# java中有几种类型的流？JDK为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？

字节流，字符流。字节流继承于InputStream OutputStream，字符流继承于InputStreamReader OutputStreamWriter。在java.io包中还有许多其他的流，主要是为了提高性能和使用方便。

# 字节流与字符流的区别

要把一片二进制数据数据逐一输出到某个设备中，或者从某个设备中逐一读取一片二进制数据，不管输入输出设备是什么，我们要用统一的方式来完成这些操作，用一种抽象的方式进行描述，这个抽象描述方式起名为IO流，对应的抽象类为OutputStream和InputStream ，不同的实现类就代表不同的输入和输出设备，它们都是针对字节进行操作的。

在应用中，经常要完全是字符的一段文本输出去或读进来，用字节流可以吗？计算机中的一切最终都是二进制的字节形式存在。对于“中国”这些字符，首先要得到其对应的字节，然后将字节写入到输出流。读取时，首先读到的是字节，可是我们要把它显示为字符，我们需要将字节转换成字符。由于这样的需求很广泛，人家专门提供了字符流的包装类。

底层设备永远只接受字节数据，有时候要写字符串到底层设备，需要将字符串转成字节再进行写入。字符流是字节流的包装，字符流则是直接接受字符串，它内部将串转成字节，再写入底层设备，这为我们向IO设别写入或读取字符串提供了一点点方便。

字符向字节转换时，要注意编码的问题，因为字符串转成字节数组，

其实是转成该字符的某种编码的字节形式，读取也是反之的道理。

讲解字节流与字符流关系的代码案例：

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.InputStreamReader;

import java.io.PrintWriter;

public class IOTest {

public static void main(String[] args) throws Exception {

String str = "中国人";

/\*FileOutputStream fos = new FileOutputStream("1.txt");

fos.write(str.getBytes("UTF-8"));

fos.close();\*/

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

fw.write(str);

fw.close();\*/

PrintWriter pw = new PrintWriter("1.txt","utf-8");

pw.write(str);

pw.close();

/\*FileReader fr = new FileReader("1.txt");

char[] buf = new char[1024];

int len = fr.read(buf);

String myStr = new String(buf,0,len);

System.out.println(myStr);\*/

/\*FileInputStream fr = new FileInputStream("1.txt");

byte[] buf = new byte[1024];

int len = fr.read(buf);

String myStr = new String(buf,0,len,"UTF-8");

System.out.println(myStr);\*/

BufferedReader br = new BufferedReader(

new InputStreamReader(

new FileInputStream("1.txt"),"UTF-8"

)

);

String myStr = br.readLine();

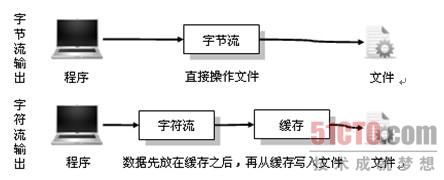
br.close();

System.out.println(myStr);

}

}

字节流在操作时本身不会用到缓冲区（内存），是文件本身直接操作的，而字符流在操作时使用了缓冲区，通过缓冲区再操作文件。



范例：使用字节流不关闭

1. **import** java.io.File;
2. **import** java.io.FileOutputStream;
3. **import** java.io.OutputStream;
4. **public** **class** OutputStreamDemo05 {
5. **public** **static** **void** main(String[] args) **throws** Exception {   // 异常抛出，  不处理
6. // 第1步：使用File类找到一个文件
7. File f = **new** File("d:" + File.separator + "test.txt"); // 声明File  对象
8. // 第2步：通过子类实例化父类对象
9. OutputStream out = **null**;
10. // 准备好一个输出的对象
11. out = **new** FileOutputStream(f);
12. // 通过对象多态性进行实例化
13. // 第3步：进行写操作
14. String str = "Hello World!!!";
15. // 准备一个字符串
16. **byte** b[] = str.getBytes();
17. // 字符串转byte数组
18. out.write(b);
19. // 将内容输出
20. // 第4步：关闭输出流
21. // out.close();
22. // 此时没有关闭
23. }
24. }

