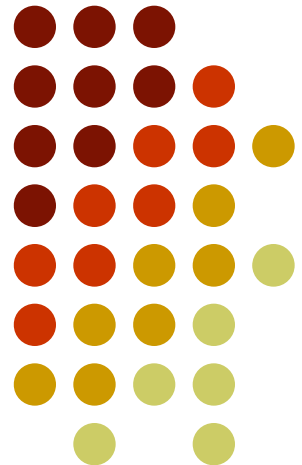




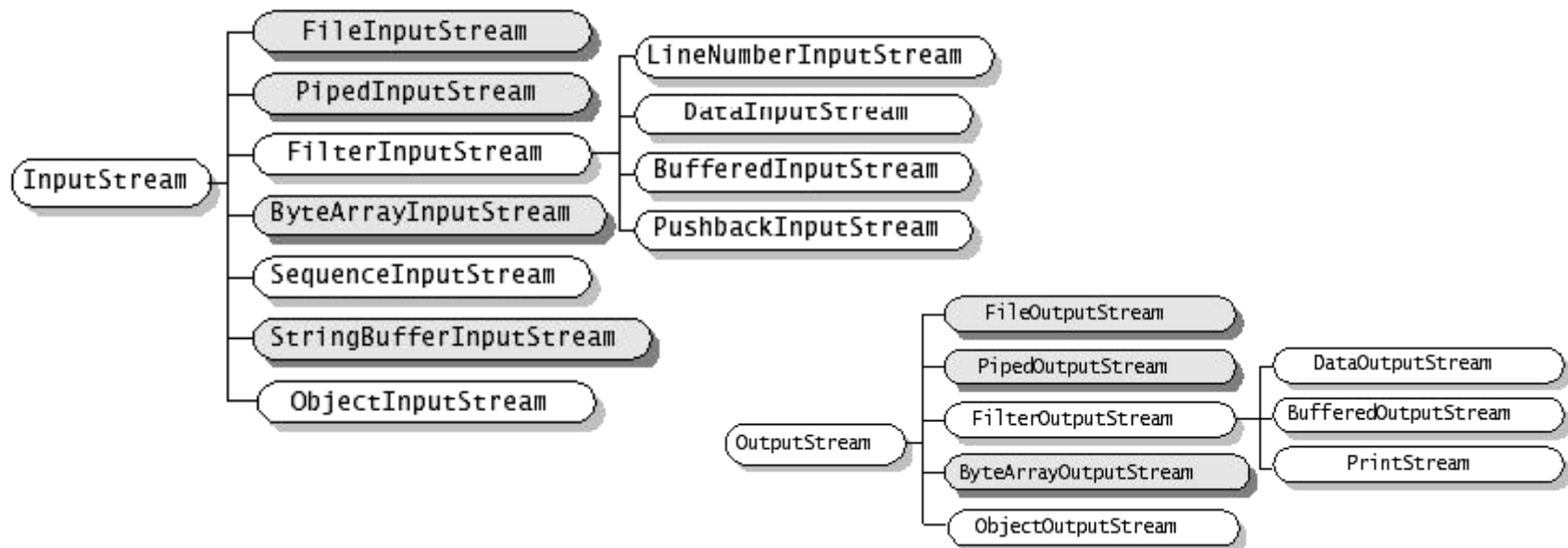
第七章 Java的输入输出

1. Java的基本输入输出
2. I/O流的类层次
3. 标准文件类File
4. 简单I/O流
5. 缓冲I/O流
6. 过滤流
7. 文件的处理
8. *其它的I/O



4种主要的I/O类

	Input	Output
Byte Streams	InputStream	OutputStream
Character Streams	Reader	Writer





7.1 Java的基本输入输出

➤ Basic I/O

- ✓ I/O Streams
- ✓ Byte Streams
- ✓ Character Streams
- ✓ Buffered Streams
- ✓ Scanning and Formatting
- ✓ I/O from the Command Line
- ✓ Data Streams
- ✓ Object Streams

参见:

</docs/api/java/io/package-tree.html>

<file:///D:/JavaEx/tutorial/essential/io/index.html>

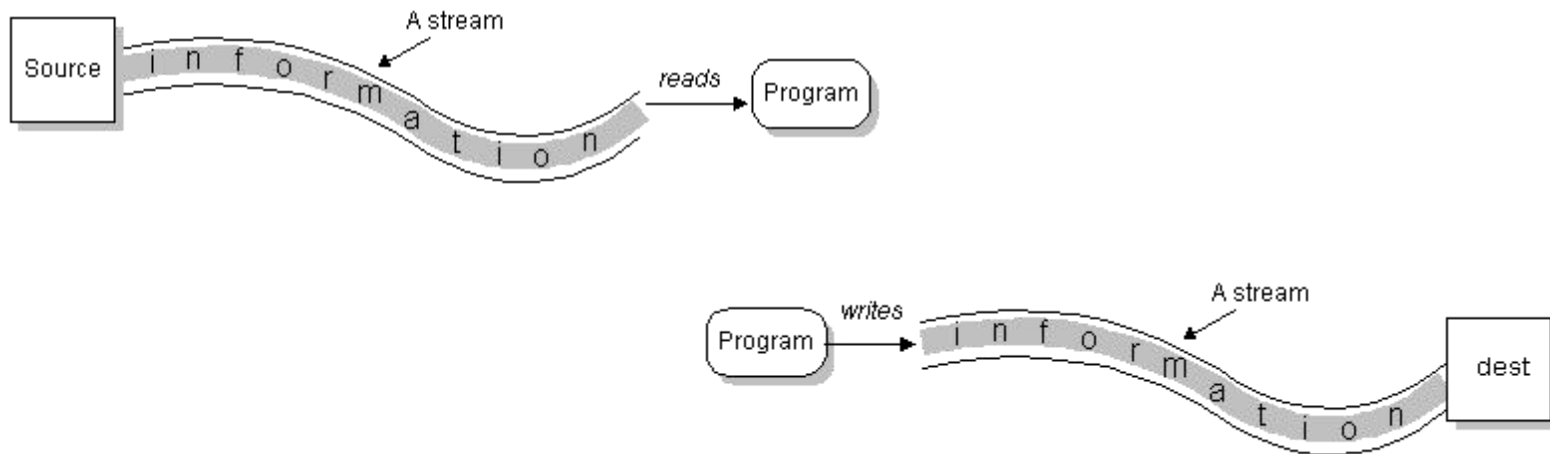
➤ File I/O

- ✓ File Objects
- ✓ Random Access Files

➤ The New I/O Packages

流的概念（Streams）

- 流可以被理解为一个“导管”。这条“导管”有两个端口：一端与数据源（当输入数据时）或数据宿（当输出数据时）相连，另一端与程序相连。
- 只要在程序和数据源/数据宿之间建立了流，用户就不需要再关心数据来自何方或送向何处，程序中输入/输出操作的复杂性就大大降低了。



流的概念（Streams）

➤ 流的类别：

- ✓ 根据流的数据传输方向：分为**输入流**和**输出流**；
- ✓ 根据流的数据的类型：将流分为**字符流（Character Streams）**和**字节流（Byte Streams）**，
- ✓ 根据流的建立方式和工作原理，将流分为**节点流（Node Streams）**与**缓冲流(Filter Streams)**。
 - 节点流是直接建立在输入、输出媒体之上的，而缓冲流必须以某一个节点流作为流的来源，可以在读/写数据的同时对数据进行处理。





7.1 Java的基本输入输出

在Java中，通过java.io包提供的类来表示流，基本的输入输出流为InputStream和OutputStream。从这两个基本的输入输出流派生出面向特定处理的流，如缓冲区读写流、文件读写流等。Java定义的流如表7.1所示。

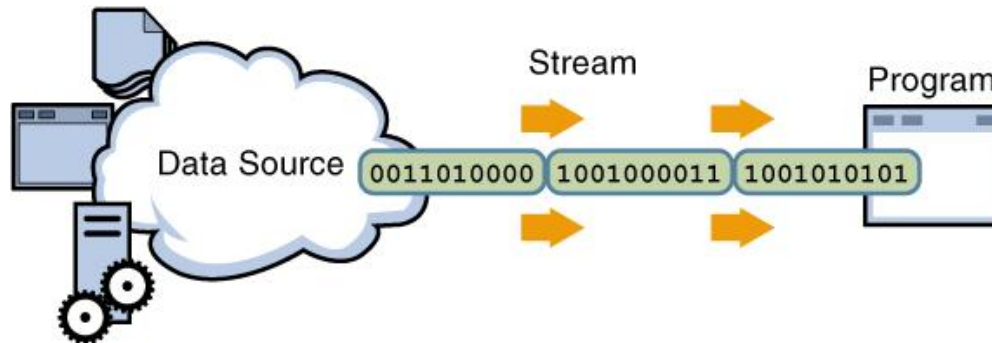
表7.1 Java定义的输入输出流

流 描 述	输 入 流	输 出 流
音频输入输出流	AudioInputStream	AudioOutputStream
字节数组输入输出流	ByteArrayInputStream	ByteArrayOutputStream
文件输入输出流	FileInputStream	FileOutputStream
过滤器输入输出流	FilterInputStream	FilterOutputStream
基本输入输出流	InputStream	OutputStream
对象输入输出流	ObjectInputStream	ObjectOutputStream
管道输入输出流	PipedInputStream	PipedOutputStream
顺序输入输出流	SequenceInputStream	SequenceOutputStream
字符缓冲输入输出流	StringBufferInputStream	StringBufferOutputStream

