Java IO流

题目难度：★★★

知识点标签：IO流

学习时长：20分钟

题目描述

面试官：谈一下IO流

解题思路

> 从字符流和字节流两方面回答包括输入输出以及两者的对比

数据流的基本概念

几乎所有的程序都离不开信息的输入和输出，比如从键盘读取数据，从文件中获取或者向文件中存入数据，在显示器上显示数据。这些情况下都会涉及有关输入/输出的处理。

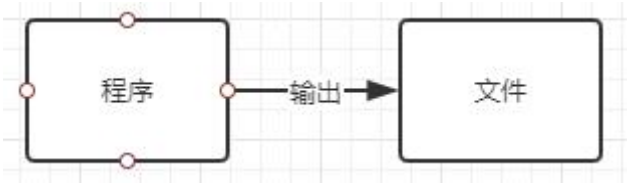
在Java中，把这些不同类型的输入、输出源抽象为流（Stream），其中输入或输出的数据称为数据流（Data Stream），用统一的接口来表示。

IO 流的分类

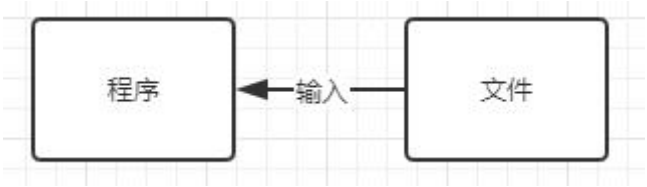
数据流是指一组有顺序的、有起点和终点的字节集合。

按照流的流向分，可以分为输入流和输出流。注意：这里的输入、输出是针对程序来说的。

输出：把程序(内存)中的内容输出到磁盘、光盘等存储设备中。



输入：读取外部数据（磁盘、光盘等存储设备的数据）到程序（内存）中。



按处理数据单位不同分为字节流和字符流。字节流：每次读取(写出)一个字节，当传输的资源文件有中文时，就会出现乱码。

字符流：每次读取(写出)两个字节，有中文时，使用该流就可以正确传输显示中文。

1字符 = 2字节； 1字节(byte) = 8位(bit)； 一个汉字占两个字节长度。

按照流的角色划分为节点流和处理流。节点流：从或向一个特定的地方（节点）读写数据。如FileInputStream。

