基于 Java 的异常处理技术①

蓝雯飞 （中南民族大学计算机科学学院 武汉 430074）

摘要：异常处理提高了程序的安全性和健壮性。本文就 Java 语言的异常处理作了详细的分析和探讨，在此基础上，结合例子说明了异常处理在程序设计中的应用。

关键词：Java 语言 异常 异常处理 类

1 引言

异常是指在程序运行期间出现的一些不正常或背离正常程序流程的运行错误，为了能够及时有效地处理程序中的运行错误，Java 语言引入了一套完备的异常处理机制。

传统的错误处理方法一般以判断返回值的方式处理错误，使得正常程序代码和错误代码混合在一起，给程序维护带来了很大的障碍。Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，具有良好的层次性。这种异常处理机制能将处理错误异常的代码和"常规"代码分开，使程序维护起来更加方便，同时也增强了程序的安全性和健壮性。

和 Java 语言的其它特性相比，异常处理显得深奥且不易掌握，因为它和 Java 运行时系统密切相关。本文着重就 Java 语言的异常处理机制及在程序设计中的应用作一个全面深入的探讨。

2 异常类结构与组成

为了能够识别并处理程序中的运行错误，Java 引入了异常这个概念，作为面向对象的语言，异常与其他语言要素一样，是面向对象规范的一部分，是异常类的对象。为了能区分不同的运行错误，Java 定义了很多系统异常类，每个异常类都代表了一种运行错误，类中包含了该运行错误的信息和处理错误的方法等内容[1]。

Java 的所有异常类都是系统类 Throwable 的子类，Java 主要异常类层次结构如图 1 所示。Throwable 直接派生了两个子类：Error 和 Exception。Error 是与系统或 Java 虚拟机（JVM）相关的异常，标志着严重的系统错误，是不可恢复的，在通常情况下是不希望被程序捕获的。Exception 用于应用程序可能捕获的异常。Exception 类有一个重要的子类 RuntimeException，和其子类一起在 Java 中被称为运行时异常。运行时异常自动为 Java 程序定义包括被零除和数组下标越界等类型的异常错误。

根据编译器是否对异常做检查，Java 的异常分为两类：非检查型异常和检查型异常。在异常类的层次结构中，Error 家族和 RuntimeException 家族都是非检查型异常（unchecked exception）；它是一种编译器不检查一个方法是否处理或指定向上抛出的异常，是在 Java 运行时系统内发生的异常，ArithmeticException、ArrayIndexOutOfBoundsException、NumberFormatException 等都是常见的非检查型异常。

Exception 及其除 RuntimeException 家族以外的所有子类都是检查型异常（checked exception）；它是一种编译器必须检查一个方法是否处理或指定向上抛出的异常，是在 Java 运行时系统外的代码中发生的异常，如果一个方法没有处理或指定向上抛出可能发生的检查型异常，那么该方法就不能通过编译器的编译。ClassNotFoundException、FileNotFoundException、IOException 等均是常见的检查型异常。

Java 类库包含了许多有用的异常类，使得程序员可以把更多的时间花在程序设计上[2]。同时，Java 允许程序员自定义异常类，用来处理应用程序中特定的逻辑运行错误，自定义异常类必须继承现有的异常类。

① 2005 年湖北省教学研究项目（20050233）

3 异常的处理

Java 的异常处理可以概括为以下几个过程：

(1) 在 Java 程序的执行过程中，如果出现了一个可识别的错误，则会生成一个与该错误相对应的异常类的对象，该异常对象将被提交给 Java 运行时系统，这个过程称为抛出异常。

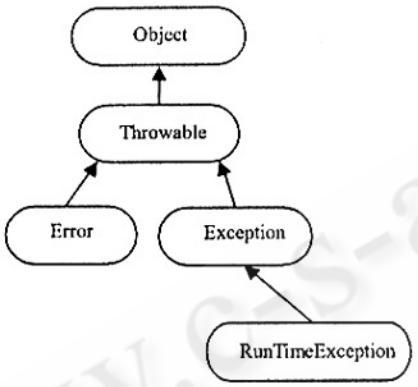

图 1 Throwable 类和它最主要的子类

(2) 当 Java 运行时系统接收到异常对象后，会寻找能处理这一异常的代码并把当前异常对象交给其处理，这一过程称为捕捉(catch)异常。

(3) 如果 Java 运行时系统找不到处理异常的程序，那么它将终止程序的执行，最后调用缺省异常处理程序来处理异常。

Java 语言有关异常的语句有 2 个，一个是 try 语句，还有一个是 throw 语句。try 语句用来抛出并处理一个或多个异常，包括 try 块、catch 块和 finally 块。在 Java 程序中，应把可能抛出异常的语句封装在 try 块中，用以捕获可能发生的异常。如果 try 块在执行过程中引发了异常，当运行时系统接收到异常后，就会根据异常的类型搜寻匹配的异常处理程序，该处理程序的目的在于处理 try 块中抛出的异常。搜索开始于最接近 try 块的 catch 块，并沿着方法调用堆栈继续，直到发现能处理该异常的 catch 块(也叫异常处理程序)。在找到异常处理程序之后，把异常传递给该处理程序，异常就被捕获了，执行流程转向该 catch 块，当执行完 catch 块，将跳过所有的 catch 块，执行 finally 块。如果 try 块在执行过程中没有引发异常，那么程序流程执行完在 try 块后，将跳过所有的 catch 块，直接执行 finally 块。

