第 9 章　异 常 处 理

要编写一个完善的程序并不只是简简单单地把程序的功能实现，它要能处理程序运行中出现的各种意外情况，也就是所谓的异常。Java 提供了一套完善的异常处理机制，通过这套机制，可以轻松地写出容错性非常高的代码。本章主要对 Java 的异常处理机制进行讲解。

9.1　异常基本知识

由于硬件问题、资源耗尽、输入错误以及程序代码编写不严谨会导致程序运行时产生异常，这些异常会导致程序中断而不能正常继续运行，所以需要对异常的情况进行妥善处理。本节主要讲解异常的基本知识。

9.1.1　什么是异常

Java 的异常实际上是一个对象，这个对象描述了代码中出现的异常情况。在代码运行异常时，在有异常的方法中创建并抛出一个表示异常的对象，然后在相应的异常处理模块中进行处理。示例如下。

```
public class ExceptionDemo{
   public static void main(String args[ ]){
       String str=null;                          //字符串的内容为 null
       int strLength=str.length();               //获取字符串的长度
       System.out.println("strLength 的长度："+strLength);
   }
}
```

这段代码是得到一个字符串的长度并把它打印出来，但是字符串并没有被实例化而是只有一个 null 值，代码编译是没有问题的，但是运行程序时却不能正常运行，它会产生如下错误。

```
Exception in thread "main" java.lang.NullPointerException
        at ExceptionDemo.main(ExceptionDemo.java:4)
```

由于程序中并没有给 str 一个实例，所以使用 String 类型的方法就是不合理的，它会抛出一个 NullPointerException 异常。它描述了程序的异常以及异常抛出的地点，在 main 方法中 ExceptionDemo.java 的第 4 行中。另外一个示例如下。

```
public class ExceptionDemo1{
       public static void printLength(){
            String str=null;                     //设置字符串的内容为 null
       int strLength=str.length();               //获取字符串的长度
       System.out.println("strLength 的长度："+strLength);
```

```
    }
  public static void main(String args[ ]){
      printLength();
  }
}
```

程序编译通过，运行会产生如下错误。

```
Exception in thread "main" java.lang.NullPointerException
        at ExceptionDemo1.printLength(ExceptionDemo1.java:4)
        at ExceptionDemo1.main(ExceptionDemo1.java:8)
```

在打印出的异常信息中给出了异常的信息，在第 4 行中，是第 8 行调用该方法的时候产生的异常，给出的异常信息方便了程序的调试。

9.1.2 异常的分类

Java 中，所有的异常类都是内置类 Throwable 的子类，所以 Throwable 在异常类型层次结构的顶部。Throwable 有两个子类，这两个子类把异常分成两个不同的分支：一个是 Exception，另一个是 Error，如图 9-1 所示。

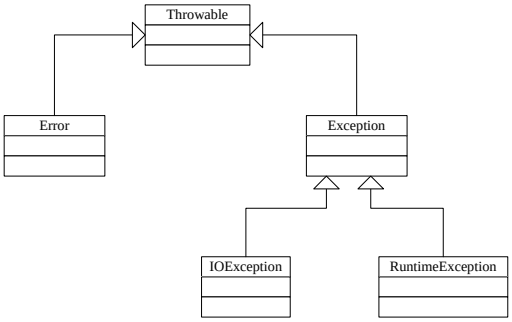

图 9-1 异常的层次结构

一个分支是 Error，它定义了 Java 运行时的内部错误。通常用 Error 类来指明与运行环境相关的错误。应用程序不应该抛出这类异常，发生这类异常时通常是无法处理的。

另一分支是 Exception，它用于程序中应该捕获的异常。Exception 类也有两个分支，一个是 RuntimeException，另一个分支是 IOException。

RuntimeException 主要用来描述由于编程产生的错误，例如前面程序演示的 NullPointerException，通过 API 可以查看它的层次，即它是 RuntimeException 的一个子类，当应用程序试图在需要对象的地方使用 null 时，抛出该异常。RuntimeException 还包括其他的一些子类。

