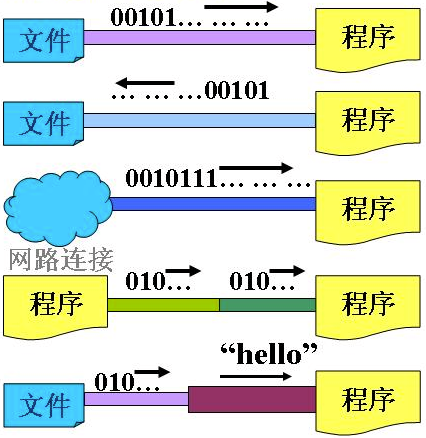
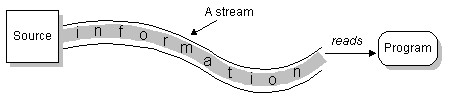
# 什么是stream

流是个抽象的概念，是对输入输出设备的抽象，Java程序中，对于数据的输入/输出操作都是以“流”的方式进行。设备可以是文件，网络，内存等。



流具有方向性，至于是输入流还是输出流则是一个相对的概念，一般以程序为参考，如果数据的流向是程序至设备，我们成为输出流，反之我们称为输入流。

可以将流想象成一个“水流管道”，水流就在这管道中形成了，自然就出现了方向的概念。



当程序需要从某个数据源读入数据的时候，就会开启一个输入流，数据源可以是文件、内存或网络等等。相反地，需要写出数据到某个数据源目的地的时候，也会开启一个输出流，这个数据源目的地也可以是文件、内存或网络等等。

# 流分类

可以从不同的角度对流进行分类：

1.处理的数据单位不同，可分为：字符流，字节流

2.数据流方向不同，可分为：输入流，输出流

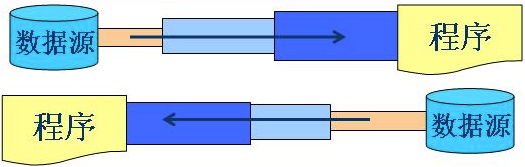
3.功能不同，可分为：节点流，处理流

1.和2. 都比较好理解，对于根据功能分类的，可以这么理解：

**节点流**：节点流从一个特定的数据源读写数据。即节点流是直接操作文件，网络等的流，例如FileInputStream和FileOutputStream，他们直接从文件中读取或往文件中写入字节流。



**处理流**：“连接”在已存在的流（节点流或处理流）之上通过对数据的处理为程序提供更为强大的读写功能。过滤流是 使用一个已经存在的输入流或输出流连接创建的，过滤流就是对节点流进行一系列的包装。例如BufferedInputStream和 BufferedOutputStream，使用已经存在的节点流来构造，提供带缓冲的读写，提高了读写的效率，以及DataInputStream和 DataOutputStream，使用已经存在的节点流来构造，提供了读写Java中的基本数据类型的功能。他们都属于过滤流。



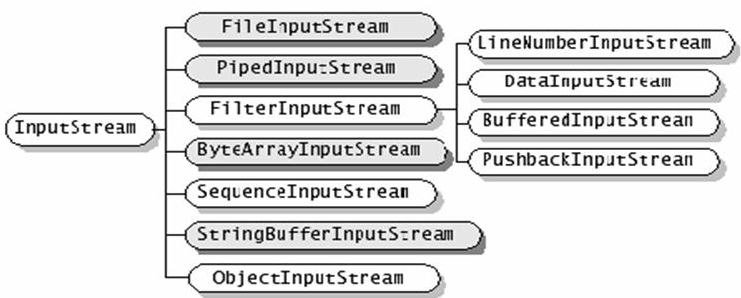
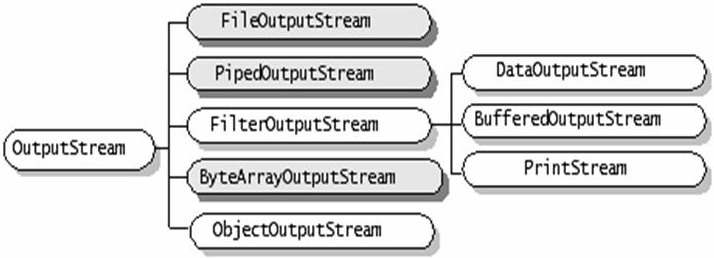
|  |
| --- |
| public static void main(String[] args) throws IOException {  // 节点流FileOutputStream直接以A.txt作为数据源操作  FileOutputStream fileOutputStream = new FileOutputStream("A.txt");  // 过滤流BufferedOutputStream进一步装饰节点流，提供缓冲写  BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(fileOutputStream);  // 过滤流DataOutputStream进一步装饰过滤流，使其提供基本数据类型的写  DataOutputStream out = new DataOutputStream(bufferedOutputStream);  out.writeInt(3);  out.writeBoolean(true);  out.flush();  out.close();  // 此处输入节点流，过滤流正好跟上边输出对应，读者可举一反三  DataInputStream in = new DataInputStream(new BufferedInputStream(  new FileInputStream("A.txt")));  System.out.println(in.readInt());  System.out.println(in.readBoolean());  in.close();  } |

# API

Java所有的流类位于java.io包中，都分别继承字以下四种抽象流类型。

|  |  |  |
| --- | --- | --- |
|  | 字节流 | 字符流 |
| 输入流 | InputStream | Reader |
| 输出流 | OutputStream | Writer |

