第11章

异常处理

（视频讲解：49分钟）

在程序中总是存在各种错误，使应用程序在运行时终止。为了在程序执行过程中发生错误时能正常运行，可以使用 Java 提供的异常处理机制捕获可能发生的异常，对异常进行处理并使程序能正常运行。

通过阅读本章，您可以：

- 了解异常的概念
- 了解异常的分类
- 掌握如何获取异常信息
- 掌握如何处理异常
- 了解如何抛出异常
- 了解自定义异常
- 了解异常的使用原则

11.1 异 常 概 述

视频讲解：光盘\TM\lx\11\异常概述.exe

假设一辆轿车发生了故障，可能是某个零件发生了问题，也可能是没有油了。如果是由于零件问题，只需要更换零件就可以解决；如果是没有油了，只需要加满油就可以正常行驶了。程序中的异常与此类似，就对程序中可能发生异常的语句进行处理，使程序能够正常执行。

在程序开发过程中，可能存在各种错误，有些错误是可以避免的，而有些错误却是意想不到的，在 Java 中把这些可能发生的错误称为异常。图 11.1 说明了异常类的继承关系。

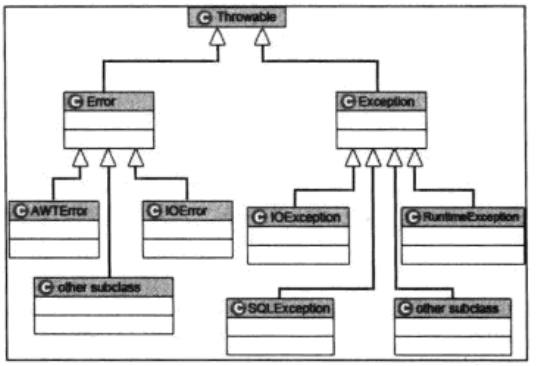

图 11.1 异常类的继承关系图

从图 11.1 中可以看出，Throwable 类是所有异常类的超类，该类的两个直接子类是 Error 和 Exception。其中，Error 及其子类用于指示合理的应用程序不应该试图捕获的严重问题，Exception 及其子类给出了合理的应用程序需要捕获的异常。

说明 图 11.1 中列出了异常类的继承关系，由于界面大小有限，而 Error 类和 Exception 类的子类又比较多，所以这里使用 other subclass 代表 Error 类和 Exception 类的其他子类。

11.2 异常的分类

视频讲解：光盘\TM\lx\11\异常的分类.exe

在 Java 中可以捕获的异常（即 Exception 类的子类）分为可控式异常和运行时异常两种类型。

11.2.1 可控式异常

在 Java 中把那些可以预知的错误，例如从文件中读取数据、对数据库进行操作等，在程序编译时就能对程序中可能存在的错误进行处理，并给出具体的错误信息，我们把这些错误称为可控式异常。表 11.1 中列出了常用的可控式异常及说明。

表 11.1　常用可控式异常及说明

异　常	说　明
IOException	当发生某种 I/O 异常时，抛出此异常
SQLException	提供关于数据库访问错误或其他错误信息的异常
ClassNotFoundException	类没有找到异常
NoSuchFieldException	类不包含指定名称的字段时产生的信号
NoSuchMethodException	无法找到某一特定方法时，抛出该异常

【例 11.1】　创建项目 01，在项目中创建 Example_01 类，在该类中加载一个不存在的类，观察发生的异常。（**实例位置：光盘\TM\sl\11\1**）

```
public class Example_01 {
    private int num=10;                                  //成员变量
    public int getNum() {                                //成员方法
        return num;                                      //返回成员变量的值
    }
    public void setNum(int num) {                        //成员方法
        this.num = num;                                  //设置成员变量的值
    }
    public Example_01(){                                 //类的构造方法
        try {
            Class.forName("com.mingrisoft.Test");        //装载 com.mingrisoft 包中的 Test 类
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        System.out.println("测试。");                    //在控制台输出“测试。”
    }
    public static void main(String[] args) {
        Example_01 exam=new Example_01();                //创建类的实例
        exam.setNum(888);                                //调用 setNum()方法设置成员变量 num 的值为 888
        System.out.println(exam.getNum());               //调用 getNum()方法输出成员变量的值 888
    }
}
```

运行结果如图 11.2 所示。

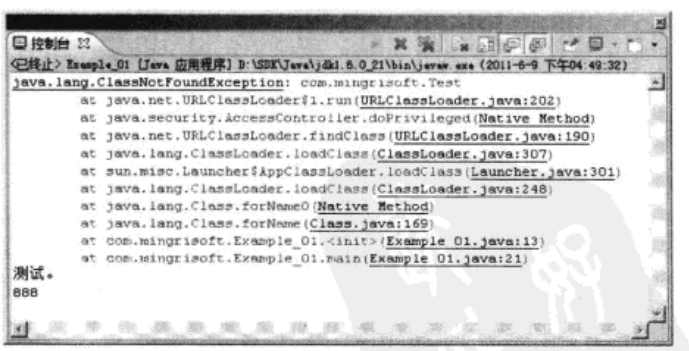

图 11.2　ClassNotFoundException 异常

由于在构造方法中加载 com.mingrisoft 包中的 Test 类时，Java 的虚拟机没有找到 Test 类，从而引发了 ClassNotFoundException 异常，所以在控制台输出了 ClassNotFoundException 这样的异常信息，由于在执行“Class.forName("com.mingrisoft.Test");”语句时发生了异常，所以导致程序中的其他代码没有执行，所以在控制台并没有输出“测试。”和“888”。

