第八章 文件和流

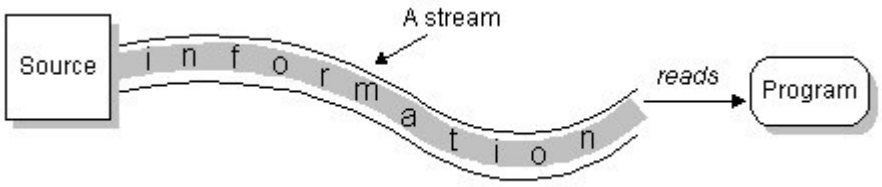
涉及到课本章节:

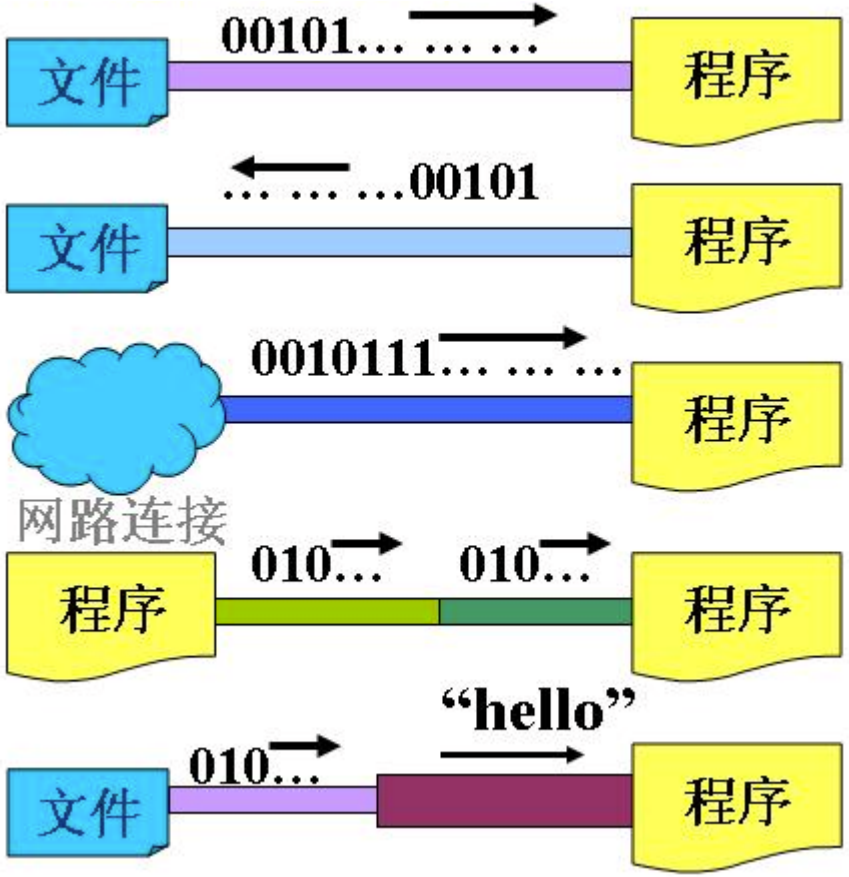
- 第5章 异常处理
- 第8章 输入输出流和文件操作

传统IO操作和文件读写

- (1)流
- (2)字节输入输出流
- (3)字符输入输出流
- (4)文件操作类

计算机中的流

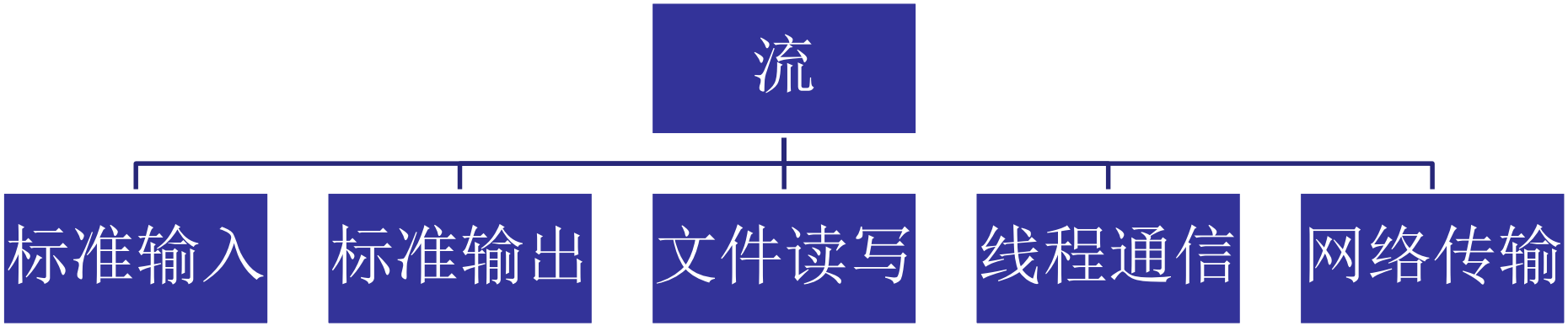

Source
A stream
i n f o r m a t i o n
reads
Program


00101… … …
文件
程序
… … …00101
文件
程序
0010111… … …
程序
网路连接
010…
010…
程序
程序
“hello”
010…
文件
程序

