第 5 章　Java 语言异常处理

面向对象程序设计特别强调软件质量的两个方面：一是程序结构方面的可扩展性与可重用性，二是程序语法与语义方面的可靠性。可靠性(Reliability)是软件质量的关键因素，一个程序的可靠性体现在两方面：一是程序的正确性(Correctness)，指程序的实现是否满足了需求；二是程序的健壮性(Robustness)，指程序在异常条件下的执行能力。在 Java 程序中，由于程序员的疏忽和环境因素的变化，经常会出现异常情况，导致程序运行时的不正常终止。为了及时有效地处理程序运行中的错误，Java 语言在参考 C++ 语言的异常处理方法和思想的基础上，提供了一套优秀的异常处理(Exception Handling)机制，可以有效地预防错误的程序代码或系统错误所造成的不可预期的结果发生。异常处理机制通过对程序中所有的异常进行捕获和恰当的处理来尝试恢复异常发生前的状态，或对这些错误结果做一些善后处理。异常处理机制能够减少编程人员的工作量，增加程序的灵活性，有效地增加程序的可读性和可靠性。

5.1　概述

在程序运行时打断正常程序流程的任何不正常的情况称为错误或异常。导致异常可能发生的原因有许多，例如下列的情形。

- 试图打开的文件不存在。
- 网络连接中断。
- 空指针异常，例如对一个值为 null 的引用变量进行操作。
- 算术异常，例如除数为 0、操作符越界等。
- 要加载的类不存在。

下面是一个简单的程序，程序中声明了一个字符串数组，并通过一个 for 循环将该数组输出。如果不认真地阅读分析程序，一般不容易发现程序中可能导致异常的代码。

【例 5.1】　Java 程序异常示例。

```
01  public class Test {
02      public static void main(String[ ] args) {
03          String friends[ ]={"Lisa", "Mary", "Bily"};
04          for(int i=0;i<4;i++){
05              System.out.println(friends[i]);
06          }
07          System.out.println("Normal ended.");
08      }
09  }
```

【运行结果】

```
Lisa
Mary
Bily
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
    at Test.main (Test.java: 5)
```

【分析讨论】

(1) 程序 Test.java 能够通过编译,但运行时出现了异常,导致程序的非正常终止。

(2) 程序在执行 for 循环语句块时,前 3 次依次输出 String 类型的数组 friends 包含的 3 个元素,即执行结果的前 3 行。但是在第 4 次循环时,由于试图输出下标为 3 的数组元素,而数组的长度为 3,从而导致数组下标越界。产生异常的是第 05 行,异常类型是 java.lang.ArrayIndexOutOfBoundsException,并且系统自动显示了有关异常的信息,指明异常的种类和出错位置。

在 Java 程序中,由于程序员的疏忽和环境因素的变化,会经常出现异常情况。如果不对异常进行处理,就将导致程序的不正常终止。为保证程序的正常运行,Java 语言专门提供了异常处理机制。Java 语言首先针对各种常见的异常定义了相应的异常类,并建立了异常类体系,如图 5.1 所示。