11.2.2 运行时异常

在 Java 中有些错误是不能被编译器检测到的，例如，在进行除法运算时，除数为零；试图把一个不是由数字组成的字符串使用 Integer 类的 parseInt()方法转换为整数等，Java 的编译器是检测不到的，因而能够正常编译，但是在运行时就会发生异常，我们把这些异常称为运行时异常。表 11.2 列出了常用的运行时异常及说明。

表 11.2 常用的运行时异常及说明

方　法	说　明
IndexOutOfBoundsException	指示某集合或数组的索引值超出范围时抛出该异常
NullPointerException	当应用程序试图在需要对象的地方使用 null 时，抛出该异常
ArithmeticException	当出现异常的运算条件时，抛出此异常
IllegalArgumentException	抛出的异常表明向方法传递了一个不合法或不正确的参数
ClassCastException	当试图将对象强制转换为不是实例的子类时，抛出该异常

【例 11.2】 通过 Integer 类的实例创建 Object 对象 o，观察将 Object 对象 o 强制转换为字符串时发生的异常。

```
Object o = new Integer(0);                    //通过 Integer 类的实例创建 Object 对象 o
System.out.println((String)o);                //将 Object 对象 o 强制转换为字符串时出错
```

上面代码首先创建一个 Object 对象 o，该对象是通过 Object 的子类 Integer 创建的，也就是说 Object 对象 o 是子类 Integer 的实例，因此当使用 String 类将对象 o 强制转换为字符串时出错了，因为对象 o 是 Object 的子类 Integer 创建的，而不是 String 类创建的，所以产生了错误，如果将对象 o 强制转换为 Integer 类型就不会出错，因为对象 o 是 Integer 类创建的。

【例 11.3】 在项目中创建 Example_02 类，在该类中创建一个数组，然后使用超出数组下标范围的值访问数组中的元素，观察发生的异常。（**实例位置：光盘\TM\sl\11\2**）

```
int[] number={ 100, 80, 50, 70, 20, 60};          //创建并初始化具有 6 个元素的数组
public void setNum(int index,int value){          //根据索引值 index 为数组元素赋值 value 的方法
    number[index]=value;                          //为数组中索引值 index 处的元素赋值 value
}
public int getNum(int index){                     //根据索引值 index 获得数组中对应元素值的方法
```

```
        return number[index];                                    //获得数组中索引值 index 处的元素值
    }
    public static void main(String[] args) {
        Example_02 ex=new Example_02();                          //创建类的实例
        //调用方法获得数组中索引值 0 处的元素值，即第一个元素的值 100
        int value=ex.getNum(0);
        System.out.println(value);                               //输出第一个元素的值 100
        //索引值 6 超出了数组下标的范围，因此将发生异常，导致程序终止，下面的代码将不会被执行
        value=ex.getNum(6);
        System.out.println(value);
        //获得数组中最后一个元素的值 60
        value=ex.getNum(5);
        System.out.println(value);                               //输出最后一个元素值 60
    }
```

运行结果如图 11.3 所示。

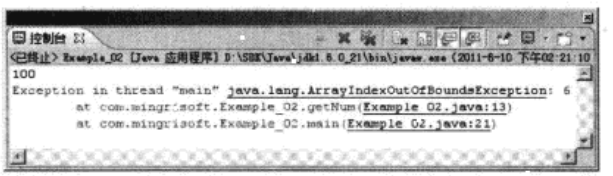

图 11.3　ArrayIndexOutOfBoundsException 异常

说明　从输出结果可以看出，程序发生了 ArrayIndexOutOfBoundsException 异常，该异常是 IndexOutOfBoundsException 异常的子异常。当程序执行到语句“value=ex.getNum(6);”时发生了异常，因为数组中有 6 个元素，数组下标的范围是从 0～5，所以用 6 作为数组下标的索引值发生了错误，因而导致下面的其他语句不会被执行，所以程序中只输出了数组中第一个元素的值 100，而没有输出最后一个元素的值 60。

11.2.3　范例 1：算术异常

算术异常即 ArithmeticException，是指整数被 0 除产生的异常。在 Java 语言中，如果一个整数被 0 除，那么将抛出 ArithmeticException，但是浮点数被 0 除，将不引发算术异常，这与数学中不同。本范例将演示出现算术异常的情况，并进行处理。运行结果如图 11.4 所示。（**实例位置：光盘\TM\sl\11\3**）