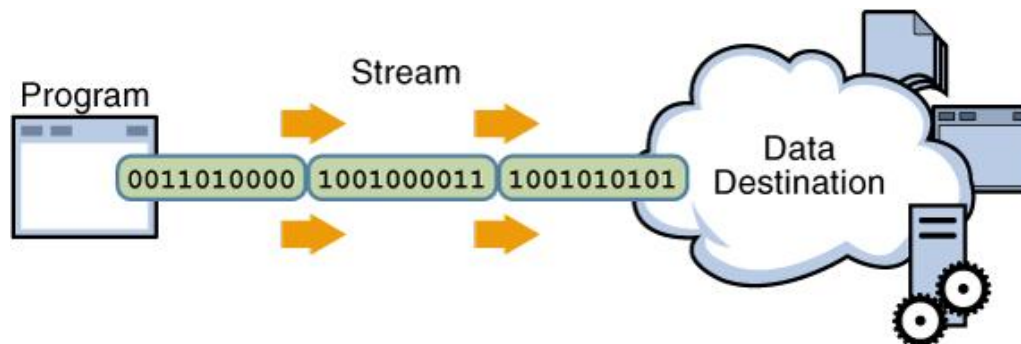
7.2 I/O Streams

- An *I/O Stream* represents an input source or an output destination.

A program uses an input stream to read data from a source

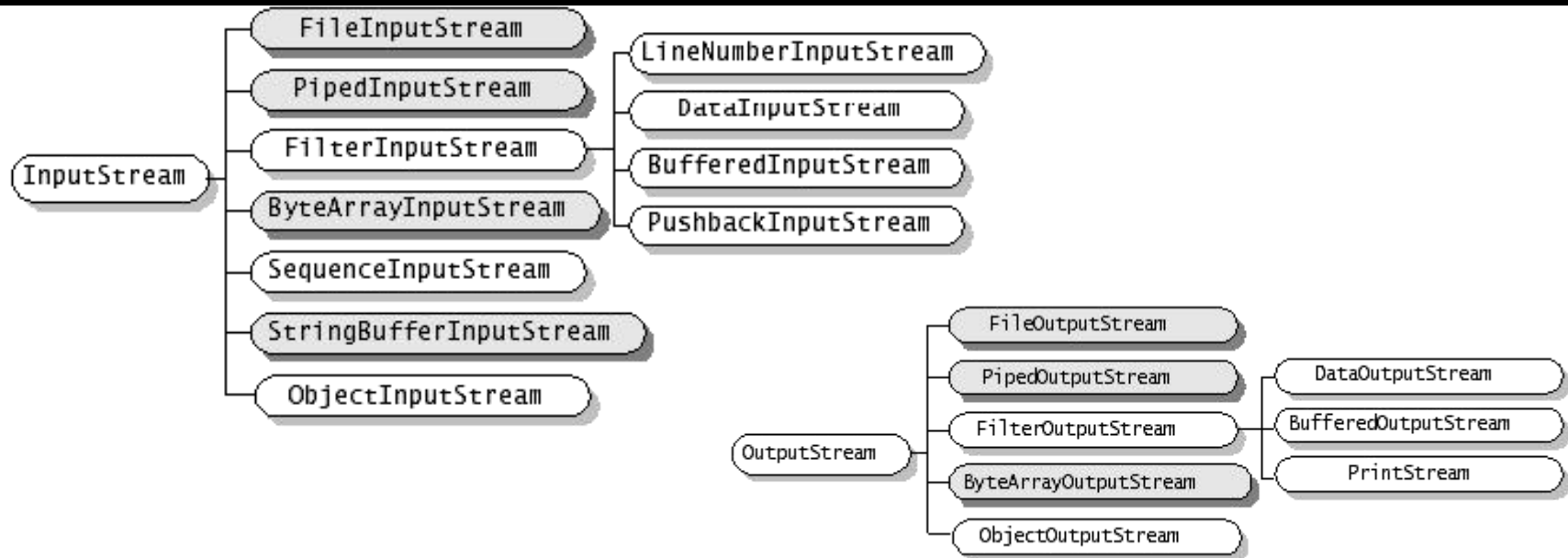


A program uses an *output stream* to write data to a destination,



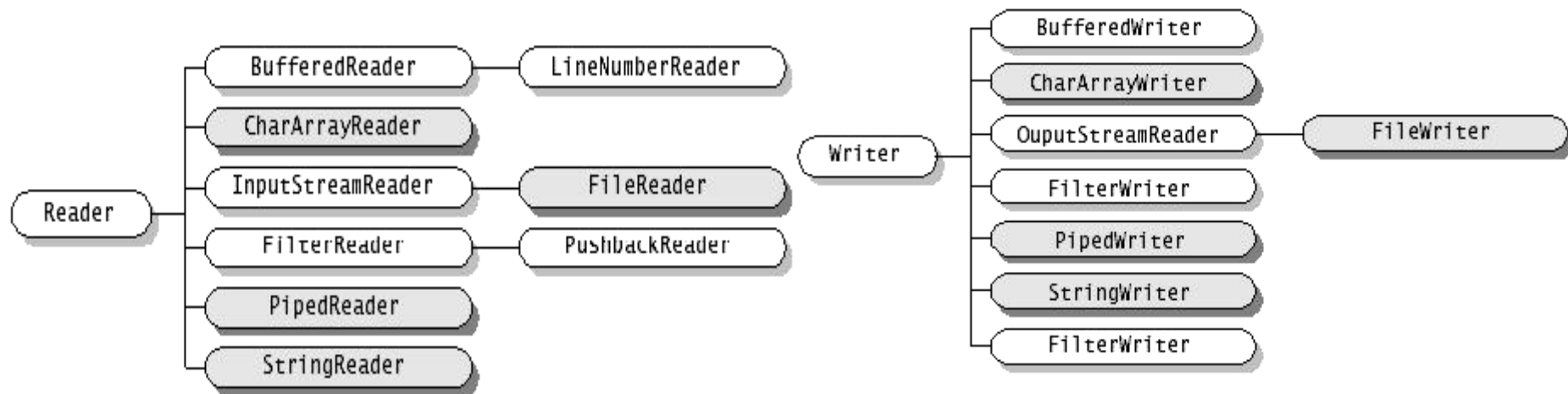
4种主要的I/O类

	Input	Output
Byte Streams	InputStream	OutputStream
Character Streams	Reader	Writer



4种主要的I/O类

	Input	Output
Byte Streams	InputStream	OutputStream
Character Streams	Reader	Writer





Hierarchy For Package java.io

- java.io.**InputStream** (implements java.io.Closeable)
 - java.io.**ByteArrayInputStream**
 - java.io.**FileInputStream**
 - java.io.**FilterInputStream**
 - java.io.**BufferedInputStream**
 - java.io.**DataInputStream** (implements java.io.DataInput)
 - java.io.**LineNumberInputStream**
 - java.io.**PushbackInputStream**
 - java.io.**ObjectInputStream** (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
 - java.io.**PipedInputStream**
 - java.io.**SequenceInputStream**
 - java.io.**StringBufferInputStream**
- java.io.**OutputStream** (implements java.io.Closeable, java.io.Flushable)
 - java.io.**ByteArrayOutputStream**
 - java.io.**FileOutputStream**
 - java.io.**FilterOutputStream**
 - java.io.**BufferedOutputStream**
 - java.io.**DataOutputStream** (implements java.io.DataOutput)
 - java.io.**PrintStream** (implements java.lang.Appendable, java.io.Closeable)
 - java.io.**ObjectOutputStream** (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
 - java.io.**PipedOutputStream**

Hierarchy For Package java.io

- java.io.**Reader** (implements java.io.Closeable, java.lang.Readable)
 - java.io.**BufferedReader**
 - java.io.**LineNumberReader**
 - java.io.**CharArrayReader**
 - java.io.**FilterReader**
 - java.io.**PushbackReader**
 - java.io.**InputStreamReader**
 - java.io.**FileReader**
 - java.io.**PipedReader**
 - java.io.**StringReader**
- java.io.**StreamTokenizer**
- java.io.**Writer** (implements java.lang.Appendable, java.io.Closeable, java.io.Flushable)
 - java.io.**BufferedWriter**
 - java.io.**CharArrayWriter**
 - java.io.**FilterWriter**
 - java.io.**OutputStreamWriter**
 - java.io.**FileWriter**
 - java.io.**PipedWriter**
 - java.io.**PrintWriter**
 - java.io.**StringWriter**