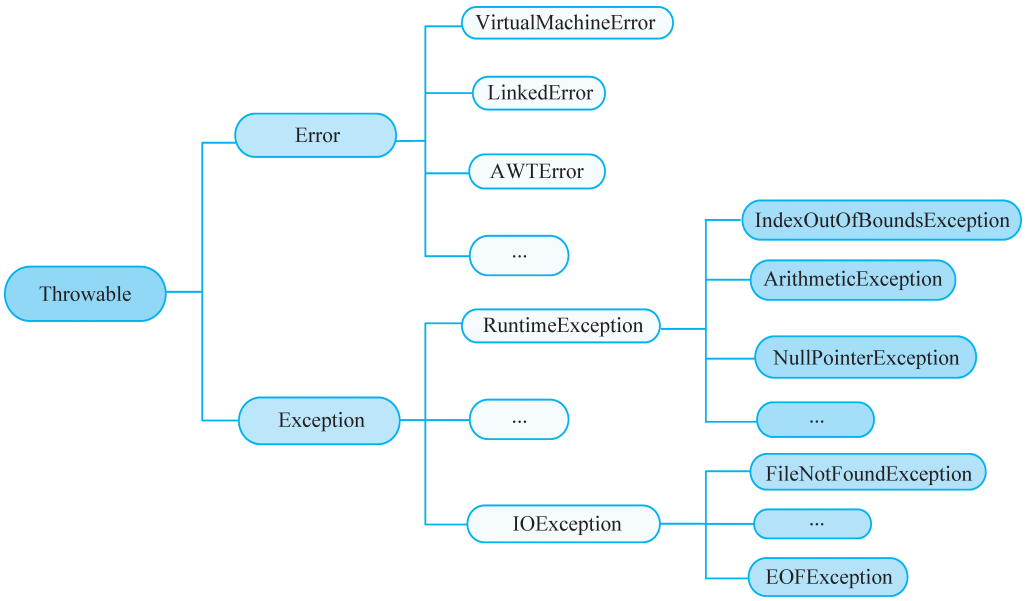

图 5.1 Java 异常类继承层次

其中,java.lang.Throwable 类是所有异常类的父类。Java 语言中只有 Throwable 类及其子类的对象才能由异常处理机制进行处理。该类提供的主要方法包括检索异常相关信息,以及输出显示异常发生位置的堆栈追踪轨迹。Java 语言异常类体系中定义了很多常见的异常,如下所示。

- ArithmeticException:算术异常,整数的除 0 操作将导致该异常的发生,例如,int i=10/0。

- NullPointerException：空指针异常，当对象没有实例化时，就试图通过该对象的变量访问其数据或方法。
- IOException：输入输出异常，即进行输入输出操作时可能产生的各种异常。
- SecurityException：安全异常，一般由浏览器抛出。例如，Applet 在试图访问本地文件、试图连接该 Applet 所来自主机之外的其他主机或视图执行其程序时，浏览器中负责安全控制的 SecurityManager 类都要抛出这个异常。

Java 语言中的异常可分为如下两类。

- 错误(Error)及其子类：JVM 系统内部错误，资源耗尽等很难恢复的严重错误，不能简单地恢复执行，一般不由程序处理。
- 异常(Exception)及其子类：其他因编程错误或偶然的外在因素导致的一般性问题，通过某种修正后程序还能继续运行，其中还可以分为以下两种情况。
  - RuntimeException(运行时异常)：也称为不检查异常，是指因为设计或实现方式不当导致的问题。例如，数组使用越界、算术运算异常、空指针异常等。也可以说是程序员的原因导致的、本来可以避免发生的情况。正确设计与实现的程序不应产生这些异常。对于这类异常，处理的策略是纠正错误。
  - 其他 Exception 类：描述运行时遇到的困难，它通常由环境而非程序员的原因引起，如文件不存在、无效 URL 等。这类异常通常是由用户的误操作引起的，可以在异常处理中进行处理。

例 5.1 中的异常即属于 RuntimeException。出错的原因是数组 friends 中只含有 3 个元素，当 for 循环执行到第 4 次时，试图访问根本不存在的第 4 个数组元素 friends[3]，因此出错。

5.2 异常处理机制

异常处理是指程序获得异常并处理，然后继续程序的执行。Java 语言要求如果程序中调用的方法有可能产生某种类型的异常，那么调用该方法的程序必须采取相应的动作处理异常。异常处理机制具体有两种方式：一是捕获并处理异常；二是将方法中产生的异常抛出。

5.2.1 捕获并处理异常

Java 程序在执行过程中如果出现异常，则会自动生成一个异常对象，该对象包含了有关异常的信息，并被自动提交给 Java 运行时系统，这个过程称为抛出异常。当 Java 运行时系统接收到异常对象时，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，这一过程称为捕获异常。如果 Java 运行时系统找不到可以捕获异常的方法，则运行时系统就将终止，相应的 Java 程序也将退出。

