**输入输出流**

输入输出操作是计算机的必备功能：

通常一个程序需要从外部获取信息（输入），这个“外部”范围很广，包括诸如键盘、显示器、文件、磁盘、网络、另一个程序等；“信息”也可以是任何类型的，例如一个对象、字符串、图像、声音等。

Java通过一种称为流的机制来实现输入输出操作。通俗来讲：

流是指数据从一处流向另一处形成的数据流，分为输入流和输出流。

输入流代表从外设流入计算机的数据序列：外设→计算机

输出流代表从计算机流向外设的数据序列：计算机→外设

**I/O流的概念**

Java没有标准的输入输出语句，将信息的输入与输出过程抽象成输入输出流。输入是指数据流入程序，输出指数据从程序流出。**一个流就是一个从源头流向目的地的数据序列**。输入输出流可以与各种数据源和目标相连。Java所有的输入输出操作都要通过I/O包中的一些流类的方法来实现。

为了从信息源获取信息，程序打开一个输入流，这个输入流就在信息源与程序之间建立连接，程序可以从输入流读取信息。

当程序需要向目标位置写信息时，也需要打开一个输出流，这个输出流就在程序与输出目标之间建立连接，程序通过输出流流向这个目标位置写信息。

通常系统中有很多种输入输出设备，一些只可以是数据源，例如键盘、鼠标、扫描仪；一些只可以是目标，例如显示器；还有一些既可以是数据源也可以是目标，例如磁盘文件、运行的程序、网络连接等。

数据流中的数据分为：二进制数据、字符流序列、数字流序列等，不同类型有不同的处理方法。

不论数据从哪来，到哪去，也不论数据本身是何类型，读写数据的方法大体都一样，如：

顺序读写数据的方法

|  |  |
| --- | --- |
| 读/输入 | 写/输出 |
| 打开一个流 | 打开一个流 |
| 读信息 | 写信息 |
| 关闭流 | 关闭流 |

流式输入输出是常见的输入输出方式。例如，磁带机是典型的流式输入输出设备。特点是必须按照数据序列的排列顺序读写数据，而不能随意选择读写位置。

**预定义的I/O流类**

java.io包：是Java语言的标准输入输出类库，包中提供了一系列支持读/写的Java各种流类。每个流类代表某种类型的输入流或输出流，用于完成某种类型的流式输入或输出。例如，FileInputStream和FileOutputStream分别代表文件输入流和文件输出流，用于读写本地磁盘文件。

输入输出流可以从几方面进行**分类**：

* 输入流或输出流（从流的方向划分）
* 节点流或处理流（从流的分工划分）
* 字符流/面向字符的流或字节流/面向字节的流（从流的内容划分）

以上是从不同角度进行分类。例如：一个流可以是面向字符的输入流，同时又属于处理流。

不仅如此，将不同类型的流连接在一起的方法也有很多，所以Java程序中输入输出部分是比较复杂的，对于初学者而言尤其不容易掌握。后续将按若干专题来介绍最常用的输入输出操作的实现方法。

java.io包中的类首先被分为：

* 面向字符的流——专门用于字符数据的。
* 面向字节的流——通常用于一般目的的输入输出。

当然从根本上说，所有的数据都是由8位的字节组成，所以逻辑上讲，所有的流都可以被称为字节流。不过用于表示字符的字节流有其特定的处理单位，因此被称为字符流。

1. **字符流——Reader和Writer类**

字符流是针对字符数据的特点进行过优化的，因而提供一些面向字符的有用

特性。**字符流的源或目标通常是文本文件**。

字符流可以实现Java程序中内部格式和文本文件、显示输出、键盘输入等外

部格式之间的转换。

Reader和Writer类是所有字符流的抽象基类。Reader提供了输入字符的API

及其部分实现，Writer则提供了输出字符的API及其部分实现。

Reader和Writer的子类又可分为两大类：一类用来从数据源读入数据或往目的地写出数据（称为**节点流**）；另一类对数据执行某种处理（称为**处理流**）。

