IO流

一、JAVA流式输入/输出原理

在Java程序中，对于数据的输入/输出操作以“流”（Stream）方式进行；J2SDK提供了各种各样的“流”类，用以获取不同种类的数据：程序中通过标准的方法输入或输出数据。

读入写出

流是用来**读写**数据的，java有一个类叫File，它封装的是文件的文件名，只是内存里面的一个对象，真正的文件是在硬盘上的一块空间，在这个文件里面存放着各种各样的数据，我们想读文件里面的数据怎么办呢?

是**通过一个流**的方式来读，咱们要想从程序读数据，对于计算机来说，无论读什么类型的数据 都是以010101101010这样的形式读取的。

怎么把文件里面的数据读出来呢?

你可以把文件想象成一个小桶，文件就是一个桶，文件里面的数据就相当于是这个桶里面的水，那么我们怎么从这个桶里面取水呢，也就是怎么从这个文件读取数据呢。

常见的**取**水的办法是我们用一根管道插到桶上面，然后在管道的另一边打开水龙头，桶里面的水就开始哗啦哗啦地从水龙头里流出来了，桶里面的水是通过这根管道流出来的，因此这根管道就叫**流**，JAVA里面的流式输入/输出跟水流的原理一模一样，当你要从文件读取数据的时候，一根管道插到文件里面去，然后文件里面的数据就顺着管道流出来，这时你在管道的另一头就可以读取到从文件流出来的各种各样的数据了。

当你要往文件**写**入数据时，也是通过一根管道，让要写入的数据通过这根管道哗啦哗啦地流 进文件里面去。

除了从文件去取数据以外，还可以**通过网络**，比如用一根管道把我和你的机子连接起 来，我说一句话，通过这个管道流进你的机子里面，你马上就可以看得到，而你说一句话，通过这根管道流到我的机子里面，我也马上就可以看到。

有的时候，一根管道不够用，比方说这根管道流过来的水有一些杂质，我们就可以在这个根管道的外面再包一层管道，把杂质给过滤掉。从程序的角度来讲，从 计算机读取到的原始数据肯定都是010101这种形式的，一个字节一个字节地往外读，当你这样读的时候你觉得这样的方法不合适，没关系，你再在这根管道的外面再包一层比较强大的管道，这个管道可以把 010101帮你转换成字符串。这样你使用程序读取数据时读到的就不再是010101这种形式的数据了，而是一些可以看得懂的字符串了。

二、输入输出流分类

Java.io 包中定义了多个流类型（类或抽象类）来实现输入/输出功能;

可以从不同的角度对其进行分类:

按数据流的方向不同可以分为输入流和输出流

按照处理数据单位不同可以分为字节流和字符流

按照功能不同可以分为节点流和处理流

我们来理解两个概念:

字节流：最原始的一个流，读出来的数据就是010101这种最底层的数据表示形式，只不过它是按照字节来读的，一个字节（Byte）是8位（bit）读的时候不是一个位一个位的来读，而是一个字节一个字节来读。

