第 7 章

流式输入输出及文件处理

面向对象程序设计

7.1 流式输入输出处理机制

在Java语言中，所有的输入输出操作都采用流式处理机制。所谓流是指具有数据源和数据目标的字节序列的抽象表示。我们可以将数据写入流中，也可以从流中读取数据，实际上流中存放着以字节序列形式表示的准备流入程序或流出程序的数据。

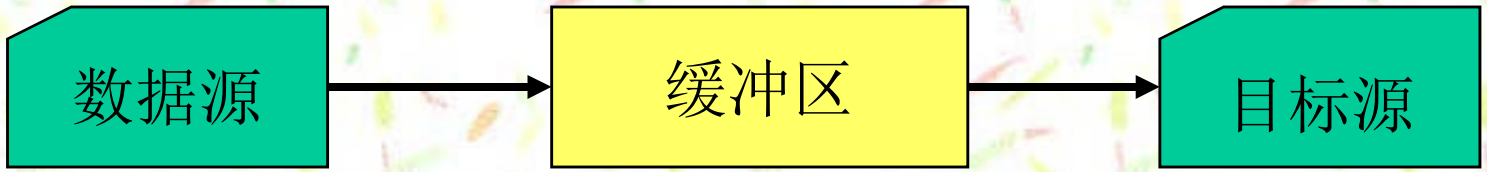
- 当试图将程序中的数据输出到输出设备时，需要将这些数据以字节序列的形式写入流中，此时的数据源是程序，数据目标是输出设备，这个流被称为输出流（**output stream**）。

- 当试图将外部的数据输入到程序中时，流中的数据源是输入设备，数据目标是程序，这个流被称为输入流（**input stream**）。

面向对象程序设计

Java程序使用流机制处理输入输出的主要好处是可以使程序中有关输入输出的代码与设备无关，这样既可以免去了解每一种设备的细节而带来的烦恼，也可以使得程序适应各种设备的输入输出。

流的基本处理单位为字节。如果每次只读写一个字节，会使得数据传输效率非常低，因此通常为流配备一个缓冲区(buffer)，我们将这种流称为缓冲流。



面向对象程序设计

在Java语言中，支持输入输出流的所有类被放置在java.io包中，其中主要包含两种类型的流，一种是二进制流（binary stream），另一种是字符流（character stream）。

当以二进制字节序列的形式写数据时，写到流中的数据与内存中的形式完全一样，即没有发生任何变化。当以字符的形式写数据时，由于Java中的字符采用Unicode编码，占据16个二进制位，因此，写入的每个字符为两个字节，先写高字节，后写低字节。

7.2 Java的输入输出流库

7.2.1 Java的输入输出流库

File 支持文件或目录操作的类

OutputStream 字节流输出操作的抽象类

InputStream 字节流输入操作的抽象类

Writer 字符流输出操作的抽象类

Reader 字符流输入操作的抽象类

RandomAccessFile 支持随机存取文件操作的类

面向对象程序设计

7.2.2 字节输入流InputStream

字节流是以字节序列的形式读写数据的方式。从输入设备或文件中读取数据使用的字节流被称为输入流，在Java语言中用InputStream类描述，并提供了下面几个用于读取数据的成员方法：

read()

read(byte[] buffer)

read(byte buffer[],int offset,int length)

skip(long n)

close()

7.2.3 字节输出流OutputStream

OutputStream类是一个抽象类，它将作为所有字节输出流类的父类，在这个类中包含下面5个主要的成员方法：

write(int b)

write(byte[] buffer)

write(byte[] buffer,int offset,int length)

flush()

close()

7.3 文件

利用文件组织和存储数据是一种常用的方式。在Java语言中，根据对文件的存取方式不同，提供了两个类用来描述文件及实现文件的各种操作。一个类是File类，用来支持顺序文件的操作；另一个类是RandomAccessFile类，用来支持随机文件的操作。