try-catch-finally 语句用于捕获程序中产生的异常，然后针对不同的异常采用不同的处理程序进行处理。try-catch-finally 语句的基本语法如下。

```
try {
    Java statements                          //一条或多条可能抛出异常的 Java 语句
```

```
} catch(ExceptionType1 e) {
    Java statements                          //当 ExceptionType1 类型的异常抛出后要执
行的代码
} catch(ExceptionType2 e) {
    Java statements                          //当 ExceptionType2 类型的异常抛出后要执
行的代码
} [finally {
    Java statements                          //执行最终清理的语句,即无条件执行的语句
} ]
```

try-catch-finally 语句把可能产生异常的语句放入 try{ }语句块中,然后在该语句块后跟一个或多个 catch 语句块,每个 catch 语句块处理一种可能抛出的特定类型的异常。在运行时刻,如果 try{ }语句块产生的异常与某个 catch 语句处理的异常类型相匹配,则执行该 catch 语句块。finally 语句定义了一个程序块,可放于 try{ }和 catch{ }块之后,用于为异常处理提供一个统一的出口,使得在控制流转到程序的其他部分以前,能够对程序的状态进行统一的管理。不论在 try{ }语句块中是否发生了异常事件,finally 块中的语句都会被执行。finally 语句是可选的,可以省略。注意,用 catch 语句进行异常处理时,可以使一个 catch 块捕获一种特定类型的异常,也可以定义处理多种类型异常的通用 catch 块。因为在 Java 语言中允许对象变量上溯造型,父类类型的变量可以指向子类对象,所以如果 catch 语句块要捕获的异常类还存在其子类,则该异常处理块就可以处理该异常类以及其所有子类表示的异常事件。这样一个 catch 语句块就是一个能够处理多种异常的通用异常处理块。例如,在下列语句中,catch 语句块将处理 Exception 类及其所有子类类型的异常,即处理程序能够处理的所有类型的异常。

```
try {
    …
}catch(Exception e) {
    System.out.println("Exception caugth: "+e.getMessage());
}
```

接下来看一看上述机制是如何处理例 5.1 中的问题的。

【例 5.2】 采用 try-catch-finally 对例 5.1 中的 RuntimeException 进行异常处理。

```
01  public class Test2 {
02      public static void main(String[ ] args) {
03          String friends[ ]={"Lisa", "Mary", "Bily"};
04          try{
05              for(int i=0;i<4;i++) {
06                  System.out.println(friends[i]);
07              }
08          }catch(ArrayIndexOutOfBoundsException e) {
09              System.out.println("Index error");
10          }
11          System.out.println("\nNormal ended.");
12      }
13  }
```

【运行结果】

```
Lisa
Mary
Bily
Index error
Normal ended
```

【分析讨论】

(1) 从输出结果中可以看出，出现异常时参数类型匹配的 catch 语句块得到执行，程序输出提示性信息后继续执行，并未异常终止，此时程序的运行仍处于程序员的控制之下。那么，既然运行错误经常发生，是不是所有的 Java 程序也都要采取这种异常处理措施呢？答案是否定的，Java 程序异常处理的原则是：

① 对于 Error 和 RuntimeException，可以在程序中进行捕获和处理，但不是必需的。

② 对于 IOException 及其他异常类，必须在程序中进行捕获和处理。

(2) 例 5.1 和例 5.2 中的 ArrayIndexOutOfBoundsException 即属于 RuntimeException，一个正确设计和实现的程序不会出现这种异常，因此可以根据实际情况选择是否需要进行捕获和处理。而对于 IOException 及其他异常，则属于另外一种必须进行捕获和处理的情况了。

再看一个 IOException 的例子，例 5.3 的类 CreatingList 要创建一个保存 5 个 Integer 对象的数组链表，并通过 copyList 方法将该链表保存到 FileList.txt 文件中。

【例 5.3】　创建链表并保存到文件中(未加任何异常处理，存在编译错误)。

```
01  import java.io.*;
02  import java.util.*;
03  class CreatingList {
04      private ArrayList list;
05      private static final int size=5;
06      public CreatingList() {
07          list=new ArrayList(size);
08          for(int i=0;i<size;i++)
09              list.add(new Integer(i));
10      }
11      //将 list 保存到 FileList.txt 文件中
12      public void copyList() {
13          BufferedWriter bw=new BufferedWriter(new FileWriter("FileList.txt"));
14          for(int i=0;i<size;i++) {
15              bw.write("Value at: "+i+" = "+list.get(i));
16              bw.newLine();
17          }
18          bw.close();
19      }
20  }
21  public class ListDemo1 {
22      public static void main(String[ ] args) {
23          CreatingList clist=new CreatingList ();
24          clist.copyList();
25      }
26  }
```