绝大多数程序使用Reader和Writer这两个抽象类的一系列子类来**读入和写出文本信息**。例如FileReader/FileWriter就是用来读/写文本文件的。

**2）字节流——InputStream和OutputStream**

很多情况下，数据源或目标中**含有非字符数据**，例如Java编译器产生的字节码文件中含有

Java虚拟机的指令。这些信息不能被解释为字符，所以必须用字节流来输入输出。

InputStream和OutputStream是用来处理8位字节流的抽象基类，程序使用

这两个类的子类来读写8位的字节信息。**这种流通常被用来读写诸如图片、声音之类的二进制数据**。事实上，绝大多数数据是被存储为二进制文件的，世界上的文本文件大约只能占到2%。

这些子类中有两个特殊的类ObjectInputStream和ObjectOutputStream，可以

用来读/写对象。PipedInputStream和PipedOutputStream主要用来完成线程之间的通信，一个线程的PipedInputStream对象能够从另外一个线程的PipedOutputStream对象中读取数据。

类似于处理字符流的类，InputStream和OutputStream的子类也可分为节点

流和处理流。

1. **标准输入输出流**

Java中有以下三个标准输入输出流对象：

* 标准输入System.in
* 标准输出System.out
* 标准错误输出System.err

它们都是System类中定义的类成员变量，System.in是InputStream类型的，代表标准输入流，这个流是已经打开了的，默认状态**对应于键盘输入**。

System.out和System.err是PrintStream类型的，代表标准输出流和标准错误信息输出流，默认状态**对应于屏幕输出**。

**注意：System.in是原始InputStream（字节流），要经过包装才能从键盘读取信息（字符）**。

下面演示如何从键盘读入数据。

**例1** 从键盘读入信息并在显示器上显示。

**import java.io.\*;**

**public class Echo {**

**public static void main(String[] args) throws IOException {**

**BufferedReader in=new BufferedReader(**

**new InputStreamReader(System.in));**

**String s;**

**while((s=in.readLine()).length()!=0){**

**System.out.println(s);}**

**}**

**}**

这里使用的System.in是在程序启动时由Java系统自动创建的流对象，由于它是原始的字节流，因而不能直接从中读取字符，需要对其做进一步处理。本例中以System.in作为参数，进一步创建了一个InputStreamReader流对象，**InputStreamReader相当于字节流和字符流之间的一座桥梁，它读取字节将其转换为字符**。**BufferedReader**用于对InputStreamReader处理后的信息进行缓冲，以提高效率。

1. **处理流**

上述例子中的BufferedReader和InputStreamReader都属于处理流，处理流不

直接与数据源或目标相连，而是与另外的流进行配合，对数据进行某种处理，例如BufferedReader对另一个流产生的数据进行缓冲。

1. **I/O异常**

多数I/O方法在遇到错误时会抛出异常，因此**调用这些方法时必须在函数头声明抛出IOException异常，或者在try块中执行I/O，然后捕获IOException**。

I/O流类有一个共同点，一旦被创建就会自动打开，通过调用其close方法，可以显式的关闭任何一个流。如果这个流对象不再被引用，Java垃圾回收机制也会隐式的关闭它。

**文件读写——I/O流的使用**

在输入输出流的应用当中，最基本和最常用的就是对磁盘文件的读写操作了。下面介绍如何读写文本文件和二进制文件。

**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，需要用到FileWriter

类。它是OutputStreamWriter类的子类，而OutputStreamWriter类又是Writer类的子类。下面演示如何写文本文件。

**例2** 在D盘根目录创建文本文件Hello.txt，并往里写入若干行文本。

**import** java.io.\*;

**public** **class** FileWriterTest {

**public** **static** **void** main(String[] args) **throws** IOException{

String fileName="D:\\Hello.txt";

**//注意'\'是转义符，需要使用'/'或'\\'**

FileWriter writer=**new** FileWriter(fileName);