(1)流:
A Java中流的定义

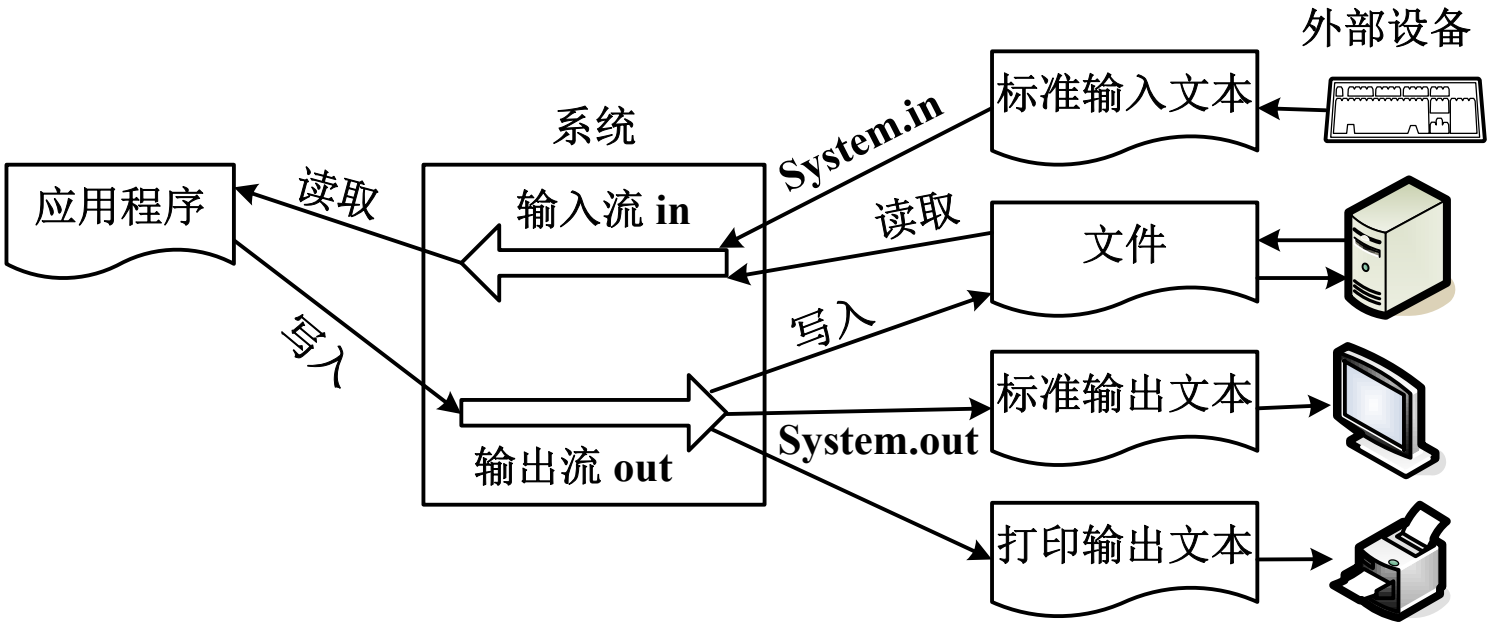
- 流是指一组有顺序的、 有起点和终点的字节集合。
- 流完成从键盘接收数据、读写文件以及打印等数据传输操作
- 流：Java中文件以流(Stream)的方式读写。
- Java中的流都在java.io包中

流的使用场景



流的存在及作用

- 面向多种设备（磁盘、键盘、显示器等）
- 每种设备拥有独立的驱动
- 提供设备无关的操作



B Java 中流的分类：

根据数据流向：

输入流：数据从程序外部流向程序内部，如读文件、网络接收数据、键盘输入。
输出流：数据从程序内部流向程序外部，如写文件、网络发送数据、屏幕输出。

根据流的处理单位：

二进制字节流：面向字节的输入和输出。
字符流：面向字符的输入和输出。是以Unicode字符为单位进行读写。

C：Java的标准输入输出

System类(java.lang)管理标准输入输出流和错误流
(public final class System extends Object)

- System.out：把输出送到缺省的显示(通常是显示器)。
  (public static final PrintStream out;)

- System.in：从标准输入获取输入(通常是键盘)。
  (public static final InputStream in;)

- System.err：把错误信息送到缺省的显示。
  (public static final PrintStream err;)

注：每当main方法被执行时,就自动生成上述三个对象，所以我们能够直接使用。

D:基于字节流的输入输出基类

- **Java中每一种字节流的基本功能都依赖于基本类InputStream和OutputStream**

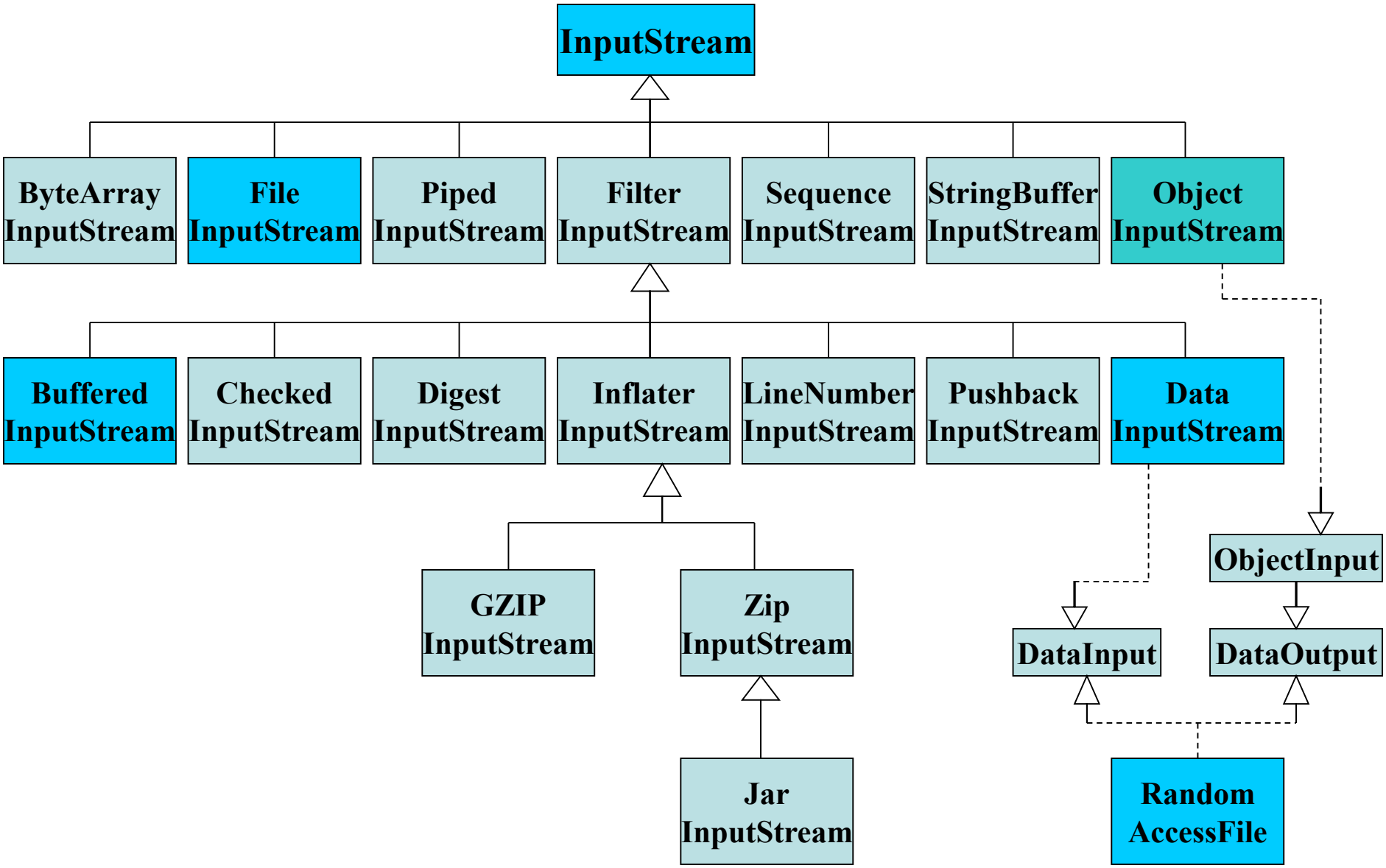
InputStream和OutputStream是抽象类,不能直接使用。(java.io)

定义如下(见JDK API)

public abstract class InputStream(OutputStream) extends Object

注:其他字节流类都是这两个类的子类。

InputStream类的子类：



InputStream类的主要方法：

三个基本的从流中读数据的方法.