【编译结果】

```
ListDemo1.java: 13: 未报告的异常 java.io.IOException; 必须对其进行捕捉或声明以便抛出 BufferedWriter bw=new BufferedWriter(new FileWriter("FileList.txt"));
1 错误
```

【分析讨论】

例 5.3 的第 13 行语句中调用了 java.io.FileWriter 的构造方法创建了一个文件输出流。该构造方法的声明如下：public FileWriter（String fileName）throws IOException。由于 copyList()方法中没有对 FileWriter 构造方法可能产生的异常进行处理，所以程序在编译时产生了上述错误。

例 5.4 在例 5.3 中加入了异常处理机制。将例 5.3 中的第 13～18 行语句放入 try 语句块中，用两个 catch 语句分别捕获 FileWriter("FileList.txt")调用中可能产生的 IOException 异常，以及 for 循环访问链表的 list.get(i)方法时可能产生的 ArrayIndexOutOfBoundsException 异常。try-catch 语句还有 finally 语句，执行程序的最后清理操作，此处是关闭程序打开的流。

【例 5.4】 对例 5.3 增加 try-catch-finally 异常处理。

```
01  import java.io.*;
02  import java.util.*;
03  class CreatingList {
04      private ArrayList list;
05      private static final int size=5;
06      public CreatingList () {
07          list=new ArrayList(size);
08          for(int i=0;i<size;i++)
09              list.add(new Integer(i));
10      }
11      public void copyList() {
12          BufferedWriter bw=null;
13          try {
14              System.out.println("Catching Exceptions");
15              bw=new BufferedWriter(new FileWriter("FileList.txt"));
16              for(int i=0;i<size;i++) {
17                  bw.write("Value at: "+i+" = "+list.get(i));
18                  bw.newLine();
19              }
20              bw.close();
21          }catch(ArrayIndexOutOfBoundsException e) {       //处理数组越界异常
                System.out.println("Caught ArrayIndexOutOfBoundsException. ");
22          }catch(IOException e) {                          //处理 I/O 异常
                System.out.println("Caught IOException.");
24          }
25          System.out.println("Closing BufferedWriter, Normal Ended! ");
26      }
27  }
28  public class ListDemo2 {
29      public static void main(String[ ] args) {
30          CreatingList clist=new CreatingList ();
31          clist.copyList();
```

```
32      }
33  }
```

【运行结果】

```
Catching Exceptions
Closing BufferedWriter, Normal Ended!
```

5.2.2　将方法中产生的异常抛出

将方法中产生的异常抛出是 Java 语言处理异常的第二种方式。如果一个方法中的语句执行时可能生成某种异常，但是并不能或不确定如何处理这种异常，则该方法应声明抛出该种异常，表明该方法将不对此类异常进行处理，而由该方法的调用者负责处理。

1. 使用 throws 关键字抛出异常

将异常抛出，可通过 throws 关键字来实现。throws 关键字通常被应用在声明方法时，用来指定方法可能抛出的异常，其语法格式如下：

```
<modifer> <returnType> methodName ([<argument list>]) throws <exceptionList>
```