//创建一个给定文件名的输出流对象

writer.write("Hello!\n");//往流里写字符数组

writer.write("This is my first text file,\n");

writer.write("You can see how this is done.\n");

writer.write("输入一行中文也可以\n");

writer.close();}//**关闭流**

}

程序运行后，打开D盘根目录下的Hello.txt文件，可以看到里面内容恰是写入的内容。每次运行该程序，都将删除已经存在的Hello.txt文件，创建新的同名文件，通过writer方法向其中写入字符。

**FileWriter类**有5个构造方法，本例通过一个字符串指定的文件名来创建。FileWriter流对象创建好之后，就会在磁盘中产生一个新的文本文件Hello.txt。如果同名文件已经存在，则会被这个新文件替代。使用FileWriter的另外一种构造方法，设定第二个**append参数为true**，就可实现在已有文件内容之后**续写**而不是完全替代。

Hello.txt是一个普通的ASCII码文本文件，每个字符占一个字节。在Java程序中的字符串每个字符占两个字节。一个**Writer类**的流可以实现内部格式到外边磁盘文件格式的转换。

**关闭流的close()方法**将清空流里的内容并关闭它，在上面例子中即结束文件操作。应该养成一个好习惯：**当一个涉及文件操作的程序结束时，都应该执行close()方法关闭文件**。如果不调用这个方法，可能系统还没有完成所有数据的写操作，程序就结束了，导致流中的最后一些数据没有被写入文件。这取决于操作系统的繁忙程度，因此会产生时好时坏的后果。流关闭后，就没有这个问题了。

**例3** 使用try、catch方法处理I/O异常。

**import** java.io.\*;

**public** **class** FileWriterTest {

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

String fileName="D:/Hello.txt";

**try**{

FileWriter writer=**new** FileWriter(fileName,true);

//注意此处多了一个参数

writer.write("Hello!\n");

writer.write("This is my first text file,\n");

writer.write("You can see how this is done.\n");

writer.write("输入一行中文也可以\n");

writer.close();

}

**catch**(IOException e){ //如果文件操作出现错误，则屏幕显示提示信息

System.***out***.println("Program writing"+fileName);

}

}

}

运行例3，会在原文件末尾后追加内容，这就是将FileWriter类构造方法的第二个append参数设置为true的效果。若将该文件属性改为只读，再运行本程序，就会出现I/O错误，程序将转入catch块中，输出出错信息。

在上述例子中，由于写入的文本很少，使用FileWriter类就够了。但若写入的内容很多，就要使用java.io包里提供的更高效的缓冲器流类BufferedWriter类。这两个类都用于输出字符流，包含的方法几乎一样，但BufferedWriter多一个newLine()方法用于换行。上述例子中的换行是在每行末尾加'\n'，但由于不同厂家生产的计算机对文字的换行方法可能不同，所以上述程序未必能在各种计算机上产生相同效果，而newLine()方法可以输出在当前计算机上正确的换行符。修改程序为例4。

**例4** 使用BufferedWriter类实现例2的功能。

**import** java.io.\*;

**public** **class** BufferedWriterTest {

**public** **static** **void** main(String[] args) **throws** IOException{

String fileName=" D:/newHello.txt";

//注意'\'是转义符，需要使用'/'或'\\'

BufferedWriter out=new BufferedWriter(

**new** FileWriter(fileName));

//将FileWriter嵌套在BufferedWriter中

out.write("Hello!");//往流里写字符数组

out.newLine();

out.write("This is another text file using BufferedWriter, ");

out.newLine();

out.write("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

out.newLine();

out.close();}//关闭流

}

用任何文本编辑器打开newHello.txt都会出现正确的换行效果，但Hello.txt可能未必正常。

1. **读文本文件**

从文本文件中读取字符需要使用FileWriter类，它继承自Reader抽象基类的子类InputStreamReader。读文本文件也有缓冲器类BufferedReader，具有readLine()方法，可以对换行符进行鉴别，一行行地读取输入流中的内容。