字符流：字符流是一个字符一个字符地往外读取数据。一个字符是2个字节 J2SDK所提供的所有流类型位于包 Java.io内，都分别继承自以下四种抽象流类型。

输入流：InPutStream（字节流），Reader（字符流）

输出流：OutPutStream（字节流），Writer（字符流）

这四个类都是抽象类，可以把这四个类想象成四根不同的管道。

一端接着你的程序，另一端接着数据源，你可以通过输出管道从数据源里面往外读数据，也可以通过输入管道往数据源里面输入数据，

总之，通过这四根管道可以让数据流进来和流出去。

io包里面定义了所有的流，所以一说流指的就是io包里面的

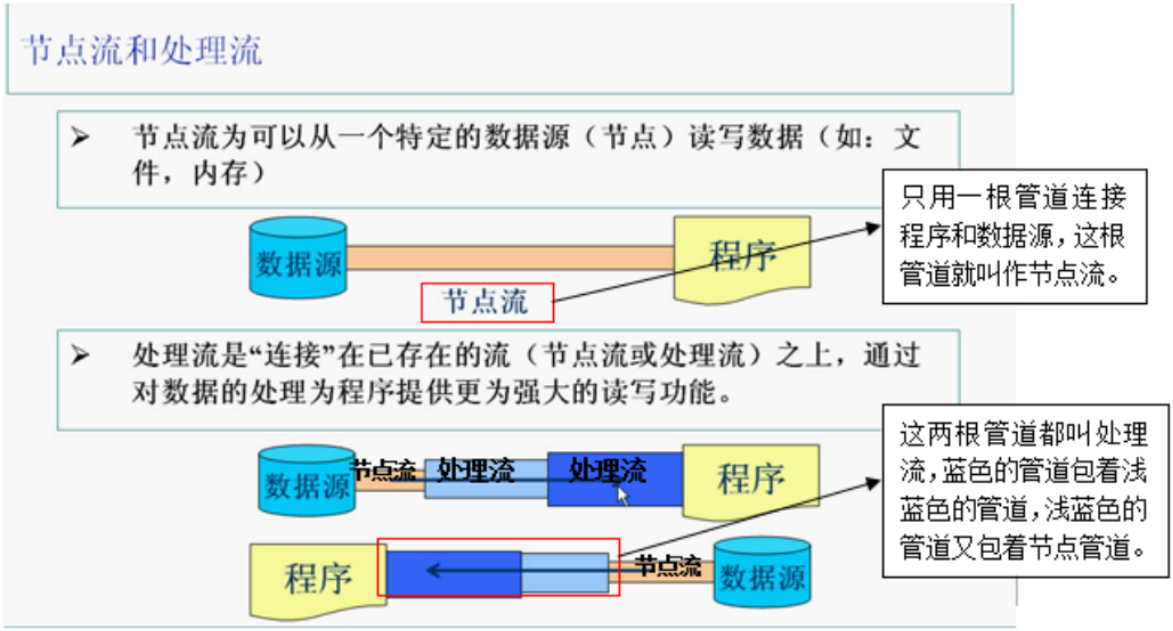
什么叫输入流？什么叫输出流？

用一根管道一端插进文件里，一端插进程序里面，然后开始读数据，那么这是输入还是输出呢？ 如果站在文件的角度上，这叫输出。

如果站在程序的角度上，这叫输入。

记住，以后说输入流和输出流都是站在程序的角度上来说。

三、节点流和处理流



你要是对原始的流不满意，你可以在这根管道外面再套其它的管道，套在其它管道之上的流叫处理流。

为什么需要处理流呢？这就跟水流里面有杂质，你要过滤它，你可以再套一层管道过滤这些杂质一样。

3.1.节点流类型

类型	字符流	字节流
File（文件）	FileReader、FileWriter	FileInputStream、FileOutputStream
Memory Array	CharArrayReader、CharArrayWriter	ByteArrayInputStream、ByteArrayOutputStream
Memory String	StringReader、StringWriter	-
Pipe（管道）	PipedReader、PipedWriter	PipedInputStream、PipedOutputStream

节点流就是一根管道直接插到数据源上面，直接读数据源里面的数据，或者是直接往数据源里面写入数据。典型的节点流是文件流：

文件的字节输入流（FileInputStream），文件的字节输出流 （FileOutputStream），文件的字符输入流（FileReader），文件的字符输出流（FileWriter）。

3.2.处理流类型

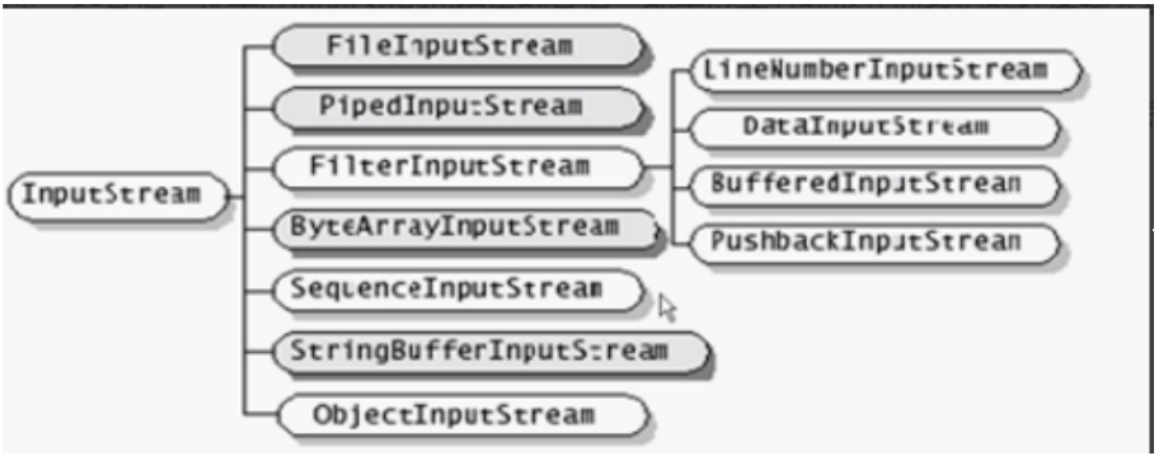
处理类型	字符流	字节流
Buffering	BufferedReader、BufferedWriter	BufferedInputStream、BufferedOutputStream
Filtering	FilterReader、FilterWriter	FilterInputStream，FilterOutputStream
Converting between bytes and chaacter	InputStreamReader、OutputStreamWriter	-
Object Serialization	-	ObjectInputStream、ObjectOutputStream
Data conversion	-	DataInputStream、DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking ahead	PusbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

处理流是包在别的流上面的流，相当于是包到别的管道上面的管道。

四、InputStream(输入流)

