第五章 输入输出

5.1 异常处理

5.1.1 异常处理的概念

异常处理：异常又称例外，是特殊的运行错误对象。它是一种程序容错机制，当程序运行时，遇到用户或环境的错误时，程序要有能力处理这些错误，并从错误中恢复出来并执行。如果无法恢复，则应该给予一些错误提示。

Java 中声明了很多异常类，每个异常类都代表一种运行错误，类中包含了该运行错误的信息和处理错误的方法。

每当 Java 程序运行过程中发生一个可识别的运行错误（Java 中有一个与之对应的类）时，即该错误有一个异常类与之相对应时，系统都会产生一个相应的异常类对象，即产生一个异常。

当遇到异常时，应该怎么做：

（1）不处理异常。但是需要声明一下，将异常抛给环境，也就是 JVM，Java 虚拟机。

（2）处理异常。

Java 异常处理机制的优点：

（1）将错误处理代码从常规代码分离出来；

（2）按错误类型和差别分组；

（3）对无法预测的错误的捕获和处理；

（4）克服了传统方法的错误信息有限的问题；

（5）把错误传播给调用堆栈。

错误分类：一类叫做错误，是致命性的，程序无法处理，Error 类是所有错误类的超类；另外一类是异常，非致命性的，可编程捕获和处理，Exception 类是所有异常类的超类。

异常分为非检查型异常和检查型异常。对于非检查型异常来说：

（1）不期望程序捕获的异常，在方法中不需要声明，编译器也不进行检查；

（2）继承自 RuntimeException；

（3）不要求捕获和声明的原因：引发 RuntimeException 的操作在 Java 应用程序中会频繁出现；

（4）它表示的问题不一定作为异常处理。如在除法运算时检查 0 值，而不适用 ArithmeticException。可以在使用引用前测试控制。

对于检查型异常来说，它是其他类型的异常。如果被调用方法抛出一个类型为 E 的检查型异常，那么调用者必须捕获或者声明抛出 E（或者 E 的父类），编译器是要检查的。

Java 预定义的一些常见异常如图 1 所示。

图 1

5.1.2 异常的处理

1.检查型异常的处理

（1）声明抛出异常。不在当前方法内处理异常，使用 throws 子句声明将异常抛出到调用方法中；如果所有的方法都选择抛出此异常，最后 JVM 将捕获它，输出相关错误信息并终止程序运行。

（2）捕获异常。使用 try{}catch{}捕获异常进行处理。

如图 2 所示，这是一个异常处理示意图。

方法一调用方法 2，然后方法 2 调用方法 3，方法是调用方法 4，如果在方法 4 中探测异常发生，那么方法 4 选择抛出异常，就会沿着调用栈方向向上抛出，也就是说抛给方法 3，然后 2，然后 1，如果方法 1 打算处理异常，那就应该捕获异常处理。

图 2

例：抛出异常

代码：

```
public void openThisFile(String fileName)
throws java.io.FileNotFoundException{
    //code for method
}
public void getCustomerInfo()
throws java.io.FileNotFoundException{
    //do something
    this.openThisFile("customer.txt");
    //do something
}
```

如果在 openThisFile 中抛出了 FileNotfoundException 异常，getCustomerInfo 将停止执行，并将此异常传送给它的调用者。

关于捕获异常的语法，见图 3 所示。

图 3

2.生成异常对象：

三种方式：由 Java 虚拟机生成；Java 类库中某些类生成；自己写的程序中生成和抛出异常对象。

抛出异常对象都是通过 throw 语句实现，异常对象必须是 Throwable 或者其子类的实例。

例：生成异常对象

代码：

```
class ThrowTest{
    public static void main(String args[]){
        try{
            throw new ArithmeticException();
        }
        catch(ArithmeticException ae){
```

```
            System.out.println(ae);
        }
        try{
            throw new ArrayIndexOutOfBoundsException();
        }
        catch(ArrayIndexOutOfBoundsException ai){
            System.out.println(ai);
        }
        try{
            throw new StringIndexOutOfBoundsException();
        }
        catch(StringIndexOutOfBoundsException si){
            System.out.println(si);
        }
    }
}
```

3.声明自己的异常类

自定义的所有异常类必须是 Exception 的子类。

声明语法如下:

```
public class MyExceptionName extend SuperclassOfMyException{
    public MyExceptionName(){
        super("Some string explaining the exception");
//继承的这个超类，可以是 Exception 也可以是 Exception 的子类
    }
}
```

