Java9:输入输出流

授课教师：邱元杰　电子邮箱：yuanjiq@126.com，微信电话: 13679081552

第9章 输入输出流

I/O基本原理

文件及文件I/O

字节流和字符流

管道输入输出流类

I/O基本原理

- 大部分程序都需要输入/输出处理，比如从键盘读取数据、向 屏幕中输出数据、从文件中读或者向文件中写数据、在一个网络 连接上进行读写操作等。
- 在Java中，对数据的输出输入操作以流的方式进行，Java SDK提供了各种各样的流用以获取不同种类的数据。

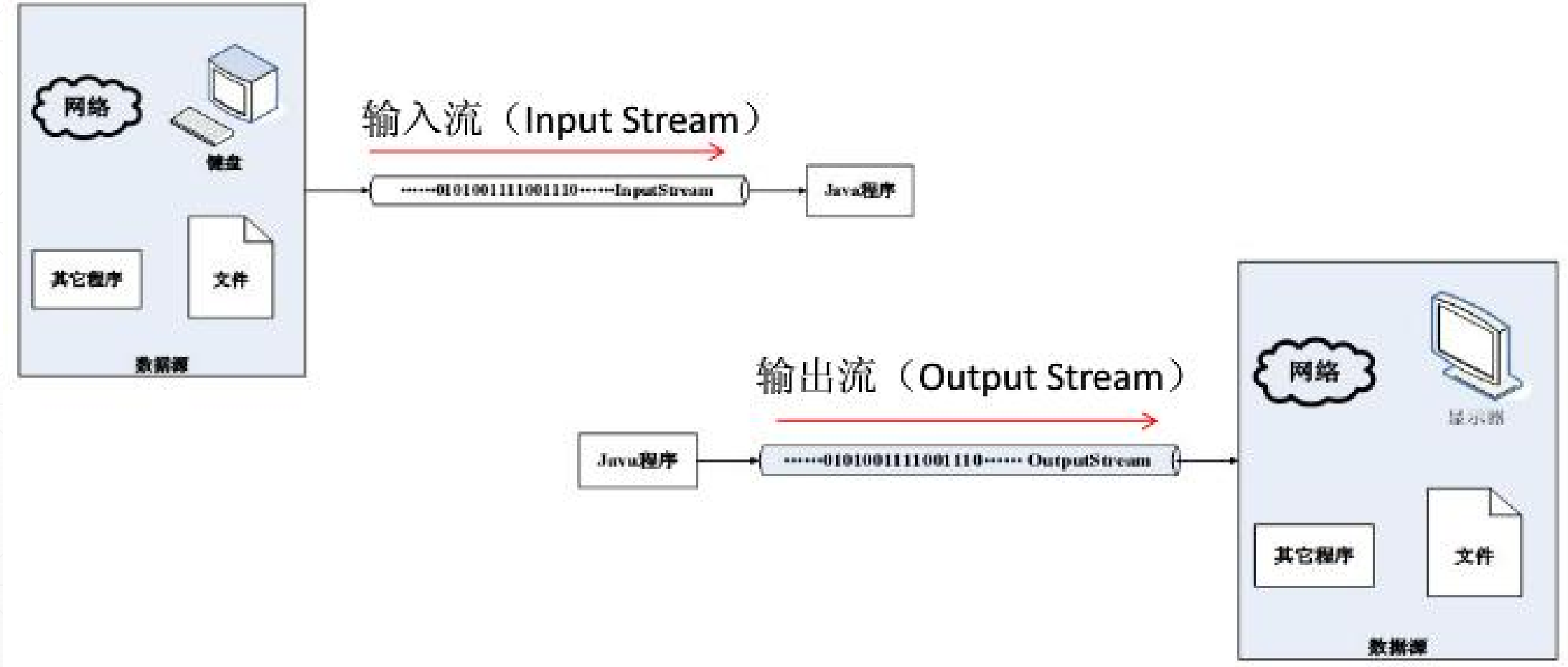
I/O基本原理

- 流(stream)是指在计算机的输入与输出之间运动的数据序列。
- 流序列中的数据既可以是未经加工的原始的二进制数据，也可以是经一定编码处理后符合某种格式规定的特定数据。
- 流通过Java的输入/输出系统与物理设备链接。尽管与它们链接的物理设备不尽相同，所有流的行为具有同样方式。
- 在Java中，把不同类型的输入、输出源(键盘、文件、网络等)抽象为流(Stream)，而其中输入或输出的数据则称为数据流(Data Stream)，用统一的方式来表示。

流分类：输入、输出流

- 流的方向是重要的，根据流的方向，流可分为两类：输入流和输出流。
- 输入流只能从中读取数据，而不能向其写出数据；输出流只能向其写出数据，而不能从中读取数据。
- 流的源端和目的端可简单地看成是字节的生产者和消费者。
  - 对输入流，可不必关心它的源端是什么，只要简单地从流中读数据。
  - 对输出流，也可不知道它的目的端，只是简单地往流中写数据。

流分类：输入、输出流


网络
键盘
其它程序
文件
数据源
输入流（Input Stream）
……01010011110011110……InputStream
Java程序
输出流（Output Stream）
Java程序
……01010011110011110…… OutputStream
网络
显示器
其它程序
文件
数据源

