭，张尧霄.基于 Java 异常处理模型的分析与研究[J].计算机工程，2004(12)：19-20.

[2] ECKEL B.Thinking in Java，3rd edition [M].Prentice Hall PTR，2002.

[3] 张聪品，孙印杰.基于 Java 的异常处理研究[J].河南师范大学学报.2005(33)：29-32.

[4] 姜元鹏，张永平，姜淑娟.测试 Java 异常处理机制的方法[J].计算机工程与设计，2007(10)：5069-5071.

(收稿日期：2009-02-13)