# [Java中的IO流系统详解](http://blog.csdn.net/jiangwei0910410003/article/details/22376895)

2014-03-28 15:05 18537人阅读 [评论](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "comments)(5) [收藏](http://blog.csdn.net/jiangwei0910410003/article/details/javascript:void(0);" \o "收藏) [举报](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "report" \o "举报)

IMG_256 分类：

Java（34） IMG_257

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "系统根据文章中H1到H6标签自动生成文章目录)[[-]](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "收起)

1. [摘要](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t0)
2. [一以字节为导向的 Stream------InputStreamOutputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t1)
   1. [InputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t2)
      1. [基类InputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t3)
      2. [1 ByteArrayInputStream -- 把内存中的一个缓冲区作为 InputStream 使用](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t4)
      3. [2 StringBufferInputStream -- 把一个 String 对象作为 InputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t5)
      4. [3 FileInputStream -- 把一个文件作为 InputStream 实现对文件的读取操作](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t6)
      5. [4 PipedInputStream 实现了 pipe 的概念主要在线程中使用 管道输入流是指一个通讯管道的接收端](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t7)
      6. [5 SequenceInputStream 把多个 InputStream 合并为一个 InputStream 序列输入流类允许应用程序把几个输入流连续地合并起来](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t8)
      7. [6 ObjectInputStream用于操作Object的stream这个在stream主要用在对象传输的过程中其中牵涉到了序列化的知识](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t9)
      8. [7 FilterInputStream是一个过滤的InputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t10)
         1. [71 BufferedInputStream使用缓冲区的stream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t11)
         2. [72 DataInputStream数字格式化的stream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t12)
      9. [OutputSteam](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t13)
         1. [基类OutputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t14)
         2. [1 ByteArrayOutputStream 把信息存入内存中的一个缓冲区中 该类实现一个以字节数组形式写入数据的输出流](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t15)
         3. [2  FileOutputStream 文件输出流是向 File 或 FileDescriptor 输出数据的一个输出流](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t16)
         4. [3 PipedOutputStream 管道输出流是指一个通讯管道的发送端 一个线程通过管道输出流发送数据](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t17)
         5. [4 ObjectOutputStream输出Object对象的stream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t18)
         6. [5  FilterOutputStream过滤的OutputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t19)
            1. [51 BufferedOutputStream带有缓冲区的stream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t20)
         7. [52 DataOutputStream具有格式化的OutputStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t21)
         8. [53 PrintStream直接输出到控制台中我们最熟悉的就是Systemout他就是一个PrintStream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t22)
3. [二 以字符为导向的 Stream  ReaderWriter](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t23)
   1. [Reader](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t24)
      1. [Reader基类基于字符流的stream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t25)
      2. [1 CharArrayReader 与 ByteArrayInputStream 对应此类实现一个可用作字符输入流的字符缓冲区](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t26)
      3. [2 StringReader 与 StringBufferInputStream 对应其源为一个字符串的字符流](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t27)
      4. [3 FileReader 与 FileInputStream 对应](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t28)
      5. [4 PipedReader 与 PipedInputStream 对应](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t29)
   2. [Writer](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t30)
      1. [1    CharArrayWriter 与 ByteArrayOutputStream 对应](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t31)
      2. [2   StringWriter无与之对应的以字节为导向的 stream](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t32)
      3. [3  FileWriter 与 FileOutputStream 对应](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t33)
      4. [4  PipedWriter与 PipedOutputStream 对应](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t34)
      5. [5 OutputStreamWriter将OutputStream转化成Writer](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t35)
      6. [6 PrintReader和PrintStream对应](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t36)
   3. [两种不同导向的 Stream 之间的转换](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t37)
      1. [1 InputStreamReader 和 OutputStreamReader](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t38)
4. [三Java IO 的一般使用原则](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t39)
   1. [一按数据来源去向分类](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t40)
   2. [二按是否格式化输出分](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t41)
   3. [三按是否要缓冲分](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t42)
   4. [四按数据格式分](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t43)
   5. [五按输入输出分](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t44)
   6. [六特殊需要](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \l "t45)