5.2 输入输出流

1.输入流和输出流

输入流是程序从空间之外的别的地方流入程序空间里面；输出流是反过来，将程序空间中的数据送到程序空间之外的地方。

预定义的 I/O 类:

从流的方向分为：输入流和输出流。

从流的分工划分：节点流（真正的去访问文件，输入输出操作的流）和处理流（在节点流之上对信息进行加工处理转换）。

从流的内容：面向字符的流（字符数据）；面向字节的流（一般目的）。

面向字符的流：源或目标通常是文本文件；能够实现内部格式和文本文件中的外部格式之间转换；内部格式是 16 位，外部是 utf 格式，包括 ascii 等。

2.面向字符的抽象流类——Reader 和 Writer

java.io 包中所有字符流的抽象超类。Reader 提供了输入字符的 API，Writer 提供了输出字符的 API。它们的子类又分为两大类：节点流和处理流。

多数程序使用这两个抽象类的一系列子类来读入/写出文本信息。例如 FileReader/FileWriter 用来读写文本文件。

面向字符的流如图 4 所示，阴影部分为节点流，其它为处理流。

图 4

5.3 文件读写

5.3.1 写文本文件

为了在磁盘上创建一个文本文件并往其中写入字符数据，需要使用到 FileWriter 类，它是 OutputStreamWriter 类的子类，而后者又是 Writer 类的子类。

例：创建文件并写入若干行文本

```
import java.io.*;
class FileWriterTester{
    public static void main(String[] args) throws IOException{
        //main 方法中声明抛出 IO 异常
        String fileName = "Hello.txt";
        FileWriter writer = new FileWriter(fileName);
        writer.write("Hello\n");
        writer.write("This is my first text file.\n");
        writer.close()
    }
}
```

我们发现在利用\n 换行的时候并不能达到我们想要的效果，这是因为\n 在不同平台下效果不一样。我们还发现，每一次运行都是删除原有的文件，重新再创建一个文件，根本不询问也不判断。

例：写入文本文件，并处理 IO 异常

```
import java.io.*;
class FileWriterTester{
    public static void main(String[] args){
        //main 方法中声明抛出 IO 异常
        String fileName = "Hello.txt";
        try{
            //将所有 IO 操作放入 try 块中
            FileWriter writer = new FileWriter(fileName,true);
            //true 表示是追加而不是覆盖
            writer.write("Hello\n");
            writer.write("This is my first text file.\n");
            writer.close()
        }
        catch(IOException iox){
            //若出错则屏幕提示
            System.out.println("Problem Writing" + fileName);
        }
    }
```

```
}
```

说明：

运行此程序，会发现源文件内容后面又追加了第二个程序的内容，这是将构造方法第二个参数设置为 true 的效果。

如果将文件属性改为只读属性，再运行本程序，就会出现 IO 错误，程序将转入 catch 块中，给出错误信息。

BufferedWriter 类

FileWriter 和 BufferedWriter 类都用于输出字符流，包含的方法几乎完全一样，但后者多提供了一个 newLine()方法用于换行。**这个方法可以跨平台**。

例：写入文本文件，使用 BufferedWriter

```
import java.io.*;
class BufferedWriterTester{
    public static void main(String[] args) throws IOException{
        //main 方法中声明抛出 IO 异常
        String fileName = "Hello.txt";
        BufferedWriter out = new BufferedWriter(new FileWriter(fileName));
        out.write("Hello");
        out.newLine();
        writer.write("This is my first text file.\n");
        out.newLine();
        out.close()
    }
}
```

5.3.2 读文本文件

从文本文件读字符数据需要使用 FileReader 类，由图 4 可以看出它继承 Reader 抽象类的子类 InputStreamReader。对应于写文本文件的缓冲器，读文本文件也有缓冲器 BufferedReader，具有 readLine()函数，可以对换行符进行识别，一行一行地读取输入流中的内容。

例：读取 Hello.txt 文本并显示在屏幕上

```
import java.io.*;
class BufferedReaderTester{
    public static void main(String[] args) throws IOException{
        //main 方法中声明抛出 IO 异常
        String fileName = "D:/Hello.txt";
        String line;
        try{
            BufferedReader in = new BufferedReader(new FileReader(fileName));
            line    = in.readLine();
            while(line ! = null)
            {
                System.out.println(line);
                line = in.readLine();
```

```
            }
            in.close();
        }
        catch(IOException iox){
            System.ourt.println("Problem reading.");
        }
    }
}
```

还有一种判断文本文件结尾的方法，那就是利用 Reader 类的 read()方法返回的一个 int 型整数。如果读到文件末尾，该方法返回-1.因此，可以将上例中读文件判断部分改为：

```
int c;
while((c = in.read())!=-1)
    System.out.println((char)c);
```