<https://docs.oracle.com/javase/8/docs/api/index.html>



Interface Hierarchy

- java.lang.**AutoCloseable**
 - java.io.**Closeable**
 - java.io.**ObjectInput** (also extends java.io.DataInput)
 - java.io.**ObjectOutput** (also extends java.io.DataOutput)
- java.io.**DataInput**
 - java.io.**ObjectInput** (also extends java.lang.AutoCloseable)
- java.io.**DataOutput**
 - java.io.**ObjectOutput** (also extends java.lang.AutoCloseable)
- java.io.**FileFilter**
- java.io.**FilenameFilter**
- java.io.**Flushable**
- java.io.**ObjectInputValidation**
- java.io.**ObjectStreamConstants**
- java.io.**Serializable**
 - java.io.**Externalizable**



流操作过程

Reading

open a stream
while more information
 read information

close the stream

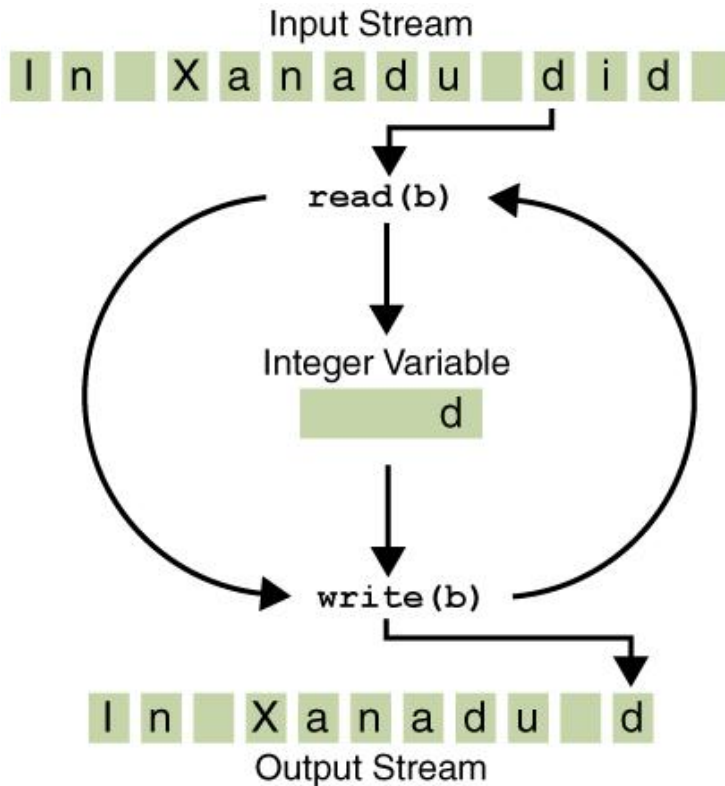
Writing

open a stream
while more information
 write information

close the stream

//Java/tutorial/essential/io/bytestreams.html

CopyBytes.java



```

public class ByteStream {
    public static void main(String[] args) throws
        IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("xan.txt");
            out = new FileOutputStream("out.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
  
```

**Always Close
Streams**



When Not to Use Byte Streams

- **ByteStream.java** seems like a normal program, but it actually represents a kind of low-level I/O that you should avoid. Since `xanadu.txt` contains character data, the best approach is to use character streams, as discussed in the next section. There are also streams for more complicated data types. Byte streams should only be used for the most primitive I/O.
- So why talk about byte streams? Because all other stream types are built on byte streams.



Character Streams

```
public class CopyCharacters {  
    public static void main(String[] args) throws IOException {  
        FileReader inputStream = null;  
        FileWriter outputStream = null;  
        try {  
            inputStream = new FileReader("xanadu.txt");  
            outputStream = new FileWriter("characteroutput.txt");  
            int c;  
            while ((c = inputStream.read()) != -1) {  
                outputStream.write(c);  
            }  
        } finally {  
            if (inputStream != null) {  
                inputStream.close();  
            }  
            if (outputStream != null) {  
                outputStream.close();  
            }  
        }  
    }  
}
```

All character stream classes are descended from Reader and Writer. As with byte streams, there are character stream classes that specialize in file I/O: FileReader and FileWriter.

JAVA的输入/输出

- JAVA的输入/输出机制：流的概念
- 输入/输出流：java.io包中的标准类
- ✓ **两个基本抽象类：**
 - InputStream, OutputStream
 - 基本类中定义了完成基本I/O操作的抽象方法
 - 读写操作：**read, write**
 - 关闭流的操作：**close**
 - 送出并清空缓冲区数据的操作：**flush**



Java.io.InputStream

- java.io.InputStream (implements java.io.Closeable)
 - java.io.ByteArrayInputStream
 - java.io.FileInputStream
 - java.io.FilterInputStream
 - java.io.BufferedInputStream
 - java.io.DataInputStream (implements java.io.DataInput)
 - java.io.LineNumberInputStream
 - java.io.PushbackInputStream
 - java.io.ObjectInputStream (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
 - java.io.PipedInputStream
 - java.io.SequenceInputStream
 - java.io.StringBufferInputStream

InputStream Method:

Type	Method	Description
int	available()	Returns an estimate of the number of bytes
void	close()	Closes this input stream and releases any system resources.
void	mark(int readlimit)	Marks the current position in this input stream.
boolean	markSupported()	Tests if this input stream supports the mark
Abstract int	read()	Reads the next byte of data from the input stream.
int	read(byte[] b)	Reads some number of bytes and stores them into the buffer array b.
int	read(byte[] b, int off, int len)	Reads up to len bytes of data from the input stream into an array of bytes.
void	reset()	Repositions this stream to the position at the time the mark method was last called.
long	skip(long n)	Skips over and discards n bytes of data.



InputStream输入流

- 管理字节（适于读取面向字节的数据，如：图像、声音等）
- 是所有表示输入字节流类的父类（抽象类）
- 三个基本方法：
 - ✓ `abstract int read()`
从输入流中读取一个字节。
 - ✓ `int read(byte[] b)`
从输入流中读若干个字节到数组中。
 - ✓ `int read(byte[] b, int off, int len)`
从输入流中读len个字节到数组中。off是写入数组的位置。



InputStream输入流

- ✓ `skip(long n)` 跳过n个字节。
- ✓ `boolean markSupported()` 流是否支持mark功能
- ✓ `mark(int readlimit)` 在当前位置做标记.
- ✓ `reset()` 回到最近一次做的标记处。
- ✓ `close()` 关闭输入流，释放与此输入流相连的系统资源。

InputStream输入流的子类

- 这些子类的构造方法都可以用某种方式指定其数据源。
- 加强输入流，对InputStream类进行功能扩充

- **ByteArrayInputStream(byte数组)**
- **FileInputStream (文件路径名 或 File对象)**
- **ObjectInputStream (InputStream in)**
- **PipedInputStream (PipedOutputStream pipe)**
- **SequenceInputStream 表示其他输入串的逻辑连接**
- **FilterInputStream (其他输入流) 定义了子类对流的进一步处理功能。**
 - **BufferedInputStream (InputStream in)**
 - **DataInputStream(InputStream in) 各种数据读入**



java.io.OutputStream

java.io.OutputStream

- java.io.ByteArrayOutputStream
- java.io.FileOutputStream
- java.io.FilterOutputStream
 - java.io.BufferedOutputStream
 - java.io.DataOutputStream (implements java.io.DataOutput)
 - java.io.PrintStream (implements java.lang.Appendable, java.io.Closeable)
- java.io.ObjectOutputStream (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
- java.io.PipedOutputStream

OutputStream输出流

- 抽象类：是所有表示输出字节流类的父类。
- 功能：接受要输出的字节并将它送往目的地。
- 方法：
 - `write(int b)` 往输出流写一个字节
 - `write(byte[] b)` 将字节数组数据写入输出流。
 - `flush()` 刷新输出流，并使缓冲区中的数据写出。
 - `close()` 关闭输出流，释放与之相连的系统资源。

OutputStream输出流子类

- `FileOutputStream` (`File`类对象或文件名),
- `ByteArrayOutputStream` ()
数据被写到无名字节数组, 该字节数组内容可利用`toByteArray()`和`toString()`分别取到指定字节数组和字符串中。
- `PipedOutputStream` (`PipedInputStream` pipe)
- `ObjectOutputStream`
- `FilterOutputStream`
 - ✓ `DataOutputStream(OutputStream out)`
包含输出各种数据类型数据的方法, 如`writeFloat()`
 - ✓ `PrintStream(OutputStream out)`
包含输出各种数据类型数据的方法, 如`print()`, `println`。
但没有对应输入流



- There are many byte stream classes. To demonstrate how byte streams work, we'll focus on the file I/O byte streams, FileInputStream and FileOutputStream.
- Other kinds of byte streams are used in much the same way; they differ mainly in the way they are constructed.

7.3标准文件类File

File Constructor Summary

File(File parent, String child)

Creates a new File instance from a parent abstract pathname and a child pathname string.

File(String pathname)

Creates a new File instance by converting the given pathname string into an abstract pathname.

