1. **IO流概述**

概述：

**IO**流简单来说就是**Input**和**Output**流，**IO**流主要是用来处理设备之间的数据传输，[**Java**](http://lib.csdn.net/base/javase)对于数据的操作都是通过流实现，而**java**用于操作流的对象都在**IO**包中。

分类：

        按操作数据分为：字节流和字符流。 如：**Reader**和**InpurStream**

        按流向分：输入流和输出流。如：**InputStream**和**OutputStream**

**IO**流常用的基类：

         \* **InputStream**    ，    **OutputStream**

字符流的抽象基类：

         \* **Reader       ，         Writer**

由上面四个类派生的子类名称都是以其父类名作为子类的后缀：

            如：**FileReader**和**FileInputStream**

## 二、字符流

### 1. 字符流简介：

**\* 字符流中的对象融合了编码表，也就是系统默认的编码表。我们的系统一般都是GBK编码。**

**\* 字符流只用来处理文本数据，字节流用来处理媒体数据。**

**\* 数据最常见的表现方式是文件，字符流用于操作文件的子类一般是FileReader和FileWriter。**

### 2.字符流读写：

注意事项：

**\* 写入文件后必须要用flush()刷新。**

**\* 用完流后记得要关闭流。**

**\* 使用流对象要抛出IO异常。**

**\* 定义文件路径时，可以用“/”或者“\\”。**

**\* 在写入一个文件时，如果目录下有同名文件将被覆盖。**

**\* 在写入一个文件时，如果目录下找不到次文件，将会创建一个新文件并写入数据。**

**\* 在读取文件时，必须保证该文件已存在，否则出异常。**

**\* 在流使用完后，记得关闭流close()。**

**示例1：在硬盘上创建一个文件,并写入一些文字数据**

FileWriter fw = **new** FileWriter("C:\\1.txt");

fw.write("hello world!");

fw.flush();

fw.close();

**示例2：FileReader的reade()方法.** **用单个字符和字符数组进行分别读取**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*字符数组进行读取\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

FileReader fr = **new** FileReader("C:\\Demo.txt");

**char** [] buf = **new** **char**[6];

//将Denmo中的文件读取到buf数组中。

**int** num = 0;

**while**((num = fr.read(buf))!=-1) {

//String(char[] value , int offest,int count) 分配一个新的String,包含从offest开始的count个字符

System.*out*.print(**new** String(buf,0,num));

}

fr.close();

**\* 空格也占长度**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*单个字母读取\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//创建一个文件读取流对象，和指定名称的文件关联。

//要保证文件已经存在，否则会发生异常：FileNotFoundException

FileReader fr = **new** FileReader("C:\\Demo.txt");

//read()方法，一次读取一个字符，并且自动往下读。如果到达末尾则返回-1

**int** ch = 0;

**while** ((ch=fr.read())!=-1) {

System.*out*.print((**char**)ch);

}

fr.close();

**示例3：对已有文件的数据进行续写**

//传递一个参数,代表不覆盖已有的数据。并在已有数据的末尾进行数据续写

FileWriterfw=**new**FileWriter("C:\\java\_Demo\\demo.txt",**true**);

fw.write(" is charactor table?");

fw.close();

### 3. 字符流的缓冲区：BufferedReader和BufferedWreiter

**\* 缓冲区的出现时为了提高流的操作效率而出现的.**

**\* 需要被提高效率的流作为参数传递给缓冲区的构造函数**

**\* 在缓冲区中封装了一个数组，存入数据后一次取出**

**BufferedReader示例**：

读取流缓冲区提供了一个一次读一行的方法readline，方便对文本数据的获取。

readline()只返回回车符前面的字符，不返回回车符。如果是复制的话，必须加入newLine()，写入回车符

newLine()是java提供的多平台换行符写入方法。

//创建一个字符读取流流对象，和文件关联

FileReader rw = **new** FileReader("buf.txt");