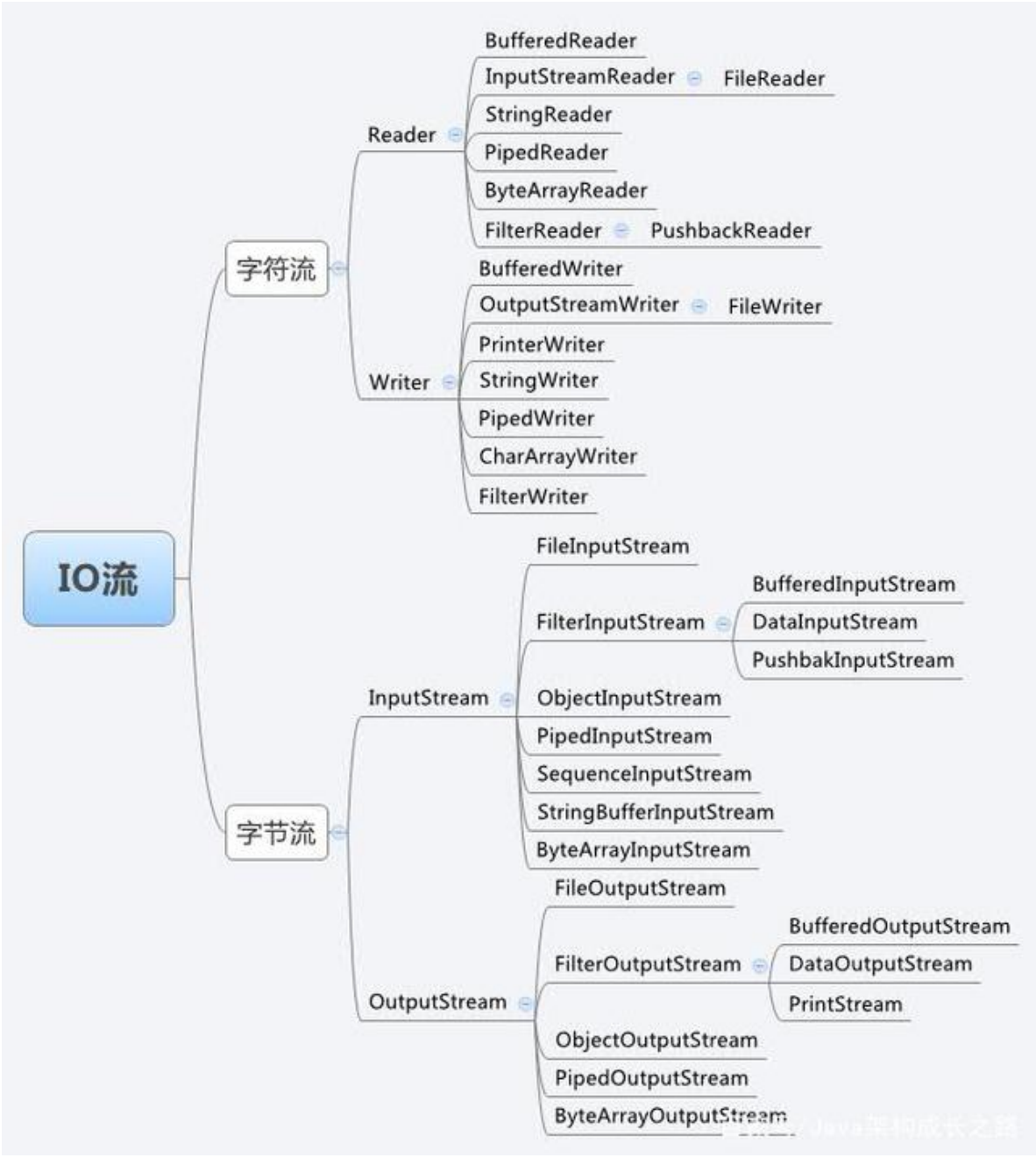
处理流（包装流）：是对一个已存在的流的连接和封装，通过所封装的流的功能调用实现数据读写。如BufferedReader。处理流的构造方法总是要带一个其他的流对象做参数。一个流对象经过其他流的多次包装，称为流的链接。

注意：一个IO流可以既是输入流又是字节流又或是以其他方式分类的流类型，是不冲突的。比如FileInputStream，它既是输入流又是字节流还是文件节点流。

Java IO 流有4个抽象基类：

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

其他流都是继承于这四大基类的。下图是Java IO 流的整体架构图：



知道了 IO 流有这么多分类，那我们在使用的时候应该怎么选择呢？比如什么时候用输出流？什么时候用字节流？可以根据下面三步选择适合自己的流：

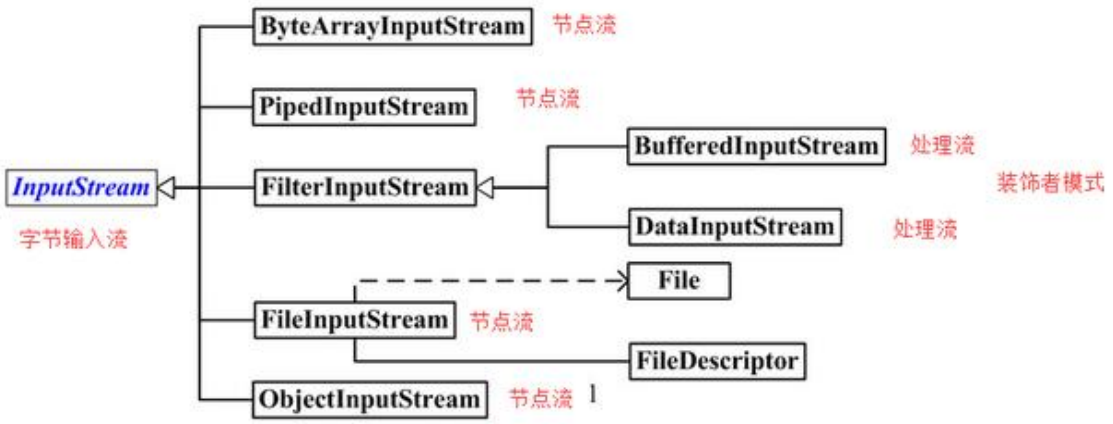
首先自己要知道是选择输入流还是输出流。这就要根据自己的情况决定，如果想从程序写东西到别的地方，那么就选择输入流，反之就选输出流；然后考虑你传输数据时，是每次传一个字节还是两个字节，每次传输一个字节就选字节流，如果存在中文，那肯定就要选字符流了。通过前面两步就可以选出一个合适的节点流了，比如字节输入流 InputStream，如果要在此基础上增强功能，那么就在处理流中选择一个合适的即可。