File(String parent, String child)

Creates a new File instance from a parent pathname string and a child pathname string.

File(URI uri)

Creates a new File instance by converting the given file: URI into an abstract pathname.

创建文件对象有
四种构造方法
(JDK6.0)



标准文件类： **File**

- **File**类的对象是文件系统中的一个个目录或文件的抽象表示。
- **File**类对象描述文件路径、名字、长度、可否读写等属性，可用来命名文件、查询文件属性、对目录进行操作，但不读写文件。
- 通过**File**类对象可以对操作系统的文件进行管理，体现了跨平台不同文件的统一管理

File类构造方法

➤ File(String path)

- ✓ 封装文件例（使用相对路径，移植性较好）

```
File f1 = new File("mydir\\myfile.txt");
```

- ✓ 封装目录例（使用绝对路径）

```
File f2 = new File("d:\\mydir\\dir1");
```

➤ File(String parent, String child)

```
File f3 = new File("d:\\d1", "a.java")
```

➤ File(File dir, String name)

```
File f4 = new File(f2, "myfile.txt");
```

windows \\
unix /

File类方法(1)

方 法	说 明
boolean <u>canRead</u> ()	判断文件是否可读。
boolean <u>canWrite</u> ()	判断文件是否可写。
int <u>compareTo</u> (File pathname)	比较两个文件名。
int <u>compareTo</u> (Object o)	将当前文件与所给对象作比较。
boolean <u>createNewFile</u> ()	自动创建一个新的空文件，文件名未曾使用过。
boolean <u>delete</u> ()	判断文件（或者目录）是否可删除。
boolean <u>equals</u> (Object obj)	判断文件路径是否与所给对象相同。
boolean <u>exists</u> ()	判断文件（或者目录）是否存在。
String <u>getAbsolutePath</u> ()	返回文件（或者目录）的绝对路径。
String <u>getName</u> ()	返回文件（或者目录）的名字。
String <u>getParent</u> ()	返回文件的上一级路径。
String <u>getPath</u> ()	返回文件（或者目录）的路径。



File类方法(2)

boolean <u>isAbsolute</u> ()	判断该文件对象是否是一个绝对路径名。
boolean <u>isDirectory</u> ()	判断是否是目录。
boolean <u>isFile</u> ()	判断是否是文件。
boolean <u>isHidden</u> ()	测试文件是否是隐藏文件，返回布尔量。
long <u>lastModified</u> ()	获得文件最后一次修改的时间。
long <u>length</u> ()	返回文件的长度。
String[] <u>list</u> ()	返回目录下的文件及子目录名列表。
String[] <u>list</u> (FilenameFilter filter)	列出一个目录中符合文件过滤器的所有文件
boolean <u>mkdir</u> ()	创建目录。
boolean <u>mkdirs</u> ()	如果该目录的父目录不存在同时创建父目。
boolean <u>renameTo</u> (File dest)	为文件重新命名。
boolean <u>setLastModified</u> (long time)	设置文件（或者目录）最后修改的时间。
boolean <u>setReadOnly</u> ()	将文件（或者目录）设置为只读。
String <u>toString</u> ()	返回文件对象的描述。



例: FileStuff.java

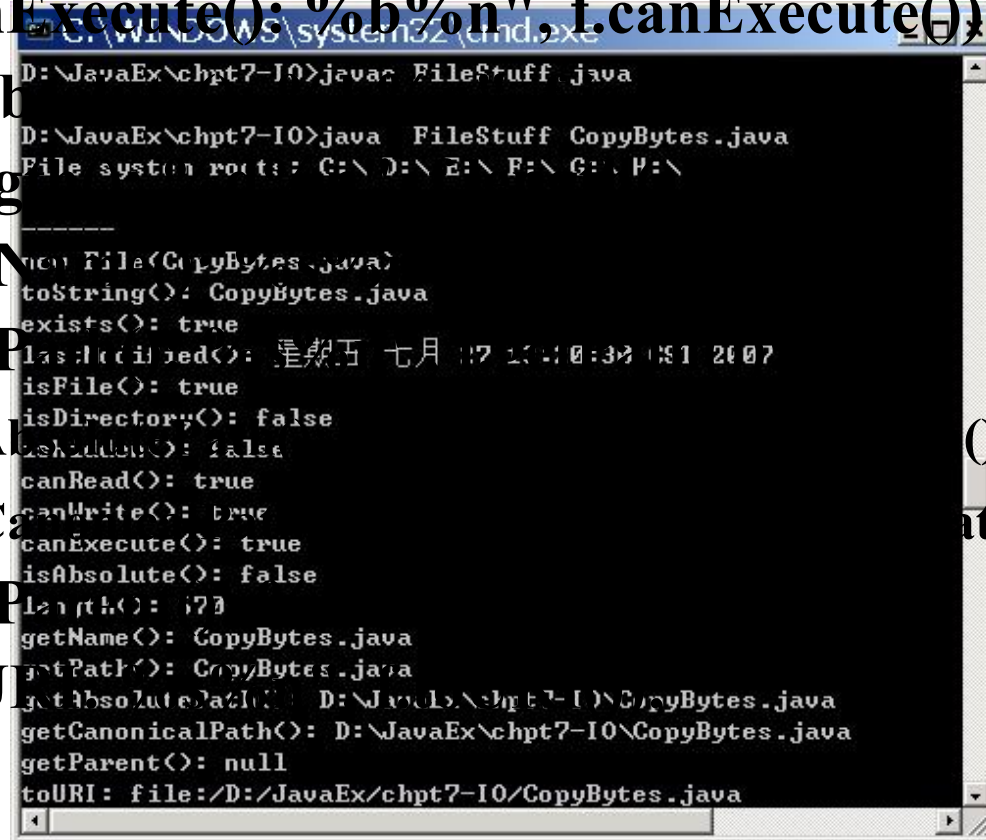
```
public class FileStuff {  
    public static void main(String args[]) throws IOException {  
        out.print("File system roots: ");  
        for (File root : File.listRoots()) {  
            out.format("%s ", root);        }        out.println();  
        for (String fileName : args) {  
            out.format("%n-----%nnew File(%s)%n", fileName);  
            File f = new File(fileName);  
            out.format("toString(): %s%n", f);  
            out.format("exists(): %b%n", f.exists());  
            out.format("lastModified(): %tc%n", f.lastModified());  
            out.format("isFile(): %b%n", f.isFile());  
            out.format("isDirectory(): %b%n", f.isDirectory());  
            out.format("isHidden(): %b%n", f.isHidden());  
        }  
    }  
}
```


运行结果

```

out.format("canRead(): %b%n", f.canRead());
out.format("canWrite(): %b%n", f.canWrite());
out.format("canExecute(): %b%n", f.canExecute());
out.format("isAbsolute(): %b%n", f.isAbsolute());
out.format("length(): %d%n", f.length());
out.format("getName(): %s%n", f.getName());
out.format("getPath(): %s%n", f.getPath());
out.format("getAbsolutePath(): %s%n", f.getAbsolutePath());
out.format("getCanonicalPath(): %s%n", f.getCanonicalPath());
out.format("getParent(): %s%n", f.getParent());
out.format("toURI(): %s%n", f.toURI());
}
}
}

```



```

D:\JavaEx\chpt7-IO>javac FileStuff.java
D:\JavaEx\chpt7-IO>java FileStuff CopyBytes.java
File system roots: C:\ D:\ E:\ F:\ G:\ P:\
-----
new File(CopyBytes.java)
toString(): CopyBytes.java
exists(): true
lastModified(): 星期五 七月 12 10:40:30 CST 2007
isFile(): true
isDirectory(): false
isHidden(): false
canRead(): true
canWrite(): true
canExecute(): true
isAbsolute(): false
length(): 173
getName(): CopyBytes.java
getPath(): CopyBytes.java
getAbsolutePath(): D:\JavaEx\chpt7-IO\CopyBytes.java
getCanonicalPath(): D:\JavaEx\chpt7-IO\CopyBytes.java
getParent(): null
toURI(): file:/D:/JavaEx/chpt7-IO/CopyBytes.java

```



File的方法（续）

➤ File类使用一个静态实例变量：

separator (public static final String)保存了当前的目录分隔符。

✓ 为本机操作系统支持的路径分隔符

✓ DOS, Windows “\\”

✓ Unix “/”

```
File f1=new File("MyDir"+File.separator+"data.txt");
```



文件的处理

- 随机访问文件:RandomAccessFile类
 - ✓ 不属于InputStream/OutputStream
 - ✓ 用于访问本地文件
 - ✓ 可以实现随机访问
 - ✓ 可读可写
 - ✓ 同时实现了DataInput和DataOutput接口，可同时进行有格式的读写
- 创建时指定对应某一文件完成读或写操作
- 用于读写一个文件中任意位置的数据
- 视文件如同一个字节类型数组，数组下标即文件指针。读写操作都会移动指针



文件I/O类:RandomAccessFile

构造方法

RandomAccessFile(File file, String mode)