一个 try 语句允许有多个 catch 块，分别处理不同类型的异常。Java 运行时系统从上到下依次对每个 catch 块处理的异常类型进行检测，直到找到类型相匹配的 catch 块为止。因此，每个 catch 块应该根据异常类型从特殊到一般排列。也可以用一个 catch 块来处理多个类型的异常，这时，它的异常类型应该是这多个异常类型的父类。除非有充分的理由这么做，一般应该对不同类型的异常作不同的处理[3]。下面是一个用 try 语句对异常进行处理的例子。

```
import java.io.*;
public class ExceptionWithCatch{
  public static void main(String[] strs){
    int a=5,b=0;
     BufferedReader br = new BufferedReader
(new InputStreamReader(System.in));
    for (int i=1;i>=0;i--){
      System.out.println("---i="+i+"---");System.out.println("请输入一个整数:");
       try{ int x = Integer.parseInt(br.readLine
());b=a/x;System.out.println("b="+b);
        }
        catch(ArithmeticException e){
         System.out.println("Catch block,i.e.
exception handling code.");}
        catch(IOException ioe){
          System.err.println(ioe.toString());Sys-
tem.exit(0);}
          finally{ System.out.println("Finally
block!");}
        System.out.println("After finally block!");
      }
    }
  }
```

当 readLine() 方法读失败将产生 IOException 检查型异常，当读成功时，我们来分析上面程序的执行流程，当 x 取非 0 时，由于除数 x 不为 0，因此 try 块不会发生异常，当 try 块执行结束，程序执行流程转向 finally 块。当 i 取 0 时，由于除数为 0，除法运算发生 Arith-

meticException 异常，执行流程转向 catch(ArithmeticException e)块，然后跳到 finally 块执行。如果方法 readLine( )发生 IOException 异常，则跳到 catch(IOException ioe)块执行，由于在 catch 块中调用了 exit( )方法而终止了程序的执行。

从以上分析我们得出 Java 的异常处理具有如下几个特点：

(1) 不论异常是否发生，finally 块总是被执行。JVM 规范定义当一个异常在本层没有对应的 catch 处理块时，执行完 finally 块后，再将这个异常抛给上层。

(2) 在没有异常发生时，不论有没有 catch 块，程序运行结果是一样的。

(3) 无论有没有 catch 块，一旦出现异常，同一 try 块中异常发生点以后的代码都不被执行。

(4) 当没有 catch 块处理相应的异常时，缺省异常处理程序打印出有关的异常信息并终止程序。

4 指定方法抛出的异常

在有些情况下，方法暂时还不能确定如何处理异常，它可以将异常进一步向上层抛出，将处理异常的任务留给调用它的方法。在 Java 中，在方法的头部使用 throws 子句指定方法向上抛出异常。例如，下面的方法头部使用了 throws 子句，表示向上抛出 FileNotFoundException 异常。

```
public void readFile ( String file ) throws FileNotFoundException{
    FileInputStream f = new FileInputStream ( file ) ;//可能产生 FileNotFoundException 异常
    ……
}
```

需要注意的是，如果一个方法在执行时可能会抛出检查型异常，那么要么在当前方法中用 try 语句处理它，见第 3 节中处理 IOException 的例子，或者在方法头部用 throws 子句列出所抛出的异常，不然无法通过编译。如果是非检查型异常可以不处理它，也可以不在方法头部指定抛出异常。如果非检查型异常要进一步向上抛出的话，建议最好也在 throws 子句中列出，这样便于提醒方法调用者处理异常，就像类库中的方法一样。

5 抛出异常

抛出异常是指，当 Java 程序在执行过程中检测到一个可识别的错误，就会产生一个与该错误相对应的异常类的对象，该对象包含了异常的类型和错误出现时程序所处的状态信息，并将它提交给 Java 运行时系统的过程。在 Java 中抛出异常有 2 种方法，一种是系统自动抛出异常，还有一种是用 throw 语句指定在程序中抛出异常。

5.1 系统自动抛出的异常

所有的系统异常都可以由系统自动抛出[1]，注意用户自定义的异常不能由系统自动抛出。

下面的程序在执行过程中引发的异常 ArrayIndexOutOfBoundsException 将由 Java 运行时系统抛出。

```
public class TestThrowSystemException{
  public static void main(String args[ ]){
    int a[ ] = {1,2,3}; System. out. println ( a[3]);
    //数组下标越界，引发系统抛出异常 ArrayIndexOutOfBoundsException
  }
}
```