1．IndexOutOfBoundsException

表示某排序索引（例如对数组、字符串或向量的排序）超出范围时抛出的异常。

```
int array1={1,2,3};
array1[3]=9;
```

长度为 3 的数组索引为 0、1、2，当试图访问 array1[3]时抛出异常。

2．ArithmeticException

当出现异常的运算条件时，抛出此异常。例如一个整数“除以零”时，抛出此类的一个实例。

除了以上的异常外，还有其他一系列的异常在这里不再一一列举。对于 Exception 类，除 RuntimeException 以外其他的类在程序中出现是必须被处理的，而不像 RuntimeException 中的异常会由系统自己处理。

对异常还可以有如此分类，RuntimeException 类的子类通常指一些运行时错误引起的异常，所以可以不处理它，而 Error 错误是无法处理的，所以把它们统称为未检查异常；其他的异常类则被称为检查异常。

9.2　异常的使用

前面介绍了异常的基本知识，这一节主要介绍 Java 中异常的基本使用，包括捕获异常、finally 语句块、printStatckTrace()方法以及方法中异常的抛出等。

9.2.1　异常捕获

知道什么是异常和有哪些异常后，现在就来学习一下如何进行异常处理。对异常的处理包括很多种，先来学习一下异常捕获。Java 使用 try-catch 语句块来进行异常的捕获，Java 中严格规定了该语句块的使用格式，它的基本使用格式如下。

```
try{
   //可能出现异常代码
}
catch(异常类型 1  异常对象)
{
   //对异常 1 的处理
}
catch(异常类型 2  异常对象)
{
   //对异常 2 的处理
}
 ┆
catch(异常对象 n  异常对象)
{
   //异常对象 n 的处理
}
```

在 try-catch 语句中捕获并处理异常，把可能出现异常的语句放入 try 语句块中，紧接在 try 语句块后的是各个异常的处理模块。

try 语句块只能有一个，而 catch 则可以有多个，catch 必须紧跟 try 语句后，中间不能有其他的任何代码。

```
public class UseTryCatchDemo{
```

```
  public static void main(String args[ ]){
      String str=null;
      int strLength=0;
      try
      {
        strLength=str.length();
      }
      catch(NullPointerException e)
      {
       System.out.println("在求字符串长度的时候产生异常");
      }
       System.out.println("strLength 的长度："+strLength);
  }
  }
```

程序的运行结果如下。

```
在求字符串长度的时候产生异常
strLength 的长度：0
```

在程序中，把求字符串长度的语句放在了 try 语句中，catch 语句处理 NullPointerException，继续执行打印语句。注意前一节的程序中，没有异常处理语句的时候并不执行最后的打印语句，而在上面的程序中，经过异常处理，可以看到程序正常结束。

当有多个 catch 语句的时候，程序运行时会自动根据抛出的异常类型去调用相应的模块，这样可以准确的处理出现的异常。示例如下。

```
public class UseTryCatchDemo2{
  public static void main(String args[ ]){
      int[ ]array1={1,2,3};
      try
      {
        array1[3]=5;
      }
      catch(NullPointerException e)
      {
       System.out.println("在求字符串长度的时候产生空指针异常");
      }
      catch(ArrayIndexOutOfBoundsException e)
      {
       System.out.println("出现的错误是数组越界异常");
      }
      System.out.println("程序正常退出");
  }
  }
```

程序的运行结果如下。

```
出现的错误是数组越界异常
程序正常退出
```

在程序中紧跟着 try 语句有两个 catch 语句块，分别捕获数组越界异常 ArrayIndexOutOfBoundsException 和空指针异常 ArrayIndexOutOfBoundsException。由于在 try 语句中访问数组越界，所以会抛出一个数组越界异常，程序运行的时候会自动地根据该异常调用捕获数组越界异常的 catch 语句块。

需要注意的是，如果有多个 catch 语句，那么捕获父类异常的 catch 语句必须放在后面，

否则它会捕获它的所有子类异常，而使得子类异常 catch 语句永远不会执行。这在 Java 中是被当作编译错误处理，示例如下。

```
public class CatchDemo{
  public static void main(String args[ ]){
     try
     {
       int a=15/0;
     }
     catch(Exception e)
     {
       System.out.println("捕获 Exception 异常");
     }
     catch(ArithmeticException e)
     {
       System.out.println("捕获 ArithmeticException 异常");
     }
  }
}
```