# ****摘要：****

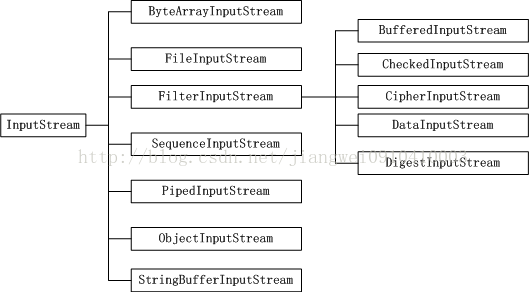
**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "http://blog.csdn.net/jiangwei0910410003/article/details/_blank)** 流在处理上分为字符流和字节流。字符流处理的单元为 2 个字节的 Unicode 字符，分别操作字符、字符数组或字符串，而字节流处理单元为 1 个字节，操作字节和字节数组。

Java 内用 Unicode 编码存储字符，字符流处理类负责将外部的其他编码的字符流和 java 内 Unicode 字符流之间的转换。而类 InputStreamReader 和 OutputStreamWriter 处理字符流和字节流的转换。字符流（一次可以处理一个缓冲区）一次操作比字节流（一次一个字节）效率高。

# ****(一)以字节为导向的 Stream------InputStream/OutputStream****

InputStream 和 OutputStream 是两个 abstact 类，对于字节为导向的 stream 都扩展这两个基类;

## ****1、 InputStream****



### ****基类InputStream:****

**构造方法:**

InputStream() 创建一个输入的stream流

**方法：**

available():返回stream中的可读字节数，inputstream类中的这个方法始终返回的是0，这个方法需要子类去实现。

close():关闭stream方法，这个是每次在用完流之后必须调用的方法。

read():方法是读取一个byte字节,但是返回的是int。

read(byte[]):一次性读取内容到缓冲字节数组

read(byte[],int,int):从数据流中的哪个位置offset开始读长度为len的内容到缓冲字节数组

skip(long):从stream中跳过long类型参数个位置

以上的方法都是很简单理解的，这里就不写代码介绍了。

下面还有三个方法：

mark(int):用于标记stream的作用

markSupported()：返回的是boolean类型，因为不是所有的stream都可以调用mark方法的，这个方法就是用来判断stream是否可以调用mark方法和reset方法

reset()：这个方法和mark方法一起使用的，让stream回到mark的位置。

上面说的可能抽象了点，下面就用代码来解释一下吧：

**[java]** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "copy) [IMG_259](https://code.csdn.net/snippets/262008)[IMG_260](https://code.csdn.net/snippets/262008/fork)

1. **package** com.io.demo;
3. **import** java.io.BufferedInputStream;
4. **import** java.io.BufferedOutputStream;
5. **import** java.io.File;
6. **import** java.io.FileInputStream;
7. **import** java.io.FileOutputStream;
8. **import** java.io.IOException;
9. **import** java.io.InputStream;
10. **import** java.io.OutputStream;
12. **public** **class** InputStreamTest {
13. **public** **static** **void** main(String[] args) **throws** IOException {
14. writeToFile();
15. readFromFile();
16. }
18. **private** **static** **void** readFromFile() {
19. InputStream inputStream = **null**;
20. **try** {
21. inputStream = **new** BufferedInputStream(**new** FileInputStream(**new** File("test.txt")));
22. // 判断该输入流是否支持mark操作
23. **if** (!inputStream.markSupported()) {
24. System.out.println("mark/reset not supported!");
25. **return**;
26. }
27. **int** ch;
28. **int** count = 0;
29. **boolean** marked = **false**;
30. **while** ((ch = inputStream.read()) != -1) {
31. System.out.print("." + ch);
32. **if** ((ch == 4) && !marked) {
33. // 在4的地方标记位置
34. inputStream.mark(10);
35. marked = **true**;
36. }
37. **if** (ch == 8 && count < 2) {
38. // 重设位置到4
39. inputStream.reset();
40. count++;
41. }
42. }
43. } **catch** (Exception e) {
44. e.printStackTrace();
45. } **finally** {
46. **try** {
47. inputStream.close();
48. } **catch** (Exception e) {
49. e.printStackTrace();
50. }
51. }
52. }
54. **private** **static** **void** writeToFile() {
55. OutputStream output = **null**;
56. **try** {
57. output = **new** BufferedOutputStream(**new** FileOutputStream(**new** File("test.txt")));
58. **byte**[] b = **new** **byte**[20];
59. **for** (**int** i = 0; i < 20; i++)
60. b[i] = (**byte**) i;
61. // 写入从0到19的20个字节到文件中
62. output.write(b);
63. } **catch** (IOException e) {
64. e.printStackTrace();
65. } **finally** {
66. **try** {
67. output.close();
68. } **catch** (IOException e) {
69. e.printStackTrace();
70. }
71. }
72. }
73. }