**例5** 从例4创建的newHello.txt中读取文本并显示在屏幕上。

**import** java.io.\*;

**public** **class** BufferedReaderTest {

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

String fileName=" D:/newHello.txt";

String line;

try{ //创建文件输入流并放入缓冲流当中

BufferedReader in= new BufferedReader (

**new** FileReader (fileName));

line=in. readLine(); //读取一行内容

while (line!=null){ //控制循环条件直到文件终点

System.***out***.println(line);

line=in. readLine();

}

in.close();//关闭缓冲流，文件输入流自动也被关闭

}

catch(IOException e){ //如出现I/O异常则进入本块处理

System.***out***.println("Program reading"+fileName);}

}

}

FileReader流对象被创建后，文件newHello.txt就被打开了，若该文件不存在，就会抛出一个IOException。

**BufferedReader对象的readLine()方法**将从一个字符输入流中读取一行文本，并将其放入字符串变量line中。输入流中不再有数据时，就会返回null。

接下来程序读取文件的每一行并将其写在监视器上，当检测到文件结尾，程序就退出了，这是一个很常用的编程模式。运行该程序，屏幕上将逐行显示出newHello.txt文件中的内容。

另有一种判断文本文件结尾的方法，即利用**Reader类的read()方法**返回一个int型整数。若读到文件尾，则返回-1。可修改例5中的读文件判断部分代码：

int c;

while ((c=in.read())!=-1){System.out.print((char)c);}

关闭输入流文件并不像关闭输出流文件那么重要，因为当检测到文件尾时输入流中就没有数据了，因而没必要去刷新它。但为了养成习惯，建议关闭。

1. **写二进制文件**

前面介绍的面向字符的流可以将数据在内部的Unicode格式和外部的本地格式互转，这对文本文件很方便。但对非字符的数据就不适用了。比如Word产生的文档文件，很多字节被解释为字体、字号、图形等非字符信息，这就是二进制文件，就不能用Reader流正确读取了。

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。DataOutputStream是其另一子类，可连接到一个FileOutputStream上，便于写各种类型的数据。

**例6** 将三个int型数字255、0、-1写入数据文件data1.dat中。

**import** java.io.\*;

**public** **class** FileOutputStreamTest {

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

String fileName="D:/data1.dat";

**int** value0=255,value1=0,value2=-1;

**try**{

DataOutputStream out=**new** DataOutputStream(

**new** FileOutputStream(fileName));

out.writeInt(value0);

out.writeInt(value1);

out.writeInt(value2);

out.close();

}

**catch**(IOException e){ //如果文件操作出现错误，则屏幕显示提示信息

System.***out***.println("Program writing"+fileName);

}

}

}

运行程序后，在D盘生成数据文件data1.dat，用写字板打开没有任何显示，用ultraEdit打开查看其二进制信息，内容为000000FF00000000FFFFFFFF，的确每个int数字都是32位的。

FileOutputStream类的构造方法负责打开文件data1.dat用于写数据，若文件不存在则创建一个新文件，若文件已存在则用新创建的文件替代。然后FileOutputStream类的对象与一个DataOutputStream对象连接，DataOutputStream类具有写各种基本数据类型的方法，如本例中的writeInt()方法可将四个字节的整型数据写入流中。最后关闭流，确保写操作完成。

**DataOutputStream类常用方法**如下：

* public DataOutputStream(OutputStream out)

//构造方法，参数为一个OutputStream对象作为其底层的输出对象

* protected int written//私有属性，代表当前已写出的字节数
* public void flush()//冲刷此数据流，使流内数据都被写出
* public final int size()//返回私有变量written的值，即已写出的字节数
* public void write(int i)

//向底层输出流输出int变量的低8位，执行后，记录写入字节数的计数器+1

* public final void writeBoolean(boolean b)

//写出一个布尔数，true为1，false为0，执行后计数器+1

* public final void writeByte(int i)

//将int参数的低8位写入，舍弃高24位，计数器+1

* public void writeBytes(String s)