- int read()：读一个字节。 范围0~255，1 byte=8 bit， ASCII表
- int read(byte[] b)：读多个字节到数组b中。
- int read(byte[] b,int off,int len)：读多个字节到数组b中从off开始长度为len的位置。 off→offset偏移坐标

- 其它的方法：
  - long skip(long n)：跳过流中若干字节数。
  - int available()：返回流中可用字节数。
  - void mark(int readlimit)：在流中标记一个位置。
  - void reset()：返回到标记过的位置。
  - boolean markSupport()：是否支持标记和复位操作。
  - void close()：关闭流。
- Reader类的方法与InputStream类似。

编码表与字节

单字节byte，8位bit

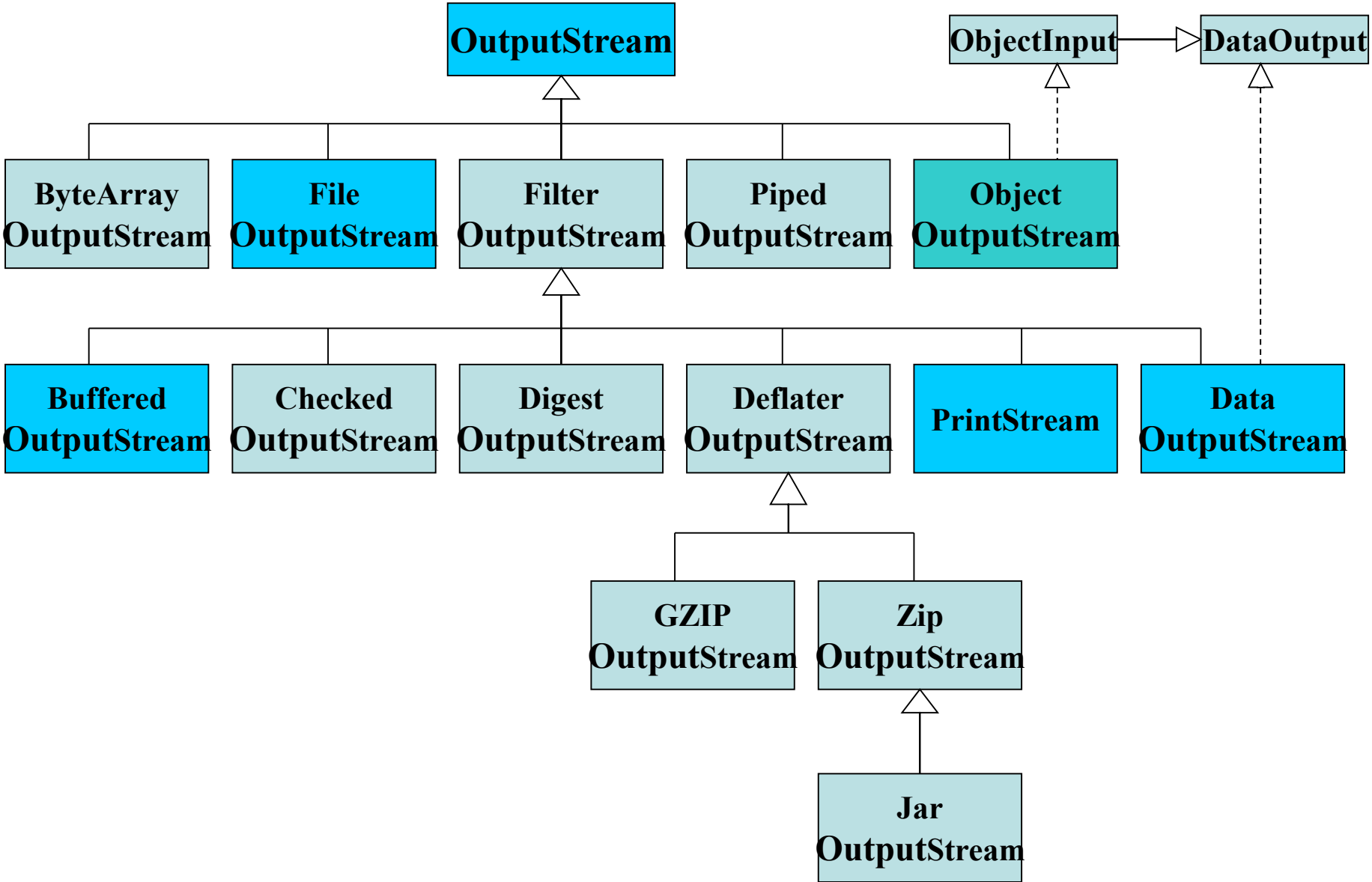
ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符 0000 0 十进制	字符	ctrl	代码	字符解释	十进制	ASCII 打印字符
0000	0	0	BLANK NULL	^@	NUL	空	16	
0001	1	1	☺	^A	SOH	头标开始	17	
0010	2	2	☻	^B	STX	正文开始	18	
0011	3	3	♥	^C	ETX	正文结束	19	
0100	4	4	♦	^D	EOT	传输结束	20	
0101	5	5	♣	^E	ENQ	查询	21	
0110	6	6	♠	^F	ACK	确认	22	
0111	7	7	•	^G	BEL	震铃	23	
1000	8	8	◘	^H	BS	退格	24	
1001	9	9	○	^I	TAB	水平制表符	25	
1010	A	10	◙	^J	LF	换行/新行	26	
1011	B	11	♂	^K	VT	竖直制表符	27	
1100	C	12	♀	^L	FF	换页/新页	28	
1101	D	13						
1110	E	14						
1111	F	15						

