第 8 章 异 常 处 理

8.1 异 常 概 述

8.1.1 异常的概念

在进行程序设计时，错误的产生是不可避免的。程序中的错误可分为三类：编译错误、逻辑错误和运行时错误。编译错误是由于没有遵循 Java 语言的语法规则而产生的，这种错误要在编译阶段排除，否则程序不可能运行。逻辑错误是指程序编译正常，也能运行，但结果不是人们所期待的。运行时错误是指程序在运行过程中出现了一个不可能执行的操作，就会出现运行时错误，运行时错误有时也可以由逻辑错误引起。异常处理的主要目的是即使在程序运行时发生了错误，也要保证程序能正常结束，避免由于错误而使正在运行的程序中途停止。

一个好的应用程序，除了具备用户要求的功能外，还要求能预见程序执行过程中可能产生的各种异常，并把处理异常的功能包括在用户程序中。异常处理机制是 Java 语言的重要特征之一。通过异常处理机制，可防止程序执行期间因出现错误而造成不可预料的结果。

例如下面的例 8-1，程序在编译阶段是没有错误的，算法设计也符合逻辑，但读者如果运行就会发现有错误出现。会有“发生异常：java.lang.ArithmeticException: / by zero”的错误提示。

所谓异常，是程序执行期间发生的各种意外或错误。比如：用户输入出错、所需文件找不到、运行时磁盘空间不够、内存不够、算术运算错（数的溢出、被零除等）、数组下标越界等。

异常是在程序运行过程中发生的非正常事件，它发生在程序运行期间，这些事件的发生将阻止程序的正常运行，干扰了正常的指令流程。

【例 8-1】 程序中所出现的算术异常。

```
package chapter8;
/**
  * 这个类生成一个算术异常
  */
class ExceptionRaised {
    /** 构造函数 */
    protected ExceptionRaised() {
    }
   /**
    * 这个方法生成一个异常
    * @param operand1 是除法中的分子
    * @param operand2 是除法中的分母
    * @return 它将返回除法的余数
    */
```

```
    static int calculate(final int operand1, final int operand2) {
        int result = operand1 / operand2;     // 用户自定义方法
        return result;
    }
}
/**
 * 这是 main 类
 *
 */
public class Chapter8_1 {
    /** 构造函数 */
    protected Chapter8_1() {
    }
    /**
     * 唯一的条目指向类和应用程序的唯一进入点
     * @param args 字符串参数的数组
     */
public static void main(String[] args) {
    ExceptionRaised obj = new ExceptionRaised();
    try {
        /* 定义变量 result 以存储结果 */
        int result = obj.calculate(9, 0);
        System.out.println(result);
    } catch (Exception e) {     // 异常对象
    System.err.println("发生异常:" + e.toString());
    e.printStackTrace();
    }
}
}
```

8.1.2　异常的分类

在 Java 程序运行过程中，产生的异常通常有以下三种类型。

（1）Java 虚拟机由于某些内部错误产生的异常，这类异常不在用户程序的控制之内，也不需要用户处理这类异常。

（2）标准异常类，由 Java 系统预先定义好。这类异常是由程序代码中的错误而产生的，例如，以零为除数的除法、访问数组下标范围以外的数组元素、访问空对象内的信息，这是需要用户程序处理的异常。

（3）根据需要在用户程序中自定义的一些异常类。本章主要来学习标准异常类及自定义异常类。

Java 中所有的异常都是用类表示的，在 Java 中预定义了很多异常类，每个代表了一种类型的运行错误。当程序发生异常时，会生成某个异常类的对象。Java 解释器可以监视程序中发生的异常，如果程序中产生的异常与系统中预定义某个异常类相对应，系统就自动产生一个该异常类的对象，就可以用相应的机制处理异常，确保程序能够安全正常地继续运行。异常对象中含有这种运行错误的信息和异常发生时程序的运行状态。

针对各种类型的异常，Java 定义了许多标准异常类，所有的 Java 异常类都是系统类库中的 Exception 类的子类，它们分布在 java.lang、java.io、java.util 和 java.net 包中。每个异常类对应一种特定的运行错误，各个异常类采用继承的方式进行组织。异常类的层次结构图如图 8-1 所示。