输出：.0.1.2.3.4.5.6.7.8.5.6.7.8.5.6.7.8.9.10.11.12.13.14.15.16.17.18.19

从输出可以看到，我们在ch==4的时候调用了mark方法进行标记了，当ch==8的时候我们调用了reset方法，就让stream回到了我们标记的地方重新读取。

上面的例子是在整个读取stream的过程中操作的，下面来看一下当我们读取完一个stream的时候再去调用reset方法的效果：

**[java]** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "copy) [IMG_261](https://code.csdn.net/snippets/262008)[IMG_262](https://code.csdn.net/snippets/262008/fork)

1. **public** **static** **void** main(String[] args) **throws** Exception{
2. BufferedInputStream bis = **new** BufferedInputStream(**new** ByteArrayInputStream("Hello World!".getBytes()));
3. **int** bytes = -1;
4. //这里进行标记
5. bis.mark(12);
6. **while**((bytes=bis.read())!=-1){
7. System.out.print(bytes+",");
8. }
9. System.out.println();
10. //让标签有效，此时bis又回到了开始点
11. bis.reset();
12. **while**((bytes=bis.read())!=-1){
13. System.out.print(bytes+".");
14. }
15. }

输出：

72,101,108,108,111,32,87,111,114,108,100,33,  
72.101.108.108.111.32.87.111.114.108.100.33.

我们可以看到当我们在stream读取完调用reset方法的话，stream就会回到开始点。

通过上面的例子我们可能知道了mark和reset方法的作用和好处：

如果想对一个stream进行多次操作的话，只要读取一次就可以了，不需要多次读取，比如：上面当读取完stream之后，想再去从头操作stream，只需要调用reset方法即可，如果没有mark的话，我们就需要从新在去读取一次数据了。

下面再来看一下mark这个方法中的参数的意义：

根据JAVA官方文档的描述，mark(int readlimit)方法表示，标记当前位置，并保证在mark以后最多可以读取readlimit字节数据，mark标记仍有效。如果在mark后读取超过readlimit字节数据，mark标记就会失效，调用reset()方法会有异常。但实际的运行情况却和JAVA文档中的描述并不完全相符。 有时候在BufferedInputStream类中调用mark(int readlimit)方法后，即使读取超过readlimit字节的数据，mark标记仍有效，仍然能正确调用reset方法重置。

事实上，mark在JAVA中的实现是和缓冲区相关的。只要缓冲区够大，mark后读取的数据没有超出缓冲区的大小，mark标记就不会失效。如果不够大，mark后又读取了大量的数据，导致缓冲区更新，原来标记的位置自然找不到了。

因此，mark后读取多少字节才失效，并不完全由readlimit参数确定，也和BufferedInputStream类的缓冲区大小有关。  如果BufferedInputStream类的缓冲区大小大于readlimit，在mark以后只有读取超过缓冲区大小的数据，mark标记才会失效。

**简言之，BufferedInputStream类调用mark(int readlimit)方法后读取多少字节标记才失效，是取readlimit和BufferedInputStream类的缓冲区大小两者中的最大值，而并非完全由readlimit确定。这个在JAVA文档中是没有提到的。**

看下面的例子Mark.java。

**[java]** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "copy) [IMG_263](https://code.csdn.net/snippets/262008)[IMG_264](https://code.csdn.net/snippets/262008/fork)

1. **package** com.io.demo;
3. **import** java.io.BufferedInputStream;
4. **import** java.io.ByteArrayInputStream;
5. **import** java.io.IOException;
7. /\*\*
8. \* @author WuDian
9. \*
10. \*/
11. **public** **class** Mark {
12. **public** **static** **void** main(String[] args) {
14. **try** {
15. // 初始化一个字节数组，内有5个字节的数据
16. **byte**[] bytes={1,2,3,4,5};
17. // 用一个ByteArrayInputStream来读取这个字节数组
18. ByteArrayInputStream in=**new** ByteArrayInputStream(bytes);
19. // 将ByteArrayInputStream包含在一个BufferedInputStream，并初始化缓冲区大小为2。
20. BufferedInputStream bis=**new** BufferedInputStream(in,2);
21. // 读取字节1
22. System.out.print(bis.read()+",");
23. // 在字节2处做标记，同时设置readlimit参数为1
24. // 根据JAVA文档mark以后最多只能读取1个字节，否则mark标记失效，但实际运行结果不是这样
25. System.out.println("mark");
26. bis.mark(1);
28. /\*
29. \* 连续读取两个字节，超过了readlimit的大小，mark标记仍有效
30. \*/
31. // 连续读取两个字节
32. System.out.print(bis.read()+",");
33. System.out.print(bis.read()+",");
34. // 调用reset方法，未发生异常，说明mark标记仍有效。
35. // 因为，虽然readlimit参数为1，但是这个BufferedInputStream类的缓冲区大小为2，
36. // 所以允许读取2字节
37. System.out.println("reset");
38. bis.reset();
40. /\*
41. \* 连续读取3个字节，超过了缓冲区大小，mark标记失效。
42. \* 在这个例子中BufferedInputStream类的缓冲区大小大于readlimit,
43. \* mark标记由缓冲区大小决定
44. \*/
45. // reset重置后连续读取3个字节，超过了BufferedInputStream类的缓冲区大小
46. System.out.print(bis.read()+",");
47. System.out.print(bis.read()+",");
48. System.out.print(bis.read()+",");
49. // 再次调用reset重置，抛出异常，说明mark后读取3个字节，mark标记失效
50. System.out.println("reset again");
51. bis.reset();
52. } **catch** (IOException e) {
53. // TODO Auto-generated catch block
54. e.printStackTrace();
55. }
56. }
57. }