程序编译后报错如下。

```
CatchDemo.java:11: 已捕捉到异常 java.lang.ArithmeticException
     catch(ArithmeticException e)
```

因为 Exception 是 ArithmeticException 类的父类，致使第二个 catch 语句成为不可达的语句。需要把二者调换，才能正常编译运行。

```
public class CatchDemo{
  public static void main(String args[ ]){
     try
     {
       int a=15/0;
     }
       catch(ArithmeticException e)
     {
       System.out.println("捕获 ArithmeticException 异常");
     }
     catch(Exception e)
     {
       System.out.println("捕获 Exception 异常");
     }
  }
}
```

程序的运行结果如下。

```
捕获 ArithmeticException 异常
```

9.2.2　printStackTrace()方法：获取异常的堆栈信息

对异常处理进行简单介绍后，来学习几个非常重要的方法。在异常类中有一个非常重要的方法 printStackTrace()，该方法将在 Throwable 类中定义，它的作用是把 Throwable 对象的堆栈信息输出至输出流。示例如下。

```
public class TestprintStackTraceDemo{
```

```
  public static void main(String args[ ]){
      method1();                    //调用 method1 方法
  }
  static void method1()
  {
      method2();                    //调用 method2 方法
  }
  static void method2()
  {
      method3();                    //调用 method3 方法
  }
  static void method3()
  {
      String str=null;              //字符串的值为 null
      int n=str.length();           //获取字符串的长度
  }
}
```

程序的运行结果如下。

```
Exception in thread "main" java.lang.NullPointerException
        at TestprintStackTraceDemo.method3(TestprintStackTraceDemo.java:16)
        at TestprintStackTraceDemo.method2(TestprintStackTraceDemo.java:11)
        at TestprintStackTraceDemo.method1(TestprintStackTraceDemo.java:7)
        at TestprintStackTraceDemo.main(TestprintStackTraceDemo.java:3)
```

另一个示例如下。

```
public class TestprintStackTraceDemo{
  public static void main(String args[ ]){
      try                           //对 method1 方法进行异常处理
      {
      method1();                    //调用 method1 方法
     }
  catch(NullPointerException e)
  {
      e.printStackTrace();          //获取异常信息
  }
  }
  static void method1()
  {
      method2();                    //调用 method2 方法
  }
  static void method2()
  {
      method3();                    //调用 method3 方法
  }
  static void method3()
  {
      String str=null;              //字符串的值为 null
      int n=str.length();           //获取字符串的长度
  }
}
```

程序的运行结果如下。

```
java.lang.NullPointerException
        at TestprintStackTraceDemo.method3(TestprintStackTraceDemo.java:23)
        at TestprintStackTraceDemo.method2(TestprintStackTraceDemo.java:18)
```

```
        at TestprintStackTraceDemo.method1(TestprintStackTraceDemo.java:14)
        at TestprintStackTraceDemo.main(TestprintStackTraceDemo.java:5)
```

先看第二个程序，在 try 语句后面的 catch 语句块中有如下语句。

```
e.printStackTrace();
```

它打印出了异常的栈信息。比较前一个程序，可以发现二者的输出是一样的，当然程序的行数不同，并且第二个程序是正常退出，但是二者的异常输出信息是相同的。原因是 NullPointerException 为未检查异常，并不是一定需要捕获的，Java 会处理它。示例如下。

```
public class DBDemo {
    public static void main(String[ ] args) {
            Class.forName("");
        }
}
```

程序在编译的时候会发生如下错误。

```
DBDemo.java:4: 未报告的异常 java.lang.ClassNotFoundException；必须对其进行捕捉或
声明以便抛出
                        Class.forName("");
                                      ^
1 错误
```

ClassNotFoundException 是检查异常，必须对其进行捕获异常，示例如下。

```
public class DBDemo {
    public static void main(String[ ] args) {
        try
        {
            Class.forName("");
        }
        catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

程序编译通过，运行时屏幕显示如下。

```
java.lang.ClassNotFoundException:
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Unknown Source)
        at DBDemo.main(DBDemo.java:5)
```

catch 语句捕获了产生的异常，并把异常信息打印出来。

9.2.3　finally 语句块

在出现异常的时候，异常处理之前的代码不会被调用。示例如下。

