I/O 学习总结

在 JAVA 中,流分两大类,既: 字节流和字符流

在正式介绍 I/O 流之前, 我们来先学一个 File 类.

File: java.io.File 类

File 可以创建和删除一个文件,也可以创建和删除一个空目录.

例子: File f1 = new File("file1.txt"); //创建一个 File 对象, 此时磁盘上还没有 file.txt 文件。f1.createNewFile(); 此时才会在磁盘上创建一个文件, 名为 file.txt

File dir = new File("p/q/s");

dir.mkdir() ; //在当前目录下的 p/q 子目录中创建 s 子目录。如果没有 p/q 子目录, 则创建不成功。

dir.mkdirs) ; //在当前目录下,如果父目录 p/q 不存在, 则会先把父目录 创建好, 再建子目录。如果父目录存在, 则直接创建子目录。

方法: isDirectory() 和 isFile() 可以用来判断是一个文件或目录。

delete() 方法可以删除一个文件或目录; 如果要删除目录, 目录必需为空才行。

deleteOnExit() 方法: 此方法不会立既删除文件或目录, 而是要等到程序运行结束后, 才会去删除; 常用于控制临时文件!

list() 方法: 返回由此抽象路径名所表示的目录中的文件和目录的名称所组成字符串数组。如果此抽象路径名并不表示一个目录, 则此方法将返回 null.

如: File home = new File("c:\temp");

```
String[] list = home.list();
For(int i=0;i<list.length;i++) {
    //可以得到此目录中的所有目录名和文件名所表示的字符串。
}
```

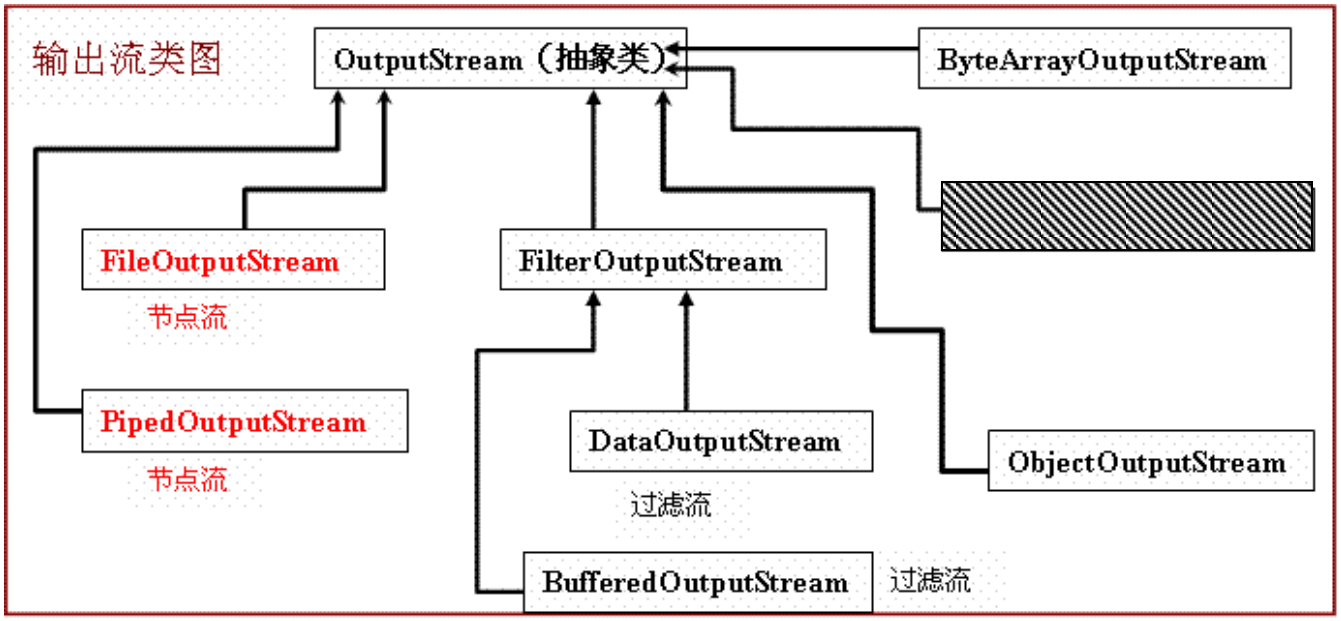
注: File 类是不可能访问得到文件的内容的!!!