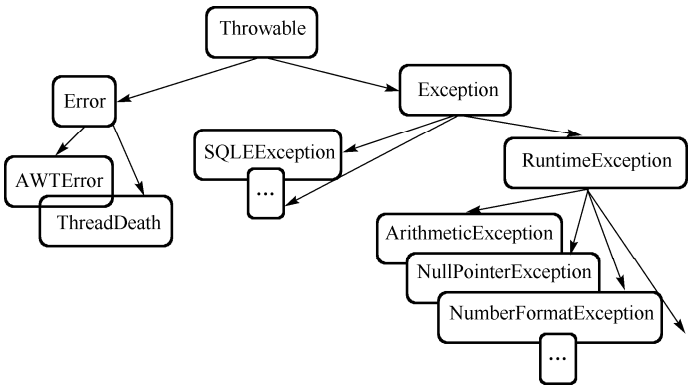

图 8-1　异常类的层次结构图

Throwable 类有两个直接子类：Error（致命错误）和 Exception（异常）。

Error 类型的异常与 Java 虚拟机本身发生的错误有关，这类异常由 Java 直接处理，用户程序一般不能做什么，只能等待系统通知用户关闭程序。

用户程序产生的错误由 Exception 的子类表示。用户程序应该处理这类异常。在 Java 中，通过 API 中 Throwable 类的众多子类描述各种不同的异常。

8.2　异常处理机制

在 Java 应用程序中，异常处理机制为——抛出异常，捕获异常。

（1）抛出异常：当一个方法出现错误引发异常时，方法创建异常对象并交付运行时系统，异常对象中包含了异常类型和异常出现时的程序状态等异常信息。运行时系统负责寻找处置异常的代码并执行。

（2）捕获异常：在方法抛出异常之后，运行时系统将转为寻找合适的异常处理器（Exception Handler）。潜在的异常处理器是异常发生时依次存留在调用栈中的方法的集合。当异常处理器所能处理的异常类型与方法抛出的异常类型相符时，即为合适的异常处理器。运行时系统从发生异常的方法开始，依次回查调用栈中的方法，直至找到含有合适异常处理器的方法并执行。当运行时系统遍历调用栈而未找到合适的异常处理器，则运行时系统终止。同时，意味着 Java 程序的终止。

对于运行时异常、错误或可查异常，Java 技术所要求的异常处理方式有所不同。由于运行时异常的不可查性，为了更合理、更容易地实现应用程序，Java 规定，运行时异常将由 Java 运行时系统自动抛出，允许应用程序忽略运行时异常。对于方法运行中可能出现的 Error，当运行方法不欲捕获时，Java 允许该方法不做任何抛出声明。因为，大多数 Error 异常属于永远不能被允许发生的状况，也属于合理的应用程序不该捕获的异常。对于所有的可查异常，Java 规定：一个方法必须捕获，或者声明抛出方法之外。也就是说，当一个方法选择不捕获可查

异常时，它必须声明将抛出异常。能够捕获异常的方法，需要提供相符类型的异常处理器。所捕获的异常，可能是由于自身语句所引发并抛出的异常，也可能是由某个调用的方法或 Java 运行时系统等抛出的异常。也就是说，一个方法所能捕获的异常，一定是 Java 代码在某处所抛出的异常。简单地说，异常总是先被抛出，后被捕获的。

8.2.1　try-catch-finally 语句捕获异常

在 Java 中，异常通过 try-catch 语句捕获。其一般语法形式为：

```
try  {
   //在此区域内或能发生异常; }
catch(异常类 1  e1)
 { //处理异常 1; }
    …
catch(异常类 n  en)
 { //处理异常 n; }
[finally
   {//不论异常是否发生,都要执行的部分; }]
```

语法结构说明：

①必须在 try 之后添加 catch 或 finally 块。try 块后可同时接 catch 和 finally 块，但至少有一个块。

②必须遵循块顺序：若代码同时使用 catch 和 finally 块，则必须将 catch 块放在 try 块之后。

③catch 块与相应的异常类的类型相关。

④一个 try 块可能有多个 catch 块。若如此，则执行第一个匹配块。即 Java 虚拟机会把实际抛出的异常对象依次和各个 catch 代码块声明的异常类型匹配，如果异常对象为某个异常类型或其子类的实例，就执行这个 catch 代码块，不会再执行其他的 catch 代码块。