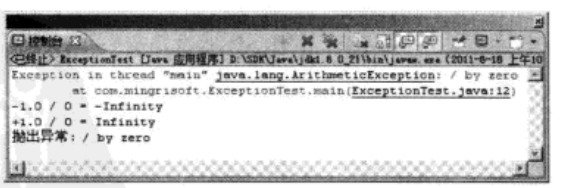

图 11.4　算术异常

（1）在 Eclipse 中创建项目 03，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建类文件，名称为 ExceptionTest。在该类的主方法中，演示出现算术异常的情况。这里将第一条可能出现异常的语句应用 try…catch 语句捕获，并输出异常信息，第二条可能出现异常的语句不作处理。代码如下：

```
public class ExceptionTest {
    public static void main(String[] args) {
        System.out.println("-1.0 / 0 = " + (-1.0 / 0));                    //演示负浮点数除 0
        System.out.println("+1.0 / 0 = " + (+1.0 / 0));                    //演示正浮点数除 0
        try{
          System.out.println("-1 / 0 = " + (-1 / 0));                      //演示负整数除 0
        }catch(Exception e){
          System.out.println("抛出异常：" +e.getMessage());
        }
        System.out.println("+1 / 0 = " + (+1 / 0));                        //演示正整数除 0
        System.out.println("输出结束。");
    }
}
```

说明 在 Java 的异常处理机制中，有一个默认处理异常的程序。当程序出现异常时，默认处理程序将显示一个描述异常的字符串，打印异常发生处的堆栈轨迹，并终止程序。

11.2.4 范例 2：数组下标越界异常

数组下标越界异常即 ArrayIndexOutOfBoundsException，当访问的数组元素的下标值大于数组的最大下标值时发生。也就是数组元素的下标值大于等于数组的长度时发生。本范例将演示出现数组下标越界异常（ArrayIndexOutOfBoundsException）的情况。运行结果如图 11.5 所示。（**实例位置：光盘\TM\sl\11\4**）

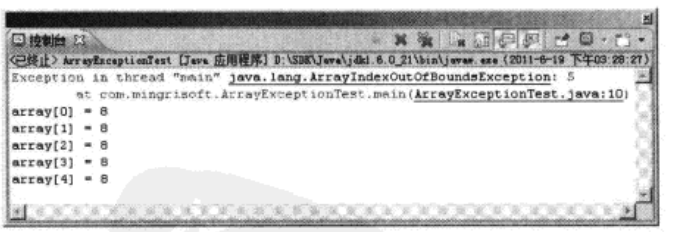

图 11.5 数组下标越界异常

（1）在 Eclipse 中创建项目 04，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建类文件，名称为 ArrayExceptionTest。在该类的 main()方法中，首先声明一个长度为 5 的整型数组，并应用 Arrays 对象的 fill()方法，将数组中所有元素赋值为 8，然后应用 for 循环遍历输出所有数组元素。代码如下：

```
public class ArrayExceptionTest {
    public static void main(String[] args) {
        int[] array = new int[5];                          //声明一个长度为 5 的整型数组
        Arrays.fill(array, 8);                             //将新声明数组的所有元素赋值为 8
        for (int i = 0; i < 6; i++) {                      //遍历输出所有数组元素
            System.out.println("array[" + i + "] = " + array[i]);
        }
    }
}
```

说明　如果要遍历数组中的全部元素，则推荐使用 foreach 循环，它可以避免数组的下标越界。如果要使用数组的下标，则需要记住数组的下标是从 0 开始计算的。如果需要使用数组的长度，则推荐使用 length 属性。另外使用 ArrayList 类也可以避免这些问题。

11.3　获取异常信息

视频讲解：光盘\TM\lx\11\获取异常信息.exe

获取异常信息就好比工厂里某个线路出现故障停电了，电工要从线路中找出故障的原因，就像程序中获取到了异常信息。

在 Java 中 java.lang.Throwable 类是所有异常类的超类，该类提供了获得异常信息的方法。表 11.3 中列出了获取异常信息的方法及说明。

表 11.3　获取异常信息的方法及说明

方　法	说　明
String getLocalizedMessage()	获得此 Throwable 的本地化描述
String getMessage()	获得此 Throwable 的详细消息字符串
void printStackTrace()	将此 Throwable 及其栈踪迹输出至标准错误流
String toString()	获得此 Throwable 的简短描述

【例 11.4】　在项目中创建 Example_03 类，在该类中使用表 11.3 中的方法输出进行除法运算时除数为 0 的异常信息。（**实例位置：光盘\TM\sl\11\5**）

```
public class Example_03 {
    public void printBugInfo(){                            //定义成员方法
        try{
            int x=100;                                     //定义局部变量 x
            int y=0;                                       //定义局部变量 y
            int z=x / y;                                   //计算 x 除以 y 的商
            //输出计算结果
            System.out.println(x+"除以"+y+"的商是："+z);
        }catch(Exception ex){
```

```
            ex.printStackTrace();                          //输出异常到标准错误流
            //使用 getMessage()方法输出异常信息
            System.out.println("getMessage 方法：    "+ex.getMessage());
            //使用 getLocalizedMessage()方法输出异常信息
            System.out.println("getLocalizedMessage 方法：    "+ex.getLocalizedMessage());
            //使用 toString()方法输出异常信息
            System.out.println("toString 方法：    "+ex.toString());
        }
    }
    public static void main(String[] args) {
        Example_03 ex=new Example_03();                    //创建类的实例
        ex.printBugInfo();                                 //调用方法
    }
}
```