//字符串中每个字符被丢掉高8位写入流中，计数器+写入的字节数，即字符个数

* public final void writeChar(int c)//将16位字符写入流中，高位在前，计数器+2
* public void writeDouble(double d)//写双精度数，计数器+8
* public void writeFloat(float f)// 写单度数，计数器+4
* public void writeInt(int i)// 写整数，计数器+4
* public void writeLong(long l)// 写长整数，计数器+8
* public final void writeShort(int i)// 写短数，计数器+2

DataOutputStream提供的size()方法，可作为计数器统计写入的字节数。

类似于文本文件中的BufferedWriter，写二进制文件也有缓冲流类BufferedOutputStream，对于大量数据的写入，使用缓冲流类可提高效率。

**例7** 向文件中写入各种数据类型的数，并统计写入的字节数。

**import** java.io.\*;

**public** **class** BufferedOutputStreamTest {

**public** **static** **void** main(String[] args) **throws** IOException{

String fileName="mixedTypes.dat";

DataOutputStream out=**new** DataOutputStream(**new** BufferedOutputStream(**new** FileOutputStream(fileName)));

out.writeInt(0);

System.***out***.println(out.size()+"bytes have been written.");

out.writeDouble(31.2);

System.***out***.println(out.size()+"bytes have been written.");

out.writeBytes("JAVA");

System.***out***.println(out.size()+"bytes have been written.");

out.close();

}

}

该程序可作为字节计数器。

**例8**向文件中写入内容为-1的一个字节，并读出。

**import** java.io.\*;

**public** **class** FileOutputStreamTest {

**public** **static** **void** main(String[] args) **throws** Exception {

DataOutputStream out=**new** DataOutputStream(

**new** FileOutputStream("D:/try.dat"));

out.writeByte(-1);

out.close();

DataInputStream in=**new** DataInputStream(

**new** FileInputStream("D:/try.dat"));

**int** a;

a=in.readByte();

System.***out***.println(Integer.*toHexString*(a));

System.***out***.println(a);

in.skip(-1);

a=in.readUnsignedByte();

System.***out***.println(Integer.*toHexString*(a));

System.***out***.println(a);

in.close();

}

}

执行结果如下：

ffffffff

-1

ff

255

用ultraEdit打开D盘try.dat文件，其内容为FF，若用DataOutputStream类的readByte()读入，其高24位都将补1，所以结果还是-1；但若用readUnsignedByte()读入，其高24位都将补0，结果就变成了255

1. **读二进制文件**

读二进制文件比较常用的类有FileInputStream、DataInputStream、

BufferedInputStream等。类似于DataOutputStream，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9 读取例6创建的数据文件中的3个int型数字，显示相加的结果。

**import** java.io.\*;

**public** **class** DataInputStreamTest {

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

String fileName="D:\\data1.dat";

**int** sum=0;

**try**{

DataInputStream in=**new** DataInputStream(**new** BufferedInputStream(**new** FileInputStream(fileName)));

sum+=in.readInt();

sum+=in.readInt();

sum+=in.readInt();

System.***out***.println("The sum is:"+sum);

in.close();

}

**catch**(IOException e){ //如果文件操作出现错误，则屏幕显示提示信息

System.***out***.println("Program writing "+fileName);

}

}

}

由于文本文件的存储方式其实也是二进制代码，在例10中使用InputStream类的方法读取例2创建的文本文件D：/Hello.txt并显示，注意与例5使用Reader类的方法的差异。

例10 用InputStream类读取文本文件并打印在屏幕上。

**import java.io.\*;**

**public class InputStreamTest {**

**public static void main(String[] args) throws IOException {**

**FileInputStream s=new FileInputStream ("D:/Hello.txt");**

**int c;**

**while((c=s.read())!=-1){**

**System.out.write(c);**

**}**

**s.close();**

**}**

**}**

1. **File类**

File类是IO包中唯一表示磁盘文件信息的对象，它定义了一些与平台无关的方法来操纵文件。通过调用File类提供的各种方法，能创建、删除、重命名文件、判断文件的读写权限及是否存在，设置和查询文件的最新修改时间等。 在Java中，目录也被当作File使用，只是多了一些目录特有的功能。具体见教材。