序运行结果：  
  
此时没有关闭字节流操作，但是文件中也依然存在了输出的内容，证明字节流是直接操作文件本身的。而下面继续使用字符流完成，再观察效果。  
  
范例：使用字符流不关闭

1. **import** java.io.File;
2. **import** java.io.FileWriter;
3. **import** java.io.Writer;
4. **public** **class** WriterDemo03 {
5. **public** **static** **void** main(String[] args) **throws** Exception { // 异常抛出，  不处理
6. // 第1步：使用File类找到一个文件
7. File f = **new** File("d:" + File.separator + "test.txt");// 声明File 对象
8. // 第2步：通过子类实例化父类对象
9. Writer out = **null**;
10. // 准备好一个输出的对象
11. out = **new** FileWriter(f);
12. // 通过对象多态性进行实例化
13. // 第3步：进行写操作
14. String str = "Hello World!!!";
15. // 准备一个字符串
16. out.write(str);
17. // 将内容输出
18. // 第4步：关闭输出流
19. // out.close();
20. // 此时没有关闭
21. }
22. }

程序运行结果：  
  
  
  
  
程序运行后会发现文件中没有任何内容，这是因为字符流操作时使用了缓冲区，而在关闭字符流时会强制性地将缓冲区中的内容进行输出，但是如果程序没有关闭，则缓冲区中的内容是无法输出的，所以得出结论：字符流使用了缓冲区，而字节流没有使用缓冲区。

# 缓冲区

缓冲区可以简单地理解为一段内存区域。

某些情况下，如果一个程序频繁地操作一个资源（如文件或数据库），则性能会很低，此时为了提升性能，就可以将一部分数据暂时读入到内存的一块区域之中，以后直接从此区域中读取数据即可，因为读取内存速度会比较快，这样可以提升程序的性能。  
在字符流的操作中，所有的字符都是在内存中形成的，在输出前会将所有的内容暂时保存在内存之中，所以使用了缓冲区暂存数据。  
如果想在不关闭时也可以将字符流的内容全部输出，则可以使用Writer类中的flush()方法完成。

范例：强制性清空缓冲区

1. **import** java.io.File;
2. **import** java.io.FileWriter;
3. **import** java.io.Writer;
4. **public** **class** WriterDemo04 {
5. **public** **static** **void** main(String[] args) **throws** Exception { // 异常抛出不处理
6. // 第1步：使用File类找到一个文件
7. File f = **new** File("d:" + File.separator + "test.txt");// 声明File
8. 对象
9. // 第2步：通过子类实例化父类对象
10. Writer out = **null**;
11. // 准备好一个输出的对象
12. out = **new** FileWriter(f);
13. // 通过对象多态性进行实例化
14. // 第3步：进行写操作
15. String str = "Hello World!!!";
16. // 准备一个字符串
17. out.write(str);
18. // 将内容输出
19. out.flush();
20. // 强制性清空缓冲区中的内容
21. // 第4步：关闭输出流
22. // out.close();
23. // 此时没有关闭
24. }
25. }

程序运行结果：  
  
此时，文件中已经存在了内容，更进一步证明内容是保存在缓冲区的。

# IO流操作中创建的流对象为什么一定close?

一个对象在没有被引用变量指向的时候它会变成垃圾，最终会被垃圾回收器从内存中清除，  
对于我们创建的流对象，干嘛还要

"调用close方法将其进行关闭呢，以释放与其相关的资源"呢？

这里我们需要注意了，当我们在程序中创建一个IO流对象的时候，同时系统也会创建  
一个叫做流的东西，在这种情况下，计算机内存中实际产生了两个事物，一个是java程  
序中类的实例对象，一个是系统本身产生的某种资源，而java垃圾回收器只能管理程序  
中类的实例对象，没办法去管理系统产生的资源，所以程序需要调用close方法，去通  
知系统释放其自身产生的资源。

