

第 5 章

异常处理与输入输出流

安全是 Java 语言考虑的重要因素之一。Java 通过异常处理机制对程序执行过程中与预期不符的各种意外情况进行捕获与处理。异常(Exception),又称为例外,是特殊的运行错误对象。当程序运行过程中遇到异常时,异常处理机制会将程序流程导向到专门的错误处理模块。

一个程序需要从外部获取(输入)信息,这个“外部”范围很广,包括诸如键盘、显示器、文件、磁盘、网络、另外一个程序等;“信息”也可以是任何类型的,例如一个对象、一串字符、图像、声音等。在 Java 程序设计中,可以通过使用 java.io 包(又称 IO 包)中的输入输出流类达到输入输出信息的目的。

这一章首先介绍 Java 的异常处理机制及方法,然后介绍输入输出流类的基本概念和常用的文件读写方式。通过这一章的学习并参考 JDK 文档,读者可以学习到如何在程序中处理各种异常情况,并可以编写涉及不同信息源和信息类型的输入输出流程序。

5.1 异常处理机制简介

5.1.1 异常处理概述

异常是在程序运行过程中发生的异常事件,例如除 0 溢出、数组越界、文件找不到等,这些事件的发生将阻止程序的正常运行。为了提高程序的鲁棒性,在进行程序设计时,必须考虑到可能发生的异常事件并做出相应的处理。

Java 中声明了很多异常类,每个异常类都代表了一种运行错误,类中包含了该异常的信息和处理异常的方法等内容。每当 Java 程序运行过程中发生一个可识别的运行错误(即有一个异常类与该错误相对应时),系统就会产生一个相应的该异常类的对象,然后采取相应的机制来处理它,确保不会对操作系统产生损害,从而保证了整个程序运行的安全性。这就是 Java 的异常处理机制。

Java 通过面向对象的方法来处理程序错误,为可能产生非致命性错误的代码段设计错误处理模块,将错误作为预定义好的“异常”捕获,然后传递给专门的错误处理模块进行处理。在一个方法的运行过程中,如果发生了异常,那么 Java 虚拟机便会生成一个代表该异常的对象,并把它交给运行时系统,运行时系统便寻找相应的代码来处理这一异常。我们把生成异常对象并提交给运行系统的过程称为抛出(throw)一个异常。

运行时系统在方法的调用栈中查找,从生成异常的方法开始进行回溯,直到找到包含处理相应异常的方法为止,这一个过程称为捕获(catch)一个异常。

使用 Java 的异常处理机制进行错误处理有以下优点：

- 将错误处理代码从常规代码中分离出来。
- 按错误类型和差别分组。
- 能够捕获和处理那些难以预测的错误。
- 克服了传统方法中错误信息有限的问题。
- 把错误传播给调用堆栈。

图 5-1 是对异常处理的示意图。可以看到,图中所示程序的调用过程是从方法 1~4 依次调用,方法 4 探测并抛出异常,由方法 1 捕获并处理。在程序实际执行过程中,如果在方法 4 中发生了异常,则程序将从该异常点沿调用栈进行回溯,直到找到对异常进行处理的方法为止,即图中的方法 1。同样,如果在方法 2 中对异常进行了捕获,那么回溯就不会到达方法 1,而是在方法 2 就完成了。

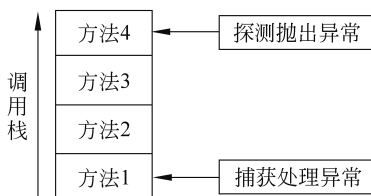


图 5-1 异常处理示意图

5.1.2 错误的分类

广义来说,程序中的错误可以分为三类,即编译错误、运行错误和逻辑错误。编译错误是编译器能够检测到的错误,一般为语法错误;运行错误是运行时产生的错误,如被零除、数组下标越界等;而逻辑错误是机器本身无法检测的,需要人对运行结果及程序逻辑进行认真分析。逻辑错误有时会导致运行错误。

Java 系统中根据错误的严重程度不同,而将错误分为两类:

- 错误(Error): 是致命性的,即程序遇到了非常严重的不正常状态,不能简单地恢复执行。
- 异常(Exception): 是非致命性的,通过某种修正后程序还能继续执行。