File类具体应用分为：

* 文件基本操作：具体见教材。
* 处理压缩文件：后续自学
* 对象序列化：后续自学
* 随机文件读写：后续自学

**输入输出流小结**

所有输入流类都派生自InputStream类，所有输出流类都派生自OutputStream

类。

**InputStream类——基本输入流**

是具有最基本输入功能的抽象类，所有其他输入流都是InputStream类的子类，这些子类不仅继承了InputStream类的基本输入功能，又根据自身特性增加了一些输入功能。

InputStream类中包含所有输入流都需要的输入方法，用于从输入流读取数据。

当Java程序需要从键盘、磁盘或其他计算机等外设数据源读出数据时，需要一个输入流对象与外设进行连接，并通过输入流对象调用输入方法实现输入操作。

**注意：**因InputStream类是不能够实例化的抽象类，所以需要创建它的某个子类对象，通过该子类对象与外设数据源建立连接，并调用输入方法实现输入操作。

**从输入流读出数据的方法**

read()方法用于从输入流中读出数据，该方法有以下三种形式：

* public int read()throws IOException

//该方法可以从输入流的当前位置读出一个字节的数据，得到的返回值却是一个int型数据，其中高字节部分全为0；如果当前位置没有数据，返回-1。

* public int read(byte[] b)throws IOException

//该方法从输入流读出多个字节的数据，并存到字节数组b中，返回值是读到的字节数。

* public int read(byte[] b,int off,int len)throws IOException

//该方法从输入流读出len个字节的数据，并把读出的第一个字节的数据存到字节数组b的off元素中，其他字节数据按顺序存到off元素之后。返回值是读到的字节数。

**使用read()方法要注意的问题：**

无论哪种形式的read()方法读出的都是二进制字节数据，这些数据没有具体意义，不能理解成整数或字符等具体数据。

如要使数据恢复为原来表示的数据类型，则需要对字节数据进行强制转换：假设变量b中存放一个字节的数据，通过(char)b可以把b转换为字符数据。

由于read()方法会产生IOException异常，因此使用时要注意异常处理。

下面程序使用read()方法从键盘(System.in) 输入字符（字节）。程序中用throws java.io.IOException声明抛出异常。

**import** java.io.\*;

**public** **class** ReadDemo{

**public** **static** **void** main(String[] args)**throws** IOException{

**byte** buf[]=**new** **byte**[64]; //字节数组

System.***out***.println("Input please:");

System.***in***.read(buf); //从键盘读出多个字符（字节），System.in表示键盘

String s=**new** String(buf); //把buf中的字节数据转换为字符串

System.***out***.println("s="+s);

System.***out***.println("Input please:");

**int d=System.in.read(); //读一个字节存到变量d中**

**System.out.println("d="+(char)d); //转换为字符**

**System.out.println("Input please:");**

**System.in.read(buf,1,3); //从键盘读3个字符**

**s=new String(buf);**

**System.out.println("s="+s);}**

**}**

**定位输入位置的方法**

流式输入按顺序读取数据。每个流都有一个位置指针，决定了read()方法在输入流中读取哪个数据。

在创建输入流时产生位置指针，并把该指针指向输入流的第一个数据，之后每次读操作都是读该处的数据，读完数据后指针自动后移，指向下一个未读数据。

InputStream类中用于控制位置指针的方法有：

* public long skip(long n) //使位置指针从当前位置向后跳过n个字节
* public void mark() //在位置指针处做一个标记
* public void reset() //将位置指针返回到有标记的位置

**OutputStream类——基本输出流**

是具有最基本输出功能的抽象类，所有其他输出流都是OutputStream类的子类，这些子类不仅继承了OutputStream类的基本输出功能，又根据自身特性增加了一些输出功能。

……

后面留给同学们自己总结。