字节输入流 InputStream

java.io 包下所有的字节输入流都继承自 InputStream，并且实现了其中的方法。InputStream 中提供的主要数据操作方法如下：

int read()：从输入流中读取一个字节的二进制数据。int read(byte[] b)：将多个字节读到数组中，填满整个数组。int read(byte[] b, int off, int len)：从输入流中读取长度为 len 的数据，从数组 b 中下标为 off 的位置开始放置读入的数据，读完返回读取的字节数。void close()：关闭数据流。int available()：返回目前可以从数据流中读取的字节数（但实际的读操作所读得的字节数可能大于该返回值）。long skip(long l)：跳过数据流中指定数量的字节不读取，返回值表示实际跳过的字节数。对数据流中字节的读取通常是按从头到尾顺序进行的，如果需要以反方向读取，则需要使用回推（Push Back）操作。在支持回推操作的数据流中经常用到如下几个方法：

boolean markSupported()：用于测试数据流是否支持回推操作，当一个数据流支持 mark() 和 reset() 方法时，返回 true，否则返回 false。void mark(int readlimit)：用于标记数据流的当前位置，并划出一个缓冲区，其大小至少为指定参数的大小。void reset()：将输入流重新定位到对此流最后调用 mark() 方法时的位置。字节输入流 InputStream 有很多子类，日常开发中，经常使用的一些类见下图：