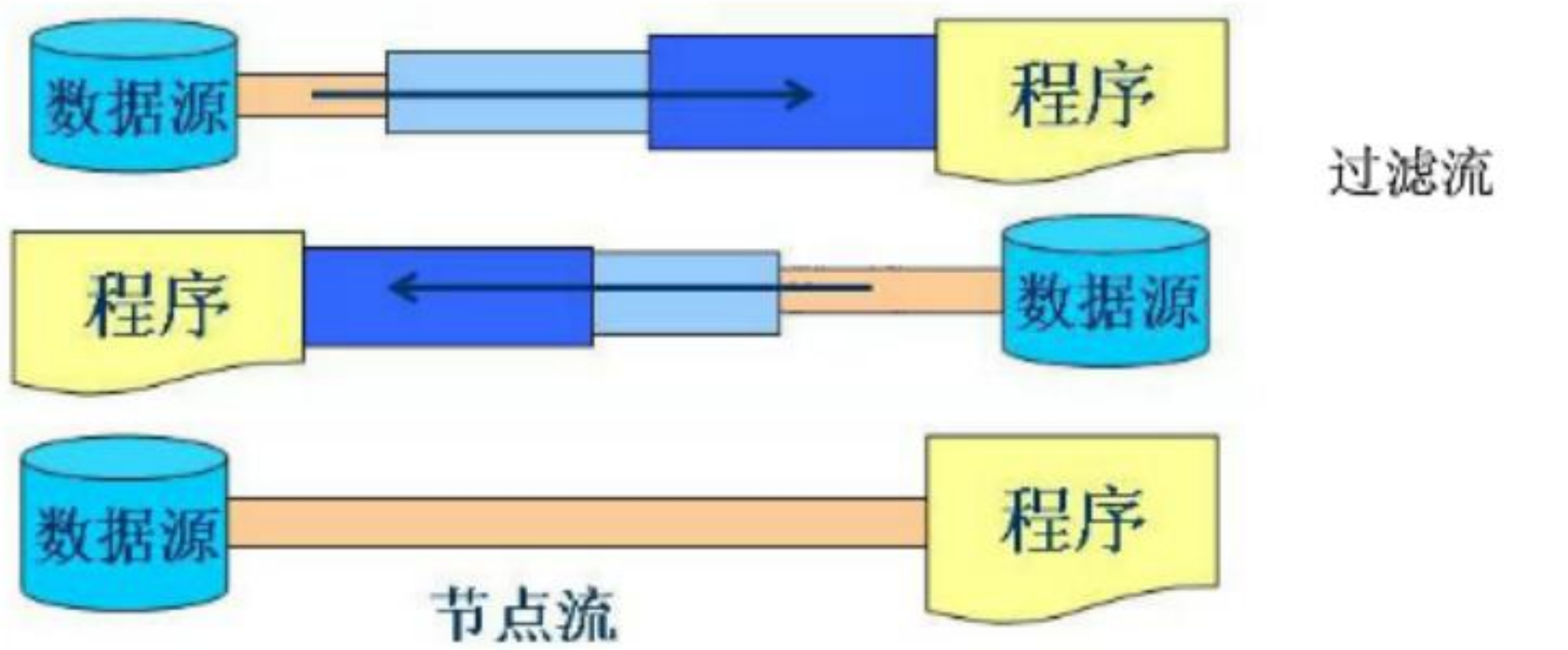
流分类：字节流、字符流

- Java把处理二进制数据的流称为字节流，字节流每次处理 一个字节的数据；以Stream结尾，说明是一字节流。
- 把处理某种格式的特定数据称为字符流，字符流每次处理一个字符的数据。以Reader Writer结尾，说明是一字符流。
- 在最底层，所有的输入/输出都是字节形式的。基于字符的流只为处理字符提供方便有效的方法。

流分类：节点流、过滤器

- 节点流(node stream)是指直接从指定的位置（如磁盘文件或内存区域）读或写。其它的流则称为过滤器。
- 过滤器(filters)输入流往往是以其它输入流作为它的输入源，经过过滤或处理后再以新的输入流的形式提供给用户，过滤器输出流的原理也类似。