其中，exceptionList 可以包含多个异常类型，用逗号隔开。

例 5.4 是使用 try-catch-finally 语句实现例 5.3 的异常处理的。下面的例 5.5 是采用异常处理的第二种方式对例 5.3 进行的改进。

【例 5.5】　采用声明抛出异常的方法对例 5.3 进行异常处理。

```
01  import java.io.*;
02  import java.util.*;
03  class CreatingList{
04    private ArrayList list;
05    private static final int size=5;
06    public CreatingList() {
07      list=new ArrayList(size);
08      for(int i=0;i<size;i++)
09        list.add(new Integer(i));
10      }
11      //声明抛出异常
12      public void copyList() throws IOException, ArrayIndexOutOfBounds Exception {
13      BufferedWriter bw=new BufferedWriter(new FileWriter("FileList.txt"));
14        for(int i=0;i<size;i++){
15          bw.write("Value at: "+i+" = "+list.get(i));
16          bw.newLine();
17        }
18        bw.close();
19      }
20  }
21  public class ListDemo3{
22    public static void main(String[ ] args){
23      try{
24          CreatingList clist=new CreatingList();
25          clist.copyList();
26      } catch(ArrayIndexOutOfBoundsException e) {          //处理数组越界异常
                System.out.println("Caught ArrayIndexOutOfBoundsException.");
```

```
27      }catch(IOException e) {                                    //处理 I/O 异常
28          System.out.println("Caught IOException.");
29      }
30      System.out.println("A list of numbers is created and stored in FileList.txt");
31    }
32  }
```

【运行结果】

```
A list of numbers is created and stored in FileLIst.txt
```

【分析讨论】

如果被抛出的异常在调用程序中未被处理,则该异常将被沿着方法的调用关系继续上抛,直到被处理。如果一个异常返回到 main()方法,并且在 main()方法中还未被处理,则该异常将把程序非正常地终止。

2. 使用 throw 关键字抛出异常

使用 throw 关键字也可抛出异常,与 throws 不同的是,throw 用于方法体内,并且抛出一个异常类对象,而 throws 用在方法声明中来指明方法可能抛出的多个异常。

通过 throw 抛出异常后,如果想由上一级代码来捕获并处理异常,则同样需要在抛出异常的方法中使用 throws 关键字在方法的声明中指明要抛出的异常;如果想在当前的方法中捕获并处理 throw 抛出的异常,则必须使用 try-catch-finally 语句。throw 语句的一般格式如下:

```
throw someThrowableObject;
```

其中,someThrowableObject 必须是 Throwable 类或其子类的对象。执行 throw 语句后,运行流程立即停止,throw 的下一条语句将暂停执行,系统转向调用者程序,检查是否有与 catch 子句能匹配的 Throwable 实例对象。如果找到相匹配的实例对象,系统转向该子句;如果没有找到,则转向上一层的调用程序。这样逐层向上,直到最外层的异常处理程序终止程序并打印出调用栈的情况。

例如,当输入的年龄为负数时,Java 虚拟机当然不会认为这是一个错误,但实际上年龄是不能为负数的,可以通过异常的方式来处理这种情况。例 5.6 中创建 People 类,该类中的 check()方法首先将传递进来的 String 型参数转换为 int 型,然后判断该 int 型整数是否为负数,若为负数则抛出异常;然后在该类的 main()方法中捕获异常并处理。

【例 5.6】 throw 关键字的使用示例。

```
01  public class People {
02    public static int check(String strage) throws Exception {
03      int age=Integer.parseInt(strage);    //转化字符串为 int 型
04      if(age<0)                          //如果 age 小于 0,则抛出一个 Exception 异常对象
05        throw new Exception("年龄不能为负数!");
06        return age;
07      }
08      public static void main(String args[ ]) {
09        try {
10            int myage=check("-101");        //调用 check()方法
11            System.out.println(myage);
```

```
12          } catch(Exception e) {              //捕获 Exception 异常
13            System.out.println("数据逻辑错误!");
14            System.out.println("原因:"+e.getMessage());
15          }
16      }
17  }
```

【运行结果】

```
数据逻辑错误!
原因:年龄不能为负数!
```

【分析讨论】

在 check()方法中将异常抛给了调用者(main()方法)进行处理。check()方法可能会抛出以下两种异常:

- 数字格式的字符串转换为 int 型时抛出的 NumberFormatException 异常;
- 当年龄小于 0 时抛出的 Exception 异常。

5.3 自定义异常类

通常使用 Java 内置的异常类就可以描述在编写程序时出现的大部分异常情况。但有时仍然需要根据需求创建自己的异常类,并将它们用于程序中来描述 Java 内置异常类所不能描述的一些特殊情况。下面就来介绍如何创建和使用自定义异常类。