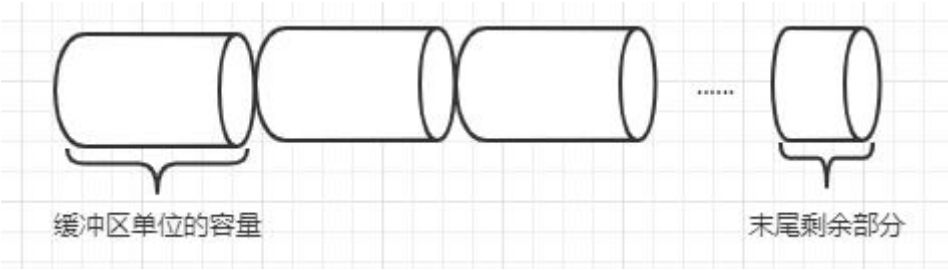
ByteArrayInputStream：字节数组输入流，该类的功能就是从字节数组 byte[] 中进行以字节为单位的读取，也就是将资源文件都以字节形式存入到该类中的字节数组中去，我们拿数据也是从这个字节数组中拿。PipedInputStream：管道字节输入流，它和 PipedOutputStream 一起使用，能实现多线程间的管道通信。FilterInputStream：装饰者模式中充当装饰者的角色，具体的装饰者都要继承它，所以在该类的子类下都是用来装饰别的流的，也就是处理类。BufferedInputStream：缓冲流，对处理流进行装饰、增强，内部会有一个缓冲区，用来存放字节，每次都是将缓冲区存满然后发送，而不是一个字节或两个字节这样发送，效率更高。DataInputStream：数据输入流，用来装饰其他输入流，它允许通过数据流来读写Java基本类型。FileInputStream：文件输入流，通常用于对文件进行读取操作。File：对指定目录的文件进行操作。ObjectInputStream：对象输入流，用来提供对“基本数据或对象”的持久存储。通俗点讲，就是能直接传输Java对象（序列化、反序列化用）。下面通过一个例子讲解InputStream中常用的方法的使用：

```java
InputStream is = null;
try {
    // 创建输入流对象，a.txt 中内容为：AAaBCDEF
    is = new FileInputStream(new File("D:\\file\\a.txt"));
    // 读数据
    int data1 = is.read(); // 获取a.txt中第一个字节
    System.out.println((char) data1); //A

    // int read(byte[] b)：读取多个字节保存在数组b中
    byte[] b = new byte[10];
    is.read(b);
    System.out.println(Arrays.toString(b)); //[65, 97, 66, 67, 68, 69, 70, 0, 0, 0]
    System.out.println(new String(b)); //AAaBCDEF[][][]

    // int read(byte[] b, int off, int len)：读取多个字节，存到数组b中，
    // 从数组下标为off处开始存储，读取5个字节
    is.read(b, 2, 5);
    System.out.println(Arrays.toString(b)); //[66, 67, 68, 69, 70]
    System.out.println(new String(b)); //"  AAaBC"
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (is != null) {
        try {
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

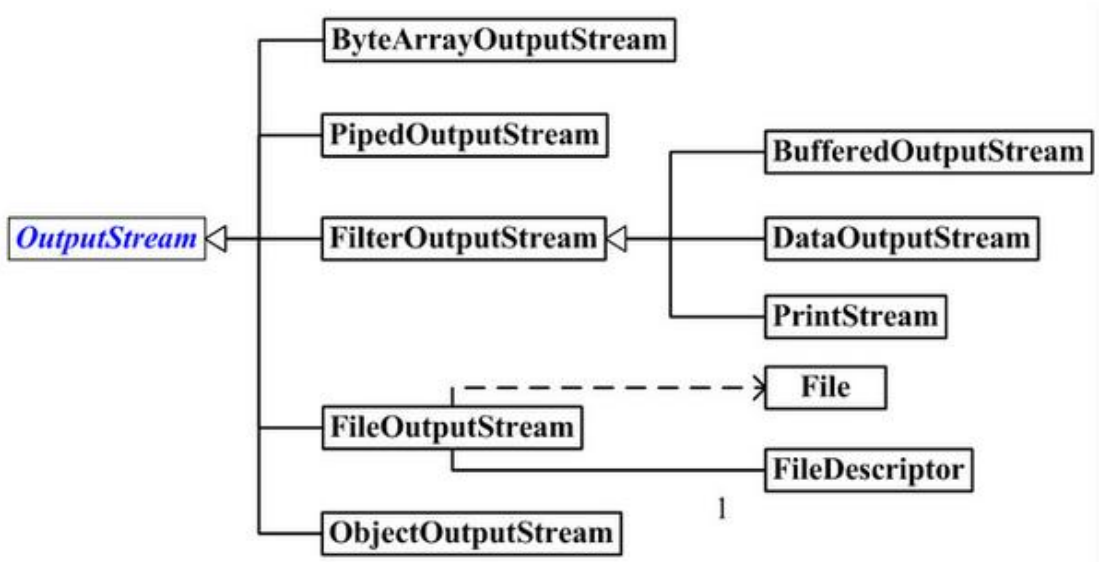
字节输出流 OutputStream

与字节输入流类似，java.io 包下所有字节输出流大多是从抽象类 OutputStream 继承而来的。OutputStream 提供的主要数据操作方法：

void write(int i)：将字节 i 写入到数据流中，它只输出所读入参数的最低 8 位，该方法是抽象方法，需要在其输出流子类中加以实现，然后才能使用。void write(byte[] b)：将数组 b 中的全部 b.length 个字节写入数据流。void write(byte[] b, int off, int len)：将数组 b 中从下标 off 开始的 len 个字节写入数据流。元素 b[off] 是此操作写入的第一个字节，b[off + len - 1] 是此操作写入的最后一个字节。void close()：关闭输出流。void flush()：刷新此输出流并强制写出所有缓冲的输出字节。为了加快数据传输速度，提高数据输出效率，又是输出数据流会在提交数据之前把所要输出的数据先暂时保存在内存缓冲区中，然后成批进行输出，每次传输过程都以某特定数据长度为单位进行传输，在这种方式下，数据的末尾一般都会有一部分数据由于数量不够一个批次，而存留在缓冲区里，调用 flush() 方法可以将这部分数据强制提交。



IO 中输出字节流的继承图可见下图：



它们的作用可以参考上面字节输入流中的各个子类的介绍，这里不再赘述。

下面通过一个例子讲解OutputStream中常用的方法的使用：