7.3.1 文件的创建与管理

1. 创建File对象

Java提供了三种创建File对象的方法：

```
File fileObject=new File("c:/jdk1.2/File.java");
File dirObject=new File("c:/jdk1.2/src/java");
==========================================
File fileObject=new File("c:/jdk1.2/io","File.java") ;
==========================================
File dirObject=new File("c:/jdk1.2/src/java/io");
File fileObject=new File(dirObject,"File.java");
```

2. 检测File对象

File类提供了一整套应用于File对象的成员方法。

exists()　检测File对象所描述的文件或目录是否存在。

isDirectory()　检测File对象所描述的是否为目录。

isFile()　检测File对象所描述的是否为文件。

isHidden()　检测File对象所描述的是否为一个隐含文件。

canRead()　检测File对象所描述的文件是否可读。

canWrite()　检测File对象所描述的文件是否可写。

equals(Object obj)　检测File对象描述的绝对路径与obj的绝对路径是否相等。

```
import java.io.*;
public class TryFile
{
    public static void main(String[] args)
    {
        File dirObject=new File("d:/java_class/java/io");
        System.out.println(dirObject+
            (dirObject.isDirectory()?" is":" is not")+" a directory.");
        File fileObject=new File(dirObject,“File.java”);
        System.out.println(fileObject+
            (fileObject.exists()?" does":" does not")+" exist");
        System.out.println("You can"+
            (fileObject.canRead()?"  ":" not")+" read "+fileObject);
        System.out.println("You can"+
            (fileObject.canWrite()?"  ":" not")+" write "+fileObject);
    }
}
```

面向对象程序设计

3. 访问File对象

在File类中，提供了一些获取文件信息的成员方法。

getName() 以String形式返回File对象描述的文件名。

getPath() 以String形式返回File对象描述的路径。

getAbsolutePath() 以String形式返回绝对路径。

getParent() 以String形式返回父目录名。

list() 返回目录中包含的全部文件名或目录名。
如果目录为空，则数组也为空。

length() 以long类型的形式返回当前文件所
占有的字节数目。

lastModified() 以long类型的形式返回File对象
描述的文件或目录最后一次被修改的时间。

4. 修改File对象

除了前面介绍的创建File对象、检测File对象和访问File对象的操作外，File类还提供了一些用来修改File对象的成员方法。

renameTo(File path) 将File对象的文件路径改为path。

setReadOnly() 将File对象设置为只读。

mkdir() 将创建由File对象指定的目录。

mkdirs() 将创建由File对象指定的目录。

createNewFile() 将创建由File对象指定的新文件。

delete() 删除File对象描述的文件或目录。

7.3.2 顺序文件的读取

1. 写文件

（1）利用FileOutputStream写文件

FileOutputStream是OutputStream的子类，它继承了OutputStream类的所有成员方法，实现了抽象方法write()，并将流的数据目标定义为磁盘文件。

利用构造方法可以将一个磁盘文件与输出流建立连接，以便利用FileOutputStream从OutputStream继承的有关成员方法向磁盘文件写入数据。

面向对象程序设计

```
import java.io.*;
public class TryWriteFile
{
        public static void main(String[] args)
        {
           byte[] info={12,34,56,76,89,54,28,90};
           String dirName="d:/test";       String fileName="test";
           try   {
              File dirObject=new File(dirName);         //创建目录路径对象
              if (!dirObject.exists())   dirObject.mkdir(); //检测目录是否存在
            File fileObject=new File(dirObject,fileName);   //创建文件对象
              fileObject.createNewFile();                  //创建空文件
              FileOutputStream outputFile=new FileOutputStream(fileObject);
              for (int i=0;i<info.length;i++)  outputFile.write(info[i]);       //写数据
              outputFile.close();                          //关闭文件
           }
           catch (IOException e)  {
             System.out.println("IOException "+e+" occurred.");
         }
      }
}
```

（2）利用DataOutputStream 写文件

FilterOutputStream是所有过滤输出流类的父类。在这个类中，重写了**OutputStream**类中的全部成员方法，增强了数据处理的能力。同时它还含有若干个子类，其中**DataOutputStream**应用的比较广泛，利用它可以将基本数据类型的数据直接写入输出流。