流分类：节点流、过滤器

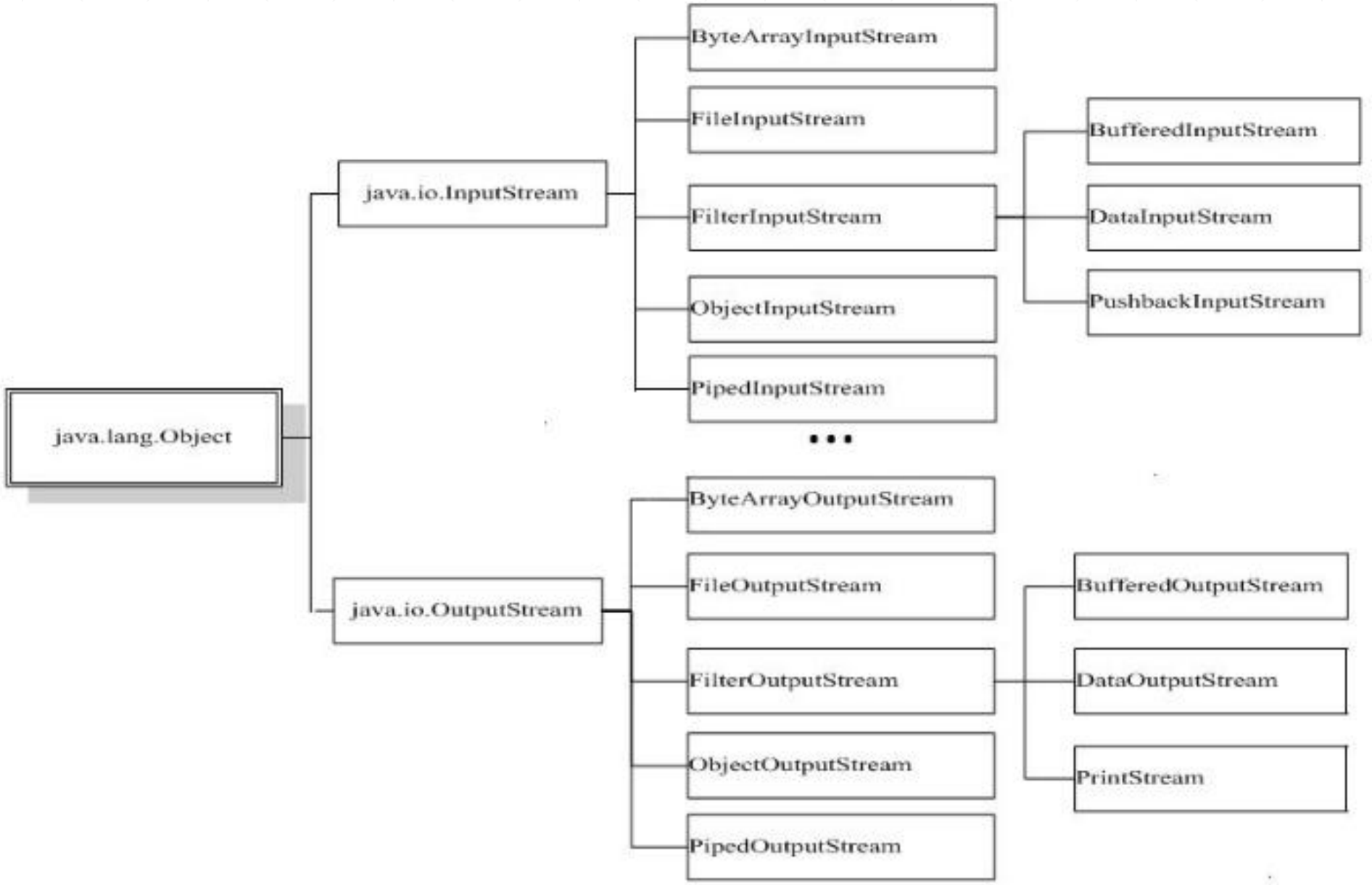


java中的流

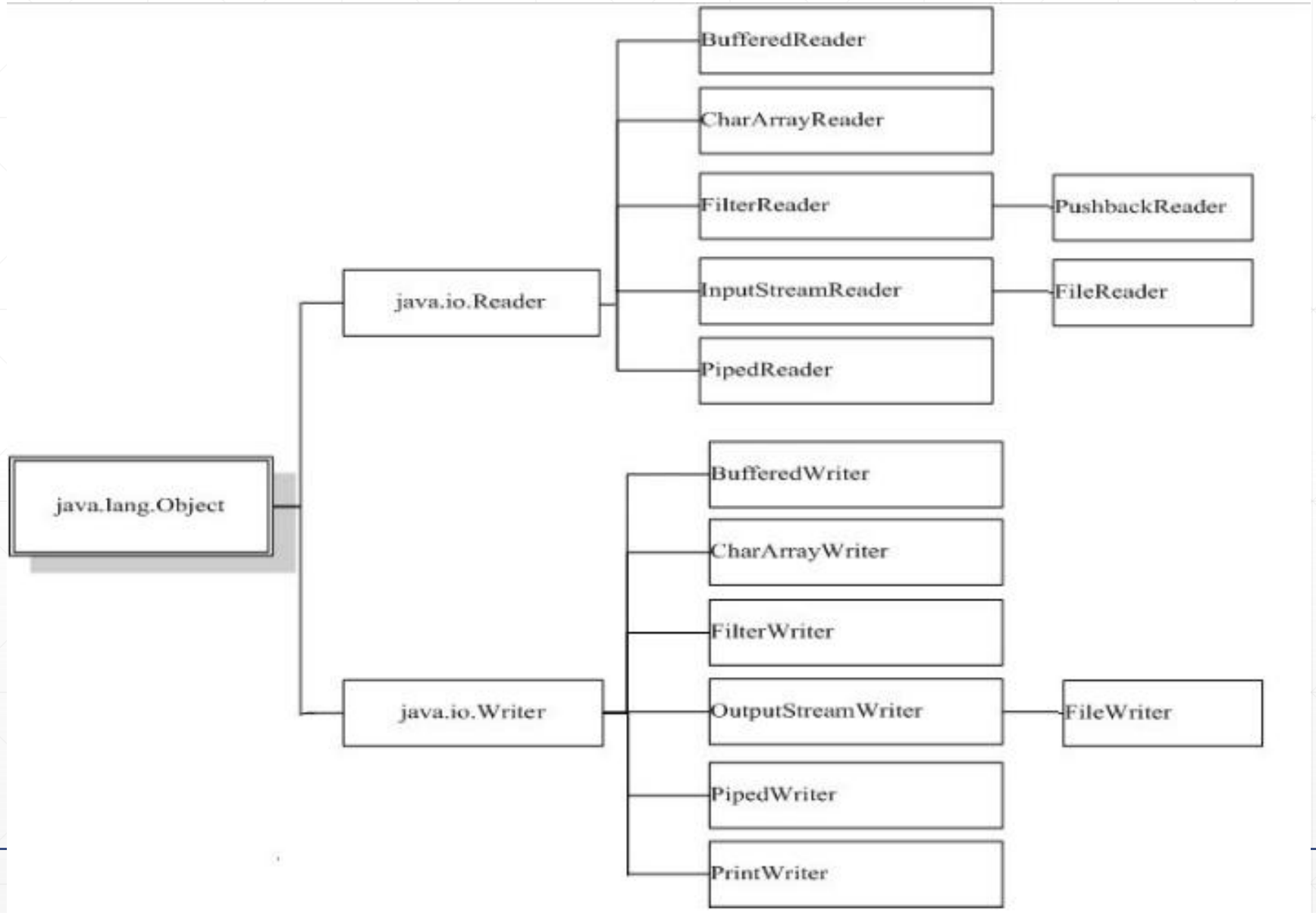
- java所提供的所有流类型位于java.io内，都分别继承自以下四种抽象流类型。