```
public class SimpleDemo {
    public static void main(String[ ] args) {
        String str=null;
        int strLength=0;
        try{
            strLength=str.length();
            System.out.println("出现异常语句之后");
```

```
            }
            catch(NullPointerException e){
                e.printStackTrace();
            }
            System.out.println("程序退出");
        }
}
```

程序的运行结果如下。

```
java.lang.NullPointerException
        at SimpleDemo.main(SimpleDemo.java:6)
程序退出
```

可以看到 try 语句块中后面的打印语句并没有执行。在 Java 中就是这样，出现异常的时候会跳出当前运行的语句块，找到异常捕获语句块，然后再跳回程序中执行 catch 后的语句。

有的时候有些语句是必须执行的，例如连接数据库的时候在使用完后必须对连接进行释放，否则系统会因为资源耗尽而崩溃。对于这些必须要执行的语句，Java 提供了 finally 语句块来执行。

finally 语句块是异常捕获里的重要语句，它规定的语句块无论如何都要执行。在一个 try-catch 中只能有一个 finally 语句块。示例如下。

```
public class SimpleDemo {
    public static void main(String[ ] args) {
            String str=null;
            int strLength=0;
            try{
                strLength=str.length();
                System.out.println("出现异常语句之后");
            }
            catch(NullPointerException e){
                e.printStackTrace();
            }
            finally
            {
                System.out.println("执行 finally 语句块");
            }
            System.out.println("程序退出");
        }
}
```

程序的运行的结果如下。

```
java.lang.NullPointerException
        at SimpleDemo.main(SimpleDemo.java:6)
执行 finally 语句块
程序退出
```

一般情况下 finally 语句块一般放在最后一个 catch 语句块后，不管程序是否抛出异常，它都会执行。示例如下。

```
public class SimpleDemo {
    public static void main(String[ ] args) {
            String str=null;
            int strLength=0;
            try{
                //strLength=str.length();
```

```
                //System.out.println("出现异常语句之后");
            }
            catch(NullPointerException e){
                e.printStackTrace();
            }
            finally
            {
                System.out.println("执行 finally 语句块");
            }
            System.out.println("程序退出");
        }
}
```

程序把本类抛出异常的语句都注释了起来，所以它们不会被执行，程序的运行结果如下。

```
执行 finally 语句块
程序退出
```

可以看到，虽然程序没有抛出异常，但是 finally 语句块仍然被执行。

9.2.4　方法抛出异常

有一种说法是，最好的异常处理是什么都不做。这样说并不是任由系统自己处理出现的错误，而是说把出现的异常留给用户自己处理。例如写一个方法，这个方法有抛出异常的可能性，最好的办法是把对异常的处理工作留给方法的调用者，因此需要在方法中定义要抛出的异常。

1．通过 throws 抛出异常

Java 方法是使用 throws 关键字来抛出异常的。看下面的程序。

```
public class ThrowsDemo {
    static void method()throws NullPointerException,
        IndexOutOfBoundsException,ClassNotFoundException{
         String str=null;
         int strLength=0;
         strLength=str.length();
         System.out.println(strLength);
    }
    public static void main(String[ ] args) {
         try {
             method();
         } catch (NullPointerException e) {
             System.out.println("NullPointerException 异常");
             e.printStackTrace();
         } catch (IndexOutOfBoundsException e) {
             System.out.println("IndexOutOfBoundsException 异常");
             e.printStackTrace();
         } catch (ClassNotFoundException e) {
             System.out.println("ClassNotFoundException 异常");
             e.printStackTrace();
         }
    }
}
```

程序的运行结果如下。

```
NullPointerException 异常
java.lang.NullPointerException
        at ThrowsDemo.method(ThrowsDemo.java:5)
        at ThrowsDemo.main(ThrowsDemo.java:10)
```

在程序中声明了一个方法，它定义了可能抛出三种异常。定义抛出异常的的一般格式如下。

```
修饰符 返回值类型 方法名() throws 异常类型 1,异常类型 2{
    //方法体
}
```

这种抛出异常的方式是称为隐式抛出异常。还有一种方式称为显示抛出异常，它是通过 throw 关键字来实现的。

2．用 throw 再次抛出异常

throw 适用于异常的再次抛出。异常的再次抛出是指当捕获到异常的时候并不对它直接处理而是把它抛出，留给上一层的调用来处理。示例如下。

