第5章　异常处理

教室里，李老师正在给同学们讲课。

我们知道，讲授一节课的正常流程是

```
教师宣布开始上课;
DO
  教师在讲台上讲解;
  板书;
WHILE 下课时间未到
教师宣布下课;
```

李老师正是按照这个流程讲到了一半。

现在李老师想写板书，在黑板上画一个矩形，但是找不到粉笔了。

“找不到粉笔”就是一个异常。

发生异常后，讲课的流程暂时中止，李老师进行异常处理。

李老师到隔壁教室找来一支粉笔。

然后继续上课。

程序的运行也是一样：一般情况下按照正常的流程运行，但总会发生异常。

程序设计时就需要考虑对异常的处理。

5.1　概　　述

异常(exception)是程序在执行期间出现的事件，这个事件打破了程序的正常执行流程。

异常是异常事件(exceptional event)的简称。

Java 的程序是由 Java 虚拟机(JVM)解释执行的。Java 程序把异常事件作为对象传递给 JVM，这个对象就称为异常对象。这个过程称为抛出异常(throwing an exception)。

异常对象中包含了错误信息以及程序在异常出现时的状态。

负责对异常进行处理的程序需要从 JVM 中取出异常对象，这个过程称为捕获。

异常对象的抛出与捕获如图 5-1 所示。

下面的程序演示了数组越界异常(ArrayIndexOutOfBoundsException)。

main 方法中首先创建了一个只具有 3 个元素的数组对象，按照 Java 语言对数组的访问规则，分别通过 a[0]、a[1]和 a[2]访问这 3 个数组对象。

然而第二条语句却试图访问 a[3]。

当运行程序执行到第二条语句时，println 方法就会向 JVM 抛出 ArrayIndexOutOfBoundsException 类型的异常对象。

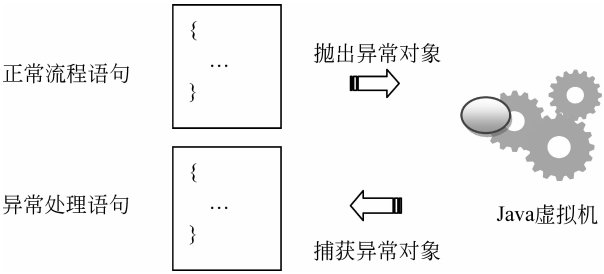

图 5-1　异常对象的抛出与捕获

代码 5-1　数组越界异常

```
/**
 * 数组的容量是 3,而程序试图访问第 4 个元素
 * 于是会抛出 ArrayIndexOutOfBoundsException 类型的异常对象
 */
public class Test {
    public static void main(String[] args) {
        int[] a={10,20,30};
        System.out.println(a[3] );
    }
}
```

由于没有任何程序捕获和处理这个异常,所以 JVM 就中止程序执行,并显示

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:3
        at Test.main(Test.java:10)
```

意思是：线程 main 中出现 java. lang. ArrayIndexOutOfBoundsException 类型的异常：索引 3 导致这个异常。抛出异常的语句位于源代码文件 Test. java 的第 10 行。

当数组的索引值为负数或大于等于数组大小时抛出 ArrayIndexOutOfBoundsException(数组索引越界异常)。

除了数组索引越界异常,常见异常还有：

- 空指针异常 NullPointerException,当应用试图在要求使用对象的地方使用了 null 时抛出该异常,例如调用 null 对象的实例方法、访问 null 对象的属性等。
- 找不到类异常 ClassNotFoundException,当 JVM 加载类时找不到对应名称的. class 文件时抛出该异常。
- 类型转换异常 ClassCastException。
- 算术异常 ArithmeticException,如除数为 0 等。

再看另外一个例子。

代码 5-2 试图通过 Scanner 对象从键盘输入一个整数,然后将输入的整数原样输出。

代码 5-2　输入不匹配异常