	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

输入输出流的类关系


java.lang.Object
java.io.InputStream
ByteArrayInputStream
FileInputStream
FilterInputStream
BufferedInputStream
DataInputStream
PushbackInputStream
ObjectInputStream
PipedInputStream
java.io.OutputStream
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
ObjectOutputStream
PipedOutputStream

Reader和Writer类关系



第9章 输入输出流

I/O基本原理

文件及文件I/O

字节流和字符流

管道输入输出流类

文件及文件I/O

- 操作系统的文件管理是向应用程序提供的最基本服务之一。管理任意复杂的分层目录系统结构和几乎任意长度的文件。
- 类File提供了一种与机器无关的方式来描述一个文件对象的属性。类File能够处理由本地文件系统维护的具体文件，并提供独立于平台的文件处理方法。

File类的主要方法

- 文件路径和属性
  - getPath()和getAbsolutePath()方法返回File对象的路径和绝对路径。
  - getName()方法返回File对象的文件名或目录名。
  - getParent()返回File对象的父目录。
- 表示文件的属性或状态：
  - canWrite()，canRead()，isDirectory()，isAbsolute()，exist()，isFile()都返回boolean型数据，分别表示文件是否写保护，是否读保护，是目录还是文件，是否使用绝对路径，是否存在。

FileTest1.java

File类的主要方法

- 创建目录和删除文件
  - mkdir()和mkdirs()用于创建目录。创建目录的位置完全取决于File对象的路径。
  - delete()用于删除文件或目录，删除目录时，应该保证所删目录是一个空目录，否则删除操作失败。
- 文件更名
  - renameTo()方法不但可以给文件更名，而且可以给目录更名。
  - equals()判断两个File对象是否相等，程序用它来判断用户给定的原文件名和新文件名是否相等，如果相等则不能进行更名操作。

FileTest2.java　FileRename.java

File类的主要方法

- 目录清单：
  - list()方法产生目录清单，它只返回指定目录中包含的文件名或子目录名，没有文件长度、修改时间、文件属性等信息。
  - lastModified()返回文件最后一次被修改的时间，其值是相对于1970年1月1日的时间毫秒数，为便于阅读，必须变成java.util.Date对象。

RandomAccessFile类

- RandomAccessFile类和输入输出流类具有读写文件的功能。它有两个构造器:
  - RandomAccessFile(String name, String mode)
  - RandomAccessFile(File file, String mode)
- 其中:
  - name 是一个String对象，表示被访问的文件名。
  - file 是一个File对象，表示被访问的文件名。
  - mode 用字符串表示被访问文件的读写模式："r"表示文件以只读方式打开，"rw"表示文件以读写方式打开。

RandomAccessFile类

- 以读写方式生成RandomAccessFile对象时，如果该文件不存在，则创建该文件，供程序进行读写操作，如果该文件已经存在，则以覆盖方式(不是改写方式)把输出数据写入到文件中，原文件中没有被覆盖的部分，仍然保留在文件之中。
- RandomAccessFile类实现了DataInput和DataOutput两个接口，这两个接口分别定义了读入和写出的方法。
- 它有针对简单数据类型读写方法，由带不同的后缀表示。

RandomTest.java

RandomAccessFile类

- RandomAccessFile类除具有读写文件的方法外，还定义了与文件操作相关的一些方法：
  - seek() 支持随机访问文件，它通过移动文件的读写指针实现文件的随机读写。其参数表示读写指针相对于文件起始位置的偏移量。
  - getFilePoint() 返回当前文件指针的位置。
  - close() 关闭文件和释放与打开文件有关的资源。在使用完文件或使用中出现异常后，都应该关闭文件。

第9章 输入输出流

I/O基本原理

文件及文件I/O

字节流和字符流

管道输入输出流类

字节流

- 字节流由两个类层次结构定义。在顶层有两个抽象类：InputStream 和 OutputStream。
- 每个抽象类都有多个具体的子类，这些子类对不同的外设进行处理。
  - 抽象类InputStream和OutputStream定义了实现其他流类的关键方法。
  - 最重要的两种方法是read()和write()，它们分别对数据以字节为单位进行读写。
  - 两种方法都在InputStream 和OutputStream中被定义为抽象方法，它们被派生的流类重载。

InputStream类的方法

- InputStream类提供了有关读入数据的三个读入方法：
  - int read()
    - 方法返回一个0至255之间的整数或-1，-1代表遇到了流的结束，其它对应读入的字节。
  - int read(byte[])
    - 方法则将字节读入参数给定的字节数组，返回值是实际读入的字节数或-1(遇到了流结束)。
  - int read(byte[],int,int)
    - 方法的后两个参数分别给出读入的起始位置和读入的最大字节数。

FileRead.java

InputStream类的方法

- InputStream类提供的其他方法：
  - void close()
    - 关闭当前流对象，并释放该流对象占用的资源。
  - int available()
    - 返回当前流对象中还没有被读取的字节数量。也就是获得流中数据的长度。
  - long skip(long)
    - 跳过当前流对象中的n个字节，而实际跳过的字节数量则以返回值的方式返回。