运行结果：

1,mark  
2,3,reset  
2,3,4,reset again  
java.io.IOException: Resetting to invalid mark  
at java.io.BufferedInputStream.reset(Unknown Source)  
at com.io.demo.Mark.main(Mark.java:51)

这样我们就对上面的三个方法进行的解释了。其实等说完所以的stream之后我们会发现，mark和reset方法只有Buffered类的stream有效，所以stream中都会有一个markSupported方法来判断一个stream中的mark和reset方法是否可用。

上面就介绍了InputStream基类中的所有方法，下面是其的子类，对于子类可能会添加以下属于自己特定的方法，我们这里就只介绍这些特定的方法

### ****1.1 ByteArrayInputStream**** -- 把内存中的一个缓冲区作为 InputStream 使用 .

**construct---**

(A)ByteArrayInputStream(byte[]) 创建一个新字节数组输入流（ ByteArrayInputStream ），它从指定字节数组中读取数据（ 使用 byte 作为其缓冲区数组）

(B)ByteArrayInputStream(byte[], int, int) 创建一个新字节数组输入流，它从指定字节数组中读取数据。

**method---**

方法都是实现了InputStream方法。

### ****1.2 StringBufferInputStream**** -- 把一个 String 对象作为 InputStream .

**construct---**

StringBufferInputStream(String) 据指定串创建一个读取数据的输入流串。

**method---**

实现了InputStream中的部分方法

注释：不推荐使用 StringBufferInputStream 方法。 此类不能将字符正确的转换为字节。

同 JDK 1.1 版中的类似，从一个串创建一个流的最佳方法是采用 StringReader 类。

### ****1.3 FileInputStream**** -- 把一个文件作为 InputStream ，实现对文件的读取操作

**construct---**

(A)FileInputStream(File name) 创建一个输入文件流，从指定的 File 对象读取数据。

(B)FileInputStream(FileDescriptor) 创建一个输入文件流，从指定的文件描述器读取数据。

(C)-FileInputStream(String  name) 创建一个输入文件流，从指定名称的文件读取数据。

**method ----**

实现了InputStream中的部分方法；

额外的两个方法：

getChannel()：这个方法返回一个FileChannel对象，这个主要用于JNIO中的通道的。

getFD()：这个方法返回一个FileDescriptor对象，这个在构造函数中使用过。

### ****1.4 PipedInputStream**** ：实现了 pipe 的概念，主要在线程中使用 . 管道输入流是指一个通讯管道的接收端。

一个线程通过管道输出流发送数据，而另一个线程通过管道输入流读取数据，这样可实现两个线程间的通讯。

**construct---**

PipedInputStream() 创建一个管道输入流，它还未与一个管道输出流连接。

PipedInputStream(PipedOutputStream) 创建一个管道输入流 , 它已连接到一个管道输出流。

**method---**

实现了InputStream中的部分方法

额外的方法：

connection(PipedOutputStream):用来连接PipedOutputStream对象

### ****1.5 SequenceInputStream**** ：把多个 InputStream 合并为一个 InputStream . “序列输入流”类允许应用程序把几个输入流连续地合并起来，

并且使它们像单个输入流一样出现。每个输入流依次被读取，直到到达该流的末尾。

然后“序列输入流”类关闭这个流并自动地切换到下一个输入流。

**construct---**

SequenceInputStream(Enumeration) 创建一个新的序列输入流，并用指定的输入流的枚举值初始化它。

SequenceInputStream(InputStream, InputStream) 创建一个新的序列输入流，初始化为首先 读输入流 s1, 然后读输入流 s2 。