参数: file 一个File对象, 它封装了一个同系统相关的文件名

mode “r”用于只读; “rw”用于读写

RandomAccessFile(String name, String mode)

参数: name 文件名(同系统相关)

mode “r”用于只读; “rw”用于读写

如 RandomAccessFile s1=new RandomAccessFile(“a.txt”, “r”);
File f1=new File(“a.txt”);
RandomAccessFile s2=new RandomAccessFile(f1, “rw”);



文件的处理（续）

➤ 随机文件读写

- ✓ `readXx()`, `writeXx()`: 实现 `DataInput/Output` 接口
- ✓ 读写位置：文件指针及其操作方法

➤ 指针操作

`long getFilePointer()` 返回文件指针的当前位置，即相对于文件头的位移量。单位：字节 该位置即下一个读写操作的位置。

`void seek(long pos)` 移动指针 (位移单位：字节) 将文件指针设置为偏移文件开始的 `pos` 字节处



文件I/O类:RandomAccessFile

`public int skipBytes(int n)`

将读取位置跳过n个字节

`close()`

关闭文件

`long length()`

取文件长度

➤ 写文件

`writeInt(int v)`

写一个整型数， 4个字节， 高字节在前



文件I/O类:RandomAccessFile

`writeBoolean(boolean v)`

写一个boolean值，一个字节，0或1

`writeUTF(String str)`

写一字符串，前两个字节标明字符串字节长度

➤ 读文件

`byte readByte()`

读取一字节

`char readChar()`

读取一字符(Unicode码2个字节)

`double readDouble()`

读取一双精度数(8个字节)

`String readUTF()`

读取一个字符串。

`readLine()`

读取以\n、\r、\r\n或者EOF终止的一行。然后该行所有字节被转换成一个Unicode字符串，并返回字符串。



RandomAccessFile

➤ 构造方法的例外:

- ✓ `IllegalArgumentException` (not "r" or "rw")
- ✓ `FileNotFoundException`
- ✓ `SecurityException`

➤ 文件操作例外:

- ✓ `EOFException` (`IOException`) 读到文件尾
- ✓ `IOException` 虽没有到文件尾, 但读不到字符
- ✓ `IOException` 文件已关闭



文件读写（文本）

➤ 输入

```
BufferedReader fin =
```

```
    new BufferedReader(new FileReader("test.txt") );
```

```
读文件 fin.readLine()
```

➤ 输出

```
PrintWriter fout =
```

```
    new PrintWriter(new FileWriter("test.txt") , true) );
```

或：

```
PrintWriter fout = new PrintWriter(new BufferedWriter(new  
                                FileWriter("test.txt")));
```

```
写文件 fout.println(...)
```

```
flush()
```

```
close()
```



7.4 简单的IO流

➤ FileInputStream/OutputStream

- ✓ 代表一个在本地文件系统中的文件的IO流，可通过文件名或文件对象创建。

```
FileInputStream fis=new FileInputStream("....\\in.txt");
```

```
FileOutputStream fos=new  
FileOutputStream("....\\out.txt");
```

```
int c;
```

```
while((c=fis.read())!= -1)    fos.write(c);
```

```
fis.close();
```

```
fos.close();
```



例:FileReader和FileWriter的综合应用

```
/* 文件以字符流形式输入和输出
import java.io.*;
public class FileRW {
public static void main(String[] args) throws IOException{
    File inputFile = new File("D:\\JavaEx\\chpt7-IO\\FileRW.java");
        // 建立一个名字为inputFile的File类对象
String str;
StringBuffer buffer= new StringBuffer();
BufferedReader inBuffer = new BufferedReader( new
    FileReader(inputFile));
    // 由FileReader类对象建立一个名字为in的BufferedReader类对象
while ((str = inBuffer.readLine()) !=null) //从文件FileRW.java中读数据
    { buffer.append(str).append("\n") ;
    }
inBuffer.close(); // 关闭输入流in
```



例:FileReader和FileWriter的综合应用

```
LineNumberReader lines=new LineNumberReader(new  
    StringReader(buffer.toString() ) );  
    File outputFile = new File("D:\\FileRW.txt");  
    // 建立一个名字为outputFile的File类对象  
    FileWriter out =new FileWriter( outputFile);  
        // 建立一个名字为out 的FileWriter类对象  
    while ((str=lines.readLine())!=null) {  
        str= String.valueOf(lines.getLineNumber()) +": "+str +"\n";  
        out.write(str);        // 将读入的数据写到文件FileRW.txt中  
        System.out.print(str);  
    }  
    out.close();        // 关闭输出流out  
}  
}
```



简单的IO流（续）

- `SequenceInputStream`
`SequenceInputStream(InputStream, InputStream)`
把两个/多个`InputStream`连接为一个`InputStream`
- `ByteArrayInput/OutputStream`
 - ✓ 从字节数组读取/向字节数组写入8位数据
- `StringBufferInputStream(String)`
 - ✓ 把一个`String` 转换为`InputStream`

7.5 过滤流(FilterInput/OutputStream)

➤ FilterInput/OutputStream

✓ 作用：给朴素的Input/OutputStream加上一些修饰——某些有用的格式

✓ 来源：Input/OutputStream

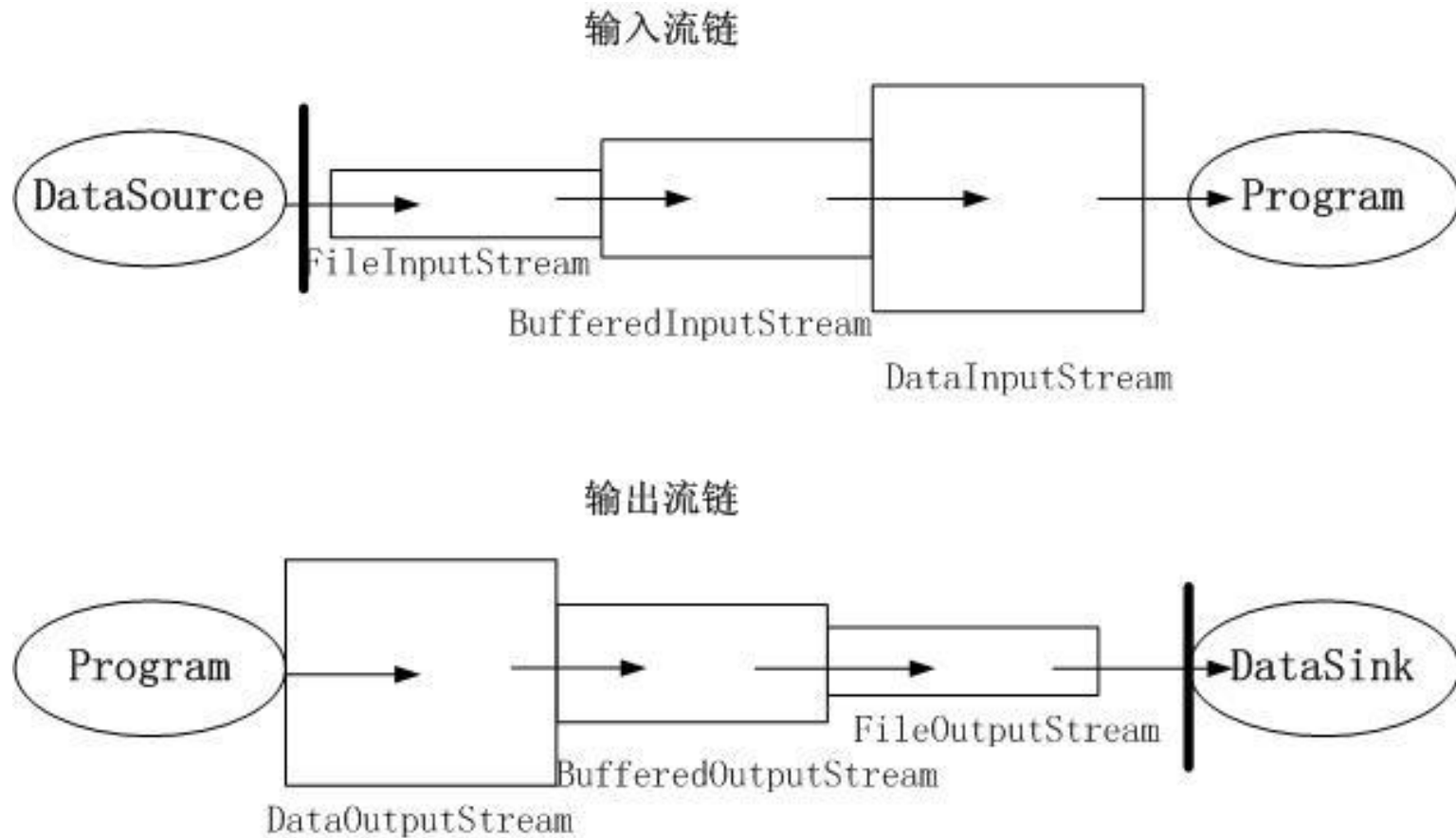
✓ 层次：

java.io.InputStream
java.io.FilterInputStream

java.io.FilterInputStream

- java.io.BufferedInputStream
- java.io.DataInputStream (implements java.io.DataInput)
- java.io.LineNumberInputStream
- java.io.PushbackInputStream

