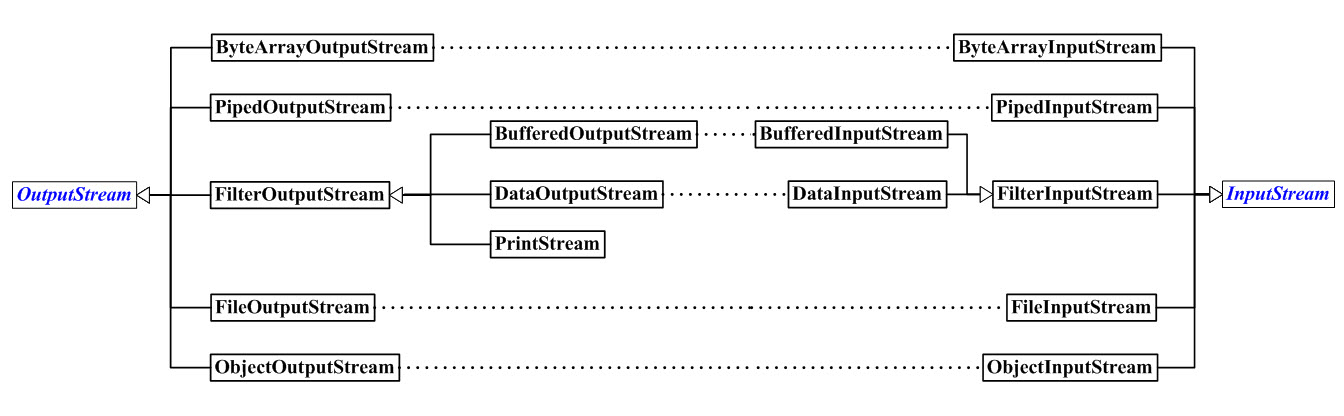
|  |  |  |
| --- | --- | --- |
| 类 | 特性 | 常用方法 |
| ByteArrayInputStream | 内部缓冲区就是一个字节数组，用字节数组保存数据 | 用字节数组初始化  read(byte[] buffer, int offset, int length)  mark(int readlimit) 标记“字节流中下一个被读取的位置” |
| ByteArrayOutputStream | 内部缓冲区就是一个字节数组，用字节数组保存数据 | ByteArrayOutputStream(int size)  write(byte[] buffer, int offset, int len)  toByteArray() 和 toString() 获取数据 |
| ObjectOutputStream | 写入对象 | 用OutputStream(file)初始化  writeObject(…)  writerInt(..)  … |
| ObjectInputStream | 读取对象 | readObject(…)  ….. |
| PipedOutputStream | 必须相互connect，写入(write(…))实际是调用相互连接的 PipedInputStream的接收方法(receive(…))写入到缓冲字节数组中 |  |
| PipedInputStream | 内部有一个缓冲字节数组 |  |
| BufferedOutputStream |  |  |
| BufferedInputStream | 作用是为另一个输入流添加一些功能，例如，提供“缓冲功能”（**会将对应的输入流的数据分批的填入到缓冲区即内部有一个缓冲数组，可以指定大小**）以及支持“mark()标记”和“reset()重置方法” |  |
| PrintStream | PrintStream 提供了自动flush 和 字符集设置功能（System.out.println out是PrintStream对象 | print(…)和println(…)(实质：将其中参数转换成字符串之后，再写入到输入流) |
| FileDescriptor.out就是机器的“标准输出(屏幕)”的文件标识符。我们可以通俗的将文件标识符就理解为，FileDescriptor.out就是代表的“标准输出” | | |



|  |  |  |
| --- | --- | --- |
| CharArrayReader | 实际上是通过内部的“字符数组”去保存数据 | 用一个字符数组来初始化 |
| CharArrayWriter | 实际上是将数据写入到内部的“字符数组”中去 | Write(…)  toCharArray() |
| InputStreamReader | 把字节输入流转成字符输入流 | 用一个字节输入流初始化 |
| OutputStreamWriter | 把字节输出流转成字符输出流 | 用一个字节输出流初始化 |
|  |  |  |
|  |  |  |