运行结果如图 11.6 所示。

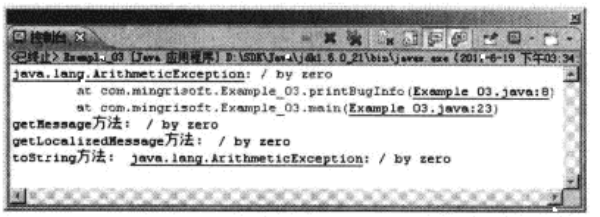

图 11.6　除数为 0 的异常信息

说明　输出结果是除数为 0 时发生的异常，其中前 3 行是 printStackTrace()方法输出的异常信息，第 4 行是 getMessage()方法输出的异常信息，第 5 行是 getLocalizedMessage()方法输出的异常信息，最后一行是 toString()方法输出的异常信息。在本实例的代码中使用了 try…catch 语句块来捕获程序中的异常信息，在 try 和 catch 之间两个大括号内是程序需要正常执行，但是却又有可能发生异常的代码，在 catch 后的两个大括号内是 try 和 catch 之间的代码发生错误时执行的代码，用于进行异常处理。

11.4　处 理 异 常

视频讲解：光盘\TM\lx\11\处理异常.exe

在 Java 语言中当程序发生异常时，可以使用 try…catch、try…catch…finally 或 try…finally 进行处理。接下来将对这 3 个语句块分别进行讲解。

11.4.1　使用 try…catch 处理异常

对于程序中可能发生异常的语句，可以将其添加到 try…catch 语句块中，这样当程序发生异常时，

就可以对其进行相应的处理。

try…catch 语句块的语法格式如下：

```
try{
    需要正常执行的语句
}catch(Exception ex){
    对异常进行处理的语句
}
```

- ☑ try 和 catch 是进行异常处理的关键字。
- ☑ try 和 catch 之间的两个大括号内是程序需要正常执行但又可能发生异常的语句。
- ☑ catch 后的两个小括号内是程序需要处理的异常类型。
- ☑ catch 后的两个大括号内是对程序发生的异常进行处理的语句。

11.4.2　使用 try…catch…finally 处理异常

对于程序中可能发生异常的语句，可以将其添加到 try…catch…finally 语句块中，这样当程序发生异常时，就可以对其进行相应的处理。

try…catch…finally 语句块的语法格式如下：

```
try{
    需要执行的语句
}catch(Exception ex){
    对异常进行处理的语句
}finally{
    一定会被处理的语句
}
```

- ☑ try、catch 和 finally 是进行异常处理的关键字。
- ☑ try 和 catch 之间的两个大括号内是程序需要正常执行但又可能发生异常的语句。
- ☑ catch 后的两个小括号内是程序需要处理的异常类型。
- ☑ catch 后的两个大括号内是对程序发生的异常进行处理的语句。
- ☑ finally 后的两个大括号内的语句，不管程序是否发生异常都要执行（也就是说程序执行完 try 和 catch 之间的语句或执行完 catch 后两个大括号内的语句都将执行 finally 后的语句），因此 finally 语句块通常用于执行垃圾回收、释放资源等操作。

在 Java 中进行异常处理时，应该尽量使用 finally 块进行资源回收，因为在 try…catch…finally 语句块中，不管程序是否发生异常，最终都会执行 finally 语句块，因此可以在 finally 块中添加释放资源的代码。

【例 11.5】　在项目中创建 IO 流，分配内存资源。使用完后，在 finally 中关闭 IO 流并释放内存资源。代码如下：（**实例位置：光盘\TM\sl\11\6**）

```
private FileInputStream in=null;                                    //声明 FileInputStream 对象 in
public void readInfo(){                                             //定义方法
    try{
        //创建 FileInputStream 对象 in
        in=new FileInputStream("src/com/mingrisoft/CloseIo.java");
        System.out.println("创建 IO 流，分配内存资源。");
    }catch(IOException io){
        io.printStackTrace();                                       //输出栈踪迹
        System.out.println("创建 IO 对象发生异常。");
    }finally{
        if (in!=null){
            try{
                in.close();                                         //关闭 FileInputStream 对象 in，释放资源
                System.out.println("关闭 IO 流，释放内存资源。");
            }catch(IOException ioe){
                ioe.printStackTrace();                              //输出栈踪迹
                System.out.println("关闭 IO 对象发生异常。");
            }
        }
    }
}
public static void main(String[] args) {
    CloseIo ex=new CloseIo();                                       //创建对象
    ex.readInfo();                                                  //调用 readInfo()方法
}
```