**method---**

实现了InputStream中的部分方法。

**[java]** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "copy) [IMG_265](https://code.csdn.net/snippets/262008)[IMG_266](https://code.csdn.net/snippets/262008/fork)

1. **public** **class** TestSequence {
2. **public** **static** **void** main(String[] args)**throws** Exception{
3. //不管怎么样，先创建两个文件。从里面读取数据，（我的文件里面有内容！）
4. InputStream in1=**new** FileInputStream(**new** File("d:\\jack.txt"));
5. InputStream in2=**new** FileInputStream(**new** File("d:\\jack2.txt"));
6. //读取，数据后要生成的文件
7. OutputStream ou=**new** FileOutputStream(**new** File("d:\\jack3.txt"));
8. //创建SequenceInputStream类，
9. SequenceInputStream si=**new** SequenceInputStream(in1,in2);
10. //因为SequenceInputStream，是一个字节一个字节读，要判断一下是否读完了。
11. **int** c=0;
12. **while**((c=si.read())!=-1){
13. //这里也是一个字节一个字节写的。
14. ou.write(c);
15. }
16. //关闭所有的资源
17. si.close();
18. ou.close();
19. in2.close();
20. in1.close();
21. }
22. }

能够将多个stream进行连接，然后输出。

### ****1.6 ObjectInputStream****：用于操作Object的stream，这个在stream主要用在对象传输的过程中，其中牵涉到了序列化的知识

**construct---**

ObjectInputStream():实例化一个ObjectInputStream对象

ObjectInputStream(InputStream):使用一个InputStream对象来实例化一个ObjectInputStream对象，其中InputStream就是对象的输入流

**method---**

实现了InputStream中的部分方法；

它自己有很多的额外的方法：

这里就只介绍一下readObject(Object)方法，就是将一个对象写入到stream中，但是这个object必须序列化

其他的还有像readInt,readFloat等这样基本类型的方法，因为基本类型对应的对象也都是Object.

### ****1.7 FilterInputStream****：是一个过滤的InputStream

**constructor---**

FilterInputStream(InputStream):使用一个InputStream为参数实例化一个FilterInputStream,其实就是来修饰InputStream的

**method---**

实现了InputStream中的所有方法

他其实没有作用，他的众多子类是很有用的：

#### ****1.7.1 BufferedInputStream****:使用缓冲区的stream

**constructor---**

BufferedInputStream(InputStream):使用InputStream为参数初始化实例

BufferedInputStream(InputStream,int):能够设置缓冲区大小的BufferedInputStream

**method---**

实现了FilterInputStream中的所有方法(其实也是实现了InputStream中的所有方法)

#### ****1.7.2 DataInputStream****:数字格式化的stream

**constructor---**

DataInputStream(InputStream):使用InputStream参数来初始化实例

**method---**

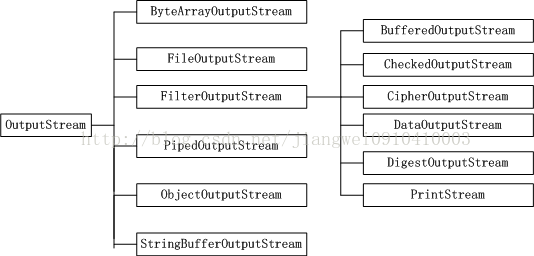
实现了FilterInputStream中的部分方法

额外的方法：

readInt,readFloat,readDouble...这样可以直接从stream中读取基本类型的数据

还有其他的类就不做解释了，因为用到的很少了，有的可能一辈子都用不到

## 2、 OutputSteam



### ****基类OutputStream****

**constructor---**

OutputStream()