⑤可嵌套 try-catch-finally 结构。

⑥在 try-catch-finally 结构中，可重新抛出异常。

【例 8-2】 try…catch 语句基本语法测试。

```
package chapter8;
import java.util.Scanner;
public class Chapter8_2 {
    /**
     输入异常测试
     */
    public static void main(String[] args) {
        Scanner input=new Scanner(System.in);
        int num=0;
        try {
            num=input.nextInt();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            System.out.print("输入的不是整数");
        }
```

```
            finally
            {
                System.out.print("总是被执行");
            }
                    System.out.print(num);
        }
    }
```

如果输入一个整数 2，则程序将输出：总是被执行 2，但如果输入一个字符或字符串，则程序将输出“输入的不是整数总是被执行 0”，这说明如果输入的不是整数，“num=input.nextInt();”将不被执行。

而 finally 里面的语句总是被执行的。

try、catch、finally 语句块的执行顺序如下。

①当 try 没有捕获到异常时：try 语句块中的语句逐一被执行，程序将跳过 catch 语句块，执行 finally 语句块和其后的语句。

②当 try 捕获到异常时，catch 语句块中有处理此异常的情况：在 try 语句块中是按照顺序来执行的，当执行到某一条语句出现异常时，程序将跳到 catch 语句块进行执行，try 语句块中，出现异常之后的语句也不会被执行，catch 语句块执行完后，执行 finally 语句块中的语句，最后执行 finally 语句块后的语句。

上面所说的执行顺序是 try 后面只跟一个 catch 语句的情况，如果 try 后面跟多个 catch 语句将是一种什么情况呢？先来看下面这个例子。

【例 8-3】 catch 语句的放置顺序。

```
package chapter8;
public class Chapter8_3 {
   public static void main(String[] args) {
      try {
            int num = 0;
            int num1 = 42 / num;
         } catch (Exception e) {
            System.out.println("父类异常 catch 子句");
        } catch (ArithmeticException ae) {  // 错误 - 不能到达实现的代码
            System.out.println("这个子类的父类是属于 exception 类，且不能到达实现");
        }     }
}
```

如果 catch 语句以这样的顺序，第二个 catch 语句将会有语法错误出现，而错误的信息就是，该 catch 语句将不能到达实现。因为 ArithmeticException 类是 Exception 的子类。所以在处理 catch 语句的顺序时，要把子类放到前面，父类放到后面。

【例 8-4】 多个 catch 语句的执行顺序。

```
package chapter8;
import java.util.InputMismatchException;
import java.util.Scanner;
public class Chapter8_4 {
    /**
    多个 catch 语句
```

```
     */
    public static void main(String[] args) {

    Scanner input=new Scanner(System.in);
        int num=0;
       try {
              num = input.nextInt();
              int num1 = 42 / num;

           } catch (ArithmeticException ae) {  // 错误 – 不能到达实现的代码
              System.out.println("算术异常");
                }
       catch (InputMismatchException e) {

              System.out.println("输入异常");
           }
  finally
       {
         System.out.println("总是执行");
       }
    }
}
```

输入 0，输出“算术异常/总是执行”。

输入“a”，输出“输入异常/总是执行”。

③ 当 try 捕获到异常，catch 语句块中有处理此异常的情况：在 try 语句块中是按照顺序来执行的，当执行到某一条语句出现异常时，程序将跳到 catch 语句块，并与 catch 语句块逐一匹配，找到与之对应的处理程序，其他的 catch 语句块将不会被执行，而 try 语句块中，出现异常之后的语句也不会被执行，catch 语句块执行完后，执行 finally 语句块中的语句，最后执行 finally 语句块后的语句。

8.2.2　异常抛出

异常抛出语句有两种，即 throws 和 throw，其中 throws 语句用于在方法头抛出异常，throw 用于在方法体内抛出异常，也就是针对某一条语句的异常抛出。

throws 语句的语法格式为：

```
methodname throws Exception1,Exception2,ExceptionN
```

方法名后的“throws Exception1,Exception2, ExceptionN”为声明要抛出的异常列表。在执行该方法的过程中，如果出现了由 throws 列出的异常，则可以抛出异常，并在程序中寻找处理这个异常的代码；如果程序中没有给出处理异常的代码，则把异常交给 Java 运行系统默认的异常处理代码进行处理。

【例 8-5】 throws 语句抛出异常。

```
package chapter8;
public class Chapter8_5 {
    public static void main(String[] args) throws ArithmeticException,
```

```
                ArrayIndexOutOfBoundsException
    {
        int a=4,b=0,c[]={1,2,3,4,5};
        System.out.println(a/b);
        System.out.println(c[a+1]);
        System.out.println("end");
}
}
```

