IO输入输出

> java.io

IO即输入输出，input指从外部（如磁盘）读入数据到内存，以java提供的某种数据类型储存，比如byte,string,; ouput指从内存输出到外部。只有把数据读入内存才能被处理，因为代码存储在内存中，数据也必须读到内存里。

IO流是一种顺序读写数据的模式，单向流动。数据类似于水在水管中流动，所以称为IO流。

一、File文件处理

输入输出离不开文件的读写处理。

（一）File类

> java.io.File

1.构造File对象

构造方法：

```
File(String pathname)
        通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例。
```

例如：

实例File类的对象：

```
File f = new File("F:\\lan\\git\\java\\4_java_IO输入输出流.md");
```

注意： Windows系统下路径必须用"\ \ "代替"\ "; Linux平台使用"/"作为路径分隔符

由于不同操作系统的差异性，最好使用separator静态变量来表示当前系统的分隔符

```
File f = new File("F:"+File.separator+"test.txt");
```

区分绝对路径、相对路径与规范路径

路径名称	含义	举例	方法
绝对路径	以根目录开头的完整路径	F:\lan\git\java\4_java_IO输入输出流.md	getPath()/getAbsolutePath()

相对路径	相对路径是省去的当前目录的路径	`// 假设当前目录是C:\Docs`
		`File f1 = new File("sub\\javac"); // 绝对路径是C:\Docs\sub\javac`
		`File f3 = new File(".\\sub\\javac"); // 绝对路径是C:\Docs\sub\javac`
		`File f3 = new File("..\\sub\\javac"); // 绝对路径是C:\sub\javac`
		可以用 `.` 表示当前目录，`..` 表示上级目录。	

规范路径	规范路径是把.和..转换成标准的绝对路径后的路径	绝对路径可以表示成 `C:\Windows\System32\..\notepad.exe`，而规范路径就是把 `.` 和 `..` 转换成标准的绝对路径后的路径：`C:\Windows\notepad.exe`。	getCanonicalPath()注意：这个方法进行了异常处理，使用时必须加try..catch

（二）类的方法

File对象即可以表示文件，也可以表示目录。构造一个`File`对象，即使传入的文件或目录不存在，代码也不会出错，因为构造一个`File`对象，并不会导致任何磁盘操作。只有当我们调用`File`对象的某些方法的时候，才真正进行磁盘操作。

判断是文件还是目录的方法：

boolean	isDirectory() 测试此抽象路径名表示的文件是否是一个目录。
boolean	isFile() 测试此抽象路径名表示的文件是否是一个标准文件。

1.当File对象表示文件时

f.getName():返回文件名 temp.dat
f.getParent():返回文件所在目录名 data
f.getPath():返回文件路径 data\temp.dat

a. 获取文件权限和大小

boolean	canExecute() 测试应用程序是否可以执行此抽象路径名表示的文件。
boolean	canRead() 测试应用程序是否可以读取此抽象路径名表示的文件。
boolean	canWrite() 测试应用程序是否可以修改此抽象路径名表示的文件。
long	length() 返回由此抽象路径名表示的文件的长度。

b.创建和删除文件

boolean	createNewFile() 当且仅当不存在具有此抽象路径名指定名称的文件时，不可分地创建一个新的空文件。
boolean	delete() 删除此抽象路径名表示的文件或目录。

注意：使用createNewFile()创建新文件时，由于内含关键字throws，所以必须进行异常处理（异常类：IOException）。

创建临时文件与程序运行后删除：

static File	createTempFile(String prefix, String suffix) 在默认临时文件目录中创建一个空文件，使用给定前缀和后缀生成其名称。
void	deleteOnExit() 在虚拟机终止时，请求删除此抽象路径名表示的文件或目录。

```
File f = File.createTempFile("tmp-", ".txt"); // 提供临时文件的前缀和后缀
f.deleteOnExit(); // JVM退出时自动删除，但是程序运行过程中这个文件是存在的
```

c.压缩文件

压缩实现在java.util.zip.*;