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	癘 7646	癙 7647	癚 7648	癛 7649	癜 764A	癝 764B	癠 764E	癡 764F	癢 7650	癣 7651	癤 7652	癥 7653	癦 7655	癧 7657	癨 7658	
50	癩 7659	癪 765A	癫 765B	癭 765D	癮 765F	癯 7660	癰 7661	癱 7662	癲 7664	癳 7665	癴 7666	癵 7667	癶 7668	癷 7669	癸 766A	癹 766C
60	癿 766D	皀 766E	皁 7670	皂 7671	皃 7672	的 7673	皅 7674	皆 7675	皇 7676	皈 7677	皉 7679	皊 767A	皌 767C	皍 767F	皎 7680	皏 7681
70	皐 7683	皑 7685	皒 7689	皓 768A	皔 768C	皕 768D	皖 768F	皗 7690	皘 7692	皙 7694	皚 7695	皛 7697	皜 7698	皝 769A	皞 769B	
80	皟 769C	皠 769D	皡 769E	皢 769F	皣 76A0	皤 76A1	皥 76A2	皦 76A3	皧 76A5	皨 76A6	皩 76A7	皪 76A8	皫 76A9	皬 76AA	皭 76AB	皮 76AC
90	皯 76AD	皰 76AF	皱 76B0	皲 76B3	皳 76B5	皴 76B6	皵 76B7	皶 76B8	皷 76B9	皸 76BA	皹 76BB	皺 76BC	皻 76BD	皼 76BE	皽 76C0	盀 76C1
A0	盁 76C3	啊 554A	阿 963F	埃 57C3	挨 6328	哎 54CE	唉 5509	哀 54C0	皑 7691	癌 764C	蔼 853C	矮 77EE	艾 827E	碍 788D	爱 7231	隘 9698
B0	鞍 978D	氨 6C28	安 5B89	俺 4FFA	按 6309	暗 6697	岸 5CB8	胺 80FA	案 6848	肮 80AE	昂 6602	盎 76CE	凹 51F9	敖 6556	熬 71AC	翱 7FF1
C0	袄 8884	傲 50B2	奥 5965	懊 61CA	澳 6FB3	芭 82AD	捌 634C	扒 6252	叭 53ED	吧 5427	笆 7B06	八 516B	疤 75A4	巴 5DF4	拔 62D4	跋 8DCB
D0	靶	把	耙	坝	霸	罢	爸	白	柏	百	摆	佰	败	拜	稗	斑

双字节byte，16位bit，编码范围为 8140-FEFE
共23940 个码位，共收录汉字21003 个

OutputStream类的子类



OutputStream类的主要方法：

三个基本的向流中写数据的方法：

- abstract void write(int b)：将一个字节输出到流中。
- void write(byte b[])：将数组中的数据输出到流中。
- void write(byte b[], int off,int len)：将数组b中从off开始len长度的数据输出到流中。

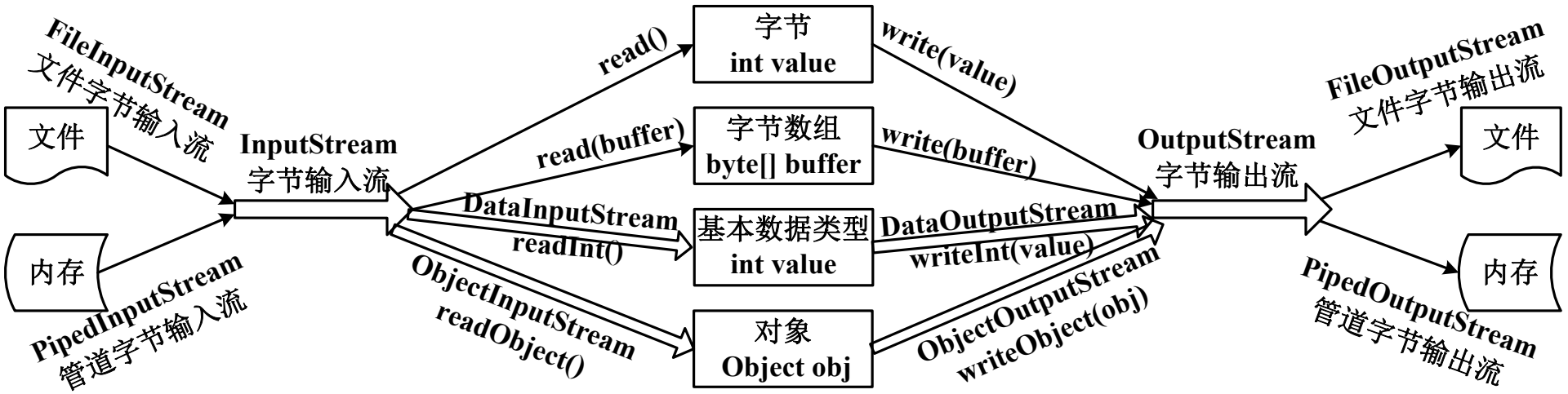
- 其它方法：
  - void flush()：将缓冲区中的数据强制送出。
  - void close()：关闭流。
- Writer类的方法与OutputStream类似。

PrintStream类的主要方法：

- 两个常用的打印方法：
  - **void print(…)**：对于各种输入参数类型都有对应的重载函数，所以能输出各种形式的数据。如：

    print(String s); print(char c);
  - **void println(…)**：类似于**print**方法，但在输出的最后自动换行。
- 两个基本的打印方法：
  - **void write(byte[] buf, int off, int len)**：写入多个字节。
  - **void write(int b)**：写入一个字节。
- 其他的方法有：
  - **void close()**：关闭流。
  - **void flush()**：将缓冲区中的数据强制送出。

各种字节流的读/写方法


FileInputStream
文件字节输入流
文件
内存
PipedInputStream
管道字节输入流
InputStream
字节输入流
read()
read(buffer)
DataInputStream
readInt()
ObjectInputStream
readObject()
字节
int value
字节数组
byte[] buffer
基本数据类型
int value
对象
Object obj
write(value)
write(buffer)
DataOutputStream
writeInt(value)
ObjectOutputStream
writeObject(obj)
OutputStream
字节输出流
FileOutputStream
文件字节输出流
文件
内存
PipedOutputStream
管道字节输出流