**method---**

write(int):写入一个字节到stream中

write(byte[])写入一个byte数组到stream中

write(byte[],int,int)：把byte数组中从offset开始处写入长度为len的数据

close()：关闭流，这个是在操作完stream之后必须要调用的方法

flush():这个方法是用来刷新stream中的数据，让缓冲区中的数据强制的输出

### ****2.1 ByteArrayOutputStream**** ： 把信息存入内存中的一个缓冲区中 . 该类实现一个以字节数组形式写入数据的输出流。

当数据写入缓冲区时，它自动扩大。用 toByteArray() 和 toString() 能检索数据。

**constructor---**

(A)--- ByteArrayOutputStream() 创建一个新的字节数组输出流。

(B)--- ByteArrayOutputStream() 创建一个新的字节数组输出流。

(C)--- ByteArrayOutputStream(int) 创建一个新的字节数组输出流，并带有指定大小字节的缓冲区容量。

**method---**

实现了OutputStream的部分方法：

额外的方法：

toByteArray() 将字节流转化成一个字节数组，用于数据的检索

toString() 将字节流转化成一个String对象，默认采用系统的编码转化，同样可以用于数据的检索

toString(String) 根据指定字符编码将缓冲区内容转换为字符串，并将字节转换为字符。

writeTo(OutputStream) 用 out.write(buf, 0, count) 调用输出流的写方法将该字节数组输出流的全部内容写入指定的输出流参数。

### ****2.2**** ****FileOutputStream****: 文件输出流是向 File 或 FileDescriptor 输出数据的一个输出流。

**constructor---**

(A)FileOutputStream(File  name) 创建一个文件输出流，向指定的 File 对象输出数据。

(B)FileOutputStream(FileDescriptor) 创建一个文件输出流，向指定的文件描述器输出数据。

(C)FileOutputStream(String  name) 创建一个文件输出流，向指定名称的文件输出数据。

(D)FileOutputStream(String, boolean) 用指定系统的文件名，创建一个输出文件。

**method---**

实现了OutputStream中的部分方法。

额外的两个方法：

getChannel()：这个方法返回一个FileChannel对象，这个主要用于JNIO中的通道的。

getFD()：这个方法返回一个FileDescriptor对象，这个在构造函数中使用过。

### ****2.3 PipedOutputStream****: 管道输出流是指一个通讯管道的发送端。 一个线程通过管道输出流发送数据，

而另一个线程通过管道输入流读取数据，这样可实现两个线程间的通讯。

**constructor---**

(A)PipedOutputStream() 创建一个管道输出流，它还未与一个管道输入流连接。

(B)PipedOutputStream(PipedInputStream) 创建一个管道输出流，它已连接到一个管道输入流。

**method---**

实现了OutputStream的部分方法

额外的方法：

connection(PipedInputStream):连接一个PipedInputStream方法

下面来看一下PipedInputStream和PipedOutputStream的例子：

**[java]** [view plain](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "view plain) [copy](http://blog.csdn.net/jiangwei0910410003/article/details/22376895" \o "copy) [IMG_268](https://code.csdn.net/snippets/262008)[IMG_269](https://code.csdn.net/snippets/262008/fork)

1. **package** com.io.demo;
3. **import** java.io.BufferedInputStream;
4. **import** java.io.BufferedReader;
5. **import** java.io.ByteArrayInputStream;
6. **import** java.io.IOException;
7. **import** java.io.InputStreamReader;
8. **import** java.io.PipedInputStream;
9. **import** java.io.PipedOutputStream;
11. **public** **class** IO {
13. **public** **static** **void** main(String[] args) **throws** Exception{
15. PipedOutputStream pos = **new** PipedOutputStream();
16. **try** {
17. PipedInputStream pis = **new** PipedInputStream(pos);
18. **new** Thread(**new** InputStreamRunnable(pis)).start();
19. **new** Thread(**new** OutputStreamRunnable(pos)).start();
20. } **catch** (IOException e) {
21. e.printStackTrace();
22. }
23. }
25. **static** **class** InputStreamRunnable **implements** Runnable{
26. **private** PipedInputStream pis = **null**;
27. **public** InputStreamRunnable(PipedInputStream pis){
28. **this**.pis = pis;
29. }
30. @Override
31. **public** **void** run() {
32. BufferedReader sr = **new** BufferedReader(**new** InputStreamReader(pis));
33. **try** {
34. System.out.println("读取到的内容:"+sr.readLine());
35. sr.close();
36. } **catch** (IOException e) {
37. e.printStackTrace();
38. }
39. }
41. }
43. **static** **class** OutputStreamRunnable **implements** Runnable{
44. **private** PipedOutputStream pos = **null**;
45. **public** OutputStreamRunnable(PipedOutputStream pos){
46. **this**.pos = pos;
47. }
48. @Override
49. **public** **void** run(){
50. **try** {
51. pos.write("Hello World!".getBytes());
52. pos.close();
53. } **catch** (IOException e) {
54. e.printStackTrace();
55. }
56. }
57. }
59. }