• 核心的类GZIPOutputStream插在File和Buffered 之间 GZIPOutputStream

```
import java.io.*;
import java.util.zip.*;

public class GZIPcompress {
  public static void main(String[] args) {
    try {
      BufferedReader in =
        new BufferedReader(
          new FileReader(args[0]));
      BufferedOutputStream out =
        new BufferedOutputStream(
          new GZIPOutputStream(
            new FileOutputStream("test.gz")));
      System.out.println("Writing file");
      int c;
      while((c = in.read()) != -1)
        out.write(c);
      in.close();
      out.close();
      System.out.println("Reading file");
      BufferedReader in2 =
```

```
          new BufferedReader(
            new InputStreamReader(
              new GZIPInputStream(
                new FileInputStream("test.gz"))));
        String s;
        while((s = in2.readLine()) != null)
          System.out.println(s);
      } catch(Exception e) {
        e.printStackTrace();
      }
    }
}
```

2.当File对象表示目录时

目录可以理解成文件夹

a.创建和删除目录

boolean	mkdir() 创建此抽象路径名指定的目录。
boolean	mkdirs() 创建此抽象路径名指定的目录，包括所有必需但不存在的父目录。

- `boolean delete()`：删除当前File对象表示的目录，当前目录必须为空才能删除成功。

b.遍历目录里面的子目录和文件

File[]	listFiles() 返回一个抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件。
File[]	listFiles(FileFilter filter) 返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录。
File[]	listFiles(FilenameFilter filter) 返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录。
static File[]	listRoots() 列出可用的文件系统根。

方法一：使用list()

列出完整的名称，返回一个字符串数组，再把这个字符串数组遍历出来

```
import java.io.*;

public class AllFiles{
    public static void main(String args[]){
        File files=new File(".");//表示当前目录
        String strFiles[]=files.list();
        for(int i=0;i<strFiles.length;i++){
            File files[]=strFiles[i];
            System.out.print(strFiles[i]);
}
```

```
AllFiles.class--(1355)
AllFiles.java--(497)
ALTest.class--(1416)
ALTest.java--(636)
```

方法二：使用listFiles()列出完整的路径，返回一个File对象数组

```
import java.io.*;

public class AllFiles{
    public static void main(String args[]){
        File files=new File(".");//表示当前目录
        File files[]=files.listFiles();
        for(int i=0;i<files.length;i++){
            System.out.print(files[i]);
}
```

```
.\AllFiles.class--(1368)
.\AllFiles.java--(481)
.\ALTest.class--(1416)
.\ALTest.java--(636)
```

（更多的实用方法请看API文件）

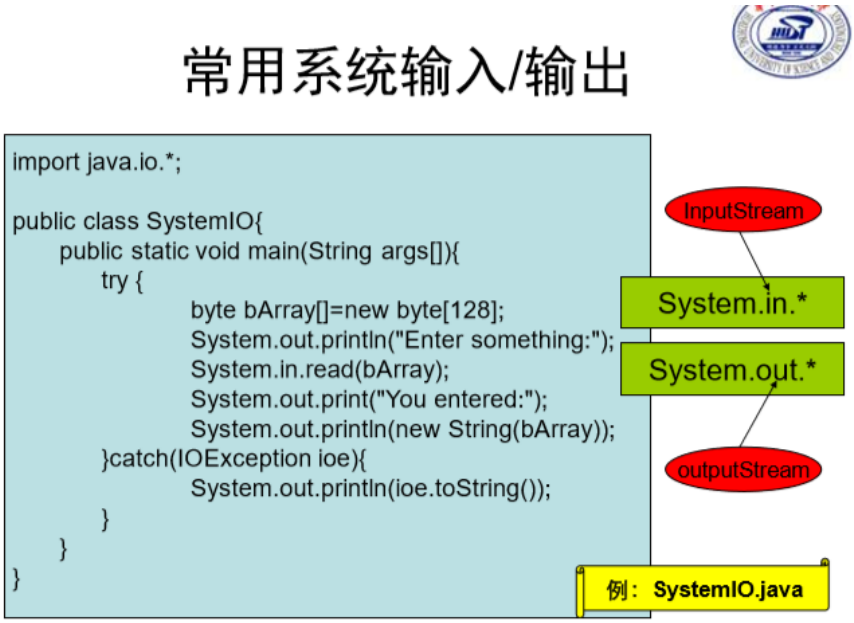
二、字节输入输出流（byte）

InputStream /OuputStream"

inputstream/outputstream都是最大的父类，自身为**抽象类**，使用时必须通过子类实例化对象。以下重点将input，output类比就行惹

（一）自身：InputStream

> java.io.InputStream



- 属于InputStream类的核心方法:
  - int read():读一个整数
  - int read(byte b[]):读多个字节到数组中
  - int read(byte,int off,int len);