Exception 类是所有异常类的超类;Error 类是所有错误类的超类。这两个类同时又是 Throwable 的子类。

异常和错误类的层次结构如图 5-2 所示。

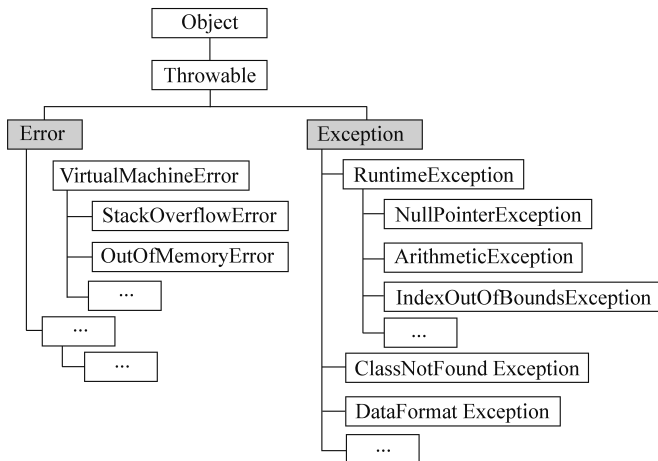


图 5-2 异常和错误类的层次结构

从图 5-2 可以看到,Error 类下的错误都是严重的错误,用户程序无法进行处理;Exception 下的异常又分为两类:检查型异常和非检查型异常。

一些因为编程错误而导致的异常,或者是不能期望程序捕获的异常(例如数组越界、除零,等等),称为非检查型异常(继承自 RuntimeException)。非检查型异常不需要在方法中声明抛出,且编译器对这类异常不做检查。

其他类型的异常称为检查型异常,Java 类必须在方法签名中声明它们所抛出的任何检查型异常。对于任何方法,如果它调用的方法抛出一个类型为 E 的检查型异常,那么调用者就必须捕获 E 或者也声明抛出 E(或者抛出 E 的一个超类),对此编译器要进行检查。

例 5-1 测试系统定义的运行异常——数组越界出现的异常。

程序如下:

```
public class HelloWorld {  
    public static void main(String args[]) {  
        int i = 0;  
        String greetings[] = {"Hello world!", "No, I mean it!", "HELLO WORLD!!"};  
        while (i < 4) {  
            System.out.println(greetings[i]);  
            i++;  
        }  
    }  
}
```

输出结果如下:

```
Hello world!  
No, I mean it!  
HELLO WORLD!!  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException  
    at HelloWorld.main(HelloWorld.java:7)
```

上面是一个简单的 Java 程序,访问数组元素时,运行时环境会根据数组的长度检查下标是否越界。由于声明数组的长度为 3,当访问数组下标为 3 的元素时出现越界,就导致了 ArrayIndexOutOfBoundsException 异常。这个异常是系统定义好的异常类,对应系统可识别的错误,所以 Java 虚拟机就会自动中止程序的执行流程,并新建一个 ArrayIndexOutOfBoundsException 类的对象,然后抛出这个异常。

Java 预定义了一些常见异常,例如:

1. ArithmeticException

整数除法中,如果除数为 0,则发生该类异常。

如:

```
int i = 12 / 0;
```

2. NullPointerException

如果一个对象还没有实例化,那么访问该对象或调用它的方法将导致 NullPointerException 异常。

如:

```
Image im [] = new Image [4];  
System.out.println(im[0].toString())
```

3. NegativeArraySizeException

数组的元素个数应是一个大于等于 0 的整数。创建数组时,如果元素个数是个负数,则会引发 NegativeArraySizeException 异常。

4. ArrayIndexOutOfBoundsException

把数组看作是对象,并用 length 变量记录数组的大小。访问数组元素时,运行时环境会根据数组的长度检查下标是否越界。如果数组下标越界,则将导致 ArrayIndexOutOfBoundsException 异常。

5. ArrayStoreException

程序试图存取数组中错误的数据类型。

6. FileNotFoundException

试图访问一个不存在的文件。

7. IOException

通常的 I/O 错误。

5.2 异常的处理

对于检查型异常,Java 程序必须进行处理。处理方法有如下两种:

- 声明抛出异常:不在当前方法内处理异常,而是把异常抛出到调用当前方法的方法中。
- 捕获异常:在当前方法中使用 catch 语句块捕获所发生的异常,并进行相应的处理。

5.2.1 声明抛出异常

如果程序员不想在当前方法内处理异常,可以使用 throws 子句声明将异常抛出到调用当前方法的方法中。如:

```
public void openThisFile(String fileName) throws java.io.FileNotFoundException {  
    //code for method  
}
```