构造带缓冲的文件数据输入/输出流





过滤流—BufferedInputStream

➤ BufferedInput/OutputStream

- ✓ 自动利用内存缓冲，不必每次存取外设
- ✓ 还可在其外面再加包装

```
new DataInputStream(  
    new BufferedInputStream(  
        new FileInputStream("File1.txt") ) );
```

- ## ➤ 缓冲流必须建立在节点流之上，对节点流中的数据进行某些加工、处理，并提供一些友好的方法供用户进行输入、输出操作以及流控制。

例如，**BufferedInputStream** 可以对任何种类的输入流进行带缓冲区的封装以达到性能的改善(可减少程序I/O操作次数，提高程序执行效率)。**Java**利用缓冲流可以在读/写数据的同时对数据进行处理。使用缓冲流时要注意：必须将缓冲流和某个输入流或输出流（节点流）连接。而在程序设计中，连接是通过在缓冲流的构造方法中指定入口参数---节点流来实现的。

➤ 例：

```
FileInputStream in=new FileInputStream(“text”);
```

```
BufferedInputStream bufin=new BufferedInputStream(in);
```

实现了缓冲流**bufin**和文件输入流**in**连接。



如何向文件中写入基本数据类型

```
import java.io.*;
class DataOut{
    public static void main(String[] args) throws IOException{
        FileOutputStream fout=new FileOutputStream("data.dat");
        DataOutputStream out=new DataOutputStream(fout);
        String[] items={"面包","笔","被子","牙膏","毛巾"};
        int[] units={5,2,1,2,3};
        float[] prices={1.5f,2.8f,125f,7.2f,5.6f};
        for(int i=0;i<items.length;i++){
            out.writeUTF(items[i]);
            out.writeChar('\t');
            out.writeFloat(prices[i]);
            out.writeChar('\t');
            out.writeInt(units[i]);
            out.writeChar('\t');
        }
        out.close();
    }
}
```



- **DataOutputStream**以**OutputStream**字节节点流为基础，因此属于字节过滤流
- 那么以字符节点流**Reader**、**Writer**为基础的过滤流，则称为字符过滤流
- 我们前面用到的**BufferedReader**就是它提供了输出缓冲功能,还增加了对整行字符的处理方法**readLine()**
- 使用**BufferedReader**和**BufferedWriter**实现文本文件的按行拷贝

过滤流（续）

➤ DataInput/OutputStream

- ✓ 对于基本数据类型进行有格式的读写
- ✓ 注意：此处为存储格式，DataOutputStream 与 DataInputStream 配套使用才有意义
- ✓ 实现了DataInput接口

readInt/Float/Char/Boolean/Long/Double()

readLine()—读入的重要方法

- ✓ 实现了DataOutput接口

writeInt/Float/Char/Boolean/Long/Double()

➤ PrintStream

- ✓ 对于基本数据类型进行有格式的显示
- ✓ 注意：此处为显示格式

Data streams

- Data streams support binary I/O of primitive data type values (boolean, char, byte, short, int, long, float, and double) as well as String values.

Order	Data type	Data description	Output Method	Input Method	Sample Value
1	double	Item price	<code>DataOutputStream .writeDouble</code>	<code>DataInputStream. readDouble</code>	19.99
2	int	Unit count	<code>DataOutputStream .writeInt</code>	<code>DataInputStream. readInt</code>	12
3	String	Item description	<code>DataOutputStream .writeUTF</code>	<code>DataInputStream. readUTF</code>	"Java T- Shir t"



例: DataStreams.java

```
public class DataStreams {
    static final String dataFile = "invoicedata.dat";
    static final double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
    static final int[] units = { 12, 8, 13, 29, 50 };
    static final String[] descs = { "Java T-shirt",
        "Java Mug",      "Duke Juggling Dolls",
        "Java Pin",      "Java Key Chain" };
    public static void main(String[] args) throws IOException {
        DataOutputStream out = null;
        try {
            out = new DataOutputStream(new
                BufferedOutputStream(new FileOutputStream(dataFile)));
            for (int i = 0; i < prices.length; i++) {
                out.writeDouble(prices[i]);
                out.writeInt(units[i]);
                out.writeUTF(descs[i]);  }
        } finally {
            out.close();    }
    }
}
```



例: DataStreams.java

```
DataStream in = null;
double total = 0.0;
try {
    in = new DataInputStream(new BufferedInputStream(new FileInputStream(dataFile)));
    double price;
    int unit;
    String desc;
    try {
        while (true) {
            price = in.readDouble();
            unit = in.readInt();
            desc = in.readUTF();
            System.out.format("You ordered %d units of %s at $%.2f%n",
                               unit, desc, price);
            total += unit * price;
        }
    } catch (EOFException e) { }
    System.out.format("For a TOTAL of: $%.2f%n", total);
} finally { in.close();
} }
```

两种流类

➤ node stream

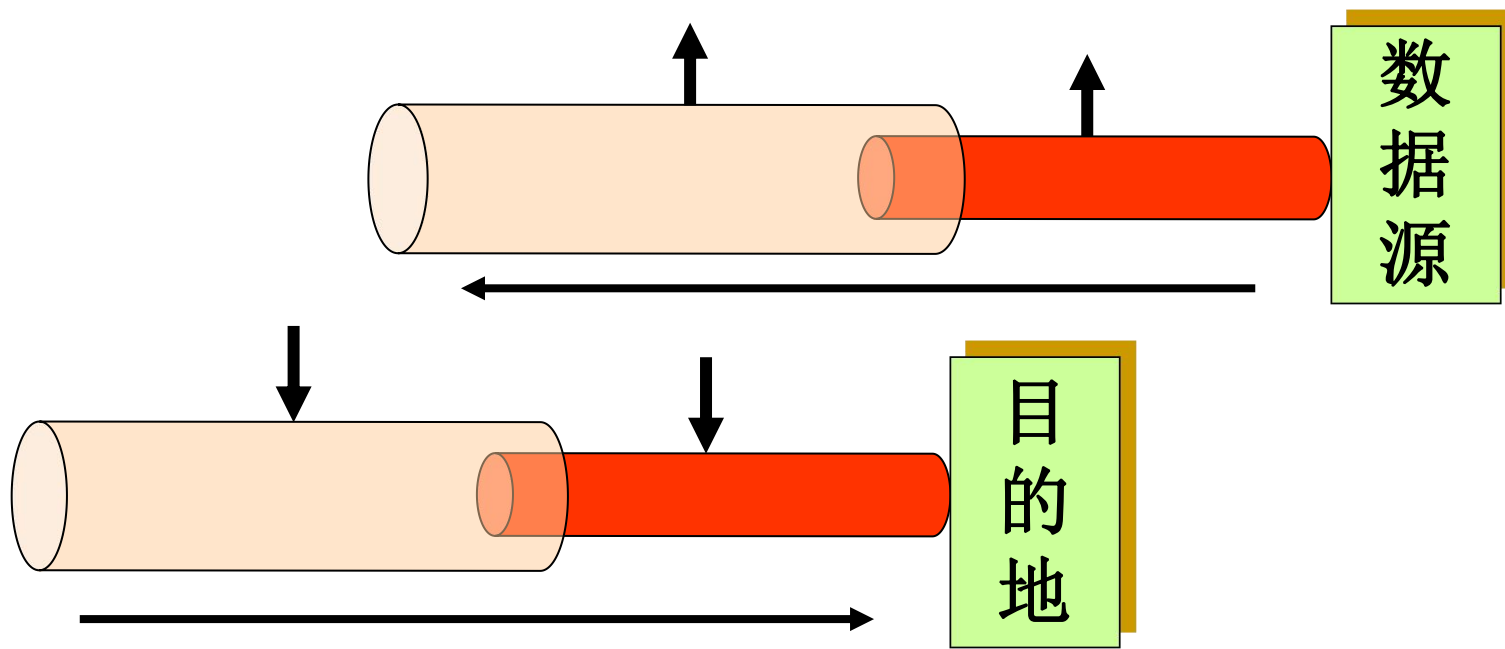


对指定的地方（磁盘文件、内存 等）读/写

➤ filter stream



一个filter 流使用node流作为输入或输出。





基本流类—node流

➤ FileInputStream — FileOutputStream

```
FileInputStream infile =  
    new FileInputStream("old.dat");
```

```
FileOutputStream outfile =  
    new FileOutputStream("new.dat");
```



基本流类—filter流

- BufferedInputStream — BufferedOutputStream
增加I/O操作的有效性

- DataInputStream — DataOutputStream

可以读写Java基本类型的数据

byte readByte()

void writeByte(byte)

long readLong()

void writeLong(long)

double readDouble()

void writeDouble(double)