- 属于OutputStream类的核心方法:
  - write(int b):将一个整数输出到流中
  - write(byte b[]):将数组中的数据输出到流中
  - write(byte b[], int off,int len):将数组b中从off指定的位置开始len长度的数据输出到流中

在使用后，都需要使用以下方法，关闭数据流，释放资源

void	close() 关闭此输入流并释放与该流关联的所有系统资源。

也可使用以下方法自动关闭资源，记得补充！！！

也可使用以下方法自动关闭资源，记得补充！！！java也提供自动关闭的方法

注意：他们的方法都使用了异常类，所以需要使用try..catch进行异常处理（异常类：IOException）

（二）子类：FileInputStream（文件）

(异常类：FileNotFoundException)

操作时必须接受File类的实例，实现文件的输入输出，输入输出都要指明被操作文件的路径。

1.构造方法

（对于构造方法，重点关注的就是入口参数类型）

FileInputStream(File file) 通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的 File 对象 file 指定。
FileInputStream(String name) 通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的路径名 name 指定。

```
File inFile=new File("file1.txt");
FileInputStream fis=new FileInputStream(inFile);
FileInputStream fis=new FileInputStream("f:"+File.separator+"gui");
```

2.常用方法

int	read() 从此输入流中读取一个数据字节。
int	read(byte[] b) 从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中。
int	read(byte[] b, int off, int len) 从此输入流中将最多 len 个字节的数据读入一个 byte 数组中。
void	close() 关闭此文件输入流并释放与此流有关的所有系统资源。

请见API

示例：

```
import java.io.*;
/*实现把文件1的内容复制到文件2*/
public class FileStream{
    public static void main(String args[]){
        try{
            File inFile=new File("file1.txt");//构造文件对象，当路径不明确时，它会在
java文件所在的目录下找
            File outFile=new File("file2.txt");
            FileInputStream fis=new FileInputStream(inFile);//创建文件流
            FileOutputStream fos=new  FileOutputStream(outFile);
            int c;
            int i=1;
            while((c=fis.read())!=-1){
                fos.write(c);
                //System.out.println("写入Buffer第"+i++);
            }
            fis.close(); //关闭文件流
            fos.close();
        }catch(FileNotFoundException e) {
            System.out.println("FileStreamsTest: "+e);
        }catch(IOException e) {
            System.err.println("FileStreamsTest: "+e);
        }
    }
}
```

实际上，程序中不断操作文件的效率是十分慢的，使用下面的孙子类更好。

（三）孙子类：BufferedInputStream（缓存）

在读取流的时候，一次读取一个字节并不是最高效的方法。很多流支持一次性读取多个字节到缓冲区，对于文件和网络流来说，利用缓冲区一次性读取多个字节效率往往要高很多。为此我们使用Buffer，将文件操作转化为内存操作 **Buffer大小可由人工指定，超过大小可执行 一次写入或读出 。**

1.构造方法

BufferedOutputStream(OutputStream out) 创建一个新的缓冲输出流，以将数据写入指定的底层输出流。
BufferedOutputStream(OutputStream out, int size) 创建一个新的缓冲输出流，以将具有指定缓冲区大小的数据写入指定的底层输出流。

注意，孙子类入口参数都要是数据流

2.常用方法

其实对于io,最重要的方法就是输入输出了。

int	read() 参见 InputStream 的 read 方法的常规协定。
int	read(byte[] b, int off, int len) 从此字节输入流中给定偏移量处开始将各字节读取到指定的 byte 数组中。

示例：

```
import java.io.*;
/*实现把文件1的内容复制到文件2*/
```

```
public class BufferedStream {
    public static void main(String args[]) {
        try {
            File inFile = new File("file1.txt");
            File outFile = new File("file2.txt");
            FileInputStream fis = new FileInputStream(inFile);
            BufferedInputStream bis = new BufferedInputStream(fis);//把家族里面的叔叔放进自己口袋
            BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(outFile), 5);//在缓存区里面操作5个字节即输出到文件里面；如果缺省默认全部操作完再输出
            int c;
            int i = 1;
            while ((c = bis.read()) != -1) {
                bos.write(c);
            }
            bis.close();
            bos.close();
        } catch (FileNotFoundException e) {
            System.out.println("FileStreamsTest: " + e);
        } catch (IOException e) {
            System.err.println("FileStreamsTest: " + e);
        }
    }
}
```