一个 throws 子句也可以声明抛出多个异常,如:

```
public Object convertFileToObject(String fileName)  
    throws java.io.FileNotFoundException, java.lang.ClassNotFoundException {
```

```
        //code for method  
    }
```

调用方法也可以将异常再抛给它的调用方法,如:

```
public void getCustomerInfo() throws java.io.FileNotFoundException {  
    //do something  
    this.openThisFile("customer.txt");  
    //do something  
}
```

上例中,如果在 openThisFile 方法中抛出了异常,getCustomerInfo 方法将停止执行,并将此异常传送给调用 getCustomerInfo 方法的方法中。

如果所有的方法都选择了抛出此异常,都没有捕获处理,最后 Java 虚拟机将捕获它们,输出相关的错误信息,并终止程序的运行。在异常被抛出之后,调用栈中的任何方法都可以捕获异常并进行相应的处理。

5.2.2 捕获异常

Java 程序设计中可以使用 try 语句括住可能抛出异常的代码段,然后紧接着用 catch 语句指明要捕获的异常及相应的处理代码。

try 与 catch 语句的语法格式如下:

```
try {  
    //此处为抛出具体的异常代码  
} catch (ExceptionType1 e) {  
    //抛出 ExceptionType1 异常时要执行的代码  
} catch (ExceptionType2 e) {  
    //抛出 ExceptionType2 异常时要执行的代码  
    ...  
} catch (ExceptionTypek e) {  
    //抛出 ExceptionTypek 异常时要执行的代码  
} finally {  
    //必须执行的代码  
}
```

其中,ExceptionType1、ExceptionType2...ExceptionTypek 是产生的异常类型。在运行时,根据产生的异常类型找到对应的 catch 语句,然后执行 catch 语句块中的代码部分。finally 语句块的作用通常用于释放资源,它不是必须的部分,如果有 finally 语句块,不论是否捕获到异常,总要执行 finally 语句块中的代码。

在有多个异常需要捕获时,异常类型的顺序很重要,具体的异常类型要放在一般异常类型的前面。例如,下面程序中的顺序就不合理,如果发生 FileNotFoundException 异常,由于第一个 catch 语句块中声明的 Exception 是 FileNotFoundException 的超类,因此异常总是会被第一个 catch 语句块捕获,永远无法到达第二个 catch 语句块。

```
public void getCustomerInfo() {  
    try {  
        //do something that may cause an Exception  
    } catch (java.lang.Exception ex) {  
        //Catches all Exceptions  
    } catch (java.io.FileNotFoundException ex) {  
        //Never reached since above two are caught first  
    }  
}
```

在 catch 块的内部,可用下面的方法处理异常对象:

- getMessage()——返回一个对产生的异常进行描述的字符串。
- printStackTrace()——输出运行时系统的方法调用序列。

例 5-2 捕获异常举例。

实现功能:读入两个整数,第一个数除以第二个数,之后输出。

```
public class ExceptionTester1 {  
    public static void main(String args[]) {  
        System.out.println("Enter the first number:");  
        int number1 = Keyboard.getInteger();  
        System.out.println("Enter the second number:");  
        int number2 = Keyboard.getInteger();  
        System.out.print(number1 + " / " + number2 + " =");  
        int result = number1 / number2;  
        System.out.println(result);  
    }  
}
```

其中,Keyboard 类的声明如下:

```
import java.io.*;  
public class Keyboard {  
    static BufferedReader inputStream = new BufferedReader(new  
    InputStreamReader(System.in));  
    public static int getInteger() {  
        try {  
            return (Integer.valueOf(inputStream.readLine().trim()).intValue());  
        } catch (Exception e) {  
            e.printStackTrace();  
            return 0;  
        }  
    }  
    public static String getString() {  
        try {
```

```
        return (inputStream.readLine());
    } catch (IOException e) {
        return "0";
    }
}
}
```

如果依次输入“143”和“24”，则输出结果如下：

```
Enter the first number:
143
Enter the second number:
24
143 / 24=5
```

如果依次输入“140”和“abc”，则系统会报告异常，结果如下：

```
Enter the first number:
140
Enter the second number:
abc
java.lang.NumberFormatException: For input string: "abc"
    at java.lang.NumberFormatException.forInputString
(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:449)
    at java.lang.Integer.valueOf(Integer.java:554)
    at Keyboard.getInteger(Keyboard.java:6)
    at ExceptionTester.main(ExceptionTester.java:7)
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExceptionTester.main(ExceptionTester.java:9)
140 / 0=Java Result: 1
```