程序运行结果为：

```
Exception in thread " main" java. lang. ArrayIndexOutOfBoundsException: 3
    at TestThrowSystemException. main ( TestThrowSystemException. java:3)
```

上述程序是一个简单的 Java Application，由于错误地用 3 作为引用数组元素的下标，运行过程中将引发 ArrayIndexOutOfBoundsException；这是系统预定义的异常类，对应系统可识别的错误，所以 Java 运行时系统遇到这样的错误就会自动中止程序的执行流程，并新建一个 ArrayIndexOutOfBoundsException 类的对象，即抛出了一个数组下标越界异常，然后调用缺省的异常处理程序在标准输出设备上打印输出该异常对象的堆栈使用轨迹。

5.2 throw 语句抛出的异常

在 Java 中可以使用 throw 语句指定在程序中抛出某种类型的异常。我们来看下面的例子：

```
public class TestThrow {
```

```
    public static void main(String args[]) throws
ArrayIndexOutOfBoundsException{
        int a[]={1,2,3}; int i=3;
        if(i>2||i<0) throw new ArrayIndexOutOf-
BoundsException();
        else System.out.println(a[i]);
    }
}
```

程序运行结果为:

```
Exception in thread "main" java.lang.ArrayIn-
dexOutOfBoundsException
    at TestThrow.main(TestThrow.java:4)
```

上述例子在 5.1 节例子的基础上由 main() 方法显式地使用 throw 语句表示直接抛出异常,抛出的异常由于 main() 方法没有为它定义异常处理程序,因此它将抛给运行时系统处理。在 Java 程序中,如果一个方法本身没有使用 throw 语句,但在方法头部使用了 throws 子句表示方法将间接抛出异常,例如下面的方法 method1() 由于调用了 method2(),又不打算处理 method2() 抛出的异常 IndexOutOfBoundsException 而使用了 throws 子句。 (代码略)

使用 throw 语句抛出异常时应注意以下几个问题:

(1) 一般地,抛出异常的语句应该被定义在满足一定条件时执行,例如把 throw 语句放在 if 语句的 if 分支中,只有当 if 条件得到满足,即用户定义的逻辑错误发生时才执行。

(2) catch 后的参数类型(T1)必须和 throw 后的表达式类型(T2)一致。即或者 T1 和 T2 一样,或者 T1 是 T2 的父类。

(3) 只能抛出一个 throwable 或其子类的异常对象,在一般情况下,应当抛出一个 Exception 或其子类的对象。

(4) 含有 throw 语句的方法,一般应该在方法头定义中增加:throws <异常类名列表>,表示将异常交给调用者处理。

6 异常处理原则

出于对程序的安全性和健壮性考虑,应该为 Java 程序添加必要的异常处理代码,但过度地增加 try 和 catch 块会影响程序的执行效率,因为它们都要耗费执行时间,下面是笔者对使用 Java 异常处理总结的几条原则:

(1) 首先,只有在程序代码无法预料的情况下,才使用 try 语句。

(2) 不要把每条语句都封装到一个 try 块,而应该把需要保护的一个程序片段作为一个整体封装到 try 块中,如果可能的话,可定义多个 try 块。

(3) 只有在可能发生异常时才使用 try 语句,不要用它们来进行简单的程序测试。因为,处理异常需要耗费很多执行时间。

(4) 在当前环境中尽可能解决问题,并将异常抛出到一个级别更高的环境中。

(5) 在 catch 块中尽可能指定具体的异常类型,必要时使用多个 catch 块。不要试图用一个 catch 块处理所有可能出现的异常。原因是太一般化的异常处理程序会增加代码出错的可能性。

(6) 充分运用 finally 子句,保证在异常发生时所有资源都被正确释放。

7 结束语

在进行异常程序设计时,一般地,在 try 块中通常会调用其他方法来检测并可能抛出异常,当被调用的方法检测到异常时,将用 throw 语句抛出某种类型的异常对象,而 try 块后的某个 catch 块将处理这种类型的异常,finally 块保证在任何情况下都能被执行。

异常处理是 Java 语言的高级特性之一。在 Java 面向对象程序设计中适当地使用异常处理,可以提高程序的安全性和健壮性。本文就 Java 语言的异常处理作了深入详细的讨论,希望对 Java 程序设计人员有所裨益。

参考文献

1 印旻,Java 语言与面向对象程序设计[M],北京 清华大学出版社,2000.249~250。

2 蓝雯飞,C++的异常处理技术探讨[J],微计算机应用,2002,23(6):375~377。

3 Mary Campione,马朝晖、陈美红译,Java 语言导学[M],北京 机械工业出版社,2003.170~171。

4 刘正林,Java 技术基础[M],武汉 华中科技大学出版社,2002.461~462。