DataOutputStram类定义的新成员方法

writeByte(int value)

writeBoolean(boolean value)

writeChar(int value)

writeShort(int value)

writeInt(int value)

writeLong(long value)

writeFloat(float value)

writeDouble(double value)

writeChars(String s)

面向对象程序设计

```
public class TryDataStream
{
    public static void main(String[] args)
    {
      int[] intArray={10,20,30,40,50,60};
      float[] floatArray={11.0f,22.0f,33.0f,44.0f,55.0f};
      String dirName="d:/MyData";
      try  {
        File dir=new File(dirName);    //创建目录路径对象
        if (!dir.exists()) dir.mkdir();    //检测目录是否存在
    else   if (!dir.isDirectory())   {
          System.out.println(dirName+" is not a directory.");   return;
        }
      File aFile=new File(dir,"data.txt");      //创建文件对象
      aFile.createNewFile();      //创建空文件
      DataOutputStream myStream=new DataOutputStream(new FileOutputStream(aFile));
      for (int i=0;i<intArray.length;i++)   myStream.writeInt(intArray[i]);
      for (int i=0;i<floatArray.length;i++)    myStream.writeFloat(floatArray[i]);
     }
     catch(IOException e)  {  System.out.println("IO exception thrown: "+e);   }
    }
}
```

（3）利用ObjectOutputStream将对象写入文件

利用DataOutputStream可以将各种基本数据类型和String类对象的内容写入文件，但不能处理一般的类对象，为此OutputStream类还提供了一个子类ObjectOutputStream可以实现将对象写入文件的功能。其操作过程为：首先创建一个ObjectOutpurStream对象，并将一个FileOutputStream对象作为参数传递给ObjectOutpurStream的构造方法，然后调用writeObject(Object obj)成员方法将通过参数传入的对象内容写入文件。

2. 读文件

(1) 利用FileInputStream读文件

正像FileOutStream是OutputStream的子类一样，FileInputStream是InputStream的子类，它继承了InputStream类的所有成员方法，实现了抽象方法read()，并将流的数据源定义为文件。

面向对象程序设计

```
public class TryReadFile
{
    public static void main(String[] args)
    {
       byte[] info=new byte[8];
       String dirName="d:/test";    String fileName="test";
       try    {
          File dirObject=new File(dirName);  //创建目录对象
          File fileObject=new File(dirObject,fileName);  //创建文件对象
          FileInputStream inputFile=new FileInputStream(fileObject);
      inputFile.read(info);    //读文件
          inputFile.close();      //关闭文件
       }
       catch(FileNotFoundException e) {
          System.out.println("FileNotFoundException"+e+" occured.");
       }
       catch (IOException e)  { System.out.println("IOException "+e+" occured.");   }
       for (int i=0;i<info.length;i++)
          System.out.print(info[i]+"   ");
    }
}
```

(2) 利用DataInputStream 写文件

FilterInputStream是所有过滤输入流类的父类。在这个类中，重写了InputStream类中的全部成员方法，增强了读取数据的能力。其中DataInputStream能够直接从输入流中读取基本数据类型和String类对象的数据。在这个类中，有一套与DataOutputStream对应的成员方法，利用它们可以根据不同的基本数据类型，一次读取若干个字节的内容。

```
public class TryDataStreamIn
{
    public static void main(String[] args)
    {
      int[] intArray=new int[6];
      float[] floatArray=new float[5];
      String dirName="d:/MyData";
      try   {
        File dir=new File(dirName);            //创建目录对象
        File aFile=new File(dir,"data.txt");       //创建文件对象
        DataInputStream myStream=new DataInputStream(new FileInputStream(aFile));
        for (int i=0;i<intArray.length;i++){
           intArray[i]=myStream.readInt();  System.out.println(intArray[i]);
        }
        for (int i=0;i<floatArray.length;i++){
          floatArray[i]=myStream.readFloat();  System.out.println(floatArray[i]);
        }
       }
       catch(FileNotFoundException e)  { System.out.println("FileNotFoundException"+e+" occured."); }
       catch(IOException e)  { System.out.println("IO exception thrown: "+e);   }
    }
}
```