```
public class ThrowDemo {
    static void connect() throws ClassNotFoundException {
      try{
            Class.forName("");
      }catch(ClassNotFoundException e){
            System.out.println("方法中把异常再次抛出");
            throw e;
      }
}
      public static void main(String[ ] args) {
            try {
                  connect();
            } catch (ClassNotFoundException e) {
                  System.out.println("主方法对异常进行处理");
            }
      }
}
```

程序的运行结果如下。

```
方法中把异常再次抛出
主方法对异常进行处理
```

可以看到，在方法中程序并没有对异常进行处理而是把它抛出，在主方法调用的时候必须放在 try-catch 语句块中，捕获上面抛出的异常并进行处理。

在上面的程序 ThrowsDemo 中定义了 3 种可能抛出的异常，那么在调用该方法的时候就必须要把方法调用语句放入 try-catch 语句块中，并在 catch 中捕获相应的异常。注意如果只是定义抛出前两种的话，因为它们都是非检查异常，所以在调用的时候可以不放在 try-catch 语句块中，但是如果方法抛出检查异常，则必须要放入 try-catch 语句块中。

3．非检查异常和检查异常使用 throws、throw 的区别

对于未检查异常，在方法抛出时可以不用 throws 来声明，而检查异常则必须在 throws 声明后才能进行 throw 抛出异常。一个示例如下。

```
public class ThrowsDemo1{
```

```
 static int amethod(int a,int b) //throws ArithmeticException
 {
       if(b==0)
          throw new ArithmeticException();
       else
          return a/b;
 }
 public static void main(String args[ ]){
        System.out.println(amethod(15,5));
        System.out.println(amethod(15,0));
   }
}
```

这段程序是没有问题的，因为在方法 amethod 中抛出的是 ArithmeticException，它是一个非检查异常。程序的运行结果如下。

```
3
Exception in thread "main" java.lang.ArithmeticException
            at ThrowsDemo1.amethod(ThrowsDemo1.java:5)
            at ThrowsDemo1.main(ThrowsDemo1.java:11)
```

另一个示例如下。

```
public class ThrowsDemo2{
 static void amethod()
 {
       System.out.println("方法内抛出异常");
       throw new IllegalAccessException();
 }
 public static void main(String args[ ]){
        amethod();
   }
}
```

在编译程序时会报错，如下所示。

```
ThrowsDemo2.java:6: 未报告的异常 java.lang.IllegalAccessException；必须对其进行
捕捉或声明以便抛出
            throw new IllegalAccessException();
            ^
1 错误
```

因为 IllegalAccessException 是一个检查异常，在方法中抛出的时候必须对它进行声明。该程序需要改进两点，在方法中声明可能抛出的异常，在调用该方法时需要放入 try-catch 中。正确的程序如下。

```
public class ThrowsDemo2{
 static void amethod() throws IllegalAccessException
 {
       System.out.println("方法内抛出异常");
       throw new IllegalAccessException();
 }
 public static void main(String args[ ]){
       try
       {
             amethod();
       }
       catch(IllegalAccessException e)
        {
```

```
                System.out.println("捕获到异常");
            }
        }
    }
```

程序的运行结果如下。

```
方法内抛出异常
捕获到异常
```

9.3 定义自己的异常

一般情况下 Java 本身提供的异常能处理大多数的错误，但是有时候还是需要创建自己的异常类来处理一些错误。这一节主要介绍如何创建使用自己的异常类。通过本节的学习，读者可以创建自己的异常类来处理一些问题。

9.3.1 创建自己的异常类

创建自己的异常类很简单，只需要继承 Exception 类并实现一些方法即可。它的一般形式如下。