```
import java.util.Scanner;
```

```
public class Test {

    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int n=sc.nextInt();   //如果此处抛出异常,则跳过 main 方法中其余代码,程序终止
        System.out.println("Your number is: "+n);
    }

}
```

正常情况下,用户按照程序的提示“Enter an integer:”输入整数 1,回车,程序显示

```
Your number is: 1
```

但是,如果用户没有仔细看程序提示,或者没有看懂,那么可能输入“1.2”:

```
Enter an integer:
1.2
```

那么,Test.java 第 10 行中的 nextInt 方法就会抛出一个 java.util.InputMismatchException 类型的异常对象:

```
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at Test.main(Test.java:10)
```

由于没有捕捉和处理该异常的语句片段,所以 JVM 输出有关该异常的源代码位置、异常类型等信息,然后中止执行。

Java 设计了保留字 try 和 catch 用于异常的捕捉和处理。捕捉和处理异常的目的是使得正常的执行流程能够继续进行,正如从隔壁教室拿来粉笔就能够继续授课一样。

代码 5-3 在代码 5-2 的基础上使用保留字 try 和 catch 捕捉和处理异常。

代码 5-3 的想法是,正常情况下,程序顺利从键盘读入了一个整数,那么就使用该整数进行后面的处理(这里仅仅是直接输出);当程序试图从键盘读入整数时抛出了异常,那么就使用默认值 0 进行后续处理。

代码 5-3　异常的捕捉和处理

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class HandleExceptionTest {

    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
```

```
        int n;

        try {
            System.out.println("Enter an integer: ");
            n=sc.nextInt();
        } catch (InputMismatchException e) {
            System.out.println("Incorrect input: An integer is required. "+
                "Default value 0 is used.");
            n=0;
        }

        System.out.println("Your number is: "+n);
    }
}
```

保留字 try 以及其后由一对大括号“{}”括起来的语句序列称为 try 块(try block)。

try 块中的语句有可能抛出异常。

保留字 catch、紧跟其后的由小括号“()”括起来的异常对象引用变量声明以及其后由一对大括号“{}”括起来的语句序列称为 catch 块。

一旦 try 块中的某条语句抛出异常对象,就不再继续执行该语句以及其后随语句,而是转移到 catch 块执行。

执行 catch 块时,通过声明的异常对象引用变量(此例中是 e)可以访问异常对象的具体内容(此例中没有访问异常对象 e)。

执行 catch 块完毕,继续执行 try-catch 语句的后随语句。

图 5-2 展示 try 块、catch 块、异常对象和 JVM 之间的关系。

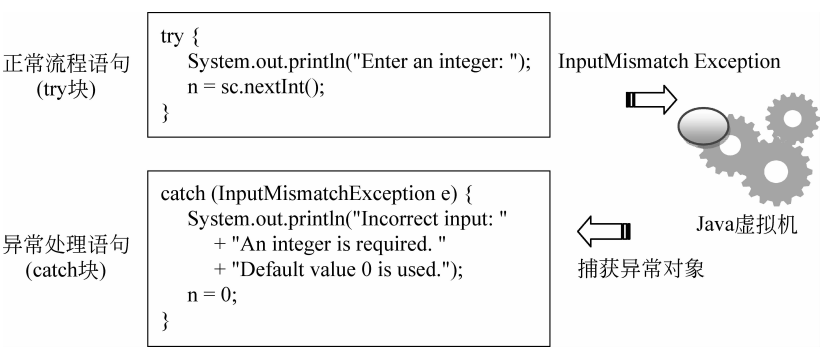

图 5-2 try-catch 语句

在某次执行中,try 块中的某条语句可能会抛出异常对象,此例中是 InputMismatchException 类型的异常对象。

这个异常对象交给了 JVM。

程序的控制发生转移,转到 catch 块执行。

catch 块通过声明的 InputMismatchException 类型的引用变量 e 访问 try 块中抛出的异常对象,一旦 e 能够引用这个异常对象,则称捕捉到了该异常。