我们看到的具体的某一些管道，凡是以InputStream结尾的管道，都是以字节的形式向我们的程序输入数据。

继承自InputStream的流都是用于向程序中输入数据，且数据的单位为字节（8bit）；下图中深色为节点 流，浅色为处理流。



4.1.InputStream的基本方法

```
//读取一个字节并以整数的形式返回（0~255）
//如果返回-1就说明已经到了输入流的末尾
int read() throws IOException
//读取一系列字节并存储到一个数组buffer
//返回实际读取的字节数，如果读取前已到输入流的末尾，则返回-1
int read(byte[] buffer) throws IOException
//读取length个字节
//并存储到一个字节数组buffer，从length位置开始
//返回实际读取的字节数，如果读取前以到输入流的末尾返回-1.
int read(byte[] buffer,int offset,int length) throws IOException
//关闭流释放内存资源
void close() throws IOException
//跳过n个字节不读，返回实际跳过的字节数
long skip(long n) throws IOException
```

read()方法是一个字节一个字节地往外读，每读取一个字节，就处理一个字节。read(byte[] buffer)方法读取数据时，先把读取到的数据填满这个byte[]类型的数组buffer(buffer是内存里面的一块缓冲区)，然后再处理数组里面的数据。这就跟我们取水一样，先用一个桶去接，等桶接满水后再处理桶里面的水。如果是每读取一个字节就处理一个字节，这样子读取也太累了。

4.2 案例

以File(文件)这个类型作为讲解节点流的典型代表

【源码查看，分析结构】

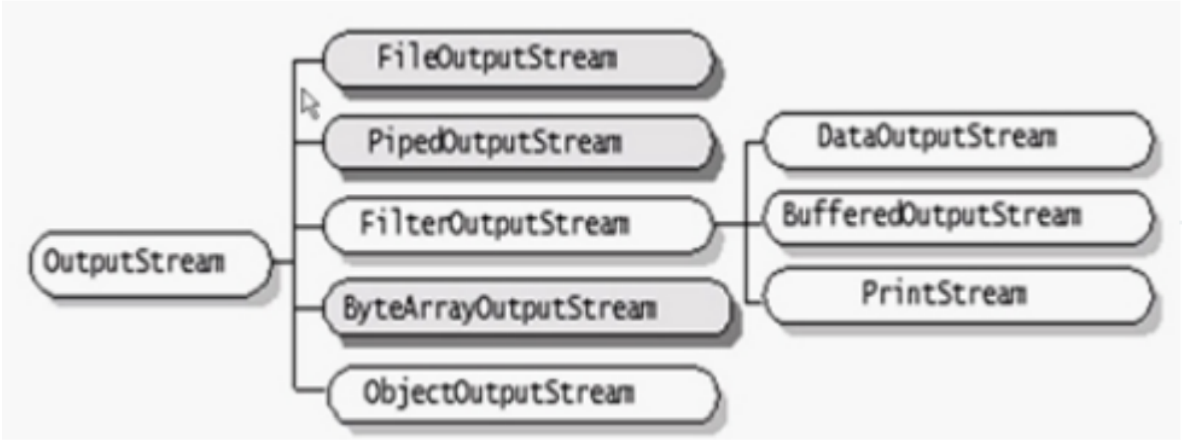
【演示：使用FileInputStream流来读取FileInputStream.java文件的内容】

```
package com.kuang.chapter;
import java.io.*;
public class TestFileInputStream {
    public static void main(String args[]) {
        int b = 0;// 使用变量b来装调用read()方法时返回的整数
        FileInputStream in = null;
        // 使用FileInputStream流来读取有中文的内容时，读出来的是乱码，因为使用InputStream流里面的read()方法读取内容时是一个字节一个字节地读取的，而一个汉字是占用两个字节的，所以读取出来的汉字无法正确显示。
        // FileReader in = null;
        // 使用FileReader流来读取内容时，中英文都可以正确显示，因为Reader流里面的read()方法是一个字符一个字符地读取的，这样每次读取出来的都是一个完整的汉字，这样就可以正确显示了。
```

```
        try {
            in = new FileInputStream("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\Student.java");
            // in = new FileReader("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\Student.java");
        } catch (FileNotFoundException e) {
            System.out.println("系统找不到指定文件！");
            System.exit(-1);// 系统非正常退出
        }
        long num = 0;// 使用变量num来记录读取到的字符数
        // 调用read()方法时会抛异常，所以需要捕获异常
        try {
            while ((b = in.read()) != -1) {
                // 调用int read() throws Exception方法时，返回的是一个int类型的整数
                // 循环结束的条件就是返回一个值-1，表示此时已经读取到文件的末尾了。
                // System.out.print(b+"\t");//如果没有使用“(char)b”进行转换，那么直接
打印出来的b就是数字，而不是英文和中文了
                System.out.print((char) b);
                // “char(b)”把使用数字表示的汉字和英文字母转换成字符输入
                num++;
            }
            in.close();// 关闭输入流
            System.out.println();
            System.out.println("总共读取了" + num + "个字节的文件");
        } catch (IOException e1) {
            System.out.println("文件读取错误！");
        }
    }
}
```

五、OutputStream(输出流)

继承自OutputStream的流是用于程序中输出数据，且数据的单位为字节（8bit）：下图中深色的为节点流，浅色为处理流。



5.1.OutputStream的基本方法

```
//向输出流中写入一个字节数据，该字节数据为参数b的低8位
void write(int b) throws IOException
//将一个字节类型的数组中的数据写入输出流
void write(byte[] b) throws IOException
//将一个字节类型的数组中的从指定位置（off）开始的len个字节写入到输出流
void write(byte[] b,int off,int len) throws IOException
//关闭流释放内存资源
void close() throws IOException
//将输出流中缓冲的数据全部写出到目的地
void flush() throws IOException
```