5.3.1 必要性与原则

Java 语言允许用户在需要时创建自己的异常类型,用于表述 JDK 中未涉及的其他异常状况,这些类型也必须继承 Throwable 类或其子类。Throwable 类有两种类型的子类,即错误(Error)和异常(Exception)。大多数 Java 应用都抛出异常,错误是指系统内部发生的严重错误。所以,一般自定义异常类都以 Exception 类为父类。由于用户自定义异常类通常属于 Exception 范畴,因此依据命名惯例,自定义异常类的名称应以 Exception 结尾。用户自定义异常类未被加入 JRE 的控制逻辑中,因此永远不会自动抛出,只能由人工创建并抛出。

例如,假设我们要编写一个可重用的链表类,该类中可能包括如下方法。

- objectAt(int n):返回链表中的第 n 个对象。
- firstObject():返回链表中的第一个对象。
- indexOf(Object n):在链表中搜索指定的对象,并返回它在链表中的位置。

对于这个链表类,在其他程序员使用时可能出现对类及方法使用不当的情况,并且即使是合法的方法调用,也有可能导致某种未定义的结果。因此希望这个链表类在出现错误时尽量强壮,对于错误能够合理地处理,并把错误信息报告给调用程序。但是由于不能预知使用该类的每个用户打算如何处理特定的错误,所以在发生一种错误时最好的处理办法是抛出一个异常。上述链表类的每个方法都有可能抛出异常,并且这些异常可能是互不相同的,例如下面的异常。

- objectAt(int n)：如果传递给该方法的参数 n 小于 0，或 n 大于链表中当前含有的对象的数目，则抛出一个异常。
- firstObject()：如果链表中不包含任何对象，则抛出一个异常。
- indexOf(Object n)：如果传递给该方法的对象不在链表中，则抛出一个异常。

通过上述分析，我们可以知道这个链表类运行中会抛出的各种异常，但是，如何确定这些异常的类型？是选择 Java 异常类体系中的一个类型，还是自己定义一种新的异常类型？下面给出一些原则，提示读者何时需要自定义异常类。满足下列任何一种或多种情形就应该考虑自己定义异常类：

- Java 异常类体系中不包含所需要的异常类型。
- 用户需要将自己所提供类的异常与其他人提供类的异常进行区分。
- 类中将多次抛出这种类型的异常。
- 如果使用其他程序包中定义的异常类，将影响程序包的独立性与自包含性。

结合上面提到的链表的例子，该链表类可能抛出多种异常，用户可能需要使用一个通用的异常处理程序对这些异常进行处理。另外，如果要把这个链表类放在一个包中，那么与该类相关的所有代码应该同时放在这个包中。因此，应该定义自己的异常类并且创建自己的异常类层次。图 5.2 给出了链表类的一种自定义异常类的层次。

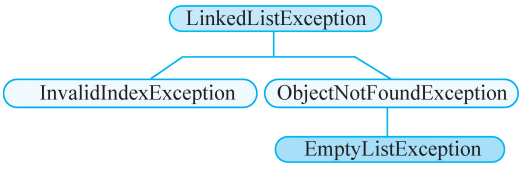

图 5.2　链表类的自定义异常类层次

LinkedListException 是链表类可能抛出所有异常类的父类。将来可以使用下列 catch 语句对该链表类的所有异常进行统一的处理：

```
catch(LinkedListException) {
    …
}
```

当然，用户也可以编写针对 LinkedListException 子类的专用的异常处理。但是，链表类的异常能够利用上述 Java 异常处理机制进行处理，还必须将这些异常类与 Java 的异常类体系融合起来。

5.3.2　定义与使用

创建自定义异常类并在程序中使用，大体可分为以下几个步骤：

(1) 创建自定义异常类。

(2) 在方法中通过 throw 抛出异常对象。

(3) 若在当前抛出异常的方法中处理异常，可使用 try-catch-finally 语句捕获并处理；否则在方法的声明处通过 throws 指明要抛出给方法调用者的异常，继续进行下一步操作。

(4) 在出现异常的方法调用代码中捕获并处理异常。

如果自定义的异常类继承自 RuntimeException 异常类，在步骤(3)中，可以不通过