InputMismatchException 异常是 Exception 的子类，而 Exception 是 Throwable 的子类。

在 JDK 中预定义了很多类似于 InputMismatchException 的异常。这些异常都是 Exception 的子孙类，而 Exception 是 Throwable 的子类。

Throwable 的子类共有两个，另一个是 Error 类。Error 类是对程序运行期间遇到的严重问题的抽象，Error 对象不能被捕获。

catch 块只能捕获 Exception 类及其子类的对象。

图 5-3 的类图展示了 Java. lang. Exception 的部分子类以及部分后代类。

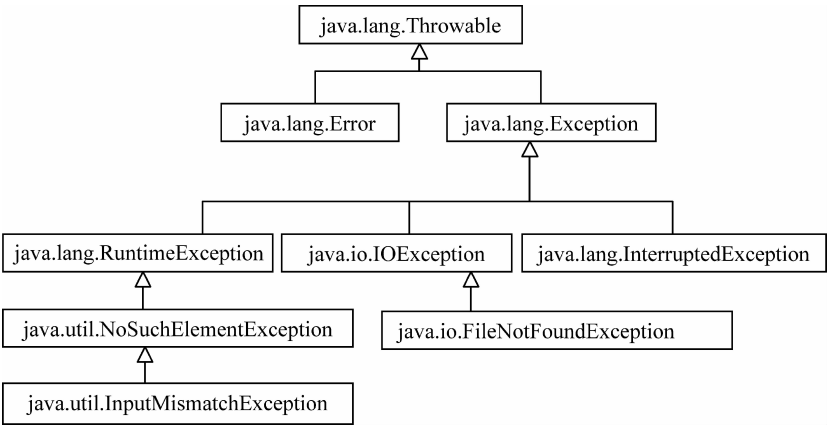

图 5-3　Exception 类的父类和子类(部分)

Exception 类有很多子类，如 RuntimeException、IOException 等。

RuntimeException 也有很多子类，如 ArithmeticException、ClassCastException、IllegalArgumentException、IndexOutOfBoundsException、NegativeArraySizeException、NoSuchElementException、NullPointerException。

IOException 的子类也有很多，如 ChangedCharSetException、CharacterCodingException、CharConversionException、EOFException、FileNotFoundException、FileSystemException。

运行时刻异常 RuntimeException 并不进行捕捉和处理，称为 unchecked 异常。

所有其他异常需要在程序中显式地声明如何进行处理，这些异常称为 checked 异常。

大多数情况下，运行时刻异常 RuntimeException 是由于程序的逻辑错误产生的，既然是逻辑错误，再运行下去就没有意义了，所以这类异常是不可恢复的，程序必须终止。

例如，当程序通过引用变量访问对象上的方法或者成员变量，而此时引用变量的值为 null 时，则会抛出空指针异常 NullPointerException 类型的异常对象。此刻程序没有必要再继续执行。

根据类的继承关系，运行时刻异常 RuntimeException 的子类也不必在程序中显式地声明如何处理，也是 unchecked 异常。

除了 RuntimeException 以外，其他 Exception 类的后代类都是 checked 异常。

对于所有 checked 异常，程序必须显式地声明如何进行处理。

下节就讨论如何显式地声明对 checked 异常的处理。

5.2 处理异常

假设异常是在方法 methodA 中抛出的，而方法 methodA 又是在另外一个方法 methodB 中被调用的，方法 methodB 称为方法 methodA 的调用者(caller)。