关闭输入流文件并不像关闭输出流文件那么重要，因为当检测到文件结尾时输入流中就没有数据了，因而没有必要去刷新它。

5.3.3 写二进制文件

Java.io 包中的 OutputStream 及其子类专门用于写二进制数据。FileOutputStream 是其子类，可用于将二进制数据写入文件，用于一般目的输出（非字符数出），用于成组字节输出此方法的二进制文件的数据可以被运行在任何平台上的 Java 程序正确读取。DataOutputStream 是 OutputStream 的另一子类，它可以链接到一个 FileOutputStream 上，便于写各种基本类型的数据。

DataOutputStream 类的成员如图 5，图 6 所示。

图 5

图 6

例：将 int 写入文件

```
import java.io.*;
class FileOutputStreamTester{
    public static void main(String[] args){
        String fileName = "myData.dat";
        int value0 = 255, value1 =0, value2 = -1;
        try{
            DataOutputStream out = new DataOutputStream
            (new FileOutputStream(fileName));
            //将 FileOutputStream 与 DataOutputStream 连接可
            //输出不同类型数据
            out.writeInt(value0);
            out.writeInt(value1);
            out.writeInt(value2);
            out.close();
        }
```

```
            catch(IOException iox){
                System.out.println("Problem writing");
            }
        }
}
```

BufferedOutputStream 类

用法示例：

```
DataOutputStream out = new DataOutputStream(new
                BufferedOutputStream((new FileOutputStream(filename)));
```

例：向文件写一个字节并读取

向文件中写入内容为-1 的一个字节，并读取出来。

```
import java.io.*;
public class FileOutputStreamTester{
    public static void main(String[] args) throws Exception{
        DataOutputStream out = new DataOutputStream(
        new FileOutputStream("trytry.dat"));
        out.writeByte(-1);out.close();
        DataInputStream in = new DataInputStream(
        new FileInputStream("trytry.dat"));
        int a = in.readByte();
        System.out.println(Integer.toHexString(a));
        System.out.println(a);
        in.skip(-1);//往后一个位置，以便下面重新读出
        a=.in.readUnsignedByte();
        System.out.println(Integer.toHexString(a));
        System.out.println(a);
    }
}
```

运行结果：

```
ffffffff    -1    ff    255
```

5.3.4 读二进制文件

读二进制文件，比较常用的类有 FileInputStream,DataInputStream,BufferedInputStream 等。DataInputStream 也提供了很多方法用于读入布尔型、字节、字符、整型、长整型、短整型、单精度、双精度等数据。

例：读取二进制文件中的 3 个 int 型数字并相加

图 7

例：通过捕获异常控制读取结束

图 8

第二个 try 就一直在读，直到读结束了，while 循环结束，然后出来，直接到了第一个 catch 这里，去执行这里面的操作。

之所以外面还有一层 try…catch 操作，就是为了对其它的 IOEXCEPTION 做准备。

例：用字节流读取文本文件

```
import java.io.*;
public class InputStreamTester{
    public static void main(String[] args) throws Exception{
        FileInputStream s =new FileInputStream("Hello.txt");
        int c;
        while((c = s.read())!=1)//读取 1 字节，结果返回-1
            System.out.write(c);
        s.close();
    }
}
```

说明：

read()方法读取一个字节，转化为[0,255]的之间的一个整数，返回一个 Int。如果读到了文件末尾，则返回-1.

wirte(int)方法写一个字节的低 8 位，忽略高 24 位。

5.3.5 FILE 类

FILE 类的作用：

（1）创建、删除文件；（2）重命名文件；（3）判断文件的读写权限以及是否存在；（4）设置和查询文件的最近修改时间等；（5）构造文件流时使用 FILE 类作为参数。

例：FILE 类举例

创建文件 Hello.txt，如果存在则删除旧文件，不存在则直接创建新的。

代码：

```
import java.io.*;
public class FileTester{
    public static void main(String[] args){
        File f = new File("Hello.txt");
        if(f.exists()) f.delete();
        else{
            try{
                f.createNewFile();
            }
            catch(Exception e){
                System.out.println(e.getMessage());
            }
        }
    }
}
```