```
class 类名 extends Exception
{
  //类体
}
```

查看 Java 的 API 可以发现 Exception 并没有定义任何方法。它从 Throwable 继承了一些方法，所以创建自定义的异常类时可以继承 Throwable 中的方法。Throwable 主要的方法有以下几种。

- public Throwable fillInStackTrace()：返回包含一个完全堆栈追踪的 Throwable 对象，这个对象可以被再次抛出。
- public Throwable getCause()：返回此 throwable 的 cause。如果 cause 不存在或未知，则返回 null。
- public String getMessage()：返回此 throwable 的详细消息字符串。
- public StackTraceElement[]getStackTrace()：返回一个包含异常堆栈追踪的数组，每个元素表示一个堆栈帧。数组的第零个元素（假定数据的长度为非零）表示堆栈顶部，它是序列中最后的方法调用。通常，这是创建和抛出该 throwable 的地方。数组的最后元素（假定数据的长度为非零）表示堆栈底部，它是序列中第一个方法调用。
- public void printStackTrace()：将异常堆栈追踪输出到标准错误流。
- public void printStackTrace(PrintStream s)：将此 throwable 及其追踪输出到指定的输出流。
- public void printStackTrace(PrintWriter s)：将此 throwable 及其追踪输出到指定的 PrintWriter。

- public String toString()：返回一个包含异常描述的 String 对象。

这些方法会被继承，也可以在创建的异常类中重写这些方法。下面是一个简单的自定义异常类。

```
public class MyException extends Exception{
        MyException()
        {
        }
        MyException(String msg)
        {
                //调用父类的方法
                super(msg);
        }
}
```

测试程序如下。

```
public class TestMyException{
  public static void main(String args[ ]){
    //创建自己的异常类对象
    MyException mec=new MyException("这是我自定义的异常类");
    System.out.println(mec.getMessage());
    System.out.println(mec.toString());
  }
}
```

程序的运行结果如下。

```
这是我自定义的异常类
MyException: 这是我自定义的异常类
```

9.3.2　使用自己的异常类

上一小节介绍了如何创建自己的异常类，这小节将讲解如何使用自己创建的异常类。示例如下。

```
class IllegalScoreException extends Exception{
  IllegalScoreException()
  {
    }
    IllegalScoreException(String msg)
  {
        //调用父类的构造方法
        super(msg);
    }
}
public class Score{
  public static void main(String args[ ])
  {
          try
          {
        String level=null;
        level=scoreLevel(90);
        System.out.println("90 分的成绩等级为："+level);
        level=scoreLevel(120);
```

```
        System.out.println("120 分的成绩等级为："+level);
          }
        catch(IllegalScoreException e)
        {
              e.printStackTrace();
              }
        }
        static String scoreLevel(int score) throws IllegalScoreException
        {
              if(score>=85&&score<=100)
                return "A";
              else if(score>=75&&score<85)
                return "B";
              else if(score>=60&&score<75)
                return "C";
              else if(score<60&&score>=0)
                return "D";
              else
                 throw new IllegalScoreException("非法的分数");
        }
}
```

程序首先定义了一个自定义异常类 IllegalScoreException。Score 类是对分数进行分级别的程序，对各个分数段的成绩分别分为不同的等级。但是由于分数有一定的范围，当分数超出范围时应该产生异常，使用前面定义的异常来处理。在程序中应该注意以下几点。

- 方法定义时要在方法中声明可能产生的异常。
- 调用方法的时候要放在 try-catch 语句中。
- catch 语句要对 IllegalScoreException 异常进行捕获。

程序的运行结果如下。

```
90 分的成绩等级为：A
IllegalScoreException: 非法的分数
        at Score.scoreLevel(Score.java:38)
        at Score.main(Score.java:19)
```

9.4 小结

在现实世界中，程序的运行不可能完全地符合程序编写者的意愿，所以程序必须能对各种情况进行合理的处理才能健壮的运行，这对程序的编写提出了很高的要求。

Java 提供的异常机制可以很方便地处理程序运行中可能产生的各种异常，并且程序员可以编写自己的异常类处理，以应对程序可能会遇到的各类异常。

在本章中首先介绍了什么是异常及 Java 异常机制的基本体系结构，然后又介绍了 Java 中异常机制的使用，并对自定义异常类进行了简单讲解。

正确的处理各类异常是非常复杂的事情，读者应该在程序编写过程中细细体会才能熟练的掌握，本章只是对 Java 异常机制进行了简单介绍，还希望读者能在实践中学习，加深对 Java 异常机制的认识。