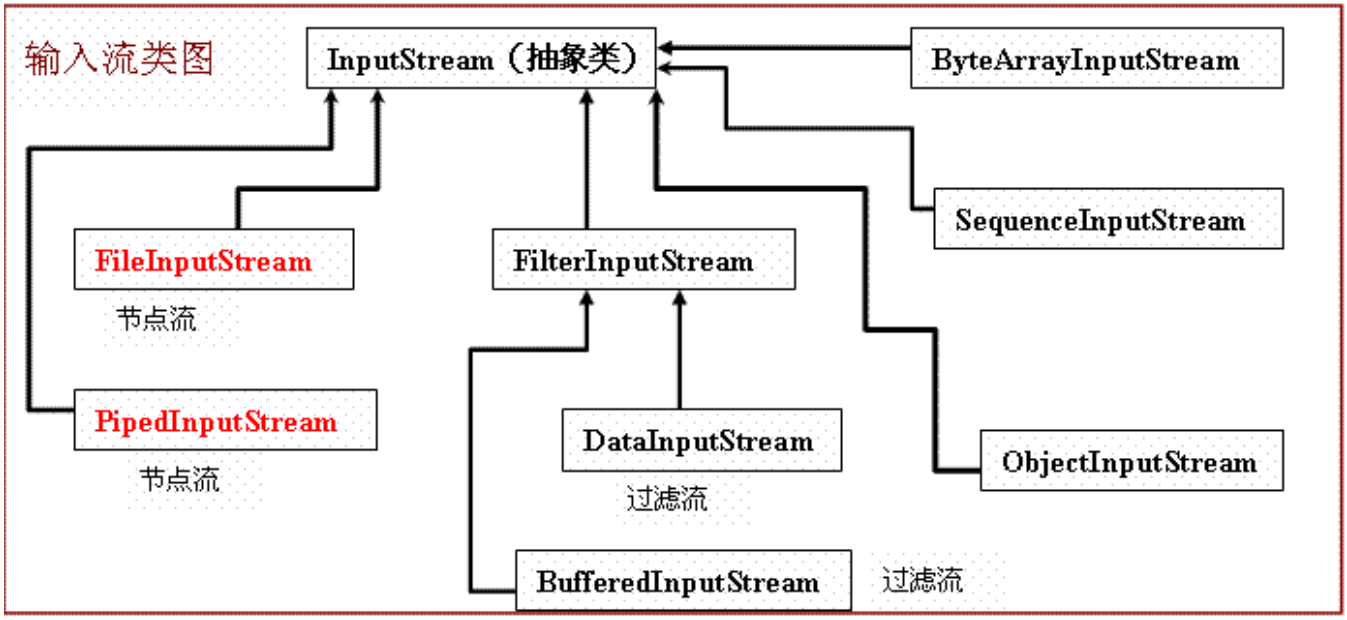
那么, 我们如何来访问和操纵文件的内容呢?

毫无疑问, 输入输出流可以来完成的我们的想法:

在 JAVA 在, 流分为两大类: **字节流和字符流**

首先, 我们来学习字节流: 我们知道, 顶层的两个抽象类: **InputStream 和 OutputStream**, 以下是常用字节输入流的类的继承关系图:





由上面两图可以看出，输入输出流都是相对应的，这样可以方便我们记忆，而且它们的操作也都是一样的！

对于节点流，拥有最基本的读写字节能力，它才是实际的去和一个文件创建连接的流。
对于过滤流：主要是负责给节点流增加一些功能。它们不能独立构造。既一定要以节点流为‘基础’。

如： BufferedInputStream br = new BufferedInputStream(new FileInputStream(“text.txt”));

在此，提一下一个应用模式，就是‘装饰模式’
流的构造就是一个装饰模式的完整应用。节点流就像是一个最原始的元素，拥有有限的能力，过滤流就像是各种装饰元素，它们通过节点流来构造一个功能强大的流，如 BufferedInputStream,DataInputStream. . .

注：在 JAVA 中，凡是跨越虚拟机的范围的一定要自己主动释放。
现在，我们来做一个用字节流实现文件的拷贝的例子：

Coding:

```
Public  class TestCopy() {
    Public  static void main(String[]  args) {
```

```
            FileInputStream  fis = new FileInputStream("源文件名");
            BufferedInputStream  bis = new BufferedInputStream(fis  );

            FileOutputStream  fos = new FileOutputStream("目标文件名");
            BufferedOutputStream  bos = new BufferedOutputStream(fos);
            byte[] b = new byte[1024]; //做为缓冲区，大小为 1K
            int len; //保存读到的实际字节数
            while((len  = bis.read(b)    !==   -1) {
                bos.write(b,0,len);    //写到 SOCKET 中。
                bos.flush();     //清空缓冲区，记得要写哦!!!!
            }
        }
    }
```