在图 5-4 中，main 方法调用了 methodA，methodA 调用了 methodB，methodB 又调用了 methodC。

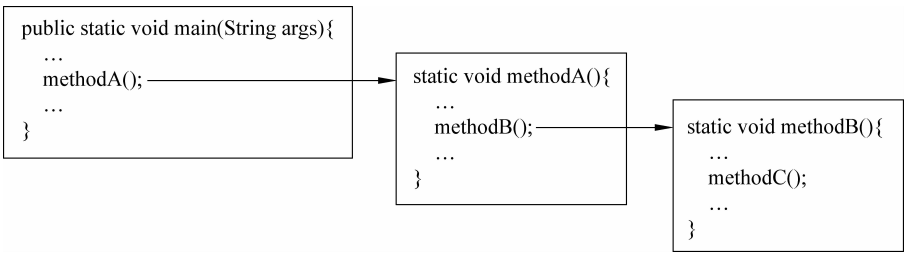

图 5-4　方法之间的调用

那么 main 是 methodA 的调用者，methodA 是 methodB 的调用者，methodB 是 methodC 的调用者。

Java 对异常的处理方式有两种：就地处理和上交调用者。

如果设计者决定就地处理，则使用 try-catch 语句，把可能产生异常的语句安排在 try 块中，把对异常进行处理的语句安排在 catch 块中。

如果设计者决定不在异常抛出的方法中进行就地处理，可能因为不知道如何处理或者懒得处理，那么就在方法的头部使用 throws 保留字声明上交调用者。

前文介绍了，输入输出异常 IOException 是一种需要在程序中显式地声明处理方式的异常。

代码 5-4 试图从文本文件“D:\java\test.txt”中读取 1 行。

代码 5-4　从文本文件中读取 1 行

```
import java.io.File;
import java.util.Scanner;

public class ExampleThrows_1 {
    public String getNextLine() {
        Scanner sc=new Scanner(new File("D:\\java\\test.txt"));
        return sc.nextLine();
    }
}
```

编译会显示

```
Unhandled exception type FileNotFoundException
```

意思是：未处理的异常类型 FileNotFoundException。

FileNotFoundException 是 IOException 的子类，对于可能抛出 FileNotFoundException 类型异常的语句必须声明如何进行处理。

根据前文所述，可选择两种方式之一进行声明：就地处理和上交调用者。

代码 5-5 在 getNextLine 方法头部使用保留字 throws 声明 getNextLine 方法可能会抛出异常，其类型为文件找不到异常 FileNotFoundException。

代码 5-5　在方法头部声明可能抛出的异常的类型

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Test {
    public String getNextLine()throws FileNotFoundException {
        Scanner sc=new Scanner(new File("D:\\java\\test.txt"));
        return sc.next();
    }
}
```

这样，如果 getNextLine 的方法中产生了 FileNotFoundException 类型的异常，那么这个异常对象就是上交给 getNextLine 方法的调用者。

由于 FileNotFoundException 是 IOException 的子类，所以 FileNotFoundException 对象也是 IOException。因此 getNextLine 方法声明抛出的异常类型也可以是父类型 IOException：

```
public class Test {
    public String getNextLine()throws IOException {
        Scanner sc=new Scanner(new File("D:\\java\\test.txt"));
        return sc.next();
    }
}
```

代码 5-6 演示了异常的“就地处理”方式。

代码 5-6　在方法体中使用 try-catch 语句就地处理异常

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Test {
    public String getPassword() {
        Scanner sc=null;
        String s=null;
        try {
            sc=new Scanner(new File("D:\\java\\test.txt"));
            s=sc.nextLine();
        } catch (FileNotFoundException e) {
```

```
            //输出异常对象中关于异常的描述信息
            System.err.println(e.getMessage());
            //终止运行
            System.exit(1);
        }
        return s;
    }
}
```

代码 5-6 把可能会抛出异常的语句序列放在了 try 块中，把产生异常后的处理语句放在了 catch 块中。

代码 5-6 中的 catch 块仅有两条语句。第一条语句调用捕捉到的异常对象上的方法 getMessage()获取异常消息并输出，第二条语句使用 System 类中的静态方法 exit 中止程序运行。