# IO流操作中，如何区分输入流和输出流呢？

我们总是认为输入流就是往文件中写入数据，输出流是从文件中读取数据。其实这种想法是错误的。

因为，我们是站在我们自己的立场去看这个输入和输出问题的，而我们现在是让程序在做输入  
和输出这件事情的，所以，我们应该站在程序的角度来看输入输出问题，对程序而言，它的输  
入，其实就是从某个文件读取数据，它的输出才是真正的向某个文件写入内容。这一点和我们  
的正常思维好像不一样，其实是我们应该转换角色来考虑这个问题。最终的结论是：IO流的输  
入输出我们应该站在程序的角度来看，也可以想象成和我们的正常想法正好相反就可以了。呵呵。

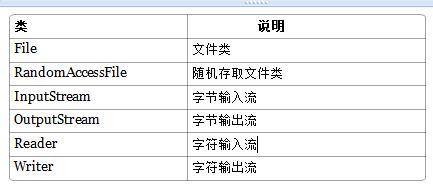
# 输入/输出

输入/输出(Input/Output)泛指对某个设备或环境进行数据的输入输出。

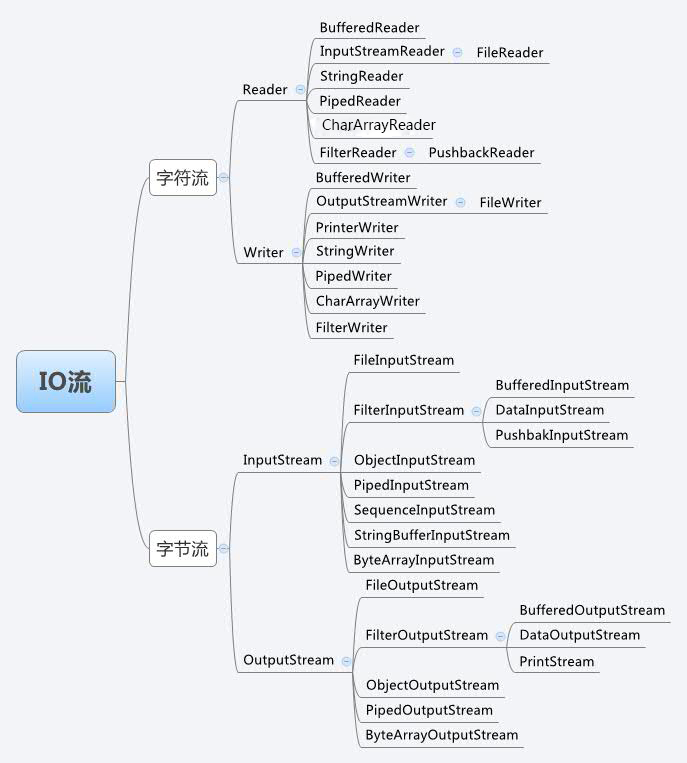
如对硬盘、视频设备、网络主机进行输入/输出。

对于输入/输出问题，Java将之抽象化为流（Stream）对象来解决。

Java流操作有关的类或接口：



java流类图结构：



## ****流的概念和作用****

流是一组有顺序的，有起点和终点的字节集合，是对数据传输的总称或抽象。即数据在两设备间的传输称为流，**流的本质是数据传输，根据数据传输特性将流抽象为各种类，方便更直观的进行数据操作。**

## ****IO流的分类****

* 根据处理数据类型的不同分为：字符流和字节流
* 根据数据流向不同分为：输入流和输出流

### 字符流和字节流

字符流的由来： 因为数据编码的不同，而有了对字符进行高效操作的流对象。本质其实就是基于字节流读取时，去查了指定的码表。 字节流和字符流的区别：

* 读写单位不同：字节流以字节（8bit）为单位，字符流以字符为单位，根据码表映射字符，一次可能读多个字节。
* 处理对象不同：字节流能处理所有类型的数据（如图片、avi等），而字符流只能处理字符类型的数据。