由例 8-5 可以看到，如果一个方法可能抛出多个必检异常，那么必须在方法的声明部分一一列出，多个异常间使用逗号进行分隔。一个方法必须通过 throws 语句在方法的声明部分说明它可能抛出而并未捕获的所有的“必检异常”，如果没有这么做，将不能通过编译。值得注意的是：如果在子类中覆盖了父类的某一方法，那么该子类方法不可以比被其覆盖的父类方法抛出更多的异常（但可以更少）。所以，如果被覆盖父类的方法没有抛出任何的“必检异常”，那么子类方法绝不可能抛出“必检异常”。

throw 语句的语法格式为：

```
throw  异常类对象名或(new 异常类名());
```

如果需要在方法内某个位置抛出异常，可以使用 throw 语句。执行 throw 语句时，程序将终止执行后面的语句，在程序中寻找处理异常的代码；如果程序中没有给出处理代码，则把异常交给 Java 运行系统处理。

【例 8-6】 throw 语句抛出异常。

```
package chapter8;
public class Chapter8_6 {
    /**
     throw 语句抛出异常
     */
    public static void main(String[] args) {
        ArithmeticException e=new ArithmeticException();
         int num1=20,num2=0;
         System.out.println("异常处理");
           if(num2==0) throw e;
            System.out.println(num1/num2);
    }
}
```

throw 语句一般和 if 语句配合使用，如果满足某个条件，则将进行异常处理，如例 8-6 所示。

8.2.3 自定义异常

尽管 Java 提供了很多异常类，但用户还是可以根据需要定义自己的异常类，即创建自定义异常类。

说明：

（1）用户自定义的异常类必须是 Throwable 类或 Exception 类的子类。

（2）自定义的异常类，一般只要声明两个构造方法，一个是不用参数的，另一个以字符串为参数。作为构造方法参数的字符串应当反映异常的信息。

自定义异常类的语法格式：

```
class MyException extends Exception{
   …
}
```

【例 8-7】 自定义异常类举例说明。

```
class ArraySizeException extends NegativeArraySizeException {
    ArraySizeException() {
        super("你传递了非法的数组长度");
    }
}
class ExceptionClass {
    ExceptionClass(int val) {
        size = val;
        try {
            checkSize();
        } catch (ArraySizeException e) {
            System.out.println(e);
        }
    }
   /** 声明变量以存储数组的大小和元素 */
    private int size;
    private int[] array;
    public void checkSize() throws ArraySizeException {
        if (size < 0) {
            throw new ArraySizeException();
        }
        array = new int[3];
        for (int count = 0; count < 3; count++) {
            array[count] = count + 1;
        }
    }
}
class Chapter8_7 {
    protected Chapter8_7() {
    }
    public static void main(String[] arg) {
        ExceptionClass obj = new ExceptionClass(Integer.parseInt(arg[0]));
    }
}
```

由例 8-7 可以看到，用户定义的异常同样要用 try-catch 捕获，但必须由用户自己抛出 throw new MyException()。

8.3 应 用 实 例

异常捕获 try-catch 语句的执行顺序和 if-else 语句是比较接近的，如果没有异常，则执行 try 后面所跟的语句，若有异常，则执行 catch 后面的语句，好像这种结构也是一种选择结构，

但实际上这两种语句的执行是不同的，try-catch 是用于防止程序出现崩溃而不能处理的。当程序估计可能会出现某种导致崩溃的情况时可以用这个语句。try 后面的是运行的代码，catch 后面的是崩溃的类型。if-else 是用于条件判断的。