捕捉到的异常对象通过在 catch 保留字后面声明的 FileNotFoundException 类型的引用变量 e 来引用。

catch 块中处理异常的代码称为异常处理器(exception handler)。

一个 try 块可以与一个或多个 catch 块相关联，如代码 5-7 所示。

代码 5-7　一个 try 块可以与一个或多个 catch 块相关联

```
try {
//可能会抛出异常的代码块
} catch (ArithmeticException e) {
    //处理 ArithmeticException 类型的异常对象的代码块
} catch (IndexOutOfBoundsException e) {
    //处理 IndexOutOfBoundsException 类型的异常对象的代码块
}
```

在代码 5-7 中，如果 try 块中抛出异常，Java 的异常处理系统就自上而下依次将该异常的类型与 catch 中声明的类型进行匹配(match)。

一旦匹配，就执行相应的 catch 块。catch 块执行完毕，则整个 try-catch 语句执行完毕，继续执行该语句的后随语句。

匹配的意思是：抛出的异常对象的引用能够合法地给异常处理器声明的参数(在代码 5-7 中该参数是 e)赋值。

匹配并不要求抛出异常的类型与异常处理器声明的异常类型严格相同。

如果抛出的异常是子类，而异常处理器声明的类型是父类，也认为匹配。

在代码 5-8 中，一个 try 块与 3 个 catch 块相关联。

代码 5-8　异常的匹配

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

```
}
catch(IOException e) {
    System.out.println(e.getMessage());
}
catch (FileNotFoundException e){
    System.out.println(e.getMessage());
}
```

自上而下，catch 块声明的异常类型分别是 Exception、IOException 和 FileNotFoundException。

根据前文介绍，Exception 是 IOException 的父类；IOException 是 FileNotFoundException 的父类。

如果在 try 块中抛出了 FileNotFoundException 类型的异常对象，由于该类型与 Exception 匹配，那么执行的是第一个异常处理器。

一般地，如果 main 方法调用了 methodA，methodA 调用了 methodB，而在 methodB 中可能会抛出异常，那么 methodB 使用 try-catch 语句进行就地处理的方式如图 5-5 所示。

