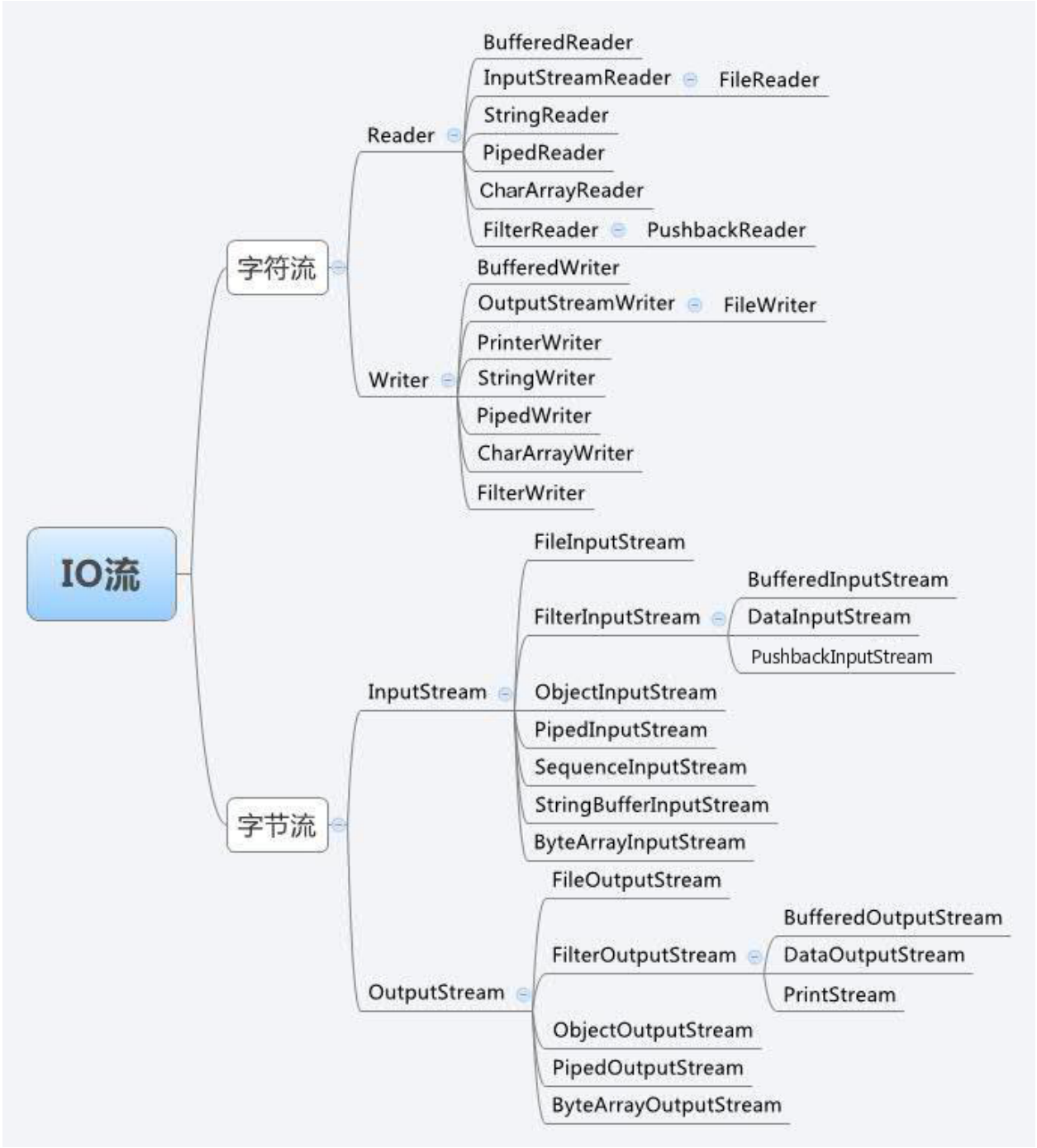
Java-输入输出流

- 总览：输入输出流的类层次图



分类

- 按流动方向
  - 输入流：向当前程序提供数据。如System.in是一个InputStream类型输入流
  - 输出流：当前程序发送数据到此处。如System.out 是一个PrintStream类型输出流
- 按读取类型
  - 字节流：一般为 `InputStream` 或 `OutputStream` 。如System.in是一个InputStream类型字节流

    - 字符流：一般带有`Writer`或`Reader`。如new InputStreamReader(System.in)是一个字符流对象
- 是否为流的源头
    - 节点流：直接操作目标设备对应的流，如文件流，标准输入输出流。
    - 处理流(也叫包装流)，是连接在已存在的流(节点流或处理流)之上，为程序提供更为强大的读写功能，也更加灵活。可以是文件和其他数据源,如BufferedReader。