运行结果如图 11.7 所示。

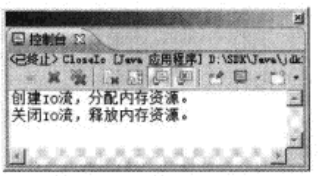

图 11.7　在控制台输出释放信息

说明 从输出结果可以看出，程序在 try 语句块中创建了 IO 对象，然后在 finally 语句块中关闭了 IO 对象，释放了内存资源。

11.4.3　使用 try…finally 处理异常

对于程序中可能发生异常的语句，可以将其添加到 try…finally 语句块中，这样当程序发生异常时，就可以在 finally 语句块中对其进行相应的处理，另外当程序没有发生异常时，执行完 try 和 finally 之间的语句后，也将执行 finally 语句块中的代码，因此可以在 finally 语句块中放置一些必须执行的代码，如释放内存资源的代码等。

try…finally 语句块的语法格式如下：

```
try{
    需要执行的语句
} finally{
    一定会被处理的语句
}
```

- ☑ try 和 finally 是进行异常处理的关键字。
- ☑ try 和 finally 之间的两个大括号内是程序需要正常执行但又可能发生异常的语句。
- ☑ finally 后两个大括号内的语句是不管程序是否发生异常最终都要执行的语句，因此 finally 语句块通常用于放置程序中必须执行的代码，如关闭数据库连接、关闭 IO 流等。

说明　在有 try…finally 语句块的程序中，只要程序执行了 try 语句块中的代码，不管 try 语句块是否发生异常，与该 try 语句块对应的 finally 语句块都一定会被执行，因此通常使用 finally 语句块进行资源释放。

【例 11.6】　在 07 项目中创建 Example_07 类，使用 try…finally 语句块对程序进行异常处理和资源释放。代码如下：（**实例位置：光盘\TM\sl\11\7**）

```
private FileReader read=null;                                    //声明 FileReader 对象 read
public void readFileInfo(){                                      //定义方法
    try{
        try {
            //创建 FileReader 对象 read
            read=new FileReader("src/com/mingrisoft/Example_07.java");
            System.out.println("找到指定的文件，创建 IO 对象成功！");
        } catch (FileNotFoundException e) {
            e.printStackTrace();                                 //输出栈踪迹
        }
    }finally{
        if (read!=null){
            try{
                read.close();                                    //关闭 FileReader 对象 read，释放资源
                System.out.println("关闭 IO 对象！");
            }catch(IOException ioe){
                ioe.printStackTrace();                           //输出栈踪迹
                System.out.println("关闭 IO 对象发生异常。");
            }
        }
    }
}
public static void main(String[] args) {
    Example_07 ex=new Example_07();                              //创建对象
    ex.readFileInfo();                                           //调用 readFileInfo()方法
}
```

运行结果如图 11.8 所示。

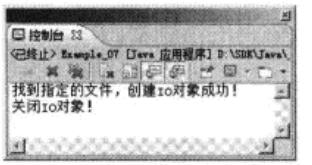

图 11.8　在控制台输出操作信息

说明 从输出结果可以看出，程序在 try 语句块中创建了 IO 对象并分配了内存资源，然后在 finally 语句块中关闭了 IO 对象并释放了内存资源。

11.5　抛 出 异 常

视频讲解：光盘\TM\lx\11\抛出异常.exe

对于程序中发生的异常，除了可以使用 try…catch 语句块处理之外，还可以使用 throws 声明或 throw 语句抛出异常。下面将分别进行讲解。

11.5.1　使用 throws 声明抛出异常

throws 通常用于方法声明，当方法中可能存在异常，却不想在方法中对异常进行处理时，就可以在声明方法时使用 throws 声明抛出的异常，然后在调用该方法的其他方法中对异常进行处理（如使用 try…catch 语句或使用 throws 声明抛出的异常）。

如果需要使用 throws 声明抛出多个异常，各异常之间要用逗号分隔。throws 声明抛出异常的语法格式如下：

```
数据类型　方法名(形参列表)　throws　异常类 1,异常类 2,……,异常类 n{
    方法体;
}
```

- ☑ 数据类型是基本数据类型或对象类型。
- ☑ 方法名是 Java 语言的合法标识符。
- ☑ throws 是抛出异常的关键字。
- ☑ 异常类是 Java 语言的异常类或自定义异常类。
- ☑ 方法体是该方法需要执行的语句。

【例 11.7】　使用 throws 抛出 Exception 异常。

```
public void showInfo() throws Exception{                    //抛出 Exception 异常
    FileInputStream in=new FileInputStream("C:/Record.txt");  //创建 IO 对象
}
```

showInfo()方法使用 throws 抛出了 Exception 异常，这样在该方法中创建 IO 对象时就不会发生异常了，但是必须在调用该方法的其他方法中对 Exception 异常进行处理，否则调用该方法的其他方法会发生异常。