(3) 利用ObjectInputStream读取文件中的对象内容

利用DataInputStream可以直接读取存储在文件中的对象内容。与将对象写入文件的过程类似，从文件中读回对象，首先需要创建一个ObjectInputStream对象，然后调用成员方法readObject()，返回一个Object类型的对象，最后根据需要将其转换成相应的类。

7.4 字符流

在Java语言中，提供了字符输出流Writer和字符输入流Reader。由于Java采用的是Unicode编码，每个字符占16个二进制位，即两个字节，所以每次读写操作以两个字节为单位。不仅如此，在读取的过程中，还要承担在Unicode编码与本地机器编码之间的转换。

1．字符输出流

在Java语言中，Writer是一个抽象类，它是所有以字符为单位的输出流的父类，其中定义了字符输出流在实现写流操作时需要的大部分成员方法，为向输出流写入字符提供了方便。

write(int c)

write(char cbuf[])

write(char cbuf[], int off, int len)

write(String str)

write(String str, int off, int len)

面向对象程序设计

2．字符输入流

与Writer对应，Reader也是一个抽象类，它是所有以字符为单位的输入流的父类，同样也定义了字符输入流在实现读流操作时需要的大部分成员方法。

read()

read(char cbuf[])

read(char cbuf[], int off, int len)

skip(long n)

ready()

close()

面向对象程序设计

7.5 对象的串行化

所谓串行化是指在外部永久性文件中存放或检索对象的过程。将对象写入文件被称为串行化对象，从文件中读取对象被称为并行化对象。如果一个对象具有串行化能力，那么将这个对象的内容永久地保存或从永久性的文件中读取出来都将是一件很容易的事情。

在Java语言，若要使对象具有串行化能力，只需要让对象所属的类实现Serializable接口即可。实际上，在Serializable接口中，没有声明任何内容，因此在实现该接口的类中也没有任何接口方法需要实现。

OutputStream类有一个子类ObjectOutputStream，利用这个类提供的writeObject(Object obj)成员方法可以实现将任何实现了Serializable接口的类对象写入永久性文件中的功能。

```
FileOutputStream output=
        new FileOutputStream("MyFile");
ObjectOutputStream objectOut=
        new ObjectOutputStream(output);
objectOut.writeObject(MyObject);
```

```
MyClass MuObject;
try
{
  FileInputStream input=new FileInputStream("MyFile");
  ObjectInputStream objectIn=new ObjectInputStream(input);
  MyObject=(MyClass)objectIn.readObject();
}
catch (IOException e){
    System.out.println(e);
}
catch(ClassNotFoundException e){
  System.out.println(e);
}
```

面向对象程序设计

```
class Employee implements Serializable
{
   private String name;     //姓名
   private double salary;    //工资
   private Date hireDay;    //开始受雇日期

public Employee(String n, double s, Date d)
{
name = n;
       salary = s;
       hireDay = d;
}
public Employee() {}
public String toString(){  return name + " " + salary + " " + hireYear();  }
public void raiseSalary(double byPercent){
                          salary *= 1 + byPercent / 100;  }
public int hireYear(){  return hireDay.getYear();  }
}
```

面向对象程序设计

```
public class ObjectFileTest
{  public static void main(String[] args)
   {  try  {
        Employee[] staff = new Employee[3];
        staff[0] = new Employee("Harry Hacker", 35000,new Date(1989,10,1));
        staff[1] = new Employee("Carl Cracker", 75000,new Date(1987,12,15));
        staff[2] = new Employee("Tony Tester", 38000,new Date(1990,3,15));
        ObjectOutputStream out =
            new ObjectOutputStream(new FileOutputStream("employee.dat"));
        out.writeObject(staff);        out.close();
         ObjectInputStream in =
             new ObjectInputStream(new FileInputStream("employee.dat"));
        Employee[] newStaff = (Employee[])in.readObject();
        int i;
        for (i = 0; i < newStaff.length; i++)  newStaff[i].raiseSalary(100);
        for (i = 0; i < newStaff.length; i++)  System.out.println( newStaff[i]);
     }
     catch(Exception e)  {System.out.print("Error: " + e);  System.exit(1);    }
   }
}
```