（四）孙子类：DataInputStream（格式化数据）

File和Buffer都是字节流，即一次读写一个 字节（虽然buffer更高效，但是在缓存区还是一个一个字节读取的），而往往我们文件内有高级数据类型， int, float等，采用数据流可直接处理高级数据类型**格式化输入输出**

1.构造方法

```
DataInputStream(InputStream in)
        使用指定的底层 InputStream 创建一个 DataInputStream。
```

2.常用方法

可以指定读写数据的类型

int	read(byte[] b) 从包含的输入流中读取一定数量的字节，并将它们存储到缓冲区数组 b 中。
int	read(byte[] b, int off, int len) 从包含的输入流中将最多 len 个字节读入一个 byte 数组中。
boolean	readBoolean() 参见 DataInput 的 readBoolean 方法的常规协定。
byte	readByte() 参见 DataInput 的 readByte 方法的常规协定。
char	readChar() 参见 DataInput 的 readChar 方法的常规协定。
double	readDouble() 参见 DataInput 的 readDouble 方法的常规协定。
float	readFloat() 参见 DataInput 的 readFloat 方法的常规协定。
void	readFully(byte[] b) 参见 DataInput 的 readFully 方法的常规协定。
void	readFully(byte[] b, int off, int len) 参见 DataInput 的 readFully 方法的常规协定。
int	readInt() 参见 DataInput 的 readInt 方法的常规协定。

```
import java.io.*;

public class DataStream{
    public static void main(String args[])throws  IOException{
        FileOutputStream fos=new FileOutputStream("file2.txt");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        DataOutputStream dos=new DataOutputStream (bos);
        try{
            dos.writeBoolean(true);
            dos.writeByte((byte)123);
            dos.writeChar('J');
            dos.writeDouble(3.141592654);
            dos.writeFloat(2.7182f);
            dos.writeInt(1234567890);
            dos.writeLong(998877665544332211L);
            dos.writeShort((short)11223);
        }catch(Exception e) {
            System.err.println("FileStreamsTest: "+e);
        }
        finally{
            dos.close();
        }
        System.in.read();
        //BufferedInputStream dis = new BufferedInputStream(new
DataInputStream(new FileInputStream("file2.txt")));
        DataInputStream dis=new DataInputStream(new
FileInputStream("file2.txt"));
        try{
            System.out.println("\t "+dis.readBoolean());
            System.out.println("\t "+dis.readByte());
            System.out.println("\t "+dis.readChar());
            System.out.println("\t "+dis.readDouble());
            System.out.println("\t "+dis.readFloat());
            System.out.println("\t "+dis.readInt());
            System.out.println("\t "+dis.readLong());
            System.out.println("\t "+dis.readShort());
```

```
        }finally{
            dis.close();
        }
    }
}
>>>  true
         123
         J
         3.141592654
         2.7182
         1234567890
         998877665544332211
         11223
```

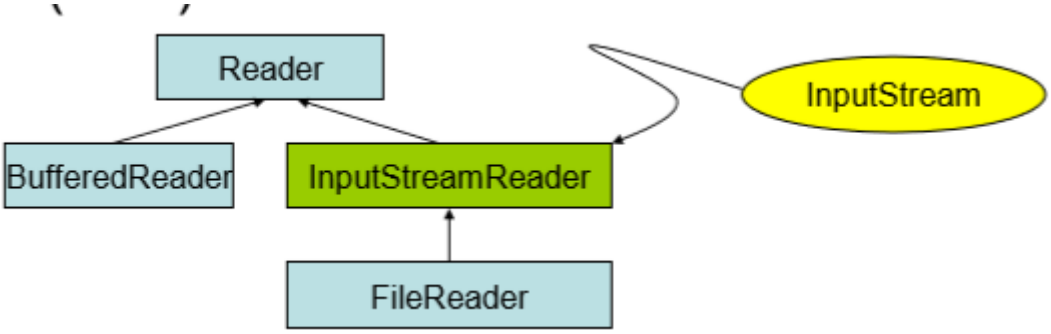
三、字符输入输出流（char）

Reader /Writer"