【例 11.8】　定义 methodName()方法，该方法调用了上面定义的 showInfo()方法，并对 showInfo()方法抛出的异常进行了处理。

```
void methodName() {
    try{
        showInfo();                                    //使用 try…catch 语句块处理 showInfo()方法抛出的异常
    }catch(Exception ex){
        System.out.println(ex.getMessage() );
    }
}
```

methodName()方法调用了 showInfo()方法，并对 showInfo()方法抛出的 Exception 异常进行了处理，否则该方法将出错。

【例 11.9】　在项目中创建 Example_08 类，在该类中创建一个使用 throws 抛出异常的 createFile()方法，然后创建一个 test()方法，在该方法中调用 createFile()方法，并进行异常处理。（**实例位置：光盘\TM\sl\11\8**）

```
public class Example_08 {
    private FileReader read=null;                      //声明 FileReader 对象 read
    public void createFile() throws Exception{         //定义方法，使用 throws 抛出 Exception 异常
        //创建 FileReader 对象 read
        read=new FileReader("src/com/mingrisoft/Example_08.java");
        System.out.println("分配内存资源。");
    }
    public void test(){
        try{
            createFile();                              //调用 createFile()方法，使用 try…catch…finally 语句处理异常
        }catch(Exception ex){
            ex.printStackTrace();                      //输出栈踪迹
            System.out.println("创建 IO 对象异常。");
        }finally{
            if (read!=null){
                try {
                    read.close();                      //关闭 IO 流
                    System.out.println("释放内存资源。");
                } catch (IOException e) {
                    e.printStackTrace();               //输出栈踪迹
                    System.out.println("关闭 IO 对象异常。");
                }
            }
```

```
        }
    }
    public static void main(String[] args) {
        Example_08 ex=new Example_08();        //创建对象
        ex.test();                             //调用 test()方法
    }
}
```

运行结果如图 11.9 所示。

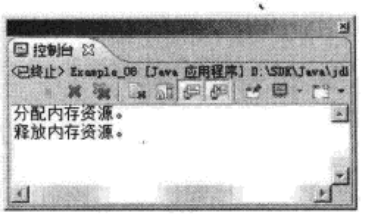

图 11.9　分配和释放内存资源提示

11.5.2　使用 throw 语句抛出异常

在通常情况下，程序发生错误时系统会自动抛出异常，而有时希望程序自行抛出异常，可以使用 throw 语句来实现。

throw 语句通常用在方法中，在程序中自行抛出异常，使用 throw 语句抛出的是异常类的实例，通常与 if 语句一起使用。

throw 语句的语法格式如下：

```
throw new Exception("对异常的说明");
```

☑ throw 是抛出异常的关键字。

☑ Exception 是异常类（通常使用自定义异常类）。

【例 11.10】 在项目中创建 Example_09 类，使用该类计算圆的面积。设定圆的半径不能小于 20，如果半径小于 20，则使用 throw 语句抛出异常，并给出提示信息。（**实例位置：光盘\TM\sl\11\9**）

```
public class Example_09 {
    final static double PI=3.14;                                    //圆周率
    public void computeArea(double r) throws Exception{             //根据半径计算圆面积的方法
        if (r<=20.0){
            //使用 throw 语句抛出异常
throw new Exception("程序异常：\n 半径为 ："+r+"\n 半径不能小于 20。");
        }else{
            double circleArea=PI*r*r;                               //计算圆的面积
            System.out.println("半径是"+r+"的圆面积是："+circleArea);
        }
    }
    public static void main(String[] args) {
        Example_09 ex=new Example_09();                             //创建对象
        try {
            ex.computeArea(10);                                     //调用方法
        } catch (Exception e) {
            System.out.println(e.getMessage());                     //输出异常信息
        }
    }
}
```

运行结果如图 11.10 所示。

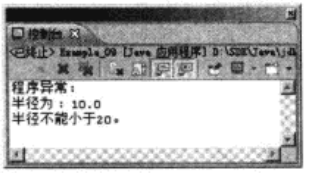

图 11.10　半径小于 20 抛出异常

说明 computeArea()方法根据圆的半径计算圆的面积，并且当圆的半径小于等于 0 时，使用 throw 语句抛出异常，由于该方法使用 throw 语句抛出了异常，所以必须在调用该方法时对其进行异常处理，本实例在主方法中使用 try…catch 语句对其进行了异常处理。

11.5.3　范例 3：方法中抛出异常

在项目开发中，通常是自顶向下进行的。在完成项目的整体设计后，需要对每个接口和类进行编写。如果一个类使用了其他类还没有实现的方法，则可以在实现其他类方法时让其抛出 UnsupportedOperationException，以便在以后进行修改完成。运行结果如图 11.11 所示。**（实例位置：光盘\TM\sl\11\10）**