结论：只要是处理纯文本数据，就优先考虑使用字符流。 除此之外都使用字节流。

### 输入流和输出流

对输入流只能进行读操作，对输出流只能进行写操作，程序中需要根据待传输数据的不同特性而使用不同的流。

## ****Java IO流对象****

### 输入字节流InputStream

IO 中输入字节流的继承图可见上图，可以看出：

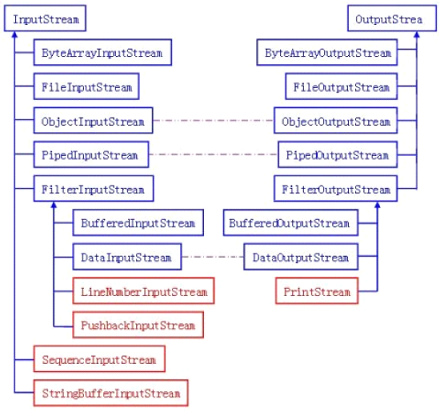
1. InputStream 是所有的输入字节流的父类，它是一个抽象类。
2. ByteArrayInputStream、StringBufferInputStream、FileInputStream 是三种基本的介质流，它们分别从Byte 数组、StringBuffer、和本地文件中读取数据。PipedInputStream 是从与其它线程共用的管道中读取数据，与Piped 相关的知识后续单独介绍。
3. ObjectInputStream 和所有FilterInputStream 的子类都是装饰流（装饰器模式的主角）。

### 2.输出字节流OutputStream

IO 中输出字节流的继承图可见上图，可以看出：

1. OutputStream 是所有的输出字节流的父类，它是一个抽象类。
2. ByteArrayOutputStream、FileOutputStream 是两种基本的介质流，它们分别向Byte 数组、和本地文件中写入数据。PipedOutputStream 是向与其它线程共用的管道中写入数据，
3. ObjectOutputStream 和所有FilterOutputStream 的子类都是装饰流。

### 3.字节流的输入与输出的对应



图中蓝色的为主要的对应部分，红色的部分就是不对应部分。紫色的虚线部分代表这些流一般要搭配使用。从上面的图中可以看出Java IO 中的字节流是极其对称的。“存在及合理”我们看看这些字节流中不太对称的几个类吧！

1. LineNumberInputStream 主要完成从流中读取数据时，会得到相应的行号，至于什么时候分行、在哪里分行是由改类主动确定的，并不是在原始中有这样一个行号。在输出部分没有对应的部分，我们完全可以自己建立一个LineNumberOutputStream，在最初写入时会有一个基准的行号，以后每次遇到换行时会在下一行添加一个行号，看起来也是可以的。好像更不入流了。
2. PushbackInputStream 的功能是查看最后一个字节，不满意就放入缓冲区。主要用在编译器的语法、词法分析部分。输出部分的BufferedOutputStream 几乎实现相近的功能。
3. StringBufferInputStream 已经被Deprecated，本身就不应该出现在InputStream 部分，主要因为String 应该属于字符流的范围。已经被废弃了，当然输出部分也没有必要需要它了！还允许它存在只是为了保持版本的向下兼容而已。
4. SequenceInputStream 可以认为是一个工具类，将两个或者多个输入流当成一个输入流依次读取。完全可以从IO 包中去除，还完全不影响IO 包的结构，却让其更“纯洁”――纯洁的Decorator 模式。
5. PrintStream 也可以认为是一个辅助工具。主要可以向其他输出流，或者FileInputStream 写入数据，本身内部实现还是带缓冲的。本质上是对其它流的综合运用的一个工具而已。一样可以踢出IO 包！System.out 和System.out 就是PrintStream 的实例！