5.2 案例

【使用FileOutputStream流往一个文件里面写入数据】

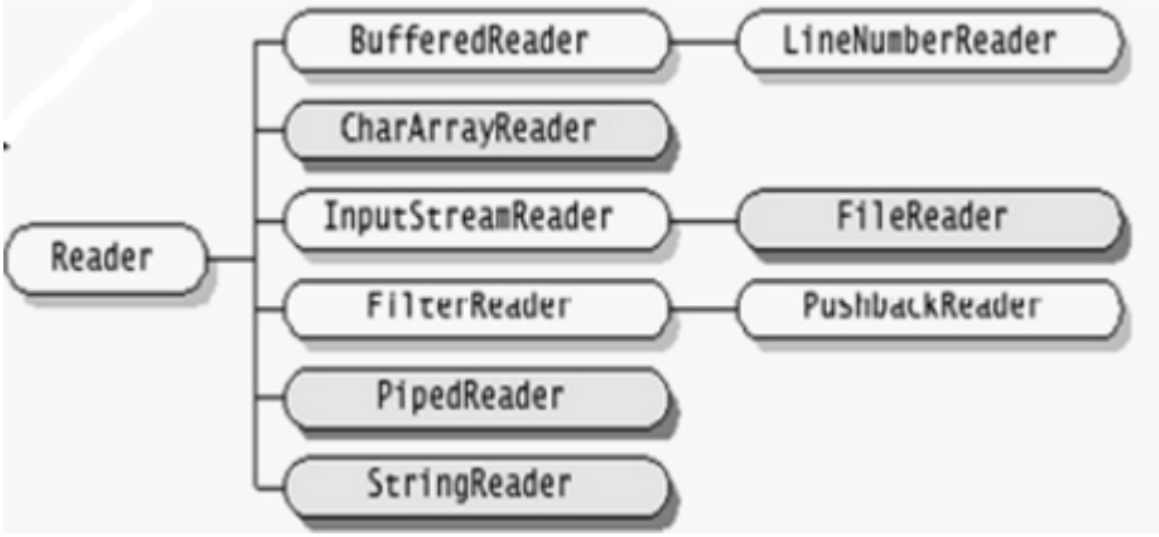
```
package com.kuang.chapter;
import java.io.*;
public class TestFileOutputStream {
    public static void main(String args[]) {
        int b = 0;
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\Student.java");
            out = new FileOutputStream("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\StudentNew.java");
            // 指明要写入数据的文件，如果指定的路径中不存在StudentNew.java这样的文件，则系
统会自动创建一个
            while ((b = in.read()) != -1) {
                out.write(b);
                // 调用write(int c)方法把读取到的字符全部写入到指定文件中去
            }
            in.close();
            out.close();
        } catch (FileNotFoundException e) {
            System.out.println("文件读取失败");
            System.exit(-1);// 非正常退出
        } catch (IOException e1) {
            System.out.println("文件复制失败！");
            System.exit(-1);
        }
        System.out.println("Student.StudentNew.java里面");
    }
}
```

FileInputStream和FileOutputStream这两个流都是字节流，都是以一个字节为单位进行输入和输出的。所以对于占用2个字节存储空间的字符来说读取出来时就会显示成乱码。

六、Reader流

Reader： 和InputStream一模一样，唯一的区别就在于读的数据单位不同

继承自Reader的流都是用于向程序中输入数据，且数据的单位为字符（16bit）



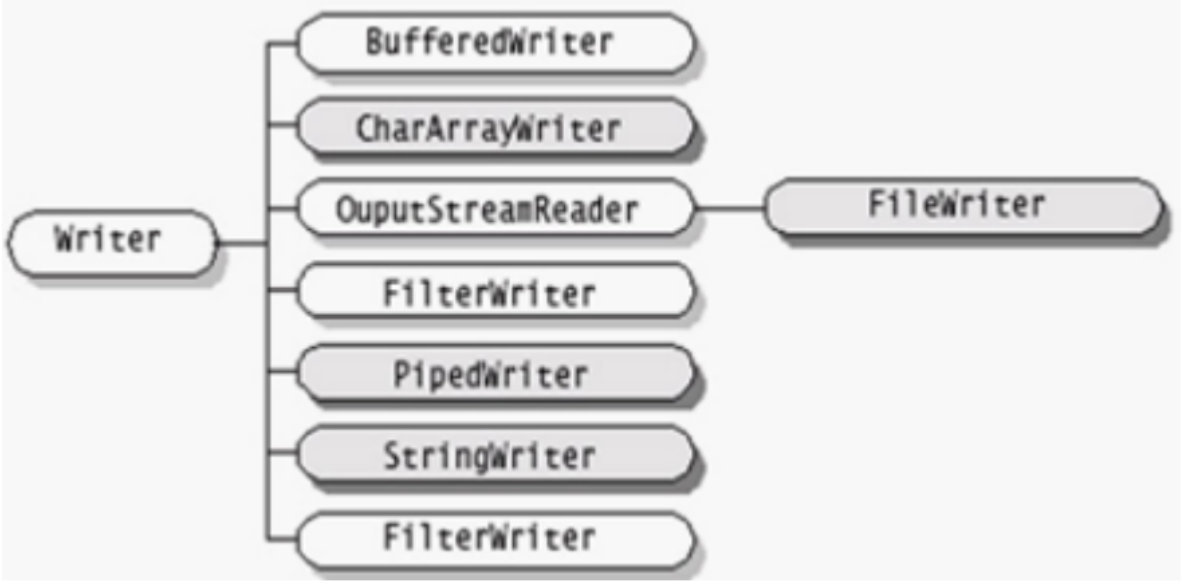
16位：一个字符也就是两个字节，使用Reader流读取数据时都是两个字节两个字节往外读的，为什么还要有这两种两个字节的读取方式呢? 因为有些字符是占2个字节的，如我们的中文字符在Java里面就是占两个字节的。如果采用一个字节一个字节往外读的方式，那么读出来的就是半个汉字，这样子Java就没有办法正确的显示中文字符的，所以有必要存在这种流，一个字符一个字符地往外读。