E:基于字符流的输入输出基类

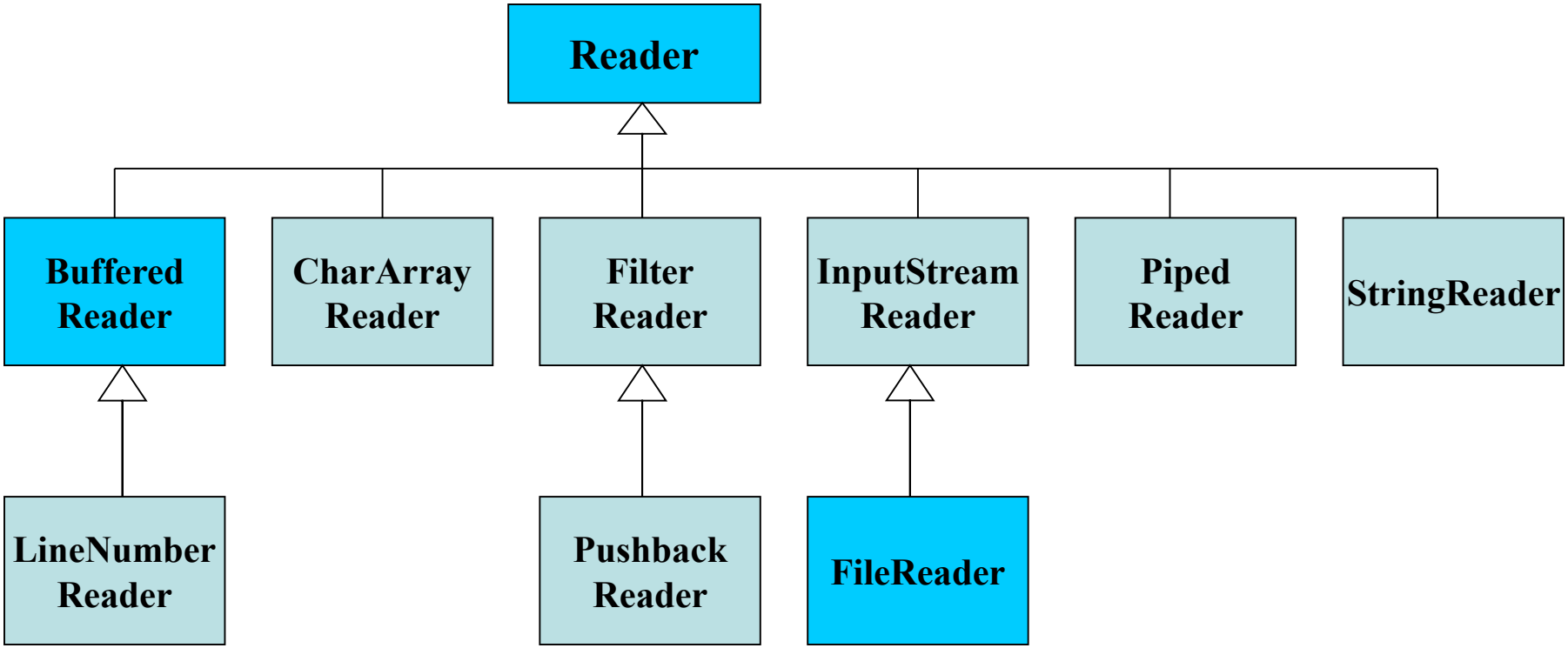
- Unicode流的基本功能依赖于Reader和Writer。这些流用于处理双字节的Unicode字符，而不是单字节字符。 范围0~65535 (0x00-0xffff)

Reader和Writer是抽象类,不能直接使用。(java.io)

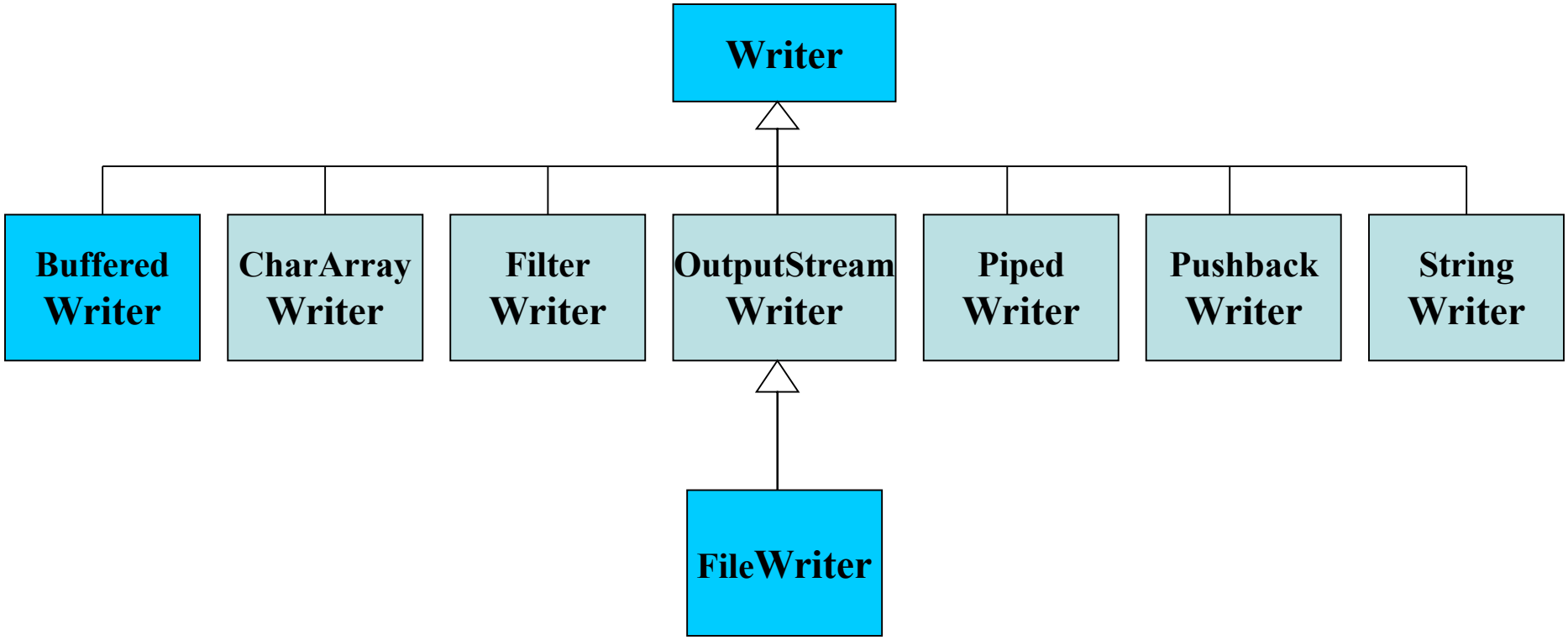
Public abastract class Reader(Writer) extends Object

其他Unicode流类都是这两个类的子类。

Reader类的子类：


Reader
Buffered Reader
CharArray Reader
Filter Reader
InputStream Reader
Piped Reader
StringReader
LineNumber Reader
Pushback Reader
FileReader

Writer类的子类：


Writer
Buffered Writer
CharArray Writer
Filter Writer
OutputStream Writer
Piped Writer
Pushback Writer
String Writer
FileWriter

字符流与字节流的区别

- 字节流操作的基本单元为字节；字符流操作的基本单元为Unicode码元。
- 字节流通常用于处理二进制数据，实际上它可以处理任意类型的数据；字符流通常处理文本数据，它支持写入及读取Unicode码元。
- 字节流默认不使用缓冲区；字符流通常使用缓冲区（编码/解码）。

- **E：读文件**

- 读文件要定义一个**FileInputStream**类型的输入流。

```
import java.io.*;
public class FileInput
{
    public static void main(String[] args) throws IOException //异常
    {
        FileInputStream in = new FileInputStream("FileTest.java");          //创建流
        int length = in.available();   //获得流中字节数(文件长度)
        char c;
        for(int i = 0; i < length; i++)
        {           c = (char)in.read(); //读取一个字节(实验一下汉字)
                    System.out.print(c);
        }
        in.close();          //关闭流
    }
}
```