二. 字符流

在字节流中可以使用 DataInputStream 中的读字符方法来读取字符，用样，也可以写，那为什么还需要要专门的字符流呢？

我们知道，在计算机里存放的就是一堆的二进制，那它是如何来表示一个字符，数字，字母，特殊符号，等，这就要靠编解码了！ 字符流就可以用来解决字符编码的问题，使用它，可以保证编解码统一，防止出现乱码。

比如：一个字符‘A’，采用 ASCII 码标准编码为 0X41。

字符编码：

1. ASCII 码，1 字符占 1 字节，对应 8 位，它最多可表示 256 个字符。
   这对于一些英语国家，这种方式不会出现问题，但对于中文，很显然，这远远不够。
   所以，如要显示中文，如果不扩展的话，那么就无法来显示中文。
2. 中国：GB2312 标准。它占 2 个字节，并向上兼容 ASCII 编码。它可以表示 65536 个字符
3. GBK 标准，对 GB2312 的扩展，增加了一些生僻字的编码，和繁体。它也是中文 OS 的默认编码方式。
4. UNICODE: 1 字符占 2 字节。支持各种字符。

。 。 。 。 。

所以，国标上有这么多的标准，如果编解码方式不统一，就会出现乱码的现象。

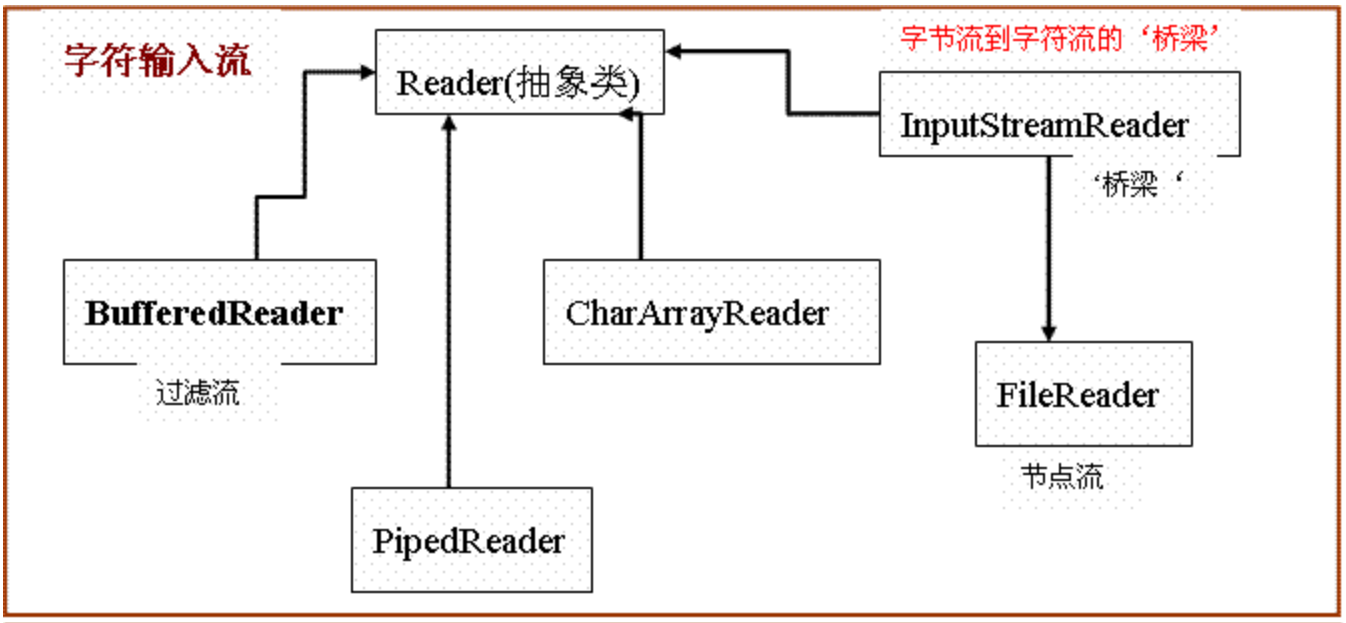
如：

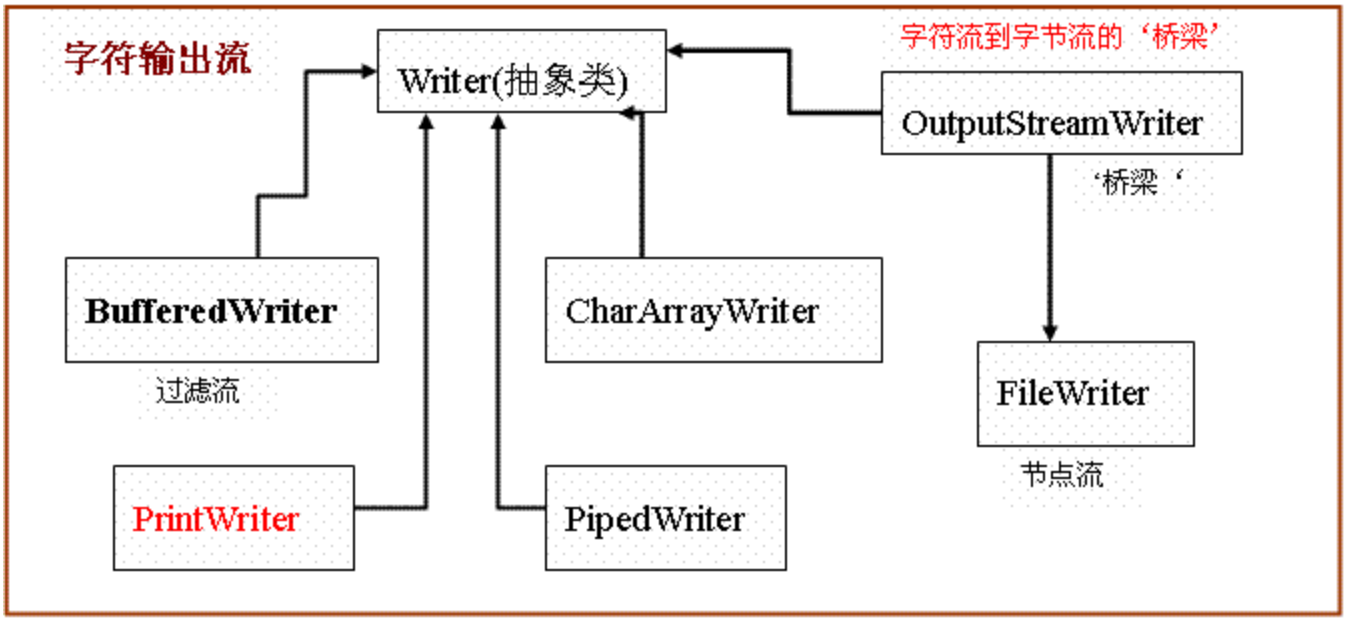
```
String s1 = "达内科技";
Byte[] b1 = s1.getBytes("UTF-8"); //指定使用 UTF-8 的编码方式。
String s2 = new String(b1,"GBK"); //使用 GBK 方式来解码，此时就会出现中文乱码。

Byte[] b2 = s2.getBytes("GBK");//采用同样方式反编回来。
String s3 = new String(b2,"UTF-8");    //解码与上面编码时一致！打印正常
```

对于 getBytes() 方法。请详见 java.lang.String 类的方法介绍。

现在： 我们来看一下字符流的常用类图：





最重要的，我们来看一下：字节流与字符流之间的‘桥接器‘，对于输出流来说，它可以把字节流转变成字符流，（InputStreamReader），而对于输入流来说，它可以把字符流转换成字节流，（OutputStreamWriter）；