6.1.Reader的基本方法

```
//读取一个字节并以整数的形式返回（0~255）
//如果返回-1就说明已经到了输入流的末尾
int read() throws IOException
//读取一系列字节并存储到一个数组buffer
//返回实际读取的字节数，如果读取前已到输入流的末尾，则返回-1
int read(byte[] buffer) throws IOException
//读取length个字节
//并存储到一个字节数组buffer，从length位置开始
//返回实际读取的字节数，如果读取前以到输入流的末尾返回-1.
int read(byte[] buffer,int offset,int length) throws IOException
//关闭流释放内存资源
void close() throws IOException
//跳过n个字节不读，返回实际跳过的字节数
long skip(long n) throws IOException
```

七、Writer流

继承自Writer的流都是用于程序中输出数据，且数据的单位为字符（16bit）；



7.1.Writer的基本方法

```
//向输出流中写入一个字节数据，该字节数据为参数b的低16位
void write(int b) throws IOException
//将一个字节类型的数组中的数据写入输出流
void write(byte[] b) throws IOException
//将一个字节类型的数组中的从指定位置（off）开始的len个字节写入到输出流
void write(byte[] b,int off,int len) throws IOException
//关闭流释放内存资源
void close() throws IOException
//将输出流中缓冲的数据全部写出到目的地
void flush() throws IOException
```

7.2 演示

【演示：使用FileWriter（字符流）向指定文件中写入数据】

```
package com.kuang.chapter;
/*使用FileWriter（字符流）向指定文件中写入数据写入数据时以1个字符为单位进行写入*/
import java.io.*;
public class TestFileWriter{
    public static void main(String args[]){
    /*使用FileWriter输出流从程序把数据写入到Uicode.dat文件中使用FileWriter流向文件写入数据时是一个字符一个字符写入的*/
    FileWriter fw = null;
    try{
        fw = new FileWriter("E:\\教学\\班级\\Test\\Lesson2\\src\\com\\kuang\\chapter\\StudentNew.java");
        //字符的本质是一个无符号的16位整数
        //字符在计算机内部占用2个字节
        //这里使用for循环把0～60000里面的所有整数都输出
        //这里相当于是把全世界各个国家的文字都0～60000内的整数的形式来表示
        for(int c=0;c<=60000;c++){
            fw.write(c);
            //使用write(int c)把0～60000内的整数写入到指定文件内
            //调用write()方法时，我认为在执行的过程中应该使用了“(char)c”进行强制转换，即把整数转换成字符来显示
            //因为打开写入数据的文件可以看到，里面显示的数据并不是0～60000内的整数，而是不同国家的文字的表示方式
```

```
        }
    /*使用FileReader(字符流)读取指定文件里面的内容读取内容时是以一个字符为单位进行读取的*/
    int b = 0;
    long num = 0;
    FileReader fr = null;
    fr = new FileReader("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\StudentNew.java");
    while((b = fr.read())!= -1){
        System.out.print((char)b + "\t");
        num++;
    }
    System.out.println();
    System.out.println("总共读取了"+num+"个字符");
    }catch(Exception e){
        e.printStackTrace();
    }
}
}
```

FileReader和FileWriter这两个流都是字符流，都是以一个字符为单位进行输入和输出的。所以读取和写入占用2个字节的字符时都可以正常地显示出来，以上是以File(文件)这个类型为例对节点流进行了讲解，所谓的节点流指定就是直接把输入流或输出插入到数据源上，直接往数据源里面写入数据或读取数据。

八、处理流讲解

8.1.第一种处理流——缓冲流(Buffering)

缓冲流要"套接"在相应的节点流之上，对读写的数据提供了缓冲的功能，提高了读写的效率，同时增加了一些新的方法。J2SDK提供了四种缓冲流，常用构造方法如下：

```
BufferedReader(Reader in)
BufferedReader(Reader in,int sz) //sz 为自定义缓冲区的大小
BufferedWriter(Writer out)
BufferedWriter(Writer out,int sz)
BufferedInputStream(InputStream in)
BufferedInputStream(InputStream in,int size)
BufferedOutputStream(InputStream in)
BufferedOutputStream(InputStream in,int size)
```