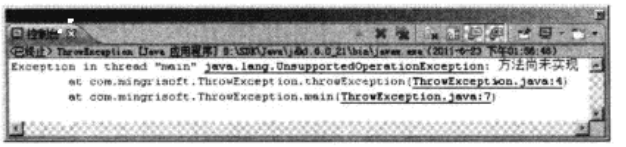

图 11.11　方法中抛出异常

（1）在 Eclipse 中创建项目 10，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 ThrowException 类，在该类中定义两个方法，一个是 throwException()方法，用于抛出异常；另一个是 main()方法，用于进行测试。代码如下：

```
public class ThrowException {
    public static void throwException() {
        throw new UnsupportedOperationException("方法尚未实现");          //抛出异常
    }
    public static void main(String[] args) {
        ThrowException.throwException();                                  //调用抛出异常的方法
    }
}
```

11.5.4　范例 4：方法上抛出异常

在方法的执行过程中，如果存在可能遇到引发问题的因素，则应该在定义方法时加以说明。例如

读取文件的方法可能遇到文件不存在的情况，此时需要在方法声明时抛出文件不存在异常。本范例将演示如何在方法上抛出异常。运行结果如图 11.12 所示。（**实例位置：光盘\TM\sl\11\11**）

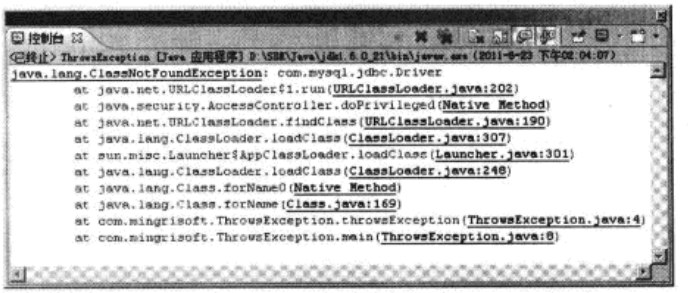

图 11.12　方法上抛出异常

（1）在 Eclipse 中创建项目 11，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 ThrowsException 类，并在该类中定义两个方法，一个是 throw Exception() 方法，用于抛出异常；另一个是 main()方法，用于进行测试。代码如下：

```
public class ThrowsException {
    public static void throwsException() throws ClassNotFoundException {          //抛出异常
        Class.forName("com.mysql.jdbc.Driver");
    }

    public static void main(String[] args) {
        try {                                                                      //捕获异常
            ThrowsException.throwsException();                                     //调用抛出异常的方法
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

11.6　自定义异常

视频讲解：光盘\TM\lx\11\自定义异常.exe

11.6.1　创建自定义异常类

创建自定义的异常类需要继承自 Exception 类，并提供含有一个 String 类型形参的构造方法，该形参就是异常的描述信息，可以通过 getMessage()方法获得。例如：

```
public class NewException extends Exception{
    public NewException(String s){
```

```
        super(s);
    }
}
```

说明 上面代码创建了一个自定义异常类 NewException，也就是说 NewException 是自定义异常类的名称，该类继承自 Exception 类，该类构造方法的形参 s 是需要传递的异常描述信息，该信息可以通过异常类的 getMessage()方法获得。

11.6.2 使用自定义异常类

创建完自定义异常类后，就可以在程序中使用了。使用自定义异常类可以通过 throw 语句抛出异常，接下来通过实例来说明自定义异常类的使用。

【例 11.11】 在项目中创建 Example_12 类，在该类中计算圆的面积，设定圆的半径不能小于 20，如果半径小于 20，则调用自定义异常类 NewException，处理程序所发生的异常。（**实例位置：光盘\TM\sl\11\12**）

自定义异常类 NewException 的代码如下：

```
public class NewException extends Exception {                    //创建自定义异常类
    public NewException(Double r) {                              //有一个 Double 类型形参的构造方法
        System.out.println("发生异常： 圆的半径不能小于 20");
        System.out.println("圆的半径为:"+r);
    }
}
```

Example_12 类的代码如下：