**所谓“流”，就是一种抽象的数据的总称，它的本质是能够进行传输。**

## Serializable

用到序列化的场合：

a）当你想把的内存中的对象状态保存到一个文件中或者数据库中时候；

b）当你想用套接字在网络上传送对象的时候；

c）当你想通过RMI传输对象的时候。

“序列化/反序列化，是专门用于的保存/恢复对象状态的机制”。

即仅支持保存/恢复类的成员变量，但不支持保存类的成员方法！

序列化对static和transient变量，是不会自动进行状态保存的

“序列化不会自动保存static和transient变量”，因此我们若要保存它们，则需要通过writeObject()和readObject()去手动读写

对于Socket, Thread类，不支持序列化。

序列化的认识已经比较深入了：

即知道了“序列化的作用和用法”，也知道了“基本类型”、“java自带的支持Serializable接口的类”和“自定义实现Serializable接口的类”都能支持序列化。

## Externalizable

(01) 实现Externalizable接口的类，不会像实现Serializable接口那样，会自动将数据保存。

(02) 实现Externalizable接口的类，必须实现writeExternal()和readExternal()接口！

否则，程序无法正常编译！

(03) 实现Externalizable接口的类，必须定义不带参数的构造函数！

否则，程序无法正常编译！

(04) writeExternal() 和 readExternal() 的方法都是public的，不是非常安全！

## FileWriter

|  |  |
| --- | --- |
| 方法 | 功能 |
| FileWriter fileWriter=new FileWriter(file) | 初始化 |
| fileWriter.write(str||char[]) | 覆写 |
| fileWriter.append(c) | 追加 |

## BufferedReader

|  |  |
| --- | --- |
| 方法 | 功能 |
| BufferedReader bufferedReader=new BufferedReader(fileWriter) | 初始化 |
| bufferedReader.readLine() | 读取一行 |

## File

|  |  |
| --- | --- |
| 方法 | 功能 |
| isDirectory() | 判断是否是目录 |
| isFile() | 判断是否是普通文件 |
| listFiles() | 返回File对象下所有的File对象所构成的数组 |
| getPath() | 获取字符型目录 |
| getName() | 获取目录名或文件名 |
| mkdirs() | 创建目录 |
| length() | 为0代表文件内容为空 |

当当前目录不存在时无法createNewFIle()

## BUfferedInputStream

private void fill() throws IOException {

byte[] buffer = getBufIfOpen(); 新建一个指向缓冲数组的引用

if (markpos < 0) 没有标记(用这个做条件是因为如果有标记就需要保留部分数据)

pos = 0;

else if (pos >= buffer.length) { 是否读完

if (markpos > 0) { 需要保留markpos到当前位置的数据

int sz = pos - markpos; 计算需要保存的长度

System.arraycopy(buffer, markpos, buffer, 0, sz);

pos = sz;

markpos = 0; 置零

} else if (buffer.length >= marklimit) {

markpos = -1; /\* buffer got too big, invalidate mark \*/

pos = 0; /\* drop buffer contents \*/

} else { /\* grow buffer \*/

int nsz = pos \* 2;

if (nsz > marklimit)

nsz = marklimit;

byte nbuf[] = new byte[nsz];

System.arraycopy(buffer, 0, nbuf, 0, pos);

if (!bufUpdater.compareAndSet(this, buffer, nbuf)) {

// Can't replace buf if there was an async close.

// Note: This would need to be changed if fill()

// is ever made accessible to multiple threads.

// But for now, the only way CAS can fail is via close.

// assert buf == null;

throw new IOException("Stream closed");

}

buffer = nbuf;

}

}

count = pos;

int n = getInIfOpen().read(buffer, pos, buffer.length - pos);

if (n > 0)

count = n + pos;

}