缓冲输入流支持其父类的mark和reset方法。

BufferedReader提供了readLine方法用于读取一行字符串

BufferedWriter提供了newLine用于写入一个行分隔符

对于输出的缓冲流，写出的数据会现在内存中缓存，使用flush方法将会使内存中的数据立刻写出

带有缓冲区的，缓冲区(Buffer)就是内存里面的一小块区域，读写数据时都是先把数据放到这块缓冲区域里面，减少io对硬盘的访问次数，保护我们的硬盘。可以把缓冲区想象成一个小桶，把要读写的数据想象成水，每次读取数据或者是写入数据之前，都是先把数据装到这个桶里面，装满了以后再做处理。这就是所谓的缓冲。先把数据放置到缓冲区上，等到缓冲区满了以后，再一次把缓冲区里面的数据写入到硬盘上或者读取出来，这样可以有效地减少对硬盘的访问次数，有利于保护我们的硬盘。

【缓冲流测试代码：BufferedInputStream】

```
```

```
package com.kuang.chapter;
import java.io.*;
public class TestBufferStream {
    public static void main(String args[]) {
        FileInputStream fis = null;
        try {
        fis = new FileInputStream("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\Student.java");
        // 在FileInputStream节点流的外面套接一层处理流BufferedInputStream
        BufferedInputStream bis = new BufferedInputStream(fis);
        int c = 0;
        System.out.println((char) bis.read());
        System.out.println((char) bis.read());
        bis.mark(100);// 在第100个字符处做一个标记
        for (int i = 0; i <= 10 && (c = bis.read()) != -1; i++) {
            System.out.print((char) c);
        }
        System.out.println();
        bis.reset();// 重新回到原来标记的地方
        for (int i = 0; i <= 10 && (c = bis.read()) != -1; i++) {
            System.out.print((char) c);
        }
        bis.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
}
```

【演示：BufferedReader】

```
package com.kuang.chapter;
import java.io.*;
public class TestBufferStream{
    public static void main(String args[]){
        try{
            BufferedWriter bw = new BufferedWriter(new FileWriter("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\Student.txt"));
            //在节点流FileWriter的外面再套一层处理流BufferedWriter
            String s = null;
            for(int i=0;i<100;i++){
                s = String.valueOf(Math.random());//“Math.random()”将会生成一系列介
于0～1之间的随机数。
                // static String valueOf(double d)这个valueOf()方法的作用就是把一个
double类型的数转换成字符串
                //valueOf()是一个静态方法，所以可以使用“类型.静态方法名”的形式来调用
                bw.write(s);//把随机数字符串写入到指定文件中
                bw.newLine();//调用newLine()方法使得每写入一个随机数就换行显示
            }
            bw.flush();//调用flush()方法清空缓冲区
            BufferedReader br = new BufferedReader(new FileReader("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\Student.txt"));
            //在节点流FileReader的外面再套一层处理流BufferedReader
            while((s = br.readLine())!=null){
```

```
            //使用BufferedReader处理流里面提供String readLine()方法读取文件中的数据时是一行一行读取的
            //循环结束的条件就是使用readLine()方法读取数据返回的字符串为空值后则表示已经读取到文件的末尾了。
                System.out.println(s);
            }
            bw.close();
            br.close();
            }catch(Exception e){
                e.printStackTrace();
            }
        }
}
```

程序的输入指的是把从文件读取到的内容存储到为程序分配的内存区域里面去。流，什么是流，流无非就是两根管道，一根向里，一根向外，向里向外都是对于我们自己写的程序来说，流分为各种各样的类型，不同的分类方式又可以分为不同的类型，根据方向来分，分为输入流和输出流，根据读取数据的单位的不同，又可以分为字符流和字节流，除此之外，还可以分为节点流和处理流，节点流就是直接和数据源连接的流，处理流就是包在其它流上面的流，处理流不是直接和数据源连接，而是从数据源读取到数据以后再通过处理流处理一遍。缓冲流也包含了四个类：BufferedInputStream、BufferedOutputStream、BufferedReader和BufferedWriter。流都是成对的，没有流是是不成对的，肯定是一个in，一个out。

8.2.第二种处理流——转换流

InputStreamReader 和 OutputStreamWriter 用于字节数据到字符数据之间的转换
InputStreamReader 需要和 InputStream “套接” 。

OutputStreamWriter 需要和 OutputStream “套接” 。 转换流在构造时可以指定其编码集合

```
InputStream isr = new InputStreamReader（System.in，"ISO8859-1"）
```

转换流非常的有用，它可以把一个**字节流**转换成一个**字符流**，转换流有两种，

一种叫 InputStreamReader，另一种叫OutputStreamWriter。InputStream是字节流，Reader是字符流， InputStreamReader就是把InputStream转换成Reader。