Reader/Writer本质上是能自动编解码（指定编解码方式）的InputStream和OutputStream。适用于文本的输入输出。

程序中一个字符等于两个字节。这两个类也是抽象类，使用子类实现。其中也暗含异常处理（**异常类：IOException**）

它与字节输入输出流极其类似，故不重复叙述。



主要子类如上，其中也包括文件读写与缓存区读写

常用方法：

int	read(char[] cbuf) 将字符读入数组。
abstract void	close() 关闭该流并释放与之关联的所有资源。

```
import java.io.*;

public class ReaderDiff {
    public static void main(String args[]) {
        try {
            File inFile1 = new File("file1.txt");
            File inFile2 = new File("file3.txt");

            BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(inFile1));
```

```
            // BufferedReader isr = new BufferedReader(new FileReader(inFile2));
            InputStreamReader isr = new InputStreamReader(new
FileInputStream(inFile2), "GBK");
            // InputStreamReader isr = new InputStreamReader(new
FileInputStream(inFile2),
            // "GB2312");
            int c1;
            byte[] bs = new byte[50];
            int c2;
            int count1 = 0;
            int count2 = 0;
            while ((c1 = bis.read()) != -1) {
                // System.out.print(new Byte((byte) c1));
                bs[count1] = (byte) c1;
                count1++;
            }
            System.out.println();
            System.out.println(new String(bs));
            System.out.println("------------------------------");
            while ((c2 = isr.read()) != -1) {
                System.out.print(new Character((char) c2));
                count2++;
            }
            System.out.println();
            System.out.println("------------------------------");
            System.out.println("Stream byte number=" + count1);
            System.out.println("Reader char number=" + count2);
            bis.close();
            isr.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

四、字节字符IO流的区别与联系

（一）区别

字节流在操作的时候本身不会用到缓存区（内存），而是通过直接操作文件的；而字符流操作的时候使用缓存区在通过缓存区操作文件。例如，如果使用字节流写文件，在字节流不关闭时，文件是可以直接显示内容的；而使用字符流时，由于字符流文件未关闭1，没有执行将缓存读到文件中的操作，文件中就看不到写的内容。但是我们可以通过**writer.flush（）**进行强制的缓存刷新，内存内容被强制读取到文件

（二）联系：InputStreamReader

它是Reader的儿子。它是从InputStreeam到Reader的转换器

```
// 持有InputStream:
InputStream input = new FileInputStream("src/readme.txt");
// 变换为Reader:
Reader reader = new InputStreamReader(input, "UTF-8");
```

常用方法：

void	close() 关闭该流并释放与之关联的所有资源。
String	getEncoding() 返回此流使用的字符编码的名称。
int	read() 读取单个字符。

现实使用时使用InputStream类更好，因为所有数据存储在硬盘中都是以字节存储的，字符流底层还是字节流，所以字节流使用更加广泛。

编码：

编码方式	
GB2312	汉字是双字节的。所谓双字节是指一个双字要占用两个BYTE的位 置（即16位），分别称为高位和低位。GB2312包括了一二级汉 字编码范围为0xb0a1到0xf7fe
GBK	GBK提供了20902个汉字，它兼容GB2312，编码范围为0x8140 到0xfefe
ISO8859_1	英文字符

一般默认为GBK或者GB2312

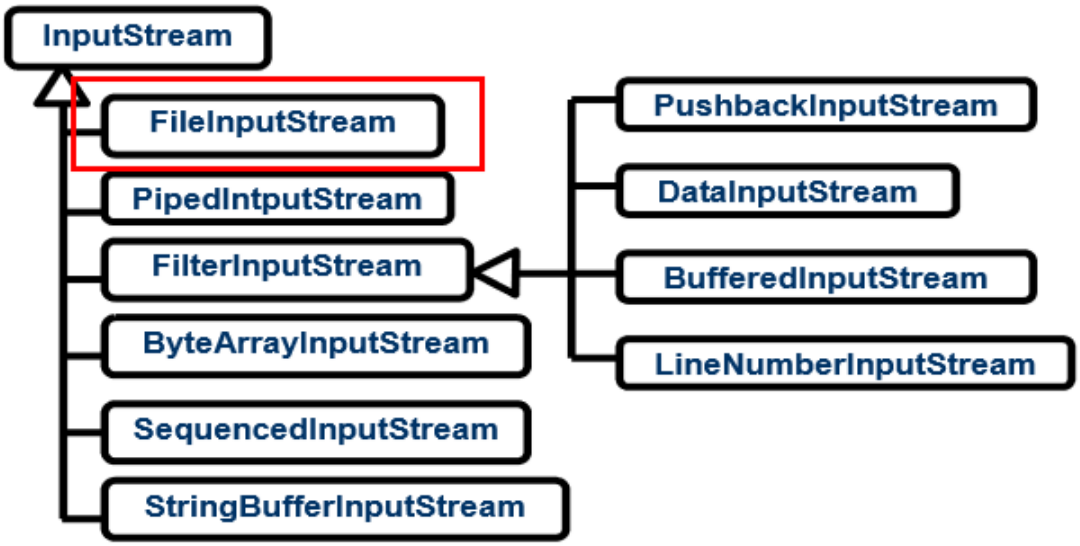
重点！InputStream 的继承关系