```
public static void showArea(double r) throws NewException {      //创建求圆面积的方法
    if (r <= 20) {
        throw new NewException(r);                               //抛出异常
    }
    double area = 3.14 * r * r;                                  //计算圆的面积
    System.out.println("圆的面积是：" + area);                    //输出圆的面积
}
public static void main(String[] args) {
    try {
        showArea(10);                                            //调用 showArea()方法，传递半径 10
    } catch (NewException ex) {
        System.out.println(ex);                                  //输出异常信息
    }
}
```

运行结果如图 11.13 所示。

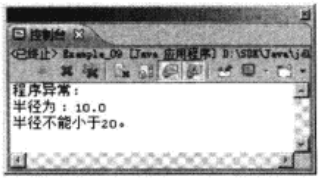

图 11.13　半径小于 20 抛出异常

11.7　异常的使用原则

视频讲解：光盘\TM\lx\11\异常的使用原则.exe

在程序中使用异常，可以捕获程序中的错误，但是异常的使用也要遵循一定的规则。下面是异常类的几项使用原则：

- ☑ 不要过多地使用异常，这样会增加系统的负担。
- ☑ 在方法中使用 try…catch 语句块捕获异常时，要对异常作出处理。
- ☑ try…catch 语句块的范围不要太大，这样不利于对异常的分析。
- ☑ 一个方法被覆盖时，覆盖它的方法必须抛出相同的异常或子异常。

11.8　经 典 范 例

前面的内容介绍了对异常的处理，下面给出几个在以后的实际开发中经常遇到的异常的例子。

11.8.1　经典范例 1：捕获单个异常

视频讲解：光盘\TM\lx\11\捕获单个异常.exe

当遇到异常时，除了可以将异常抛出，还可以将其捕获。抛出异常虽然简单，但是有时却不得不使用捕获来处理异常。如果程序遇到异常而没有捕获，则程序会直接退出。这在大多数情况下是不能被接受的，至少需要保存程序当前状态才能退出。本范例将演示如何捕获单个异常。运行结果如图 11.14 所示。（**实例位置：光盘\TM\sl\11\13**）

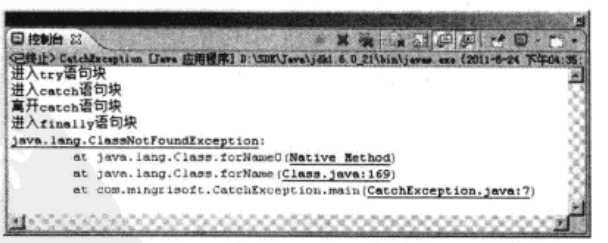

图 11.14　捕获单个异常

（1）在 Eclipse 中创建项目 14，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 CatchException 类，在该类的 main()方法中应用 try…catch…finally 语句捕获单个异常。代码如下：

```
public class CatchException {
    public static void main(String[] args) {
        try {                                                    //定义 try 语句块
            System.out.println("进入 try 语句块");
            @SuppressWarnings("unused")
            Class<?> clazz = Class.forName("");                  //得到一个空的 Class 对象
            System.out.println("离开 try 语句块");
        } catch (ClassNotFoundException e) {                     //定义 catch 语句块
            System.out.println("进入 catch 语句块");
            e.printStackTrace();
            System.out.println("离开 catch 语句块");
        } finally {                                              //定义 finally 语句块
            System.out.println("进入 finally 语句块");
        }
    }
}
```

说明 Java 中捕获异常是通过 try…catch…finally 语句来完成的。其中 try 语句块是必需的，catch 和 finally 语句块可以选择一个或者两个。try 语句块用来放置可能出现问题的语句，catch 语句块用来放置异常发生后执行的代码。finally 语句块用来放置无论是否发生异常都需要执行的代码。

11.8.2　经典范例 2：数据库操作异常

视频讲解：光盘\TM\lx\11\数据库操作异常.exe

数据库操作异常即 SQLException，通常发生在出现数据库访问错误时。本范例将演示出现数据库操作异常的情况。运行结果如图 11.15 所示。（**实例位置：光盘\TM\sl\11\14**）

图 11.15　数据库操作异常

（1）在 Eclipse 中创建项目 15，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 ExceptionTest 类，在该类的 main()方法中，编写数据库连接的代码，并且捕获可能抛出的异常。代码如下：

```
public class ExceptionTest {
    public static void main(String[] args) {
        String URL = "jdbc:mysql://localhost:3306/db_database";          //MySQL 数据库的 URL
        String DRIVER = "com.mysql.jdbc.Driver";                         //MySQL 数据库的驱动
        String USERNAME = "mr";                                          //数据库的用户名
        Connection connection = null;
        try {
            Class.forName(DRIVER);                                       //加载驱动
            //建立连接
            connection = DriverManager.getConnection(URL, USERNAME, "");
        } catch (SQLException e) {                                       //捕获 SQLException
            e.printStackTrace();
        } catch (ClassNotFoundException e) {                             //捕获 ClassNotFoundException
            e.printStackTrace();
        } finally {
            try {
                connection.close();                                      //释放资源
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

11.9 本章小结

本章向读者介绍的是 Java 中的异常处理机制。通过本章的学习，读者应了解异常的概念、几种常见的异常类，掌握异常处理技术，以及如何创建、激活用户自定义的异常处理器。Java 中的异常处理是通过 try…catch 语句来实现的，也可以使用 throws 语句向上抛出。建议读者不要将异常抛出，应该编写异常处理语句。对于异常处理的使用原则，读者也应该理解。

11.10 实战练习

1．编写一个异常类 MyException，再编写一个 Student 类，该类有一个产生异常的 speak(int m)方法。要求参数 m 的值大于 1000 时，方法抛出一个 MyException 对象。最后编写主类，在主方法中创建 Student 对象，让该对象调用 speak()方法。（**答案位置：光盘\TM\sl\11\15**）

2．创建 Number 类，通过类中的 count()方法可得到任意两个数相乘的结果，并在调用该方法的主方法中使用 try…catch 语句捕捉可能发生的异常。（**答案位置：光盘\TM\sl\11\16**）

3．创建 Computer 类，该类中有一个计算两个数的最大公约数的方法，如果向该方法传递负整数，该方法就会抛出自定义异常。（**答案位置：光盘\TM\sl\11\17**）