使用I/O流读写文件

➤ 输出

```
DataOutputStream out =  
new DataOutputStream(new FileOutputStream("test3"))  
写文件: out.writeUTF("wang hong");  
out.writeFloat(485.2F);
```

➤ 输入

```
DataInputStream in =  
new DataInputStream(new FileInputStream("test3"));  
读文件: in.readUTF()  
in.readFloat()
```

2022-05-06 11#

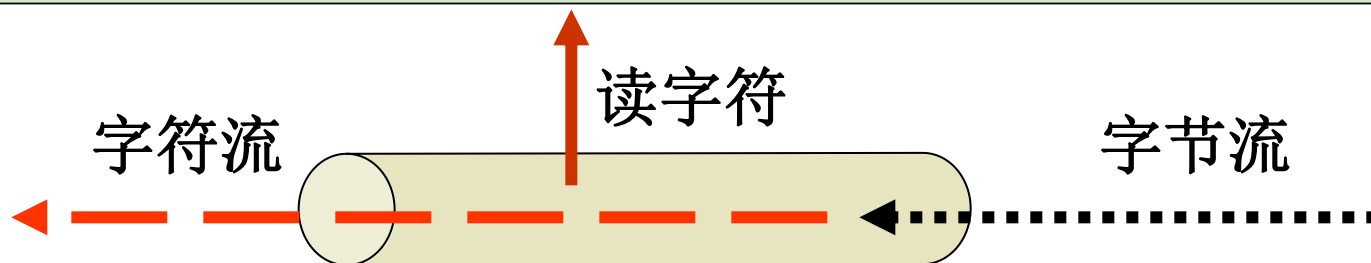


7.6 字符流 - Reader类

- 是所有读取字符流类的父类抽象类(面向Unicode字符操作) Java使用Unicode码表示字符和字符串。
- 方法:
 - ✓ `boolean ready()` 输入字符流是否可读
 - ✓ `int read()` 读取一个字符
 - ✓ `int read(char[] cbuf)` 读取一串字符(到字符数组)
 - ✓ `long skip(long n)` 跳过n个字符
 - ✓ `mark(int readAheadLimit)` 在当前位置做一标记
 - ✓ `reset()` 将读取位置恢复到标记处
 - ✓ `close()` 关闭字符流

Reader类子类

- `CharArrayReader(char[] buf)`
- `PipedReader(PipedWriter)`
- `StringReader(String s)`
- `BufferedReader(Reader in)`
 - ✓ 提供有效读的方法，如： `readLine`



`InputStreamReader(InputStream in)`



`FileReader(File对象 或 文件名)`



Writer类

- 是所有表示输出字符流类的父类（抽象类）。
- 功能：接受要输出的字符并将它送往目的地。
- 方法：
 - ✓ write(String str)
 - ✓ write(char[] cbuf)
 - ✓ flush()
 - ✓ close()

Writer类的子类

- `CharArrayWriter()` `toCharArray()` and `toString()`.
- `StringWriter()` 方法 `toString`
- `PipedWriter(PipedReader)`
- `BufferedWriter(Write out)` 提高I/O效率
 - ✓ 方法: `write(...)` 写字符或字符串
- `PrintWriter (OutputStream类或Writer类对象)`
 - ✓ 方法: `print` `println` 实现各种类型数据的输出



`OutputStreamWriter(OutputStream out)`



`FileWriter(File对象或文件名)`



TestReaderWriter.java

```
public class TestReaderWriter {  
    public static void main(String[] args) {  
        try {  
            //分别创建可处理中文字符的文件输入输出流  
            BufferedReader br = new BufferedReader(new  
                FileReader("TestReaderWriter.java"));  
            BufferedWriter bw = new BufferedWriter(new FileWriter("temp.txt"));  
            String s = br.readLine();//readLine从char流中读入一行文本String对象  
            while ( s!=null ) {  
                bw.write(s);//将读出的一行内容写入到临时文件temp.txt中  
                bw.newLine(); //每行后面追加一个行分隔符  
                s = br.readLine();//从输入流中读取下一行,读取内容不包含行分隔符  
            }  
            br.close(); //关闭流  
            bw.close();  
        }  
    }  
}
```




TestReaderWriter.java (2)

```
//读取并显示复制后生成的文件temp.txt的内容
    br = new BufferedReader(new FileReader("temp.txt"));
    String s1=new String();
    System.out.println(" == 以下是所读取的文件内容 ==");
    while ((s1 = br.readLine())!= null ) {
        System.out.println(s1);
    }
    br.close();
} catch (IOException e) {
    e.printStackTrace();
} }
}
```



7.7 对象输入输出流

```
FileOutputStream fout = new  
    FileOutputStream("t.tmp");  
ObjectOutputStream oOut = new  
    ObjectOutputStream(fout);  
oOut.writeInt(12345);  
oOut.writeObject("Today");  
oOut.writeObject(new Date());  
oOut.flush();  
fout.close();
```

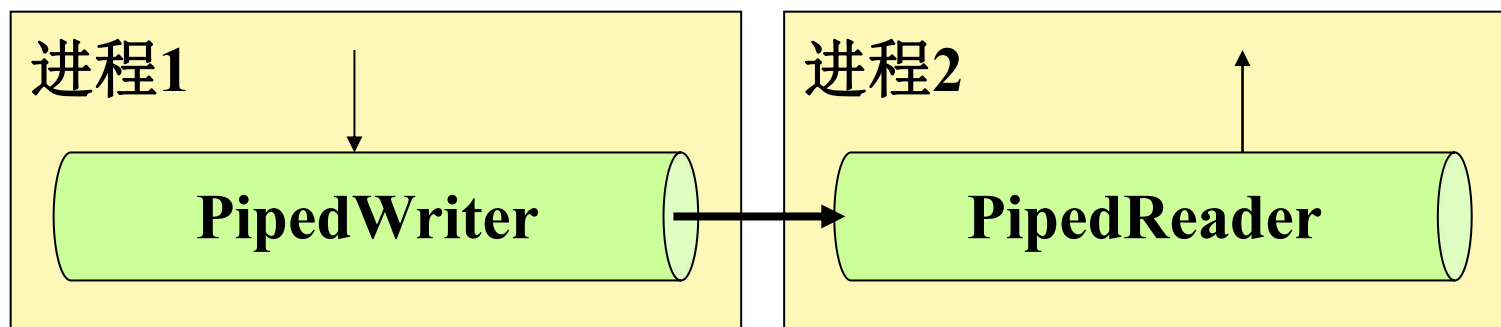


对象输入输出流

```
FileInputStream fin = new  
    FileInputStream("t.tmp");  
ObjectInputStream oln = new  
    ObjectInputStream(fin);  
int i = oln.readInt();  
String today = (String)oln.readObject();  
Date date = (Date)oln.readObject();  
fin.close();
```

管道与流

- 标准输入/输出
 - 改道 — 标准输入/输出 (> <)
 - 管道 — 将一个程序的标准输出作为另一个程序的标准输入 (|)
 - sort 对输入数据排序，然后输出
 - more 分页显示
- dir | sort | more





输出管道流

- 构造函数
 - ✓ `PipedWriter(PipedReader snk)`
建立一个与指定piped reader 相连的piped writer 。
 - ✓ `PipedWriter()`
建立一个还未与piped reader 相连的piped writer 。
- 方法：
 - ✓ `connect(PipedReader snk)`
与指定的piped reader 相连接。
 - ✓ `write(int c)`
写指定字符
 - ✓ `write(String s)`
写指定字符串



输入管道流

- 构造函数
 - ✓ `PipedReader(PipedWriter src)`
建立一个与指定piped writerr 相连的piped reader 。
 - ✓ `PipedReader()`
建立一个还未与piped Writer 相连的piped Reader 。
- 方法：
 - ✓ `connect(PipedWriter src)`
与指定的piped Writer 相连接。
 - ✓ `read(int c)`
读指定字符
 - ✓ `read(String s)`
读指定字符串

管道连接

- `pw = new PipedWriter(PipedReader snk)`
`new PipedReader()`
 ↖
- `pr = new PipedReader(PipedWriter src)`
`new PipedWriter()`
 ↖
- `pw = new PipedWriter()`
 `pr = new PipedReader()`
 `pw.connect(pr) 或 pr.connect(pw);`

- <file:///D:/JavaEx/tutorial/essential/io/pipedstreams.html>
- <http://download.oracle.com/javase/6/docs/api/java/math/BigDecimal.html>
- **RhymingWords.java**
/*将Words.txt中的单词反转，排序，再反转
原文件按第一个字母排序，生成文件按最后一个字母排序
*/

7.8 System类的输入和输出

- java.io包中提供了多种输入/输出流，但是Java把最基本的输入/输出流类放在了java.lang.System类中。
 - ✓ 标准输入/输出流。标准输入流用于从标准输入设备输入数据；标准输出流用于向标准输出设备输出数据。
 - ✓ 标准设备指计算机启动后默认的设备，比如，键盘是标准输入设备，显示器是标准输出设备。
 - ✓ 当不需要从标准输入设备（如键盘）上获取数据或者想将数据输出至标准输出设备以外的其它地方（如磁盘），我们就要重新设置输入流或输出流的方向，Java把这种操作称为重定向。
- System类包含三个I/O流成员用于系统标准输入/输出
 - ✓ public static InputStream **System.in**
 - ✓ public static PrintStream **System.out**
 - ✓ public static PrintStream **System.err**