//将需要被提高效率的流作为参数传递给缓冲区的构造函数即可

BufferedReader brw = **new** BufferedReader(rw);

**for**(;;) {

String s = brw.readLine();

**if**(s==**null**) **break**;

}

brw.close(); //关闭输入流对象

**BufferedWriter示例：**

FileWriter fw = **new** FileWriter("buf.txt");

//只要将需要被提高效率的流作为参数传递给缓冲区的构造函数即可

BufferedWriter bfw = **new** BufferedWriter(fw);

//bfw.write("abc\r\nde");

//用到缓冲区就必须要刷新

**for**(**int** x = 1; x < 5; x++) {

bfw.write("abc");

bfw.newLine(); //java提供了一个跨平台的换行符newLine();

bfw.flush();

}

bfw.flush(); //刷新缓冲区

bfw.close(); //关闭缓冲区，但是必须要先刷新

fw.close(); //关闭输入流对象

## 三、字节流

### 1.字节流简介：

**1、字节流和字符流的基本操作是相同的，但是要想操作媒体流就需要用到字节流。**

**2、字节流因为操作的是字节，所以可以用来操作媒体文件。（媒体文件也是以字节存储的）**

**3、读写字节流：InputStream   输入流（读）和OutputStream  输出流（写）**

**4、字节流操作可以不用刷新流操作。**

**5、InputStream特有方法： int available();//返回文件中的字节个数**

注：可以利用此方法来指定读取方式中传入数组的长度，从而省去循环判断。但是如果文件较大，而虚拟机启动分配的默认内存一般为64M。当文件过大时，此数组长度所占内存空间就会溢出。所以，此方法慎用，当文件不大时，可以使用。

### 2.字节流的读写：

**示例：复制一张图片C:\java\_Demo\ 1.jpg到C:\java\_Demo \2.jpg**

FileInputStream fis = **null**;

FileOutputStream fos = **null**;

**try** {

//写入流关联文件

fis = **new** FileInputStream("C:\\javaDemo\\1.jpg");

//读取流关联文件

fos = **new** FileOutputStream("C:\\javaDemo\\2.jpg");

**byte**[] copy = **new** **byte**[1024];

**int** len = 0;

**while**((len=fis.read(copy))!=-1) {

fos.write(copy,0,len);

}

}

**catch** (IOException e) {

e.printStackTrace();

**throw** **new** RuntimeException("复制文件异常");

}

### 3. 字节流缓冲区：

**\* 字节流缓冲区跟字符流缓冲区一样，也是为了提高效率。**

**注意事项：**

1. read()：会将字节byte()提升为int型值

2. write()：会将int类型转换为byte()类型，保留最后的8位。

**示例：复制MP3文件 1.MP3 --> 2.MP3**

**class** MyBufferedInputStream {

**private** InputStream in; //定义一个流对象

**private** **byte** [] buf = **new** **byte**[1024\*4];

**private** **int** count = 0,pos = 0;

**public** MyBufferedInputStream(InputStream in){

**this**.in = in;

}

**public** **int** MyRead() **throws** IOException{

**if**(count==0) { //当数组里的数据为空时候，读入数据

count = in.read(buf);

pos = 0;

**byte** b = buf[pos];

count--;

pos++;

**return** b&0xff;

} **else** **if**(count > 0) {

**byte** b = buf[pos];

pos++;

count--;

**return** b&0xff;

}

**return** -1;

}

**public** **static** **void** copy() {

MyBufferedInputStream bis = **null**;

BufferedOutputStream bos = **null**;

**try** {

//匿名类，传入一个InputStream流对象

bis = **new** out(**new** FileInputStream("C:\\CloudMusic\\你在终点等我.mp3"));

//输出路径和文件名

bos = **new** BufferedOutputStream(**new** FileOutputStream("C:\\Users\\ls\\Desktop\\Cover.mp3"));