OutputStream是字节流，Writer是字符流， OutputStreamWriter就是把OutputStream转换成Writer。把OutputStream转换成Writer之后就可以一个字符一个字符地通过管道写入数据了，而且还可以写入字符串。我们如果用一个 FileOutputStream 流往文件里面写东西，得要一个字节一个字节地写进去，但是如果我们在FileOutputStream流上面套上一个字符转换流，那我们就可以一个字符串一个字符串地写进去。

【转换流测试代码】

```
import java.io.*;
public class TestTransform1 {
    public static void main(String args[]) {
        try {
            OutputStreamWriter osw = new OutputStreamWriter(
                new FileOutputStream("D:/java/char.txt"));
            osw.write("MircosoftsunIBMOracleApplet");// 把字符串写入到指定的文件中去
            System.out.println(osw.getEncoding());// 使用getEncoding()方法取得当前系统的默认字符编码
            osw.close();
```

```
            osw = new OutputStreamWriter(new
FileOutputStream("D:\\java\\char.txt", true), "ISO8859_1");
            // 如果在调用FileOutputStream的构造方法时没有加入true，那么新加入的字符串就会
替换掉原来写入的字符串，在调用构造方法时指定了字符的编码
            osw.write("MircosoftsunIBMOracleApplet");// 再次向指定的文件写入字符串，新
写入的字符串加入到原来字符串的后面
            System.out.println(osw.getEncoding());
        osw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;
public class TestTransform2{
    public static void main(String args[]){
        try{
            InputStreamReader isr = new InputStreamReader(System.in);
            //System.in这里的in是一个标准的输入流，用来接收从键盘输入的数据
            BufferedReader br = new BufferedReader(isr);
            String s = null;
            s = br.readLine();//使用readLine()方法把读取到的一行字符串保存到字符串变量s
中去
            while(s != null){
                System.out.println(s.toUpperCase());//把保存在内存s中的字符串打印出来
                s = br.readLine();//在循环体内继续接收从键盘的输入
                if(s.equalsIgnoreCase("exit")){
                    //只要输入exit循环就结束，就会退出
                    break;
                }
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

8.3.第三种处理流——数据流

DataInputStream 和 DataOutputStream 分别继承自InputStream 和 OutputStream， 它属于处理流，需要分别“套接”在InputStream 和 OutputStream类型的节点流上。

DataInputStream 和 DataOutputStream 提供了可以存取与机器无关的Java原始类型数据（int，double等）的方法。

DataInputStream 和 DataOutputStream 的构造方法

【数据流测试代码】

```
package com.kuang.chapter;
import java.io.*;
public class TestDataStream{
    public static void main(String args[]){
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        //在调用构造方法时，首先会在内存里面创建一个ByteArray字节数组
        DataOutputStream dos = new DataOutputStream(baos);
```

```
        //在输出流的外面套上一层数据流，用来处理int，double类型的数
        try{
            dos.writeDouble(Math.random());//把产生的随机数直接写入到字节数组ByteArray
中
            dos.writeBoolean(true);//布尔类型的数据在内存中就只占一个字节
            ByteArrayInputStream bais = new
            ByteArrayInputStream(baos.toByteArray());
            System.out.println(bais.available());
            DataInputStream dis = new DataInputStream(bais);
            System.out.println(dis.readDouble());//先写进去的就先读出来，调用
readDouble()方法读取出写入的随机数
            System.out.println(dis.readBoolean());//后写进去的就后读出来，这里面的读取
顺序不能更改位置，否则会打印出不正确的结果
            dos.close();
            bais.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

通过bais这个流往外读取数据的时候，是一个字节一个字节地往外读取的，因此读出来的数据无法判断是字符串还是bool类型的值，因此要在它的外面再套一个流，通过dataInputStream把读出来的数据转换就可以判断了。注意了：读取数据的时候是先写进去的就先读出来，因此读ByteArray字节数组数据的顺序应该是先把占8个字节的double类型的数读出来，然后再读那个只占一个字节的boolean类型的数，因为double类型的数是先写进数组里面的，读的时候也要先读它。这就是所谓的先写的要先读。如果先读Boolean类型的那个数，那么读出来的情况可能就是把double类型数的8个字节里面的一个字节读了出来。

8.4.打印流——Print

PrintWriter 和 PrintStream 都属于输出流，分别针对与字符和字节

PrintWriter 和 PrintStream 提供了重载的print Println方法用于多种数据类型的输出

PrintWriter和PrintStream 的输出操作不会抛出异常，用户通过检测错误状态获取错误信息

PrintWriter 和 PrintStream 有自动flush功能

```
PrintWriter（Writer out）
PrintWriter（Writer out，boolean autoFlush）
PrintWriter（OutputStream out）
PrintWriter（OutputStream out，boolean autoFlush）
PrintStream（OutputStream out）
PrintStream（OutputStream out，boolean autoFlush）
```

