一． Input和Output

1. stream代表的是任何有能力产出数据的数据源，或是任何有能力接收数据的接收源。在Java的IO中，所有的stream（包括Input和Out stream）都包括两种类型：

1.1 以字节为导向的stream

以字节为导向的stream，表示以字节为单位从stream中读取或往stream中写入信息。以字节为导向的stream包括下面几种类型：

1) input　stream：

1) ByteArrayInputStream：把内存中的一个缓冲区作为InputStream使用

2) StringBufferInputStream：把一个String对象作为InputStream

3) FileInputStream：把一个文件作为InputStream，实现对文件的读取操作

4) PipedInputStream：实现了pipe的概念，主要在线程中使用

5) SequenceInputStream：把多个InputStream合并为一个InputStream

2) Out　stream

1) ByteArrayOutputStream：把信息存入内存中的一个缓冲区中

2) FileOutputStream：把信息存入文件中

3) PipedOutputStream：实现了pipe的概念，主要在线程中使用

4) SequenceOutputStream：把多个OutStream合并为一个OutStream

1.2 以Unicode字符为导向的stream

以Unicode字符为导向的stream，表示以Unicode字符为单位从stream中读取或往stream中写入信息。以Unicode字符为导向的stream包括下面几种类型：

1) Input　Stream

1) CharArrayReader：与ByteArrayInputStream对应

2) StringReader：与StringBufferInputStream对应

3) FileReader：与FileInputStream对应

4) PipedReader：与PipedInputStream对应

2) Out　Stream

1) CharArrayWrite：与ByteArrayOutputStream对应

2) StringWrite：无与之对应的以字节为导向的stream

3) FileWrite：与FileOutputStream对应

4) PipedWrite：与PipedOutputStream对应

以字符为导向的stream基本上有与之相对应的以字节为导向的stream。两个对应类实现的功能相同，只是在操作时的导向不同。如CharArrayReader：和ByteArrayInputStream的作用都是把内存中的一个缓冲区作为InputStream使用，所不同的是前者每次从内存中读取一个字节的信息，而后者每次从内存中读取一个字符。

1.3 两种不同导向的stream之间的转换

InputStreamReader和OutputStreamReader：把一个以字节为导向的stream转换成一个以字符为导向的stream。

2. stream添加属性

2.1 “为stream添加属性”的作用

运用上面介绍的Java中操作IO的API，我们就可完成我们想完成的任何操作了。但通过FilterInputStream和FilterOutStream的子类，我们可以为stream添加属性。下面以一个例子来说明这种功能的作用。

如果我们要往一个文件中写入数据，我们可以这样操作：

FileOutStream fs = new FileOutStream(“test.txt”);

然后就可以通过产生的fs对象调用write()函数来往test.txt文件中写入数据了。但是，如果我们想实现“先把要写入文件的数据先缓存到内存中，再把缓存中的数据写入文件中”的功能时，上面的API就没有一个能满足我们的需求了。但是通过FilterInputStream和FilterOutStream的子类，为FileOutStream添加我们所需要的功能。

2.2 FilterInputStream的各种类型

2.2.1 用于封装以字节为导向的InputStream

1) DataInputStream：从stream中读取基本类型（int、char等）数据。

2) BufferedInputStream：使用缓冲区

3) LineNumberInputStream：会记录input stream内的行数，然后可以调用getLineNumber()和setLineNumber(int)

4) PushbackInputStream：很少用到，一般用于编译器开发

2.2.2 用于封装以字符为导向的InputStream

1) 没有与DataInputStream对应的类。除非在要使用readLine()时改用BufferedReader，否则使用DataInputStream

2) BufferedReader：与BufferedInputStream对应

3) LineNumberReader：与LineNumberInputStream对应

4) PushBackReader：与PushbackInputStream对应

2.3 FilterOutStream的各种类型

2.2.3 用于封装以字节为导向的OutputStream

1) DataIOutStream：往stream中输出基本类型（int、char等）数据。

2) BufferedOutStream：使用缓冲区

3) PrintStream：产生格式化输出

2.2.4 用于封装以字符为导向的OutputStream

1) BufferedWrite：与对应

2) PrintWrite：与对应

3. RandomAccessFile

1) 可通过RandomAccessFile对象完成对文件的读写操作

2) 在产生一个对象时，可指明要打开的文件的性质：r，只读；w，只写；rw可读写

3) 可以直接跳到文件中指定的位置

4. I/O应用的一个例子

Java codeimport java.io.\*;

public class TestIO{

public static void main(String[] args)

throws IOException{//1.以行为单位从一个文件读取数据

BufferedReader in =

new BufferedReader(

new FileReader("F:\\nepalon\\TestIO.java"));