**int** buf = 0;

**while**((buf=bis.MyRead())!=-1) {

bos.write(buf);

}

}

**catch** (IOException e) {

e.printStackTrace();

**throw** **new** RuntimeException("复制失败");

}

}

## 四、流操作规律

### 键盘读取，控制台打印。

/\*================从键盘录入流，打印到控制台上================\*/

**public** **static** **void** InOutDemo(){

//键盘的最常见的写法

BufferedReader bufr = **null**;

BufferedWriter bufw = **null**;

**try** {

bufr = **new** BufferedReader(**new** InputStreamReader(System.*in*));

//InputSteamReader:读取字节并将其解码为字符

bufw = **new** BufferedWriter(**new** OutputStreamWriter(System.*out*));

//OutputStreamWriter:要写入流中的字符编码成字节

String line = **null**;

**while**((line = bufr.readLine())!=**null**){

**if**("over".equals(line)) **break**;

bufw.write(line.toUpperCase()); //打印

bufw.newLine(); //写入换行符

bufw.flush(); //刷新

}

**if**(bufw!=**null**) {

bufr.close();

bufw.close();

}

}

**catch** (IOException e) {

e.printStackTrace();

}

}

### 2. 转换流

为了让字节流可以使用字符流中的方法，我们需要转换流。

InputStreamReader:字节流转向字符流；

1. 获取键盘录入对象。

InputStream in=System.*in*;

b、将字节流对象转成字符流对象，使用转换流。

InputStreamReader isr = **new** InputStreamReader(in);

c、为了提高效率，将字符串进行缓冲区技术高效操作。使用BufferedReader

  BufferedReader br = **new** BufferedReader(isr);

//键盘录入最常见写法

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

**什么时候使用转换流呢？**

1，源或者目的对应的设备是字节流，但是操作的却是文本数据，可以使用转换作为桥梁。提高对文本操作的便捷。

2，一旦操作文本涉及到具体的指定编码表时，必须使用转换流 。

### 3. 打印流

**分为两类**

一个类是字节打印流 **PrintStream** 针对字节.   
一个类是字符打印流 **PrintWriter** 针对字符.

两个都有自动刷新功能 意思就说不用flush就可以写入数据   
**\*这里必须强调一点：PrintWriter的自动flush功能，只有在new PrintWriter对象的时候调用PrintWriter（Writer out, boolean autoFlush），并将 autoFlush设置为true才会自动flush，否则不会自动flush，必须手动printWriter.flush()才能成功打印数据。**

字节打印流：**PrintStream**   
两个作用 1.打印方便 2.可以打印异常日志信息   
PrintStream对普通字节输出流的增强

**System.out对应的类型就是PrintStream.**

对于基本数据类型的数据,print和println方法会先将它们转换成字符串的形式后再输出,而不是输出原始的字节内容,对于一个非基本数据类型的对象,print和println方法会先调用对象的toString方法，然后再输出toString方法返回的字符串。

字符打印流：**PrintWriter**   
构造函数可以接收四种类型的值。   
1，字符串路径。  
2，File对象。   
对于1，2类型的数据，还可以指定编码表。也就是字符集。   
3，OutputStream   
4，Writer   
对于3，4类型的数据，可以指定自动刷新。   
注意：该自动刷新值为true时，只有三个方法可以用：println,printf,format   
如果想要既有自动刷新，又可执行编码。如何完成流对象的包装？   
 PrintWrter pw = **new** PrintWriter(**new** OutputSteamWriter(**new** FileOutputStream(“a.txt”),”utf-8”),**true**);

如果想要提高效率。还要使用打印方法。   
 PrintWrter pw = **new** PrintWriter(**new** BufferdWriter(**new** OutputSteamWriter( **new** FileOutputStream(“a.txt”),”utf-8”)),**true**);