其实PipedInputStream/PipedOutputStream相当于消费者和生产者机制。

### ****2.4 ObjectOutputStream****:输出Object对象的stream

**constructor---**

ObjectOutputStream:实例化一个ObjectOutputStream对象

ObjectOutputStream(OutputStream):使用OutputStream对象来实例化一个ObjectOutputStream对象，将对象写入到OutputStream中

**method---**

实现了OutputStream中的部分方法

额外的方法：

这里只介绍一个writeObject(Object):将一个对象Object写入到OutputStream中

同时他也有writeInt,writeFloat..这样的基本类型的方法，因为基本类型对应的对象类型都是Object的子类

关于ObjectInputStream/ObjectOutputStream主要用与将一个对象Object写入到本地或者是在网络中进行传输的，所以这些对象需要进行序列化操作。

### ****2.5  FilterOutputStream****:过滤的OutputStream

**constructor---**

FilterOutputStream(OutputStream):使用OutputStream参数实例化FilterOutputStream

**method---**

实现了OutputStream的所有方法

和FilterInputStream一样，他其实也没有什么用途，他的子类很有用的

#### ****2.5.1 BufferedOutputStream****:带有缓冲区的stream

**constructor---**

BufferedOutputStream(OutputStream):使用OutputStream参数初始化BufferedOutputStream类

BufferedOutputStream(OutputStream,int)：在初始化实例的时候指定缓冲区的大小

**method---**

实现了FilterOutputStream中的部分方法

### ****2.5.2 DataOutputStream****:具有格式化的OutputStream

**constructor---**

DataOutputStream(OutputStream):使用OutputStream参数来初始化DataOutputStream实例

**method---**

实现了FilterOutputStream中的部分方法

writeInt,writeFloat,writeDouble....能够给直接向stream中写入基本类型的方法

### ****2.5.3 PrintStream****:直接输出到控制台中：我们最熟悉的就是System.out他就是一个PrintStream

**constructor---**

PrintStream(OutputStream):使用OutputStream参数来实例化PrintStream

**method---**

实现了FilterOutputStream中的部分方法

print()有多个重载的方法，8中基本类型和String类型，同时他还可以进行格式化输出。

其他还有一些stream可能用不到，这里就不做介绍了。

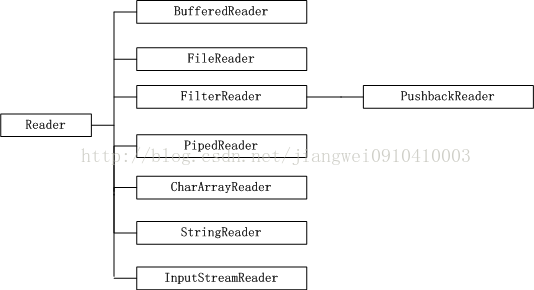
# ****( 二 )以字符为导向的 Stream  Reader/Writer****

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。

Reader/Writer 为 abstact 类

以 Unicode 字符为导向的 stream 包括下面几种类型：

## ****1. Reader****



### ****Reader基类****：基于字符流的stream

**constructor--**

Reader():无参的构造方法

**method---**

这些方法和InputStream中的方法是一一对应的，就是读取的类型不同，InputStream中读取的是byte,而Reader中读取的是char

### ****1.1 CharArrayReader**** ：与 ByteArrayInputStream 对应此类实现一个可用作字符输入流的字符缓冲区

**constructor---**

CharArrayReader(char[]) 用指定字符数组创建一个 CharArrayReader 。

CharArrayReader(char[], int, int) 用指定字符数组创建一个 CharArrayReader

**method---**

实现了部分Reader的方法

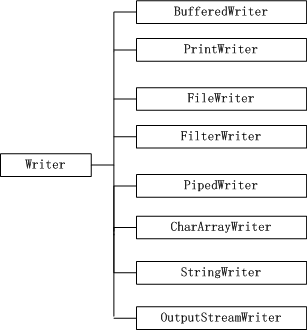
### ****1.2 StringReader**** ： 与 StringBufferInputStream 对应其源为一个字符串的字符流。