```
OutputStream os = null;
try {
    //创建输出流的对象，第二个参数是Boolean类型，
    // true表示后面写入的文件会追加到数据后面，false表示覆盖原来的数据
    os = new FileOutputStream(new File("D\\file\\a.txt"), true);
    // 写数据到文件
    os.write(65); //将A写入到文件中
    os.write('A'); //将A写入到文件中
    os.write("Ab".getBytes()); //将Ab写入文件中
    os.write("abcdefg".getBytes(), 1, 3); //将bcd写入文件
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (os != null) {
        try {
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

下面展示一个字节输入流和字节输出流综合使用的案例：复制文件。

```java
InputStream is = null;
OutputStream os = null;
try {
    // 创建输入流/输出流对象
    is = new FileInputStream(new File("D:\\file\\a.txt"));
    os = new FileOutputStream(new File("D:\\file\\b.txt"));
    // 读取和写入操作
    byte[] buffer = new byte[10]; // 创建一个长度为10的字节数组，存储已经读取的数据
    int len; // 表示已经读取了多少字节，如果是-1，表示已经读到文件末尾
    while ((len = is.read(buffer)) != -1) {
        os.write(buffer, 0, len);
    }
    os.flush();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        if (is != null) {
            is.close();
        }
        if (os != null) {
            os.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

字符流

字符流

从JDK1.1开始，java.io 包中加入了专门用于字符流处理的类，它们是以Reader和Writer为基础派生的一系列类。

同其他程序设计语言使用ASCII字符集不同，Java使用Unicode字符集来表示字符串和字符。ASCII字符集以一个字节（8bit）表示一个字符，可以认为一个字符就是一个字节（byte）。但Java使用的Unicode是一种大字符集，用两个字节（16bit）来表示一个字符，这时字节与字符就不再相同。为了实现与其他程序语言及不同平台的交互，Java提供一种新的数据流处理方案，称作读者（Reader）和写者（Writer）。

字符输入流 Reader

Reader是所有的输入字符流的父类，它是一个抽象类。

Reader及其一些常用子类：

CharReader和SringReader是两种基本的介质流，它们分别将Char数组、String中读取数据。

PipedReader 是从与其它线程共用的管道中读取数据。

BufferedReader很明显是一个装饰器，它和其他子类负责装饰其他Reader对象。

FilterReader是所有自定义具体装饰流的父类，其子类PushBackReader对Reader对象进行装饰，会增加一个行号。

InputStreamReader是其中最重要的一个，用来在字节输入流和字符输入流之间作为中介，可以将字节输入流转换为字符输入流。

FileReader 可以说是一个达到此功能、常用的工具类，在其源代码中明显使用了将FileInputStream 转变为Reader 的方法。

Reader 中各个类的用途和使用方法基本和InputStream 中的类使用一致。

字符输出流 Writer

Writer是所有的输出字符流的父类，它是一个抽象类。

Writer及其一些常用子类：

CharWriter、StringWriter 是两种基本的介质流，它们分别向Char 数组、String 中写入数据。

PipedWriter 是向与其它线程共用的管道中写入数据。BufferedWriter 是一个装饰器为Writer 提供缓冲功能。

PrintWriter 和PrintStream 极其类似，功能和使用也非常相似。OutputStreamWriter是其中最重要的一个，用来在字节输出流和字符输出流之间作为中介，可以将字节输出流转换为字符输出流。

FileWriter 可以说是一个达到此功能、常用的工具类，在其源代码中明显使用了将OutputStream转变为Writer 的方法。

Writer 中各个类的用途和使用方法基本和OutputStream 中的类使用一致。

下面展示一个字符输入流和字符输出流综合使用的案例：复制文件。

```
/**
 * 将文件a.txt复制到b.txt中
 */
BufferedReader reader = null;
BufferedWriter writer = null;
try {
    // 1.创建源和目标
    File srcFile = new File("D:\\file\\a.txt");
    File targetFile = new File("D:\\file\\b.txt");
    // 2.创建输入流、输出流对象
    InputStream is = new FileInputStream(srcFile);
    OutputStream os = new FileOutputStream(targetFile);
    // 3.将字节流转换为字符流
    reader = new BufferedReader(new InputStreamReader(is));
    writer = new BufferedWriter(new OutputStreamWriter(os));
    // 4.读取和写入操作
    String temp; // 记录当前行的数据
    while ((temp = reader.readLine()) != null) { // readLine() 方法表示读取一行数据
        System.out.println(temp); // 打印读取的数据
        writer.write(temp); // 将这一行数据写入到b.txt文件中
        writer.newLine(); // b.txt文件进行换行
    }
    writer.flush();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    // 5.关闭流
    try {
        if (reader != null) {
            reader.close();
        }
        if (writer != null) {
            writer.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