### 4.对象流

**ObjectInputStream ,ObjectOutputStream**   
不同于其他类型的流这里只能用字节对对象进行操作。

ObjectOutputStream对象的序列化：   
将java程序中的对象写到本地磁盘里用ObjectOutputStream eg：将Person类的对象序列化到磁盘

1.创建Person类

注1：此类要实现Serializable接口，此接口为标志性接口

注2：此类要有无参的构造函数

注3：一旦序列化此类不能再修改

**class** Person **implements** Serializable{

**public** Person(){

}

}

2.创建对象流对象

注：要增强功能可以将传入文件缓冲流

ObjectOutputStream oos =**new** ObjectOutputStream(

**new** FileOutputStream(**new** File("文件路径")));

3.写入对象 ，一般会将对象用集合存储起来然后直接将集合写入文件

List<Person> list =**new** ArrayList<>();

list.add(**new** Person());

...(可以添加多个）

oos.writeObject(list);

4.关闭流，处理异常

oos.flush();

oos.close();

}

### 5.中文乱码问题

List<String> lines=**new** ArrayList<String>();

String fileName = "C:\\Users\\ls\\Desktop\\1.txt";

BufferedReader br=**new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(fileName),"UTF-8"));

String line= **null**;

**while** ((line= br.readLine()) != **null**){

lines.add(line);

}

System.*out*.println(lines);

br.close();

**用InputStreamReader代替FileReader，InputStreamReaderisr=new  
InputStreamReader(new FileInputStream(fileName),"UTF-8");这样读取文件就会直接用UTF-8解码，不用再做编码转换。**

### 6. 流操作的基本规律

一、两个明确：(明确体系)

1. 明确源和目的

    源：输入流  InputStream  Reader

    目的：输出流  OutputStream Writer

2. 操作的数据是否是纯文本

    是： 字符流

    否： 字节流

二、明确体系后要明确具体使用的对象

通过设备区分：内存，硬盘，键盘

目的设备：内存，硬盘，控制台

在Java中IO操作也是有相应步骤的，以文件操作为例，主要的操作流程如下：

1 使用File类打开一个文件

2 通过字节流或字符流的子类，指定输出的位置

3 进行读/写操作

4 关闭输入/输出

IO 操作属于资源操作，一定要记得关闭

## 五、字节流与字符流的区别

**java**中提供了专用于输入输出功能的包**Java.io,**其中包括：  
     **InputStream,OutputStream,Reader,Writer**

**inputStream** 和**OutputStream**,两个是为字节流设计的,主要用来处理字节或二进制对象,  
     **Reader**和 **Writer**.两个是为字符流（一个字符占两个字节）设计的,主要用来处理字符或字符串.

===================我们还可以看到：==================  
Reader类的read()方法返回类型为int ：作为整数读取的字符（占两个字节共16位），范围在 **0 到 65535** 之间 (0x00-0xffff)，如果已到达流的末尾，则返回 -1

inputStream的read()虽然也返回int，但由于此类是面向字节流的，一个字节占8个位，所以返回 **0 到 255** 范围内的 int 字节值。如果因为已经到达流末尾而没有可用的字节，则返回值 -1。因此对于不能用0-255来表示的值就得用字符流来读取！比如说汉字.

**字节流和字符流使用是非常相似的，那么除了操作代码的不同之外，还有哪些不同呢？**

字节流在操作的时候本身是不会用到缓冲区（内存）的，是与文件本身直接操作的，而字符流在操作的时候是使用到缓冲区的

字节流在操作文件时，即使不关闭资源（close方法），文件也能输出，但是如果字符流不使用close方法的话，则不会输出任何内容，说明字符流用的是缓冲区，并且可以使用flush方法强制进行刷新缓冲区，这时才能在不close的情况下输出内容

**那开发中究竟用字节流好还是用字符流好呢？**

在所有的硬盘上保存文件或进行传输的时候都是以字节的方法进行的，包括图片也是按字节完成，而字符是只有在内存中才会形成的，所以使用字节的操作是最多的。