### ****1.3 FileReader**** ： 与 FileInputStream 对应

### ****1.4 PipedReader**** ：与 PipedInputStream 对应

**1.5 InputStreamReader**：将InputStream转化成Reader

## ****2.  Writer****



### ****2.1****    CharArrayWriter： 与 ByteArrayOutputStream 对应

### ****2.2****   ****StringWriter****：无与之对应的以字节为导向的 stream

### **2.3**FileWriter： 与 FileOutputStream 对应

### ****2.4****  PipedWriter：与 PipedOutputStream 对应

### ****2.5**** ****OutputStreamWriter****：将OutputStream转化成Writer

### ****2.6**** ****PrintReader****：和PrintStream对应

## ****3、两种不同导向的 Stream 之间的转换****

### ****3.1 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 。

InputStreamReader 类是从字节流到字符流的桥梁：它读入字节，并根据指定的编码方式，将之转换为字符流。

使用的编码方式可能由名称指定，或平台可接受的缺省编码方式。

InputStreamReader 的 read() 方法之一的每次调用，可能促使从基本字节输入流中读取一个或多个字节。

为了达到更高效率，考虑用 BufferedReader 封装 InputStreamReader ，

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

例如： // 实现从键盘输入一个整数

**[java]** [view plain](http://blog.csdn.net/ilibaba/article/details/3955799" \o "view plain" \t "http://blog.csdn.net/jiangwei0910410003/article/details/_blank)[copy](http://blog.csdn.net/ilibaba/article/details/3955799" \o "copy" \t "http://blog.csdn.net/jiangwei0910410003/article/details/_blank)

1. String s = **null**;
2. InputStreamReader re = **new** InputStreamReader(System.in);
3. BufferedReader br = **new** BufferedReader(re);
4. **try** {
5. s = br.readLine();
6. System.out.println("s= " + Integer.parseInt(s));
7. br.close();
8. }
9. **catch** (IOException e)
10. {
11. e.printStackTrace();
12. }
13. **catch** (NumberFormatException e)// 当应用程序试图将字符串转换成一种数值类型，但该字符串不能转换为适当格式时，抛出该异常。
14. {
15. System.out.println(" 输入的不是数字 ");
16. }

InputStreamReader(InputStream) 用缺省的字符编码方式，创建一个 InputStreamReader

InputStreamReader(InputStream, String) 用已命名的字符编码方式，创建一个 InputStreamReader

OutputStreamWriter 将多个字符写入到一个输出流，根据指定的字符编码将多个字符转换为字节。

每个 OutputStreamWriter 合并它自己的 CharToByteConverter, 因而是从字符流到字节流的桥梁。

# ****（三）Java IO 的一般使用原则 ：****

## 一、按数据来源（去向）分类：

1 、是文件： FileInputStream, FileOutputStream, ( 字节流 )FileReader, FileWriter( 字符 )

2 、是 byte[] ： ByteArrayInputStream, ByteArrayOutputStream( 字节流 )

3 、是 Char[]: CharArrayReader, CharArrayWriter( 字符流 )

4 、是 String: StringBufferInputStream, StringBufferOuputStream ( 字节流 )StringReader, StringWriter( 字符流 )

5 、网络数据流： InputStream, OutputStream,( 字节流 ) Reader, Writer( 字符流 )

## 二、按是否格式化输出分：

1 、要格式化输出： PrintStream, PrintWriter

## 三、按是否要缓冲分：

1 、要缓冲： BufferedInputStream, BufferedOutputStream,( 字节流 ) BufferedReader, BufferedWriter( 字符流 )

## ****四、按数据格式分：****

1 、二进制格式（只要不能确定是纯文本的） : InputStream, OutputStream 及其所有带 Stream 结束的子类

2 、纯文本格式（含纯英文与汉字或其他编码方式）； Reader, Writer 及其所有带 Reader, Writer 的子类

## ****五、按输入输出分：****

1 、输入： Reader, InputStream 类型的子类

2 、输出： Writer, OutputStream 类型的子类

## ****六、特殊需要：****

1 、从 Stream 到 Reader,Writer 的转换类： InputStreamReader, OutputStreamWriter

2 、对象输入输出： ObjectInputStream, ObjectOutputStream

3 、进程间通信： PipeInputStream, PipeOutputStream, PipeReader, PipeWriter

4 、合并输入： SequenceInputStream

5 、更特殊的需要： PushbackInputStream, PushbackReader, LineNumberInputStream, LineNumberReader

决定使用哪个类以及它的构造进程的一般准则如下（不考虑特殊需要）：

首先，考虑最原始的数据格式是什么： 原则四

第二，是输入还是输出：原则五

第三，是否需要转换流：原则六第 1 点

第四，数据来源（去向）是什么：原则一

第五，是否要缓冲：原则三 （特别注明：一定要注意的是 readLine() 是否有定义，有什么比 read, write 更特殊的输入或输出方法）

第六，是否要格式化输出：原则二