运行结果：

因为在前面的例子中已经创建 Hello.txt，所以第一次运行将删除这个文件；第二次运行则又创建了一个此名的空文件。

分析：

在试图打开文件之前，可以使用 File 两类的 isFile 方法来确定 File 对象是否代表一个文件而非目录。

还可以通过 exists 方法判断同名文件或者路径是否存在，进而采取正确的方法，避免造成误操作。

例：改进的文件复制程序

```
import java.io.*;
public class NewCopyBytes{
    public static void main(String[] args){
        DataInputStream instr;
        DataOutputStream outstr;
        if(args.length != 2){
            System.out.println("Please Enter file names!");
            return;
        }
        File inFile = new File(args[0]);
        File outFile = new File(args[1]);
        if(outFile.exists()){
            System.out.println(args[1]+"already exists.");
        }
        if(!inFile.exists()){
            System.out.println(args[0] + "does not exist.");
        }
        try{
            instr = new DataInputStream(new BufferedInputStream(
            new FileInputStream(inFile)));
            outstr = new DataOutputStream(new BufferedOutputStream(
            new FileOutputStream(outFile)));
            try{
                int data;
                while(true){
                data = instr.readUnsignedByte();
                outstr.writeByte(data);
                }
            }
            catch(EOFException eof){
            outstr.close();instr.close();return;}
        }
        catch (FileNotFoundException nfx){/*代码不再写*/}
        catch(IOException ios){/*代码不再写出*/}
    }
}
```

5.3.6 处理压缩文件

java.util.zip 包中提供了一些类可以压缩格式对流进行读写。它们都继承自字节流类 OutputStream 和 InputStream。其中 GZIPOutputStream 和 ZipOutputStream 可分别把数据压缩成 GZIP 和 zip 格式，GZIPInputStream 和 ZipInputStream 可以分别把压缩成 GZIP 和 ZIP 格式的数据解压缩恢复原状。

例：压缩和解压缩 Gzip 文件

将文本文件 Hello.txt 压缩为文件 test.gz，再解压缩该文件，显示其中内容，并另存为 newHello.txt。

代码见图 9、图 10 所示。

图 9

图 10

图 9 首先是创建一个文件流对象 in，然后创建一个对象 out，它是在文件流外侧套一个 GZIPOutputStream，这便是创建了压缩文件“test.gz”。然后就是和前面复制文件一样了，先读出 hello.txt 文件的内容，然后再写入 gz 文件中。

在图 10 中，是一个解压缩过程。构造输入流，也是在输入流最底下原声字节输入流外面接一个解压缩输入流，经过这个解压缩的输入流就可以自然的把从压缩文件中读出的数据恢复成普通的解压缩后的数据了。然后接下来我们指导它是一个文本文件，又希望我们最后把它恢复到一个文本文件，所以我们还得用读文本文件的方式去按行读，解压缩以后它还是一个二进制流，所以我们就要用 InputStreamReader，它是面向字节的流和面向字符的流之间的一个桥梁，经过它以后再经过一个 BufferedReader 那么我们就可以按行读了，所以在 while 中按行读，读一行在显示器显示一行。显示完关闭一行。接下来我们依然可以用刚才的方式，按行读取文本。由于仅仅是要解压缩并且把解压缩的结果写到新闻界去，我们也可以不管它是不是文本文件就简单解压缩就可以了，所以就只在 FileInputStream 外面套了一个 GZIPInputStream 这个解压缩的输入流，然后就用另外的复制文件方法那样一步一步完成复制。最后关闭。

运行结果：

首先生成了压缩文件“test.gz”，再读取显示其中的内容，和”Hello.txt”中的内容完全一样；再解压缩文件“newHello.txt”，和“Hello.txt”中的内容也完全相同。

例：Zip 文件的压缩与解压缩

代码：

图 11

图 12

操作文件的是这个 FileOutputStream 二进制流，然后套了一个缓冲的流，最外层是 ZIP 的流。压缩多个文件，就要有多个文件之间的间隔。首先用 for 循环依次去打开不同的输入文件，所以在每一轮我们取一个命令行参数作为名来作为输入流，然后在压缩每个文件前首先将 zipentry 这个文件开始标记写到文件中去。所以用压缩源来源文件名做参数，构造一个 zipentry，写进去。接下来就是 read 和 write 操作。然后关闭输入流。再循环，循环后结束，关闭输出流。

图 13

接下来的部分是解压缩。利用一个 ZipInputStream 用来解压缩，也是在普通的二进制流和缓冲流基础上加一个解压缩流。每次外层循环判断还有文件不，在内层循环中解压缩，然后往显示器上写。等外层循环完了，关闭所有的流。

运行结果：

在命令行输入两个文本文件名后，将生成 test.zip 文件；在屏幕上显示解压后每个文件的内容；在资源管理器窗口，可以使用 winzip 软件解压缩该文件，恢复出和原来文件相同的两个文本文件。