字符流与字节流之间的转化

- 输入字节流转为字符流需要用到InputtreamReader的构造方法：
    - `InputStreamReader(InputStream in)`
- 输出字符流转为字节流用到OutputStreamWriter或PrintWriter的构造方法：
    - `OutputStreamWriter(OutputStream out)`
    - `PrintWriter(OutputStream out)`

```
InputStreamReader ins1 = new InputStreamReader(System.in);
InputStreamReader ins2 = new InputStreamReader(new FileInputStream(“test.txt”));

OutputStreamWriter outs = new OutputStreamWriter(new
FileOutputStream(“test.txt”));
```

常见流的介绍

- BufferedInputStream和BufferedOutputStream， 缓存作用，用于装配文件磁盘、网络设备、终端等读写开销大的节点流，提高读写性能
- BufferedReader的使用：用于缓存字符流，可以一行一行的读。常见方法见下。
- DataInputStream和DataOutputStream， 可从字节流中写入、读取Java基本数据类型，不依赖于机器的具体数据类型，方便存储和恢复数据。
    - DataOutputStream
        - writeInt()
        - writeDouble()
        - writeUTF(): 写入字符串
    - DataInputStream
        - readInt()
        - readDouble()
        - readUTF(): 写入字符串

控制台读写

使用Scanner读入

- 创建对象：`Scanner s = new Scanner(System.in);`

- 读取内容：next() 与 nextLine()
  - next():
    - 一定要读取到有效字符后才可以结束输入。
    - 对输入有效字符之前遇到的空白，next() 方法会自动将其去掉。
    - 只有输入有效字符后才将其后面输入的空白作为分隔符或者结束符。
    - next() 不能得到带有空格的字符串。
  - nextLine():
    - 以Enter为结束符,也就是说 nextLine()方法返回的是输入回车之前的所有字符。
    - 可以获得空白。
- 输入之前最好先使用 hasNext() / hasNextLine() 方法进行验证，再使用 nextXxx() 来读取
- 如果要输入 int 或 float 类型的数据，在 Scanner 类中也有支持。如nextInt()和nextFloat().但在读入前最好使用hasNextInt()进行验证。

使用Bufferedreader读入

- 创建对象：
  `BufferedReader br = new BufferedReader(new InputStreamReader(System.in));`
- 读入数据
  - read()：读入单个字符，返回int类型。流结束时返回-1.
  - readLine()：读入一行字符，返回String类型。

控制台输出

- 在此前已经介绍过，控制台的输出由 print( ) 和 println() 完成。这些方法都由类 PrintStream 定义，System.out 是该类对象的一个引用。
- `System.out.println()` 和 `System.out.printf()`

文件的读写

使用FileInputStream 和 FileOutputStream，在文件和流之间搭建桥梁。

FileInputStream

- 该流用于从文件读取数据，它的对象可以用关键字 new 来创建。有多种构造方法可用来创建对象。
- 可以使用字符串类型的文件名来创建一个输入流对象来读取文件：

```
InputStream f = new FileInputStream("C:/java/hello");
```

- 也可以使用一个文件对象来创建一个输入流对象来读取文件。我们首先得使用 File() 方法来创建一个文件对象：

```
File f = new File("C:/java/hello");
InputStream in = new FileInputStream(f);
```

FileOutputStream

- 该类用来创建一个文件并向文件中写数据。如果该流在打开文件进行输出前，目标文件不存在，那么该流会创建该文件。有两个构造方法可以用来创建 FileOutputStream 对象。
- 使用字符串类型的文件名来创建一个输出流对象：

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

- 也可以使用一个文件对象来创建一个输出流来写文件。我们首先得使用File()方法来创建一个文件对象：

```
File f = new File("C:/java/hello"); OutputStream fOut = new FileOutputStream(f);
```

示例代码

```
import java.io.*;
public class fileStreamTest2 {
        public static void main(String[] args) throws IOException {
                File f = new File("a.txt");
                FileOutputStream fop = new FileOutputStream(f);
                // 构建FileOutputStream对象,文件不存在会自动新建
                OutputStreamWriter writer = new OutputStreamWriter(fop, "UTF-8");
                // 构建OutputStreamWriter对象,参数可以指定编码,默认为操作系统默认编码,windows上是gbk
                writer.append("示例"); // 写入到缓冲区
                writer.append("\r\n"); // 换行
                writer.close(); // 关闭写入流,同时会把缓冲区内容写入文件,所以上面的注释掉
                fop.close(); // 关闭输出流,释放系统资源

                FileInputStream fip = new FileInputStream(f); // 构建FileInputStream对象
                InputStreamReader reader = new InputStreamReader(fip, "UTF-8");
// 构建InputStreamReader对象,编码与写入相同
                StringBuffer sb = new StringBuffer();
                while (reader.ready()) {
                        sb.append((char) reader.read()); // 转成char加到StringBuffer对象中
                }
                System.out.println(sb.toString());
                reader.close(); // 关闭读取流
                fip.close(); // 关闭输入流,释放系统资源
```

```
        }
}
```