```
public abstract class InputStream
extends Object
implements Closeable
```

此抽象类是表示字节输入流的所有类的超类。

需要定义 InputStream 子类的应用程序必须总是提供返回下一个输入字节的方法。

InputStream是最大的爸爸，其继承树关系如下：



特别值得关注的是：

构造方法摘要

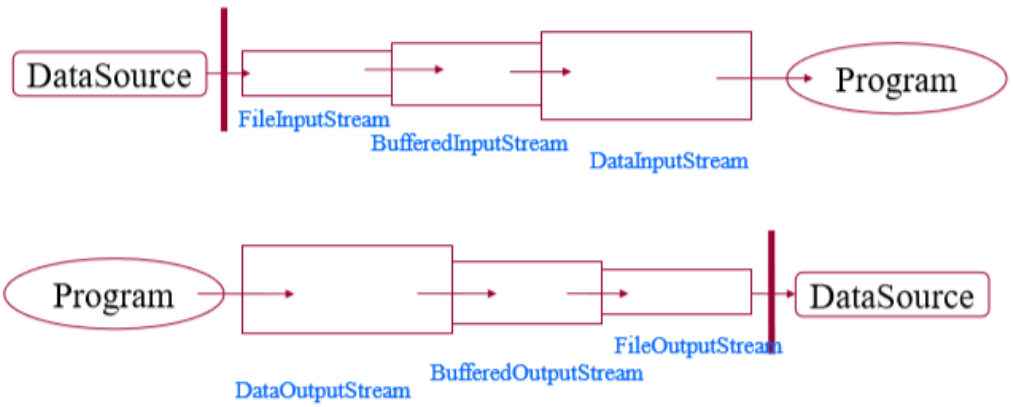
protected FilterInputStream(InputStream in)
将参数 in 分配给字段 this.in，以便记住它供以后使用，通过这种方式创建一个 FilterInputStream。

BufferedInputStream(InputStream in)
创建一个 BufferedInputStream 并保存其参数，即输入流 in，以便将来使用。

DataInputStream(InputStream in)
使用指定的底层 InputStream 创建一个 DataInputStream。

FilterInputStream作为InputStream的第一任儿子，用爸爸的类作为构造参数入口，而BufferedInputStream作为FilterInputStream的儿子，InputStream的孙子居然也是用爷爷的类作为构造函数入口参数类型。而且FilterInputStream的所有儿子都是用爷爷的类构造入口参数的。说明他们之间可以互相套娃。沟通的桥梁就是InputStream这个爷爷。家族所有类都能传入家族下任何一个类里面。（同理，OutputStream这个超类也是这样的家族，而Reader和Writer也有类似的地方）

一个程序很少使用单个的流对象，可以使用多个流对象形成一个链来合作处理数据。



```
FileOutputStream fos=new FileOutputStream("file2.txt");
BufferedOutputStream bos = new BufferedOutputStream(fos);
DataOutputStream dos=new DataOutputStream (bos);
```