## 字节流API

继承自InputStream/OutputStream的流都是用于向程序中输入/输出数据，且数据的单位都是字节(byte=8bit):  

### Int InputStream#read():

从输入流中读取一个字节并返回，或返回值为-1

### Int InputStream#read(byte b[]):

从流中读取b.length个字节，从b第0个位置开始放读到的字节，并返回读取的个数或-1（流中已经没有字符）

### Int InputStream#read(byte b[],int off,int len):

从流中读取len个字节，从b第off个位置开始放读到的字节，并返回读取的个数或-1（流中已经没有字符）

### void OutputStream#write(int b):

写指定字节

### void OutputStream#write(byte[] b):

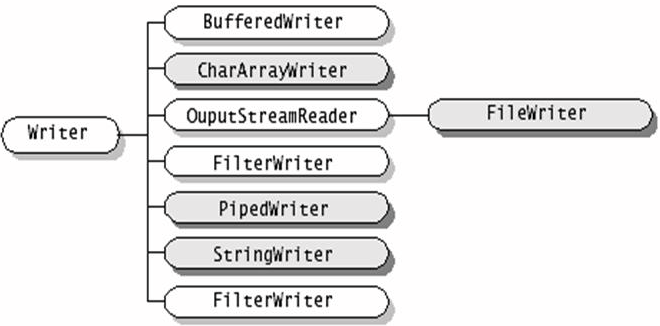
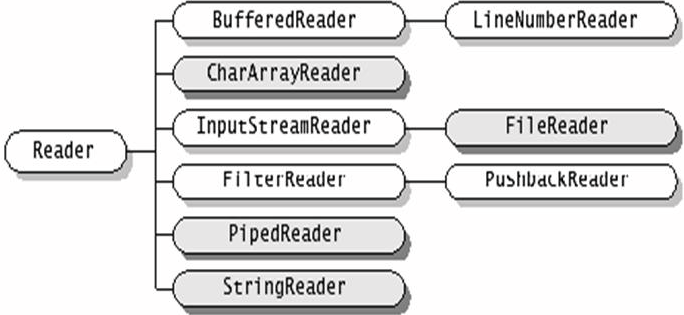
写字节数组(注off=0,len=b.length)

### void OutputStream#write(byte[] b,int off,int len):

从字节数组off位置取len长度写入到流中。

## 字符流API

继承自Reader/Writer的流都是用于向程序中输入/输出数据，且数据的单位都是字符(2byte=16bit):



### int Reader#read()

\* Reads a single character. This method will block until a character is

\* available, an I/O error occurs, or the end of the stream is reached.

读一个字节并返回(值为0~255)，如果没有则返回-1.

### int Reader#read(char cbuf [])

\* Reads characters into an array.

从流中读取cbuf.length个字符，从cbuf第0个位置开始放读到的字符，并返回读取的个数或-1（流中已经没有字符）

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

\* Reads characters into an array.

从流中读取len个字符，从cbuf第off个位置开始放读到的字符，并返回读取的个数或-1（流中已经没有字符）

### void Writer#write(char buf[])

\* Array of characters to be written

将字符数组写入到流中

### void Writer#write(char buf[],int off,int len)

\* Writes a portion of an array of characters

从字符数组off位置取len长度写入到流中。

### void Writer#write(String str)

\* str to be written

将字符串写入到流中

### void Writer#write(String str,int off,int len)

\* Writes a portion of an array of characters

从字符串off位置取len长度写入到流中。

### Void Write#flush()

将流中刷新到终端设备（比如文件）.

## 常见流类介绍：

**节点流类型：**

* 字符流中有对文件操作的FileReader/FileWriter、，
* 字节流中有对文件操作的FileInputStream/FileOutputStream、。

**处理流（包装流）类型：**

* 缓冲流：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，提高了读写效率，同事增加了一些新的方法。

字节缓冲流有BufferedInputStream/BufferedOutputStream，

字符缓冲流有BufferedReader/BufferedWriter，字符缓冲流分别提供了读取和写入一行的方法ReadLine和NewLine方法。

　　对于输出地缓冲流，写出的数据，会先写入到内存中，再使用flush方法将内存中的数据刷到硬盘。所以，在使用字符缓冲流的时候，一定要先flush，然后再close，避免数据丢失。

* 转换流：用于字节数据到字符数据之间的转换。

字符流-->字节流：无。

字节流-->字符符流：InputStreamReader/OutputStreamWriter。

* 数据流：提供了读写Java中的基本数据类型的功能。

　　DataInputStream和DataOutputStream分别继承自InputStream和OutputStream，需要“套接”在InputStream和OutputStream类型的节点流之上。

* 对象流：用于直接将对象写入写出。

　　流类有ObjectInputStream和ObjectOutputStream，本身这两个方法没什么，但是其要写出的对象有要求，该对象必须实现Serializable接口，来声明其是可以序列化的。否则，不能用对象流读写。

　　还有一个关键字比较重要，transient，由于修饰实现了Serializable接口的类内的属性，被该修饰符修饰的属性，在以对象流的方式输出的时候，该字段会被忽略。

# 标题1

## 标题1.1

### 标题1.1.1

#### 标题1.1.1.1

##### 标题1.1.1.1.1