- **F：写文件**

写文件要定义一个**FileOutputStream**类型的输出流。

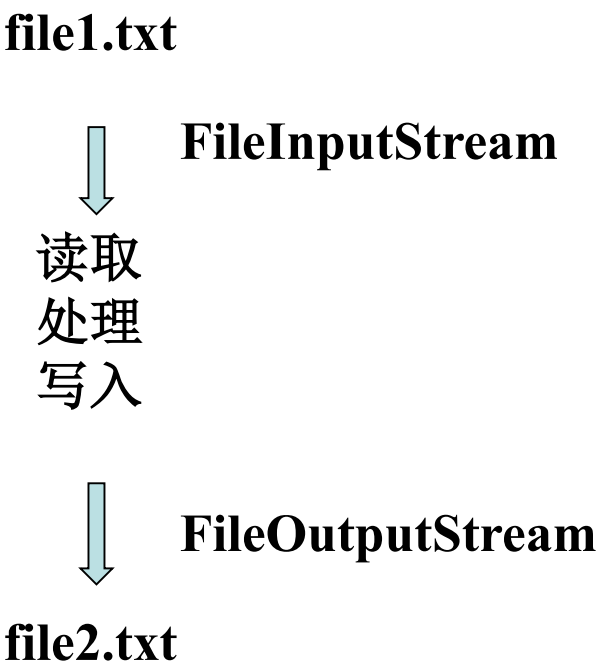
```
import java.io.*;
public class FileOutput
{
    public static void main(String[] args) throws IOException
                                                    //异常
    {
        FileOutputStream out = new FileOutputStream("abc.txt");
//创建流
        for(int i = 'a'; i <= 'z'; i++)
        {
            out.write(i);       //写入一个字节
        }
        out.close();        //关闭流
    }
}
```

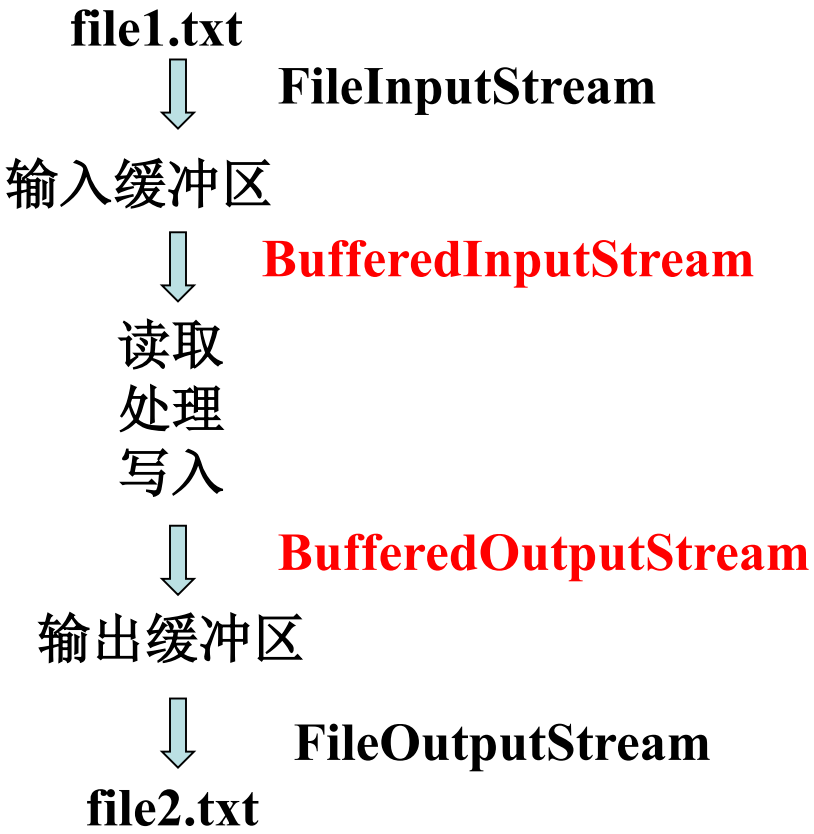
- **G:缓冲区**

【类比】搬砖，一趟一块，一趟多块

读写文件时增加缓冲区的优势：减少访问硬盘的次数，提高读写效率。
缓冲区填满时，将若干字节一次性发送到相应设备，中间也可flush()

不设缓冲区	设置缓冲区

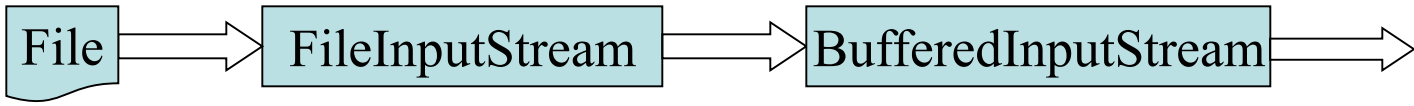




带缓冲区读文件：

读文件要定义一个**FileInputStream**类型的输入流。

```
import java.io.*;
public class FileInput
{   public static void main(String[] args) throws IOException
    { FileInputStream in = new FileInputStream("FileInput.java");
      BufferedInputStream bufIn = new BufferedInputStream(in);
      int length = bufIn.available();
      char c;
      for(int i = 0; i < length; i++)
      {           c = (char)bufIn.read();
                  System.out.print(c);
      }
      bufIn.close();
      in.close();
    }
}
```

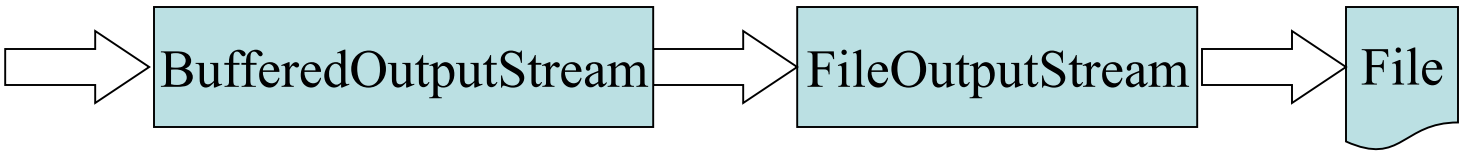


带缓冲区写文件

- 写文件要定义一个**FileOutputStream**类型的输出流。

```
import java.io.*;
public class FileOutput
{
    public static void main(String[] args) throws IOException
    {
      FileOutputStream out = new FileOutputStream("abc.txt");
      BufferedOutputStream bufOut = new BufferedOutputStream(out);
      for(int i = 'a'; i <= 'z'; i++)
      {
                bufOut.write(i);
      }
      bufOut.close();
      out.close();
    }
}
```

【断点对比】FileOutput不带缓冲区写入abc.txt



文件读写总结:

- 文件读写有关的类都在java.io包中。
- 四个基本类：
  - InputStream, OutputStream：字节流读写
  - Reader, Writer：Unicode流(字符流)读写
- 常用类：
  - FileInputStream, FileOutputStream：文件读写类
  - BufferedInputStream, BufferedOutputStream：缓冲区读写类
  - DataInputStream, DataOutputStream：数据读写类
  - RandomAccessFile：随机读写文件类