在本节的应用实例中，将介绍一个模拟登录、注册及抽奖的程序，在该程序的输入判断中将会用到异常捕获。

【例 8-8】 运用异常捕获的知识，实现模拟登录注册游戏。

```
import java.util.Scanner;
public class Demo {
    static Scanner input=new Scanner(System.in);
    static String username;
    static String pwd;
    static boolean reg=false;       //注册标记
    static boolean login=false;     //登录标记
    /**
     * 主函数
         */
    public static void main(String[] args) {
        String flag="y";
        do
        {
            showMenu();
            System.out.println("输入 y 继续操作");
            flag=input.next();
            if(!"y".equals(flag))
            {
                System.out.println("输入代码有误!");
            }
        }while("y".equals(flag));
    }
    /**
     * 显示系统菜单
         */
    public static void showMenu()
    {
        System.out.println("*****欢迎登录*****");
        System.out.println("\t1:注册");
        System.out.println("\t2:登录");
        System.out.println("*****************");
        System.out.println("输入操作代码");

            try {//异常捕获部分
                int num=input.nextInt();
                switch(num)
                {
                case 1:
                    reg();
                    break;
                case 2:
```

```
                    login();
                    break;
                default:
                    System.out.println("输入操作代码有误");
                }
            } catch (Exception e) {
                input.next();
                System.out.println("输入操作代码有误");
            }
    }
    /**
     * 注册
         */
    public static void reg()
    {
        System.out.println("*****注册*****");
        System.out.println("输入注册名:");
        username=input.next();
        System.out.println("输入密码:");
        pwd=input.next();
        int member=(int)(Math.random()*9000+1000);
        reg=true;
        System.out.println("注册成功!");
        System.out.println("用户名:"+username+"密码:"+pwd+"会员号:"+member);
    }
    /**
     *登录
         */
    public static void login()
    {
        if(reg)
        {
            System.out.println("*****登录*****");
            for (int i = 0; i < 3; i++) {
                System.out.println("输入用户名:");
                String username2=input.next();
                System.out.println("输入密码:");
                String pwd2=input.next();
                if(username2.equals(username)&&pwd2.equals(pwd))
                {
                    System.out.println("登录成功");
                    login=true;
                    break;
                }
                else
                {
                    System.out.println("还剩"+(2-i)+"次登录机会");
                }
            }
        }
```

```
        else
        {
            System.out.println("请先注册!");
        }
    }
}
```

习　题

一、选择题

1．请问所有的异常类皆继承哪一个类？（　）

A．java.lang.Throwable　　B．java.lang.Exception

C．java.lang.Error　　D．java.io.Exception

2．哪个关键字可以抛出异常？（　）

A．transient　　B．throw　　C．finally　　D．catch

3．对于已经被定义过可能抛出异常的语句，在编程时（　）。

A．必须使用 try-catch 语句处理异常，或用 throw 将其抛出

B．如果程序错误，必须使用 try-catch 语句处理异常

C．可以置之不理

D．只能使用 try-catch 语句处理

4．下面程序段的执行结果是（　）。

```
public class Foo{
  public static void main(String[] args){
    try{
       return;}
       finally{System.out.println("Finally");
     }
  }
}
```

A．编译能通过，但运行时会出现一个例外

B．程序正常运行，并输出“Finally”

C．程序正常运行，但不输出任何结果

D．因为没有 catch 语句块，所以不能通过编译

5．下面的方法是一个不完整的方法，其中的方法 unsafe()会抛出一个 IOException，那么在方法的①处应加入哪条语句，才能使这个不完整的方法成为一个完整的方法？（　）

①　_____

②　{ if(unsafe())　{//do something…}

③　　else　if(safe())　{//do the other…}

④　}

A．public IOException methodName()

B．public void methodName() throw IOException

C．public void methodName()

D．public void methodName() throws IOException

E．public void methodName() throws Exception

6．如果下列的方法能够正常运行，在控制台上将显示什么？（　　）

```
public void example( ){
  try{
      unsafe();
      System.out.println("Test1");
     }
  catch(SafeException e)
     {System.out.println("Test 2");}
  finally{System.out.println("Test 3");}
  System.out.println("Test 4");
}
```

A．Test 1　　B．Test 2　　C．Test 3　　D．Test 4

二、简答题

1．什么是异常？简述 Java 的异常处理机制。

2．系统定义的异常与用户自定义的异常有何不同？如何使用这两类异常？

3．在 Java 的异常处理机制中，try 程序块、catch 程序块和 finally 程序块各起什么作用？try-catch-finally 语句如何使用？

4．说明 throws 与 throw 的作用。

三、程序填空

```
public class ServerTimedOutException extends Exception {
    private int port;
    public ServerTimedOutException(String message, int port) {
          super(message);
          this.port = port;
       }
    public int getPort() {
          return port;
} }
class Client {// 在下行横线处填上声明抛弃 ServerTimedOutException 例外的语句
public void connectMe(String serverName) _____ {
    int success;
    int portToConnect = 80;
    success = open(serverName, portToConnect);
    if (success == -1) {
// 在下行横线处填上抛出 ServerTimedOutException 例外的语句

    _____
         }}
    private int open(String serverName, int portToConnect) {
          return 0;
    }}
```