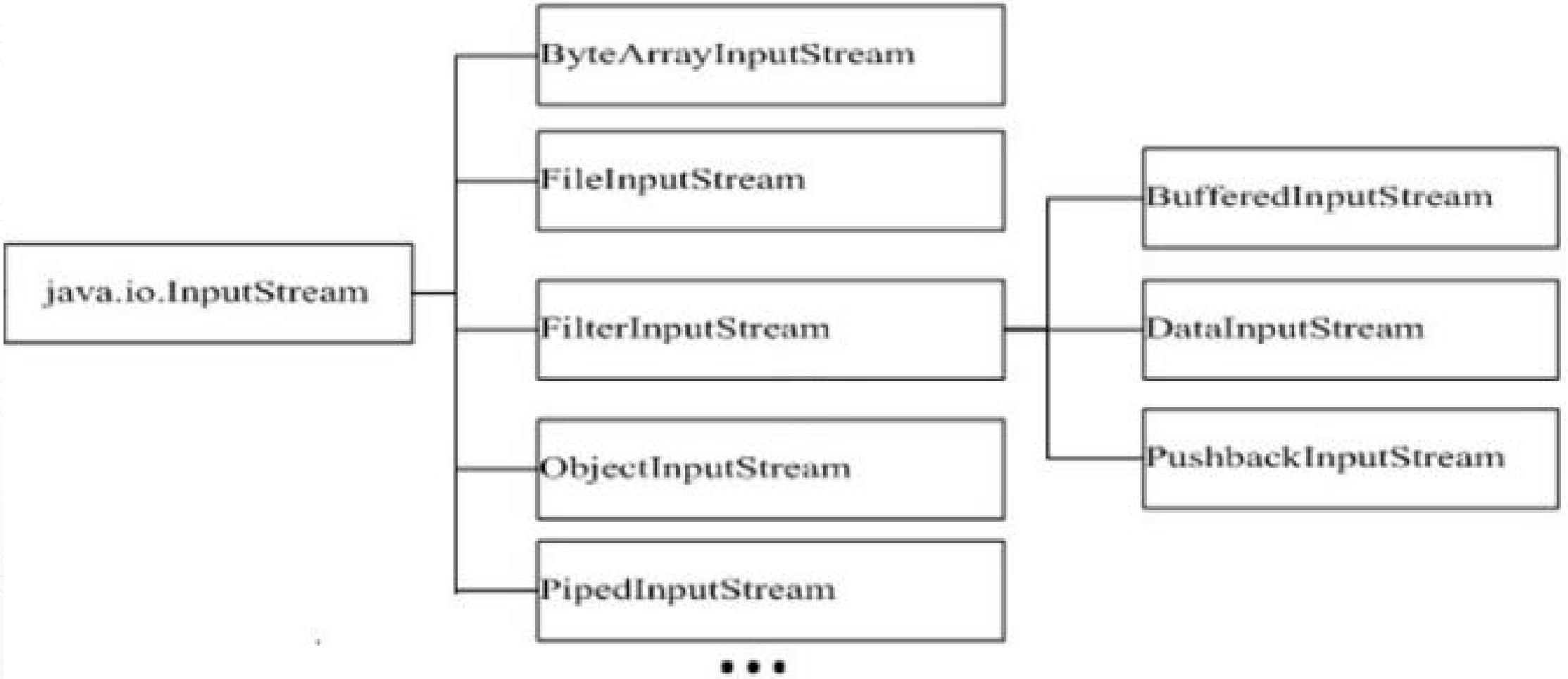
InputStream类的方法

- InputStream类提供的其他方法：
  - boolean markSupported()
    - 判断流是否支持标记(mark)。标记可以方便的回到原来读过的位置。
  - void mark(int)
    - 为流中当前的位置设置标志，使得以后可以从该位置继续读取。
  - void reset()
    - 使流读取的位置回到设定标记的位置。

InputStream的子类

- 所有InputStream的子类都是针对不同的输入数据源，其类名的前缀清楚地表示出输入数据源，如文件的数据源是FileInputStream类，PipedInputStream类的数据源是管道等等。
- FilterInputStream类及其子类是加强流，它的功能更加强大，使用更加灵活。

输入流及其子类


java.io.InputStream
ByteArrayInputStream
FileInputStream
FilterInputStream
ObjectInputStream
PipedInputStream
BufferedInputStream
DataInputStream
PushbackInputStream

OutputStream类的方法

- OutputStream类提供了有关写数据的三个方法：
  - int write (int)
    - 向流的末尾写入一个字节的数据。
  - int write (byte[])
    - 将数组b中的数据依次写入当前的流对象中。
  - int write (byte[],int,int)
    - 将数组中从开始下标(包含)，后续长度的数据依次写入到流对象中。

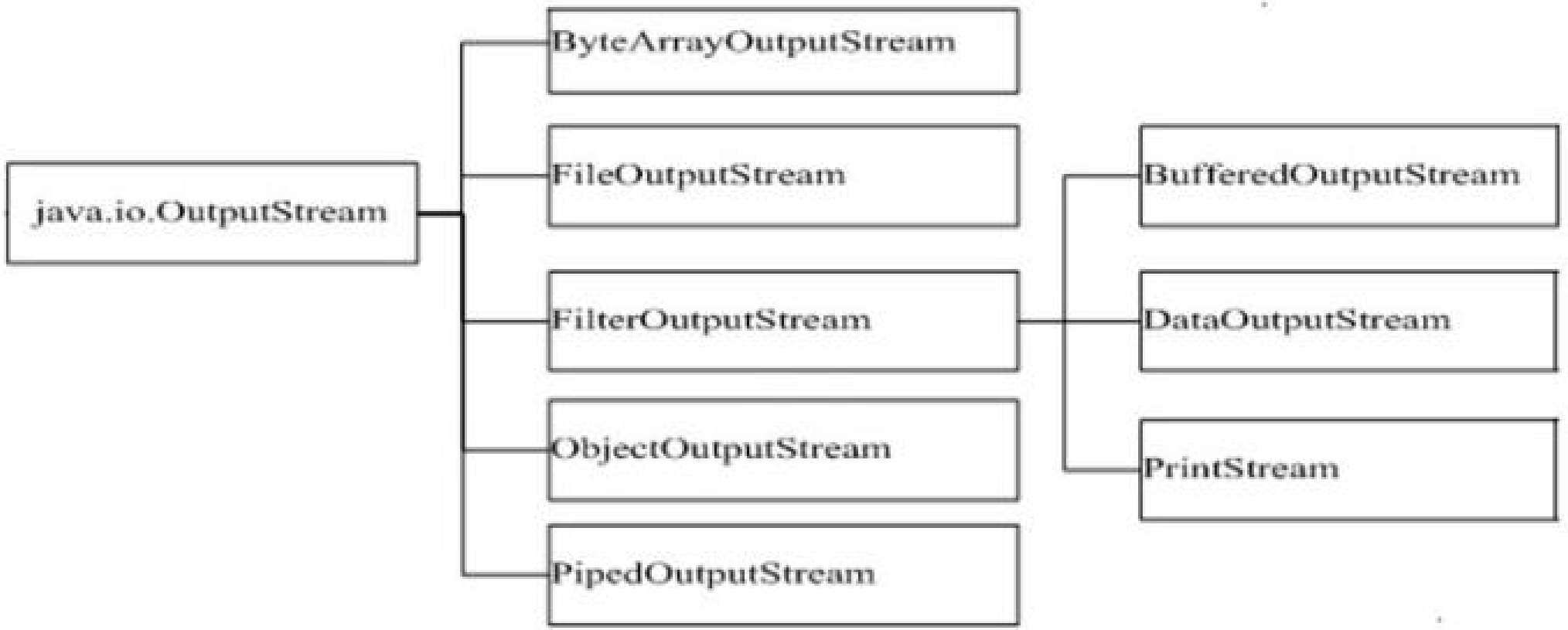
OutputStream类的方法

- OutputStream类提供的其他方法：
  - void close()
    - 关闭当前流对象，并释放该流对象占用的资源。
  - void flush()
    - 将当前流对象中的缓冲数据强制输出出去。使用该方法可以实现立即输出。

OutputStream的子类

- 所有OutputStream的子类与InputStream的子类相似，针对不同的输出数据源，其类名的前缀清楚地表示出输出数据源，如文件的数据源是FileOutputStream类，PipedOutputStream类的数据源是管道等等。
- FilterInputStream类及其子类是加强流，它的功能更加强大，使用更加灵活。

OutputStream的子类



FileInputStream和FileOutputStream类

- 这两个类属于节点流，分别完成对文件的输入输出（即读写）操作。
- FileInputStream类的构造器有：
  - FileInputStream(String)
    - 参数String对象表示文件名。
  - FileInputStream(File)
    - 参数File对象表示文件名。
  - FileInputStream(FileDescriptor)
    - 参数FileDescriptor定义一个本地文件系统对象表示的文件名。

FileInputStream和FileOutputStream类

- FileOutputStream类和FileInputStream类在构造器和方法的参数定义上都十分相似，只是一个针对输入，另一个针对输出。
- FilterOutputStream类的构造器有：
- FileOutputStream(String)
- FileOutputStream(String,boolean)
- FileOutputStream(File)
- FileOutputStream(FileDescriptor)

FileInputStream和FileOutputStream类

- 生成FileOutputStream对象时，如果文件不存在，则创建该文件供程序输出数据；
- 如果文件已经存在，则有改写和附加两种输出数据的方式：
- 改写的含义是先把原文件长度截为零，原文件数据被丢弃，然后再输出数据。(第二个构造器的boolean参数为false值和其它构造器的对象)
- 附加的含义是在原文件末尾追加输出数据，原文件数据仍然存在。(第二个构造器的boolean参数为true)

BufferedInputStream类和BufferedOutputStream类

- 这两个类是过滤器流，它们在标准读写功能和应用程序之间，增加输入输出缓冲机制，从而显著地提高输入输出的速度。
- 它的构造器使用输入流对象，在生成BufferedInputStream对象时可以设置输入缓冲区的大小，也可使用缺省值，其值是2048字节。生成BufferedInputStream对象后，应用程序在读入数据时，直接取自于缓存区，从而提高读入数据的速度。

BufferTest.java

DataInputStream和DataOutputStream类

- 这两个类创建的对象分别被称为数据输入流和数据输出流。
- 它们分别实现了DataInput接口和DataOutput接口。
- 它们允许程序按与机器无关的风格读写Java数据。
- 这两个流也是过滤器流，常以其它流如InputStream或OutputStream作为它们的输入或输出。

DataInputStream和DataOutputStream类

- 它们输入和输出几乎是对应的，每种基本数据类型的读写方法可以从其后缀名字识别。
- 例：readInt()　　writeInt()
- 　readBoolean()　writeBoolean()
- 　readChar()　　writeChar()
- 　readDouble()　writeDouble()

字节流和字符流

- SequenceInputStream类
  - SequenceInputStream类可以将两个或几个输入流不露痕迹地接合在一起，生成一个长长的接合流，在读入数据时，它忽略前面几个输入流的结束符EOF， 直到最后一个流的结束符EOF时，才完成流的输入

字节流和字符流

- 字符流主要是用来处理字符的。它们在读写流内数据时是以字符为单位。
- 字符流类由两个类层次结构定义。顶层有两个抽象类：Reader和Writer。这些抽象类处理统一编码的字符流。
- 抽象类Reader和Writer定义了实现其他流类的关键方法。其中两个最重要的是read()和write()，它们分别进行字符数据的读和写。这些方法被派生流类重载。
- Reader及Writer类和它们子类的方法，与InputStream及OutputStream类及它们子类的使用方法非常类似。

字节流和字符流

- 字符流Reader类、Writer的子类：
  - InputStreamReader类和OutputStreamWriter类
    - 在构造这两个类对应的流时，它们会自动进行转换，将平台缺省的编码集编码的字节转换为Unicode字符。对英语环境，其缺省的编码集一般为ISO8859-1。
  - BufferedReader类和BufferedWriter类
    - 这两个类对应的流使用了缓冲，能大大提高输入输出的效率。这两个也是过滤器流，常用来对InputStreamReader和OutputStreamWriter进行处理。

字节流和字符流示例

- 示例：
  - 实现对文件中关键字的查找，把包含有指定字符的行显示出来或写到文件中。
  - 用FileReader类实现文件读入功能，BufferedReader类用于提高文件读入速度。与输入相似的输出类FileWriter类和BufferedWriter类。
  - 程序用readLine()方法一次读入一行字符，读入的一行字符中不包括行结束符。程序用write()方法向文件中一次写出一字符串，方法newLine()写一个行结束符。

字节流和字符流示例

```
import java.io.*;
public class Find {
  public static void main(String args[]) {
    BufferedReader bRead = null;
    BufferedWriter bWrite = null;
    String buffer;
    boolean outFile = false;
    int i = 0;
    if(args.length < 2) {
      System.out.println("Use:java Find <input> <String>"
                         +" [output]");
      System.exit(0);
    }
```

字节流和字符流示例

```
try {
      FileReader fr = new FileReader(new File(args[0]));
      bRead = new BufferedReader(fr);
    }catch(IOException e) {
      System.out.println("Cannot find " + args[0]);
      System.exit(-1);
  }
 if(args.length == 3) {
         try {
            FileWriter fw = new FileWriter(new File(args[2]));
            bWrite = new BufferedWriter(fw);
            outFile = true;
         } catch (IOException e) {
            System.out.println("Cannot find " + args[2]);
            System.exit(-1);
         }
     }
```

字节流和字符流示例

```
try {
        buffer = bRead.readLine();
        while(buffer != null) {
          if(buffer.indexOf(args[1]) != -1) {
             if(outFile) {
                bWrite.write(buffer);
                bWrite.newLine();
             }
             System.out.println(buffer);
             i++;
          }
          buffer = bRead.readLine();
        }
    } catch(IOException e) {
        System.err.println(e);
```

字节流和字符流示例

```
} finally {
        try {
            bRead.close();
            if(bWrite != null) bWrite.close();
        } catch(Exception e) {System.err.println(e);}
    }
    System.out.println("\n\n\n----------------------");
    System.out.println("Searched completely");
    if( i != 0) {
      System.out.println("Found " + i + " lines have \""
                         + args[1] + "\"" );
    } else
      System.out.println("Can't find \""+args[1]
                         +"\"in file" + args[0]);
  }
}
```

管道输入输出流类

- PipedInputStream和PipedOutputStream类
  - 管道是UNIX的发明，它大大增强了流的概念。
  - 管道(pipe)提供一种线程之间的通信方法，可用于IPC(进程间通信) 或是ITC(线程间通信)。
  - 一个输入管道是用来接收一个输出管道所写出的数据。
  - 这两个类必须同时使用，所以它们除了不带参数的构造器外，互为构造器中的参数。
  - PipedInputStream(PipedOutputStream)
  - PipedOutputStream(PipedInputStream)

管道输入输出流类示例

- 示例：
- 下面的程序使用两个线程：一个线程模拟数据采集，用Random类随机生成数值，另一个线程模拟数据处理，使用数据，计算其平均值。

管道输入输出流类示例

```
import java.io.*;
import java.util.Random;
//数据处理
class RunningAverage extends
Thread {
  private DataInputStream in;
  double total = 0;
  long count = 0;
  RunningAverage(InputStream i) {
    in = new DataInputStream(i);
  }
```

```
public void run() {
    while (true){
     try {
       double num = in.readDouble();
       total += num;
       count++;
       System.out.println(count+":"+num+"\t
avg="+total/count);
     } catch (IOException e) { e.printStackTrace(); }
    }
  }
}
```

管道输入输出流类示例

```
//数据采集
class NumberGenerator extends Thread {
  private DataOutputStream out;
  private Random gen = new Random();
  private final long RANGE = 1000;
  NumberGenerator(OutputStream o) {
    out = new DataOutputStream(o);
  }
  public void run(){
    while (true){
     try{
      double num = gen.nextDouble()*RANGE;
      out.writeDouble(num);
      out.flush();
      sleep(500);
     }catch(IOException e){e.printStackTrace();
     }catch(InterruptedException
e){e.printStackTrace();}
    }
  }
}
```

管道输入输出流类示例

```
//主类
class PipeTest {
  public static void main(String[] args){
    try {
      PipedOutputStream producer = new PipedOutputStream();
      PipedInputStream consumer =
                        new PipedInputStream(producer);
      RunningAverage avg = new RunningAverage(consumer);
      NumberGenerator gen = new NumberGenerator(producer);
      gen.start();
      avg.start();
      try{
        Thread.sleep(5000);
      }catch (InterruptedException e) {}
      gen.stop();
      avg.stop();
```

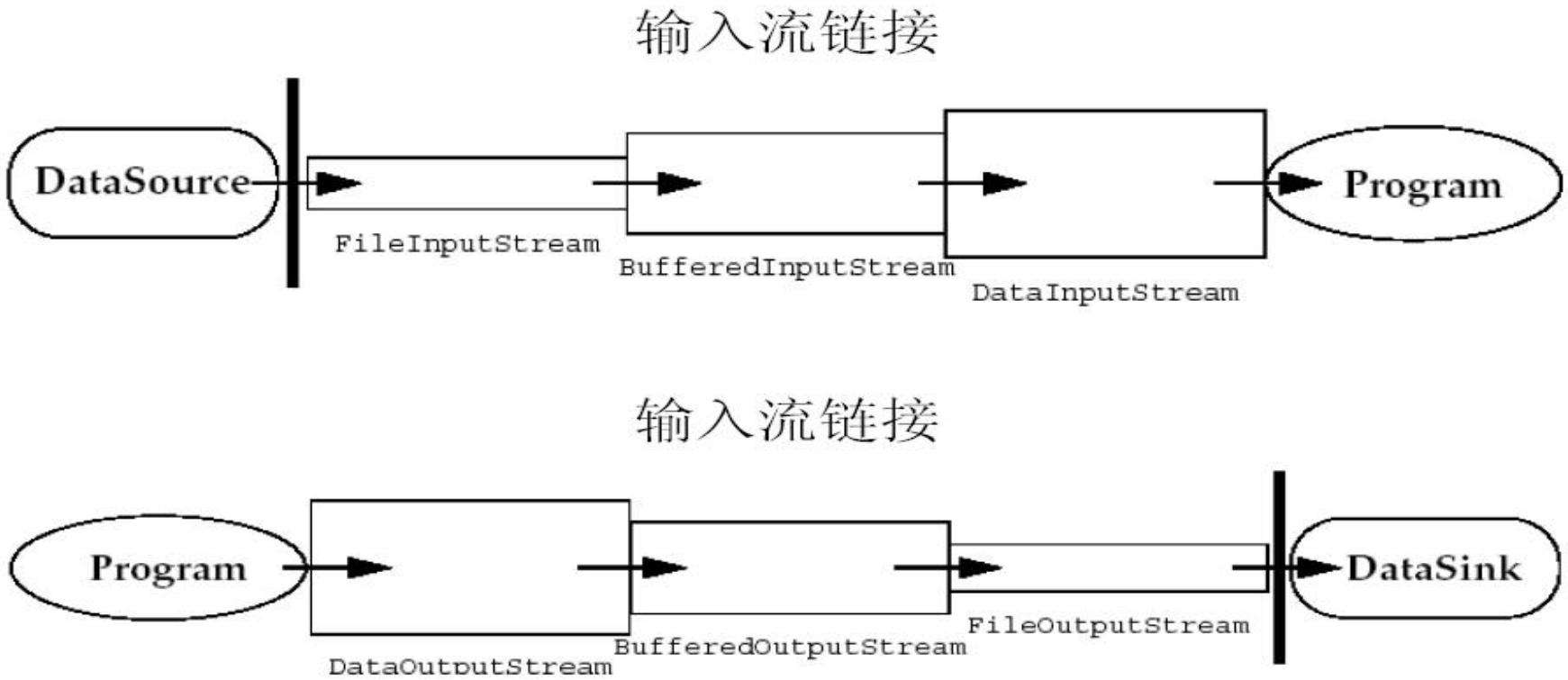
管道输入输出流类示例

```
        producer.close();
        consumer.close();
    } catch (IOException e){
        e.printStackTrace();
    }
  }
}
```

- 程序用PipedOutputStream()生成管道输出流producer，用PipedInputStream(producer)生成管道输入流consumer，同时实现了两个管道的连接。
- 类RunningAverage和NumberGenerator第一个类实现生成数据，用管道输出这些数据，后一个类从输入管道接收数据，然后计算平均值。

I/O流链接及处理流

- 将一个流的输出链接到一个流的输入，达到更好的I/O效果。如下示意图所示：



I/O流链接及处理流

➢字符流与字节处理流的关系对照表如下：

类型	字符流	字节流
缓冲	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
过滤	FilterReader FilterWriter	FilterInputStream FilterOutputStream
字节和字符间转换	InputStreamReader OutputStreamWriter	
对象序列化		ObjectInputStream ObjectOutputStream
数据转换		DataInputStream DataOutputStream
计算	LineNumberReader	LineNumberInputStream
向前查看	PushbackReader	PushbackInputStream
打印	PrintWriter	PrintStream

对象序列化处理

- “对象序列化”（Object Serialization）是Java一种特性。类实现Serializable接口。
  - 实现了Serializable接口的对象，可将它们转换成一系列字节，并可在以后完全恢复回原来的样子。
  - 这一过程可通过网络进行。这意味着序列化机制能自动补偿操作系统间的差异。
  - 可以实现“有限持久化”。意味着对象的“生存时间”并不取决于程序是否正在执行——它存在或“生存”于程序的每一次调用之间。

对象序列化处理

- 序列化一个对象：
  - 首先要创建某些OutputStream对象，然后将其封装到ObjectOutputStream对象内。
  - 再需调用writeObject()即可完成对象的序列化，并将其发送给OutputStream。
  - 相反将一个InputStream封装到ObjectInputStream内，然后调用readObject()。
  - 和往常一样，最后获得的是指向一个上溯造型Object的句柄，所以必须下溯造型，以便能够直接设置。