字节流与字符流的区别

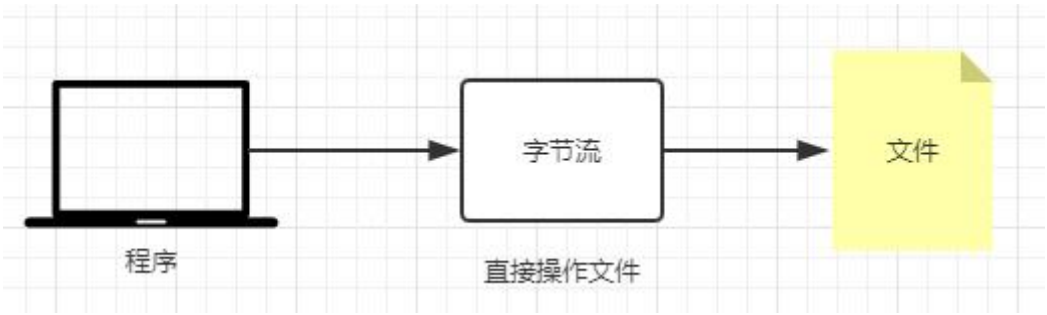
1、要把一片二进制数据数据逐一输出到某个设备中，或者从某个设备中逐一读取一片二进制数据，不管输入输出设备是什么，我们要用统一的方式来完成这些操作，用一种抽象的方式进行描述，这个抽象描述方式起名为IO流，对应的抽象类为OutputStream和InputStream ，不同的实现类就代表不同的输入和输出设备，它们都是针对字节进行操作的。

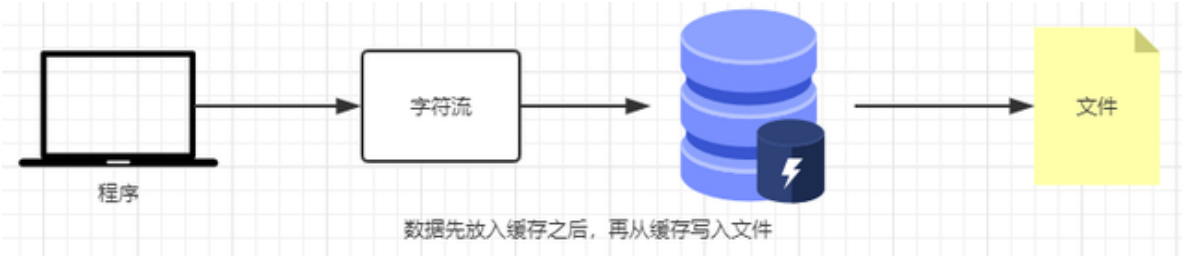
2、在应用中，经常要完全是字符的一段文本输出去或读进来，用字节流可以吗？计算机中的一切最终都是二进制的字节形式存在。对于“中国”这些字符，首先要得到其对应的字节，然后将字节写入到输出流。读取时，首先读到的是字节，可是我们要把它显示为字符，我们需要将字节转换成字符。由于这样的需求很广泛，人家专门提供了字符流的包装类。

3、底层设备永远只接受字节数据，有时候要写字符串到底层设备，需要将字符串转成字节再进行写入。字符流是字节流的包装，字符流则是直接接受字符串，它内部将串转成字节，再写入底层设备，这为我们向IO设别写入或读取字符串提供了一点点方便。

4、字符向字节转换时，要注意编码的问题，因为字符串转成字节数组，其实是转成该字符的某种编码的字节形式，读取也是反之的道理。

字节流在操作时本身不会用到缓冲区（内存），是文件本身直接操作的；而字符流在操作时使用了缓冲区，通过缓冲区再操作文件。





字节流与字符流的使用场景

字节流一般用来处理图像，视频，以及PPT，Word类型的文件。字符流一般用于处理纯文本类型的文件，如TXT文件等。字节流可以用来处理纯文本文件，但是字符流不能用于处理图像视频等非文本类型的文件。