其他用法

- ChineseFileInput
- ChineseBufferFileInput
- BufferReaderFromConsole

New Java IO

- Java NIO（New IO）可以替代标准IO API
- Java NIO提供了与标准IO不同的工作方式：

Java NIO: Channels and Buffers（通道和缓冲区）

标准的IO基于字节流和字符流进行操作的，而NIO是基于通道（Channel）和缓冲区（Buffer）进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中

Java NIO: Non-blocking IO（非阻塞IO）

Java NIO可以让你非阻塞的使用IO，例如：当线程从通道读取数据到缓冲区时，线程还是可以进行其他事情。当数据被写入到缓冲区时，线程可以继续处理它。从缓冲区写入通道也类似。

Java NIO: Selectors（选择器）

Java NIO引入了选择器的概念，选择器用于监听多个通道的事件（比如：连接打开，数据到达）。因此，单个的线程可以监听多个数据通道。

资源链接：**https://www.ibm.com/developerworks/cn/education/java/j-nio/j-nio.html#**
http://tutorials.jenkov.com/java-nio/index.html
http://ifeve.com/java-nio-all/

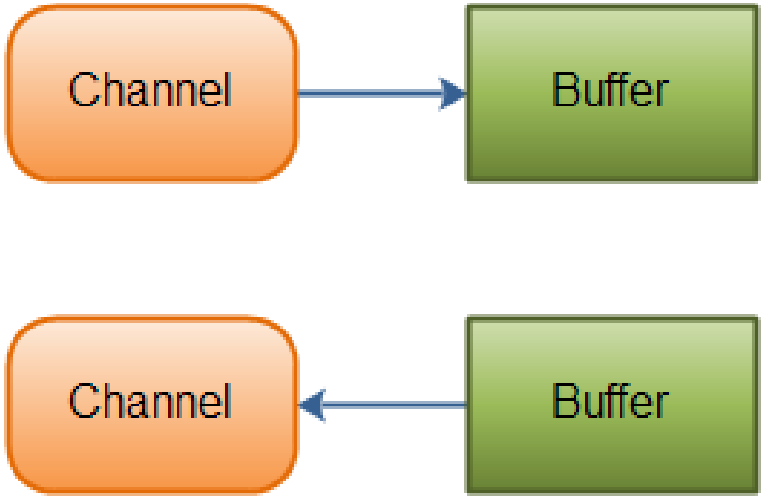
Java NIO核心组件

A ： Channel，Buffer

所有的 IO 在NIO 中都从一个Channel 开始。

Channel 有点象流。

数据可以从Channel读到Buffer中，也可以从Buffer 写到Channel中



Channel 的实现类：

–FileChannel-----文件中读写数据

–DatagramChannel----通过UDP读写网络中的数据

–SocketChannel--------通过TCP读写网络中的数据

–ServerSocketChannel-----可以监听新的TCP连接请求。

Buffer 的实现类：

- ByteBuffer
- CharBuffer
- DoubleBuffer
- FloatBuffer
- IntBuffer
- LongBuffer
- ShortBuffer

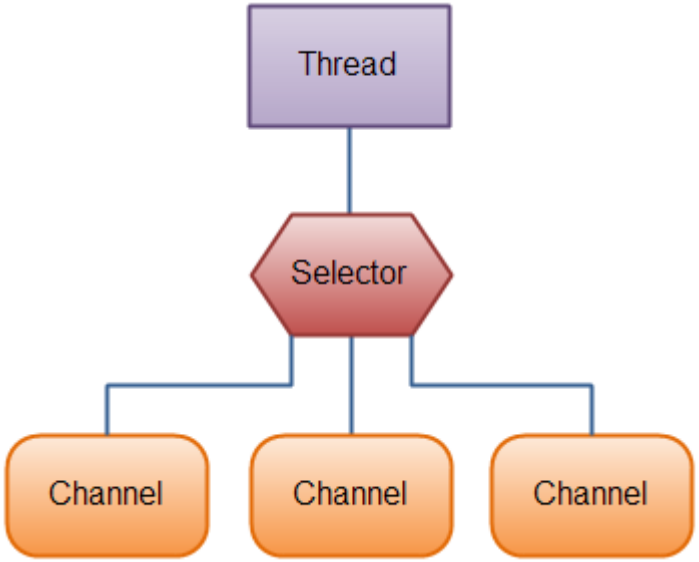
Java NOI核心组件

A ： Selector

Selector允许单线程处理多个 Channel。

如果你的应用打开了多个连接（通道），但每个连接的流量都很低，使用Selector就会很方便。

要使用Selector，得向Selector注册Channel，然后调用它的select()方法



一个单线程中使用一个Selector处理3个Channel

Buffer的具体使用方法

- Java NIO中的Buffer用于和NIO Channel进行交互
- 缓冲区本质上是一块可以写入数据，然后可以从中读取数据的内存。这块内存被包装成NIO Buffer对象，并提供了一组方法，用来方便的访问该块内存。
- 使用Buffer读写数据一般需要4个步骤：
  写入数据到Buffer
  调用flip()方法切换到读模式
  从Buffer中读取数据
  调用clear()方法或者compact()方法