在 Keyboard.getInteger 方法中，捕获任何 Exception 类的异常，并输出相关信息。

为了处理输入的字母(而非数字)，可以使用方法 Keyboard.getString 先取得字符串后，然后再做转换。

例 5-3 捕获 NumberFormatException 类型的异常。

```
public class ExceptionTester2 {
    public static void main(String args[]) {
        int number1 = 0, number2 = 0;
        try {
            System.out.println("Enter the first number:");
            number1 = Integer.valueOf(Keyboard.getString()).intValue();
            System.out.println("Enter the second number:");
            number2 = Integer.valueOf(Keyboard.getString()).intValue();
        }
```

```
        } catch (NumberFormatException e) {
            System.out.println("Those were not proper integers!quit!");
            System.exit(-1);
        }
        System.out.print(number1 + " / " + number2 + "=");
        int result = number1 / number2;
        System.out.println(result);
    }
}
```

运行结果如下：

```
Enter the first number:
abc
Those were not proper integers! I quit!
```

例 5-4 捕获被零除的异常（ArithmeticException 类型的异常）。

```
public class ExceptionTester3 {
    public static void main(String args[]) {
        int number1 = 0, number2 = 0, result = 0;
        try {
            System.out.println("Enter the first number:");
            number1 = Integer.valueOf(Keyboard.getString()).intValue();
            System.out.println("Enter the second number:");
            number2 = Integer.valueOf(Keyboard.getString()).intValue();
            result = number1 / number2;
        } catch (NumberFormatException e) {
            System.out.println("Invalid integer entered!");
            System.exit(-1);
        } catch (ArithmeticException e) {
            System.out.println("Second number is 0, cannot do division!");
            System.exit(-1);
        }
        System.out.print(number1 + " / " + number2 + "=" + result);
    }
}
```

输出结果如下：

```
Enter the first number:
143
Enter the second number:
0
Second number is 0, cannot do division!
```

下面对程序进行改进：重复提示输入，直到输入合法的数据。为了避免代码重复，可将数据存入数组。

例 5-5 可以提示重复输入的程序。

```
public class ExceptionTester4 {
    public static void main(String args[]) {
        int result;
        int number[] = new int[2];
        boolean valid;
        for (int i = 0; i < 2; i++) {
            valid = false;
            while (!valid) {
                try {
                    System.out.println("Enter number " + (i + 1));
                    number[i] = Integer.valueOf(Keyboard.getString()).intValue();
                    valid = true;
                } catch (NumberFormatException e) {
                    System.out.println("Invalid integer entered. Please try again.");
                }
            }
        }
        try {
            result = number[0] / number[1];
            System.out.println(number[0] + " / " + number[1] + "=" + result);
        } catch (ArithmeticException e) {
            System.out.println("Second number is 0, cannot do division!");
        }
    }
}
```

运行结果如下：

```
Enter number 1
abc
Invalid integer entered. Please try again.
Enter number 1
efg
Invalid integer entered. Please try again.
Enter number 1
143
Enter number 2
abc
Invalid integer entered. Please try again.
Enter number 2
40
143 / 40=3
```

5.2.3 生成异常对象

前面所提到的异常或者是由 Java 虚拟机生成,事实上,在程序中也可以自己生成异常对象,即不是因为出错而产生异常,而是人为地抛出异常。

不论哪种方式,生成异常对象都是通过 throw 语句实现,例如:

```
throw new ThrowableObject();  
ArithmeticException e = new ArithmeticException();  
throw e;
```

注意: 生成的异常对象必须是 Throwable 或其子类的实例。

例 5-6 生成异常对象举例。

程序如下:

```
class ThrowTest {  
    public static void main(String args[]) {  
        try {  
            throw new ArithmeticException();  
        } catch (ArithmeticException ae) {  
            System.out.println(ae);  
        }  
        try {  
            throw new ArrayIndexOutOfBoundsException();  
        } catch (ArrayIndexOutOfBoundsException ai) {  
            System.out.println(ai);  
        }  
        try {  
            throw new StringIndexOutOfBoundsException();  
        } catch (StringIndexOutOfBoundsException si) {  
            System.out.println(si);  
        }  
    }  
}
```

运行结果如下:

```
java.lang.ArithmeticException  
java.lang.ArrayIndexOutOfBoundsException  
java.lang.StringIndexOutOfBoundsException
```

5.2.4 自定义异常类

除了使用系统预定义的异常类外,用户还可以自己定义异常类,所有自定义的异常类都必须是 Exception 的子类。一般的声明方法如下: