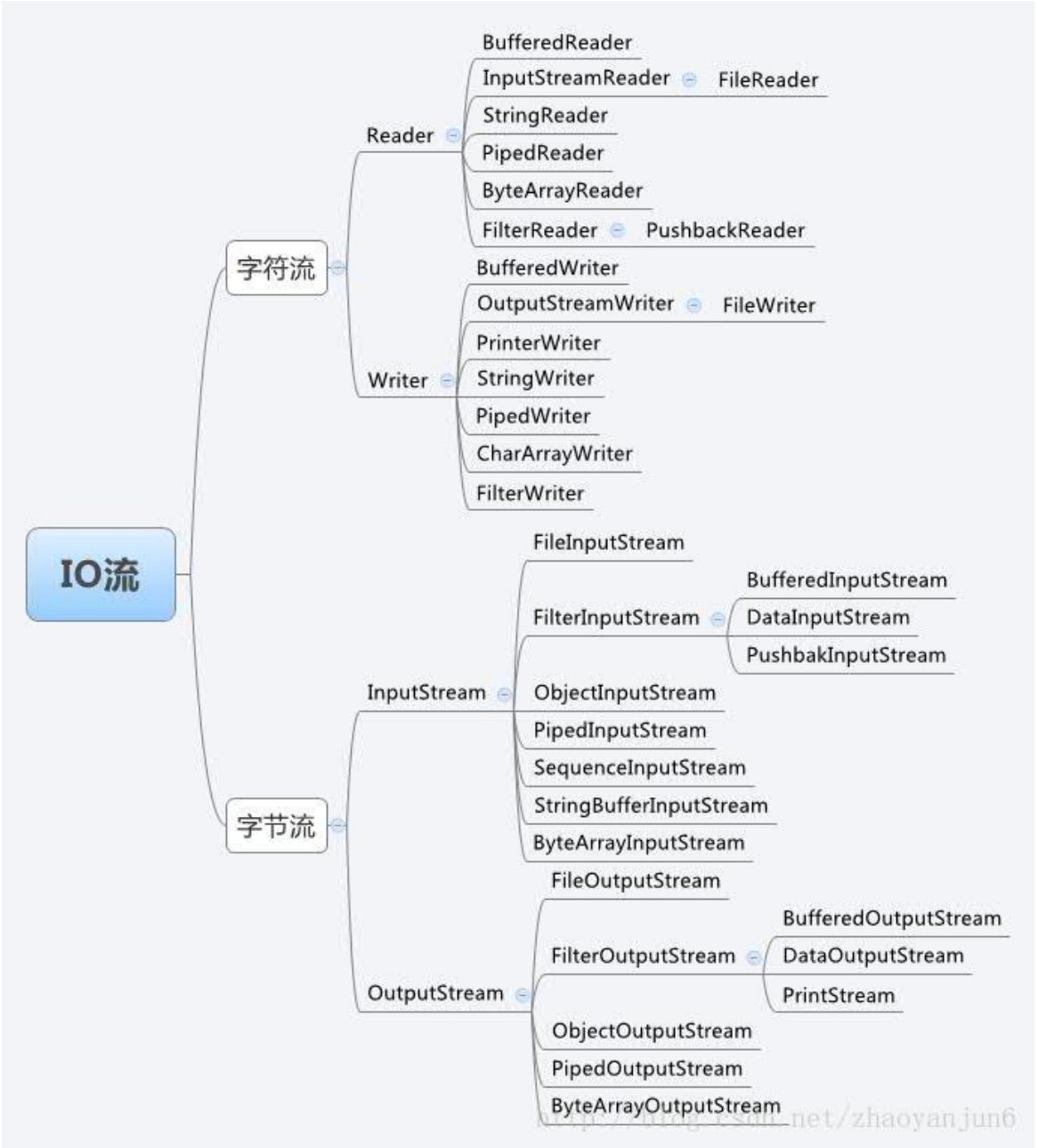


数据源

数据库、文件、其他程序、内存、网络连接、IO设备。

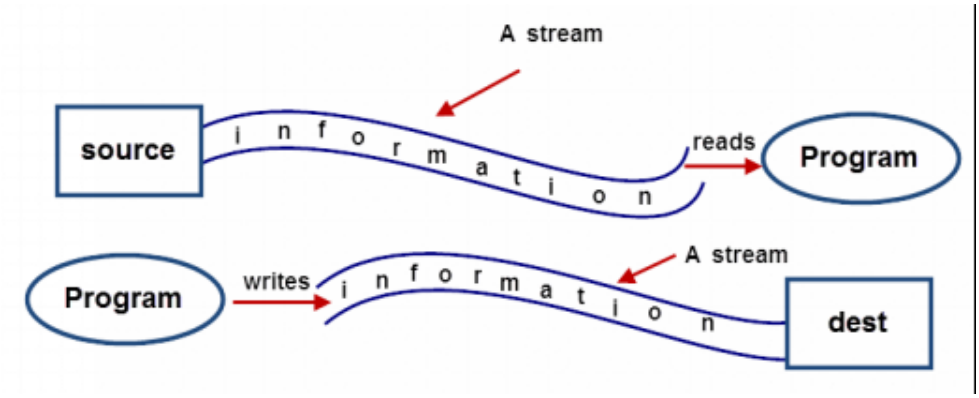
流

流是一个抽象、动态的概念，是一连串连续动态的数据集合。输入/输出流的划分是相对程序而言的，并不是相对数据源。

输入流（读）和输出流（写）：

- 输入流：InputStream或者Reader：从文件中读到程序中；

- 输出流：OutputStream或者Writer：从程序中输出到文件中；



处理的数据单元不同

- **字节流：以字节为单位获取数据，命名上以Stream结尾的流一般是字节流，如FileInputStream、FileOutputStream。**
- **字符流：以字符为单位获取数据，命名上以Reader/Writer结尾的流一般是字符流，本质其实就是基于字节流读取时，去查了指定的码表。如FileReader、FileWriter。**

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。

字节流：一次读入或读出是8位二进制。

字符流：一次读入或读出是16位二进制。

设备上的数据无论是图片或者视频，文字，它们都以二进制存储的。二进制的最终都是以一个8位为数据单元进行体现，所以计算机中的最小数据单元就是字节。意味着，字节流可以处理设备上的所有数据，所以字节流一样可以处理字符数据。

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。

处理的数据对象不同

- **节点流：可以直接从数据源或目的地读写数据，如FileInputStream、FileReader、DataInputStream等**
- **处理流：不直接连接到数据源或目的地，是”处理流的流”。通过对其他流的处理提高程序的性能，如BufferedInputStream、BufferedReader等。处**

理流也叫包装流。

输入/输出流体系

分　类	字节输入流	字节输出流	字符输入流	字符输出流
抽象基类	*InputStream*	*OutputStream*	*Reader*	*Writer*
访问文件	FileInputStream	FileOutputStream	FileReader	FileWriter
访问数组	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
访问管道	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
访问字符串			StringReader	StringWriter
缓冲流	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
转换流			InputStreamReader	OutputStreamWriter
对象流	ObjectInputStream	ObjectOutputStream		
抽象基类	*FilterInputStream*	*FilterOutputStream*	*FilterReader*	*FilterWriter*
打印流		PrintStream		PrintWriter
推回输入流	PushbackInputStream		PushbackReader	
特殊流	DataInputStream	DataOutputStream		

经典案例

- 在实际的项目中，所有的IO操作都应该放到子线程中操作，避免堵住主线程。

```
import  java.io.*;
public  class  TestIO2 {
    public  static  void  main(String[] args) {
        FileInputStream fis = null;
        try  {
            fis = new  FileInputStream("d:/a.txt"); // 内容是：abc
            StringBuilder sb = new  StringBuilder();
            int  temp = 0;
            //当temp等于-1时，表示已经到了文件结尾，停止读取
            while  ((temp = fis.read()) != -1) {
```

```
                sb.append((char) temp);
            }
            System.out.println(sb);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                //这种写法，保证了即使遇到异常情况，也会关闭流对象。
                if (fis != null) {
                    fis.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```