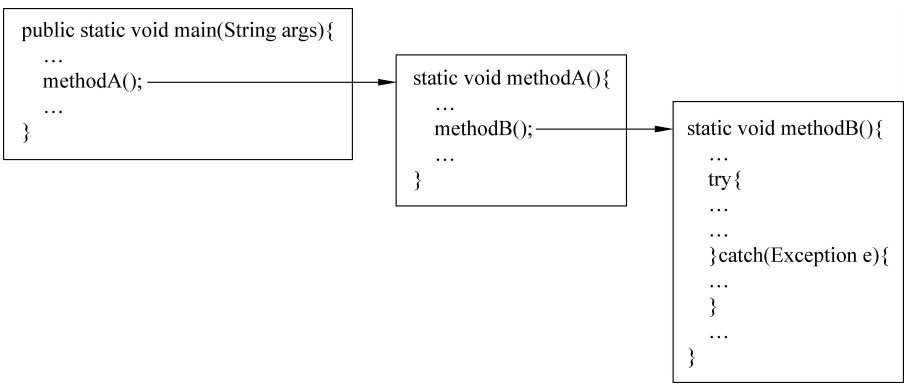

图 5-5　使用 try-catch 进行异常处理的方式

使用 throws 保留字声明异常抛出类型，交由调用者进行处理的方式如图 5-6 所示。

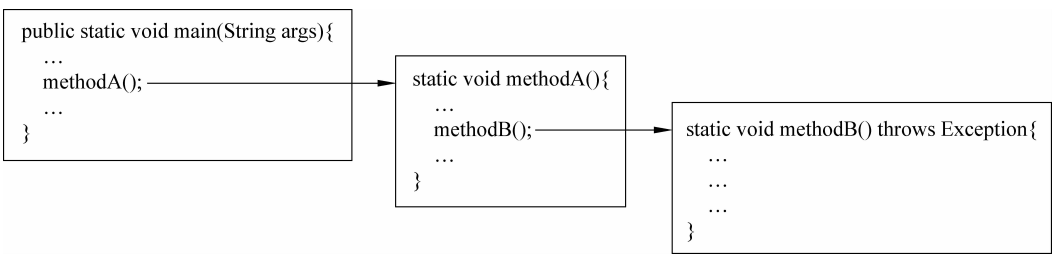

图 5-6　在方法头部声明抛出

在这种方式中，methodA 是 methodB 的调用者，将来 methodA 在执行含有 methodB 调用的语句时可能会产生异常。此时 methodA 同样有两种异常处理方式可供选择：使用 try-catch 语句就地处理或使用 throws 声明交给调用者。

如果 methodA 使用 throws 声明把异常交给调用者处理，那么在其调用者 main 方法中

又遇到同样问题。当然 main 方法同样有两种异常处理方式可供选择：使用 try-catch 语句就地处理或使用 throws 声明交给调用者。

但是 main 方法是整个程序的入口，所以，如果一个异常对象由 main 方法声明抛出，交由调用者处理，那么 JVM 就会输出一些异常对象的信息，然后中止程序执行。

一旦某条语句执行期间抛出异常，控制就会立即停止该语句的执行，转移到能够对这个异常进行处理的地方。

如果抛出异常的语句在 try 块内，那么控制就转移到与 try 块相伴的 catch 块。当 catch 块执行完毕时，继续执行 try-catch 语句的后随语句。

注意：执行完 catch 块后绝不可能回到抛出异常的语句继续执行。

如果抛出异常的语句不在任何 try 块内，那么：

(1) Java 会到包含该语句的方法的调用者中寻找相应的调用语句，假设该语句为 S，检查 S 是否在 try 块中。

(2) 如果是，则执行与 try 块相伴的 catch 块，catch 块执行完毕后，控制继续执行包含语句 S 的 try-catch 块的后随语句。

(3) 如果在调用者中发现调用语句没有在 try 块中，那么 Java 继续到语句 S 所在方法的调用者中寻找相应的调用语句。

(4) 如果寻找到 main 方法也没有找到处理该异常的 try-catch 块，Java 的异常处理系统输出关于该异常的一些信息并终止程序。

总之，Java 异常处理系统的任务就是当程序中的异常发生后，为其寻找异常处理器。

如果把一个方法看作一个结点，方法的调用是结点之间的有向边，那么 main 方法调用方法 methodA，methodA 方法又调用方法 methodB，等等，就形成了一个方法调用链。如图 5-7 所示。

图 5-7　方法调用链

对异常处理器的寻找就是沿着方法调用链反向进行。

如果直到 main 方法也没有找到异常处理器，那么异常处理系统向控制台输出一些消息，并终止程序执行。

寻找异常处理器的过程就称为捕捉异常(catching an exception)。

如果抛出异常的语句在 try 块中，但是该异常不能与 try-catch 语句中的任何 catch 块声明的异常类型相匹配，那么该异常就沿着方法调用链向调用者方向传播，如图 5-8 所示。

在图 5-8 中，main 调用了方法 methodA，而 methodA 调用了 methodB。

假设在方法 methodB 抛出异常。

如果该异常类型匹配 ExceptionB，那么这个异常就被捕捉到并进行处理。

如果该异常类型不匹配 ExceptionB，那么停止执行 methodB，控制返回到调用者 methodA 对方法 methodB 的调用语句处。

如果异常类型与 methodB 所在 try-catch 语句中的异常处理器声明的异常类型 ExceptionA 匹配，那么就执行该异常处理器。

如果该异常类型也不匹配 ExceptionA，那么停止执行 methodA，控制返回到调用者