例：解压缩 zip 文件，并恢复原来的路径

具体代码见课本

5.3.7 对象序列化

需要将程序编写时的一些数据永久保存，那么就要对对象进行序列化，即按照整体写到文件中，再整体读出来。

ObjectInputStream/ObjectOutStream 类

1.实现对象的读写：通过前者把对象读入程序；通过后者把对象写入磁盘文件。

注意：transient 和 static 类型的变量不会被保存。对象要想实现序列化，其所属类必须实现 serializable 接口。

2.实现方法：

OBjectOutputStream

必须通过另一个流构造 ObjectOutputStream：

```
FileOutputStream out = new FileOutputStream(“theTime”);
ObjectOutputStream s = new ObjectOutputStream(out);
s.writeObject(“Today”);
s.writeObject(new Date());
s.flush();
```

ObjectInputStream:

必须通过另一个流构造 ObjectInputStream：

```
FileInputStream in = new FileInputStream(“theTime”);
ObjectInputStream s = new ObjectInputStream(out);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

3.Seriealizable

Seriealizable 的定义

```
package java.io;
public interface Serializable{
//there's nothing in here!};
```

这是一个典型的代码。实现接口的语句：

```
public class MyClass implements Serializable {/*程序体*/}
```

使用关键字 transient 可以阻止对象的某些成员被自动写入文件。

例：创建一个书籍对象输出并读出

创建一个书记对象，并把它输出到一个文件 book.dat 中，然后再把该对象读出来，在屏幕上显示对象信息。

程序代码：

```
import java.io.*;

public class SerializableTester{
    public static void main(String args[]) throws IOException,
    ClassNotFoundException{
        //新建一个 Book 类的对象
        Book book = new Book(100032, "Java", "Wang",30);
        ObjectOutputStream oos =new ObjectOutputStream(
        new FileOutputStream("book.dat"));//创建一对象输出流
        oos.writeObject(book);//向流中写对象
        oos.close();//关闭输出流
        book = null;
        ObjectInputStream ois = new ObjectInputStream(
        new FileInputStream("book.dat"));//创建一对象输入流
        book= (Book) ois.readObject();//读入对象并强制转型为 BOOK 类
        ois.close();
        System.out.println("ID is" + book.id);//读取对象信息并显示
        //剩余输出信息不再累述
    }
}

class Book implements Serializable{
    int id;
    String name;
    String author;
    float price;
    public Book(int id, String name,
    String author, float price){
        this.id =id;
        this.name = name;
        this.author = author;
        this.price = price;
    }
}
```

如果希望增加 Book 类的功能，使其还能够具有借书方法 borrowBook，并保存借书人的借书号 borrowID，课对 BOOK 类添加如下内容：

```
transient int borrowerID;
public void borrowBook(int ID){
    thi.borrowerID = ID;
```

```
}
//在 main 方法中创建了 Book 类的一个对象后，紧接着调用此方法，输入 ID 为 2018：
book.borrowBook(2018);
//最后再要求从读入的对象中输出 borrowerID
System.out.println(book.borrowerID);
```

可以发现 borrowID 为 0，这是因为声明为 transient 了，所以保存和读出对象时都不会进行处理，如果去掉 transient 关键字，则可以正确读出 2018。

5.3.8 随机文件读写

RandomAccessFile 类

（1）可跳转到文件的任意位置读写数据；

（2）课在随机文件中插入数据，而不破坏该文件的其它数据。

注意：所有的位置这些东西都需要自己计算。

（3）实现了 DataInput 和 DataOutput 接口，可使用普通的读写方法；

（4）有个位置指示器，指向当前读写处的位置。刚打开文件时，文件指示器指向文件的开头出。对文件指针显式操作的方法有：

int skipBytes(int n)：把文件指针向前移动指定的 n 个字节；

void seek(long)：移动文件指针到指定的位置；

long getFilePointer()：得到当前的文件指针。

（5）在等长记录格式文件的随机读取时有很大的优势，但仅限于操作文件，不能访问其它 IO 设备，如网络、内存映像等。

（6）构造方法：

public RandomAccessFile(File file, String mode) throws FileNotFoundException

public RandomAccessFile(String name, String mode) throws FileNotFoundException

（7）构造对象时，应指出操作：仅读，还是读写。

new RandomAccessFile("farrago.txt", "r");

new RandomAccessFile("farrago.txt", "rw");

要实现随机读写，文件的记录必须是等长的！