String s, s2 = new String();

while((s = in.readLine()) != null)

s2 += s + "\n";

in.close();

//1b. 接收键盘的输入

BufferedReader stdin =

new BufferedReader(

new InputStreamReader(System.in));

System.out.println("Enter a line:");

System.out.println(stdin.readLine());

//2. 从一个String对象中读取数据

StringReader in2 = new StringReader(s2);

int c;

while((c = in2.read()) != -1)

System.out.println((char)c);

in2.close();

//3. 从内存取出格式化输入

try{

DataInputStream in3 =

new DataInputStream(

new ByteArrayInputStream(s2.getBytes()));

while(true)

System.out.println((char)in3.readByte());

}

catch(EOFException e){

System.out.println("End of stream");

}

//4. 输出到文件

try{

BufferedReader in4 =

new BufferedReader(

new StringReader(s2));

PrintWriter out1 =

new PrintWriter(

new BufferedWriter(

new FileWriter("F:\\nepalon\\ TestIO.out")));

int lineCount = 1;

while((s = in4.readLine()) != null)

out1.println(lineCount++ + "：" + s);

out1.close();

in4.close();

}

catch(EOFException ex){

System.out.println("End of stream");

}

//5. 数据的存储和恢复

try{

DataOutputStream out2 =

new DataOutputStream(

new BufferedOutputStream(

new FileOutputStream("F:\\nepalon\\ Data.txt")));

out2.writeDouble(3.1415926);

out2.writeChars("\nThas was pi:writeChars\n");

out2.writeBytes("Thas was pi:writeByte\n");

out2.close();

DataInputStream in5 =

new DataInputStream(

new BufferedInputStream(

new FileInputStream("F:\\nepalon\\ Data.txt")));

BufferedReader in5br =

new BufferedReader(

new InputStreamReader(in5));

System.out.println(in5.readDouble());

System.out.println(in5br.readLine());

System.out.println(in5br.readLine());

}

catch(EOFException e){

System.out.println("End of stream");

}

//6. 通过RandomAccessFile操作文件

RandomAccessFile rf =

new RandomAccessFile("F:\\nepalon\\ rtest.dat", "rw");

for(int i=0; i<10; i++)

rf.writeDouble(i\*1.414);

rf.close();

rf = new RandomAccessFile("F:\\nepalon\\ rtest.dat", "r");

for(int i=0; i<10; i++)

System.out.println("Value " + i + "：" + rf.readDouble());

rf.close();

rf = new RandomAccessFile("F:\\nepalon\\ rtest.dat", "rw");

rf.seek(5\*8);

rf.writeDouble(47.0001);

rf.close();

rf = new RandomAccessFile("F:\\nepalon\\ rtest.dat", "r");

for(int i=0; i<10; i++)

System.out.println("Value " + i + "：" + rf.readDouble());

rf.close();

}

}

关于代码的解释（以区为单位）：

1区中，当读取文件时，先把文件内容读到缓存中，当调用in.readLine()时，再从缓存中以字符的方式读取数据（以下简称“缓存字节读取方式”）。

1b区中，由于想以缓存字节读取方式从标准IO（键盘）中读取数据，所以要先把标准IO（System.in）转换成字符导向的stream，再进行BufferedReader封装。

2区中，要以字符的形式从一个String对象中读取数据，所以要产生一个StringReader类型的stream。

4区中，对String对象s2读取数据时，先把对象中的数据存入缓存中，再从缓冲中进行读取；对TestIO.out文件进行操作时，先把格式化后的信息输出到缓存中，再把缓存中的信息输出到文件中。

5区中，对Data.txt文件进行输出时，是先把基本类型的数据输出屋缓存中，再把缓存中的数据输出到文件中；对文件进行读取操作时，先把文件中的数据读取到缓存中，再从缓存中以基本类型的形式进行读取。注意in5.readDouble()这一行。因为写入第一个writeDouble()，所以为了正确显示。也要以基本类型的形式进行读取。

6区是通过RandomAccessFile类对文件进行操作。