### 4.字符输入流Reader

在上面的继承关系图中可以看出：

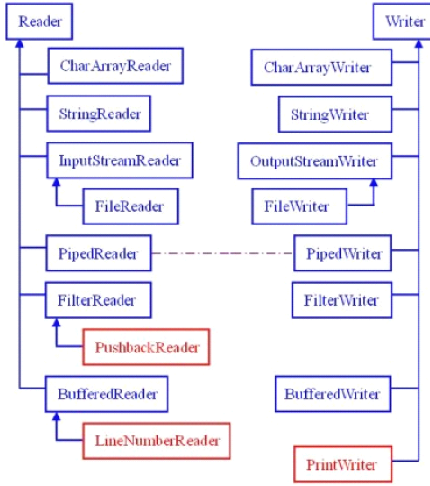
1. Reader 是所有的输入字符流的父类，它是一个抽象类。
2. CharReader、StringReader 是两种基本的介质流，它们分别将Char 数组、String中读取数据。PipedReader 是从与其它线程共用的管道中读取数据。
3. BufferedReader 很明显就是一个装饰器，它和其子类负责装饰其它Reader 对象。
4. FilterReader 是所有自定义具体装饰流的父类，其子类PushbackReader 对Reader 对象进行装饰，会增加一个行号。
5. InputStreamReader 是一个连接字节流和字符流的桥梁，它将字节流转变为字符流。FileReader 可以说是一个达到此功能、常用的工具类，在其源代码中明显使用了将FileInputStream 转变为Reader 的方法。我们可以从这个类中得到一定的技巧。Reader 中各个类的用途和使用方法基本和InputStream 中的类使用一致。后面会有Reader 与InputStream 的对应关系。

### 5.字符输出流Writer

在上面的关系图中可以看出：

1. Writer 是所有的输出字符流的父类，它是一个抽象类。
2. CharArrayWriter、StringWriter 是两种基本的介质流，它们分别向Char 数组、String 中写入数据。PipedWriter 是向与其它线程共用的管道中写入数据，
3. BufferedWriter 是一个装饰器为Writer 提供缓冲功能。
4. PrintWriter 和PrintStream 极其类似，功能和使用也非常相似。
5. OutputStreamWriter 是OutputStream 到Writer 转换的桥梁，它的子类FileWriter 其实就是一个实现此功能的具体类（具体可以研究一SourceCode）。功能和使用和OutputStream 极其类似，后面会有它们的对应图。

### 6.字符流的输入与输出的对应



### 7.字符流与字节流转换

* 从字节流到字符流：InputStreamReader、OutputStreamWriter类可以实现。
* 从字符流到字节流：可以从字符流中获取char[]数组，转换为String，然后调用String的API函数getBytes() 获取到byte[]，然后就可以通过ByteArrayInputStream、ByteArrayOutputStream来实现到字节流的转换。

### 8.File类

File类是对文件系统中文件以及文件夹进行封装的对象，可以通过对象的思想来操作文件和文件夹。 File类保存文件或目录的各种元数据信息，包括文件名、文件长度、最后修改时间、是否可读、获取当前文件的路径名，判断指定文件是否存在、获得当前目录中的文件列表，创建、删除文件和目录等方法。

### 9.RandomAccessFile类

该对象并不是流体系中的一员，其封装了字节流，同时还封装了一个缓冲区（字符数组），通过内部的指针来操作字符数组中的数据。 该对象特点：

1. 该对象只能操作文件，所以构造函数接收两种类型的参数：a.字符串文件路径；b.File对象。
2. 该对象既可以对文件进行读操作，也能进行写操作，在进行对象实例化时可指定操作模式(r,rw)

**注意：该对象在实例化时，如果要操作的文件不存在，会自动创建；如果文件存在，写数据未指定位置，会从头开始写，即覆盖原有的内容。** 可以用于多线程下载或多个线程同时写数据到文件。