备注：clear()方法会清空整个缓冲区。

compact()方法只会清除已经读过的数据。任何未读的数据都被移到缓冲区的起始处，新写入的数据将放到缓冲区未读数据的后面

Buffer的三大属性

- Capacity　表示最大容量。不论读写，含义相同
- Position---

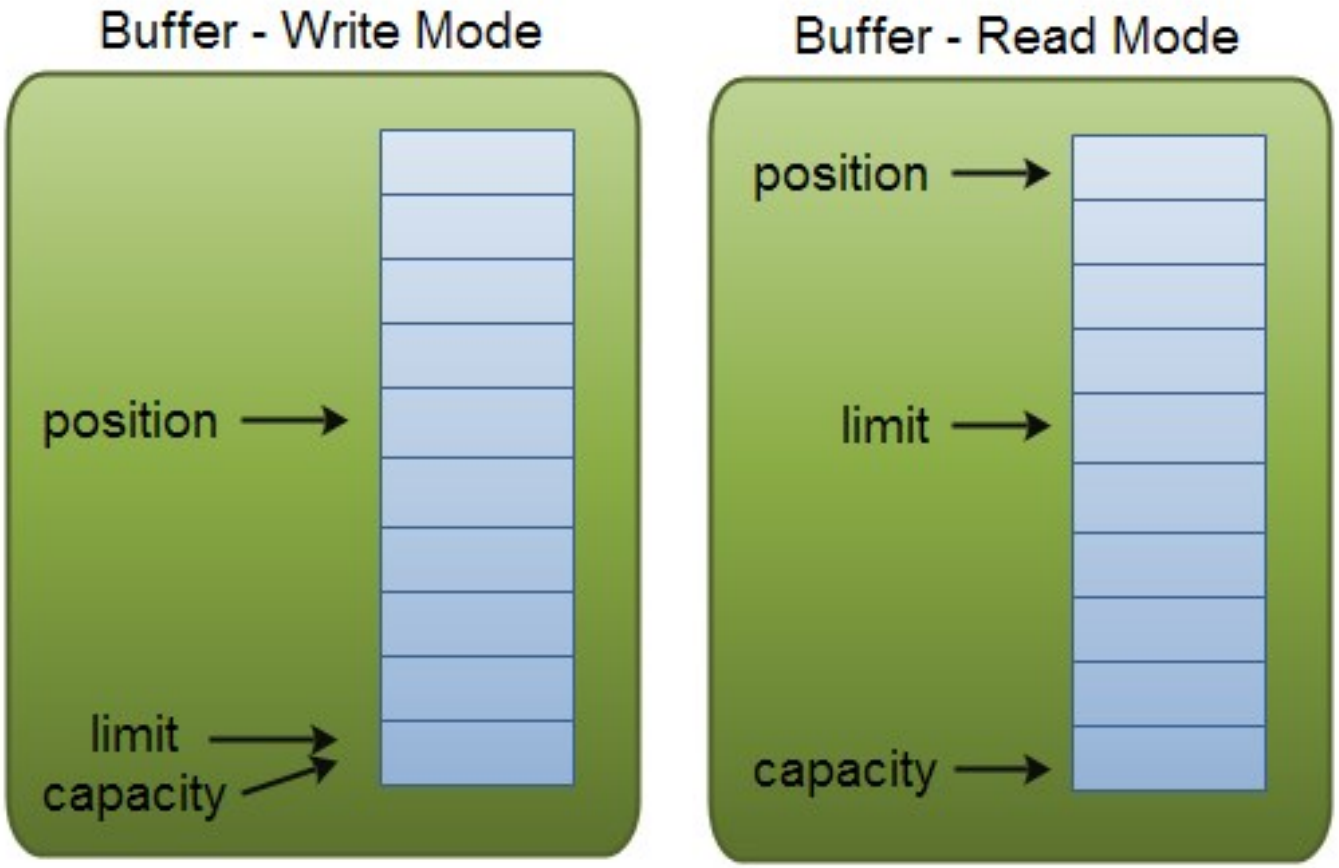
  写数据时，position表示当前的位置。初始的position值为0. position最大可与capacity相同

  读取数据时，也是从某个特定位置读。当将Buffer从写模式刚切换到读模式，position会被重置为0。当从Buffer的position处读取数据时，position向前移动到下一个可读的位置。
- Limit---

  在写模式下，Buffer的limit表示你最多能往Buffer里写多少数据。 写模式下，limit等于Buffer的capacity。

  当切换Buffer到读模式时， limit表示你最多能读到多少数据。因此，当切换Buffer到读模式时，limit会被设置成写模式下的position值。

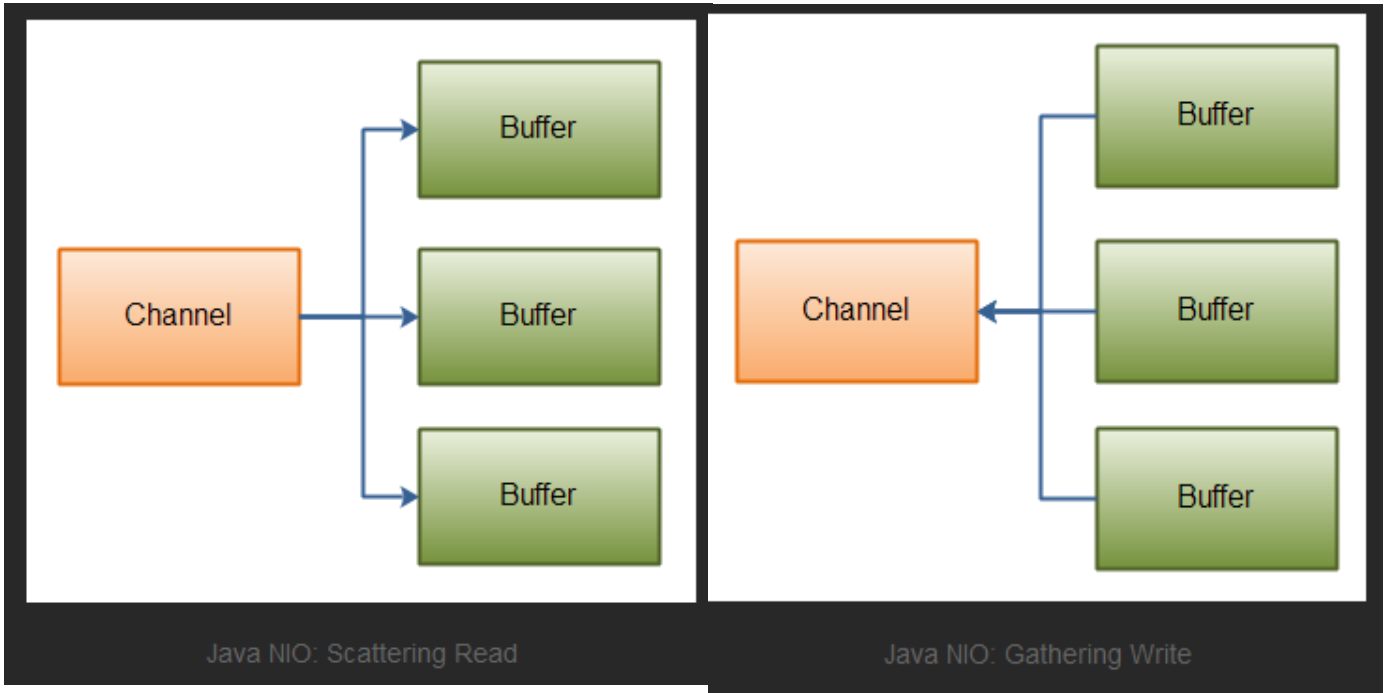
Buffer属性示意图



NIO之Scatter和Gather

分散（scatter）----从Channel中读取是指在读操作时将读取的数据写入多个buffer中。因此，Channel将从Channel中读取的数据“分散（scatter）”到多个Buffer中。

聚集（gather）-----写入Channel是指在写操作时将多个buffer的数据写入同一个Channel，因此，Channel 将多个Buffer中的数据“聚集（gather）”后发送到Channel。



通道与流的区别

- 通道是双向的
- 流只是在一个方向上移动(一个流必须是 InputStream 或者OutputStream 的子类)
- 通道 可以用于读、写或者同时用于读写