【测试代码】

```
/*这个小程序是重新设置打印输出的窗口，
* 把默认在命令行窗口输出打印内容设置成其他指定的打印显示窗口
*/
import java.io.*;
public class TestPrintStream{
    public static void main(String args[]){
        PrintStream ps = null;
        try{
```

```
            FileOutputStream fos = new FileOutputStream("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\log.txt");
            ps = new PrintStream(fos);//在输出流的外面套接一层打印流，用来控制打印输出
            if(ps != null){
                System.setOut(ps);//这里调用setOut()方法改变了输出窗口，以前写
System.out.print();//默认的输出窗口就是命令行窗口.
            //但现在使用System.setOut(ps)将打印输出窗口改成了由ps指定的文件里面，通过这样
设置以后，打印输出时都会在指定的文件内打印输出
            //在这里将打印输出窗口设置到了log.txt这个文件里面，所以打印出来的内容会在
log.txt这个文件里面看到
        }
        for(char c=0;c<=1000;c++){
            System.out.print(c+"\t");//把世界各国的文字打印到log.txt这个文件中去
        }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

8.5. 对象流——Object

直接将Object 写入或读出

transient关键字

transient：透明的，用它来修饰的成员变量在序列化的时候不予考虑，也就是当成不存在。

serializable接口

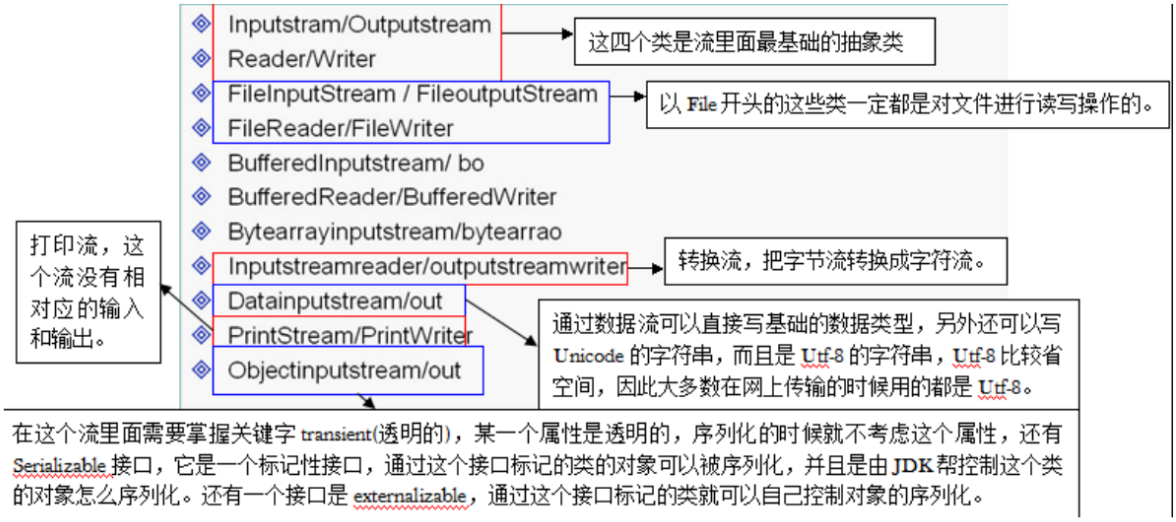
externaliazble接口

```
package com.kuang.chapter;
import java.io.*;
public class TestObjectIo {
    public static void main(String args[]) {
        T t = new T();
        t.k = 8;// 把k的值修改为8
        try {
            FileOutputStream fos = new FileOutputStream("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\TestObjectIo.txt");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            // ObjectOutputStream流专门用来处理Object的，在fos流的外面套接
ObjectOutputStream流就可以直接把一个Object写进去
            oos.writeObject(t);// 直接把一个t对象写入到指定的文件里面
            oos.flush();
            oos.close();
            FileInputStream fis = new FileInputStream("E:\\教学\\班级
\\Test\\Lesson2\\src\\com\\kuang\\chapter\\TestObjectIo.txt");
            ObjectInputStream ois = new ObjectInputStream(fis);
            // ObjectInputStream专门用来读一个Object的
            T tRead = (T) ois.readObject();
            // 直接把文件里面的内容全部读取出来然后分解成一个Object对象，并使用强制转换成指定
类型T
            System.out.print(tRead.i + "\t" + tRead.j + "\t" + tRead.d +"\t"+
tRead.k);
            ois.close();
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
/*
* 凡是要将一个类的对象序列化成一个字节流就必须实现Serializable接口
* Serializable接口中没有定义方法，Serializable接口是一个标记性接口，用来给类作标记，
只是起到一个标记作用。
* 这个标记是给编译器看的，编译器看到这个标记之后就可以知道这个类可以被序列化 如果想把某个
类的对象序列化，就必须得实现Serializable接口
*/
class T implements Serializable {
// Serializable的意思是可以被序列化的
    int i = 10;
    int j = 9;
    double d = 2.3;
    int k = 15;
// transient int k = 15;
// 在声明变量时如果加上transient关键字，那么这个变量就会被当作是透明的，即不存在。
}
```

直接实现Serializable接口的类是JDK自动把这个类的对象序列化，而如果实现public interface Externalizable extends Serializable的类则可以自己控制对象的序列化，建议能让JDK自己控制序列化的就不要让自己去控制

九、IO流总结



要写文件就创建一个output/writer管道，相当于把程序的东西送出去（out）外面给文件

要读文件就得创建input/reader管道，相当于文件把他的东西送进（in）程序