System类及标准输入输出

- static `PrintStream err` 标准错误输出流
 - static `InputStream in` 标准输入流
 - static `PrintStream out` 标准输出流

 - `OutputStream`
 - ✓ `FilterOutputStream`
 - `PrintStream`
 - 提供输出各种类型数据的方法（`print`, `println`）
 - 不抛出I/O例外（可用`checkError`方法检测）
 - 可设置自动刷新
- `System.out.println(...)`



System类及标准输入输出

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

- `stdin.read()` 读一个字符
- `String readLine()` 读一行
- `read(char[] cbuf, int off, int len)` 将指定数量(`len`)字符读到字符数组中的指定位置 (`off`)
- `InputStreamReader` 没有`readLine()` 方法
- `BufferedReader(Reader)` 构造方法



System.in.read()

- 使用键盘输入一行文字，保存在文件中KeyTyechar.txt中

```
1.  try{
2.      System.out.println("please Input from Keyboard");
3.      int count,n=512;
4.      byte buffer[]=new byte[n];
5.      count=System.in.read(buffer);
6.      FileOutputStream wf=new FileOutputStream("typechar.txt");
7.      wf.write(buffer,0,count);
8.      //wf.flush();
9.      wf.close();      //关闭流。
10.     System.out.println("Save to the typechar.txt");
11. }catch(IOException IOe){
12.     System.out.println("File Write Error!");
13. }
```

KeyBoardIn.java

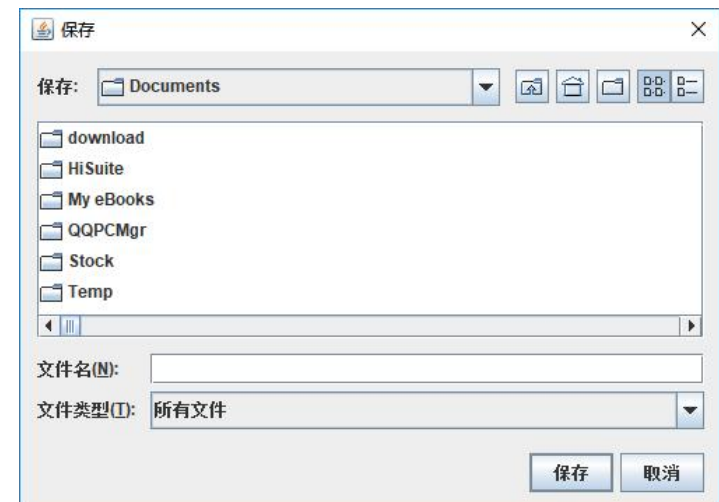
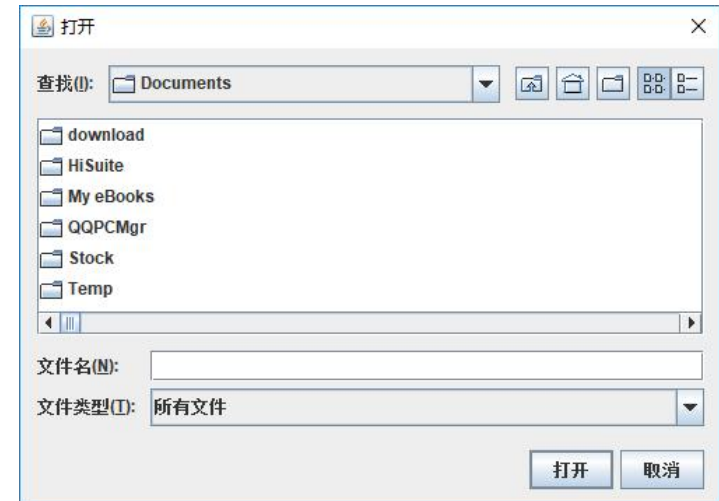


JOptionPane

```
private void closeFile() {  
    int option = JOptionPane.showConfirmDialog(null, "文件已修改， 是否  
保存？ ", "保存文件？ ", JOptionPane.YES_NO_OPTION,  
        JOptionPane.WARNING_MESSAGE, null);  
    switch (option) {  
        case JOptionPane.YES_OPTION:  
            saveFile();  
            break;  
        case JOptionPane.NO_OPTION:  
            closeWin();  
            break;  
        // dispose();  
    }  
}  
JOptionPane.showMessageDialog(null, e.toString(), "无法建立新文件",  
    JOptionPane.ERROR_MESSAGE);
```

JFileChooser 类

```
private JFileChooser fileChooser=new  
JFileChooser() ;
```





JFileChooser 的实例

```
private void openFile() { // 显示文件选取的对话框
    int option = fileChooser.showDialog(null, null);
    if (option == JFileChooser.APPROVE_OPTION) { // 使用者按下确认键
        try { // 开启选取的文件
            BufferedReader buf = new BufferedReader(new FileReader(
                fileChooser.getSelectedFile())); // 设定文件标题
            setTitle(fileChooser.getSelectedFile().toString());
            // 读取文件并附加至文字编辑区
            String text;
            while ((text = buf.readLine()) != null) {
                textArea.append(text);
            }
            buf.close();
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, e.toString(), "开启文件失败",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}
```



7.10 Interface Serializable

➤ 对象序列化

- 将程序中的对象输出到外部设备（如文件、网络）中，称为**对象序列化（serialization）**。反之，从外部设备将对象读入程序中称为**对象反序列化（deserialization）**。
- 一个类的对象要实现序列化，必须实现**java.io.Serializable**接口，该接口的定义如下。

```
public interface Serializable{
```




对象序列化与对象流

- **ObjectInputStream**是对象输入流，继承了InputStream类，实现了ObjectInput接口。
- **ObjectOutputStream**是对象输出流，继承了OutputStream类，实现了ObjectOutput接口。

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
private void readObjectNoData() throws ObjectStreamException;
```

小结

- java.io包中的输入/输出流类
- 面向字节的输入输出流
 - ✓ InputStream / OutputStream
- 面向字符的输入输出流
 - ✓ Reader / Writer
- 节点流
 - ✓ 连接流到具体的数据源或数据目的地。
Byte, File, Pipe, Sequence, String
- 过滤流
 - ✓ 加强流的处理功能。
 - ✓ 低级写出器将字符传给各自目标，高级写出器将组织好的字符发送给其他写出器。



tutorial/essential/io/index.html

I/O Streams

- [Byte Streams](#) handle I/O of raw binary data.
- [Character Streams](#) handle I/O of character data, automatically handling translation to and from the local character set.
- [Buffered Streams](#) optimize input and output by reducing the number of calls to the native API.
- [Scanning and Formatting](#) allows a program to read and write formatted text.
- [I/O from the Command Line](#) describes the Standard Streams and the Console object.
- [Data Streams](#) handle binary I/O of primitive data type and `String` values.
- [Object Streams](#) handle binary I/O of objects.

File I/O (Featuring NIO.2)

- [What is a Path?](#) examines the concept of a path on a file system.
- [The Path Class](#) introduces the cornerstone class of the `java.nio.file` package.
- [Path Operations](#) looks at methods in the `Path` class that deal with syntactic operations.
- [File Operations](#) introduces concepts common to many of the file I/O methods.
- [Checking a File or Directory](#) shows how to check a file's existence and its level of accessibility.
- [Deleting a File or Directory](#).
- [Copying a File or Directory](#).
- [Moving a File or Directory](#).
- [Managing Metadata](#) explains how to read and set file attributes.
- [Reading, Writing and Creating Files](#) shows the stream and channel methods for reading and writing files.
- [Random Access Files](#) shows how to read or write files in a non-sequentially manner.
- [Creating and Reading Directories](#) covers API specific to directories, such as how to list a directory's contents.
- [Links, Symbolic or Otherwise](#) covers issues specific to symbolic and hard links.
- [Walking the File Tree](#) demonstrates how to recursively visit each file and directory in a file tree.
- [Finding Files](#) shows how to search for files using pattern matching.
- [Watching a Directory for Changes](#) shows how to use the watch service to detect files that are added, removed or updated in one or more directories.
- [Other Useful Methods](#) covers important API that didn't fit elsewhere in the lesson.
- [Legacy File I/O Code](#) shows how to leverage `Path` functionality if you have older code using the `java.io.File` class. A table mapping `java.io.File` API to `java.nio.file` API is provided.



课后作业

- 1. Implement a pair of classes, one Reader and one Writer, that count the number of times a particular character, such as e, is read or written. The character can be specified when the stream is created. Write a program to test your classes. You can use readme.txt as the input file.
- 2. The file datafile begins with a single long that tells you the offset of a single int piece of data within the same file. Using the RandomAccessFile class, write a program that gets the int piece of data. What is the int data?