所以，用一个字节输入流来构造一个字符流如下：

```
BufferedReader br = new BufferedReader(new InputStreamReader(new InputStream());
```

如果想从键盘获得收入：可以这样封装一个 BufferedReader;

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

当然，对于‘桥梁’，是可以指定字符编码方式的。 详见 API(~^~);

注：我们在多线程讲过，等待数据输入是会产生阻塞的。所以 read() 方法是会产生阻塞 的•
而对于 BufferdReader 中的 readLine() 方法，只有读到换行，它才会打破阻塞！一定要注意！

对于 FileReader/FileWriter ，如不指定编码方式，都会采用当前系统的默认的字符集。如中文 OS 的‘GBK’.

现在，我们来介绍如何让一个对象持入化呢？
实现一个接口： Serializable 接口
这是一个标记接口，不需要实现任何方法
实现了 Serializable 接口的类的对象就可以序列化，注：如果此类中包含其它的类对象，则那个类也必需要实现 Serializable 接口。
其实，序列化一个对象，就是序列化它所包含的属性。
那么如何有先择地序列化其中某些属性，而某些属性是不序列化呢？
关键字： transient 它用来修饰实例变量
表示此实例变量不被序列化。

```
如： class Student implements Seriailizable {
        String name;
        int transient age; //在写对象时，age 属性将不被序列化，在读到此属性时，它的值为 0；
    }
```

注：Serializable 接口不能自己定义序列化逻辑，它的子接口：Externalizable 接口，它可以让你自己定义序列化对象的逻辑。 详见 API， 这里不多做介绍

小结：

1. 使用 ObjectInputStream 和 ObjectOutputStream 来读写对象
2. 对象必须要实现 Serializable 接口
3. 属性是对象，也要实现 Serializable 接口
4. 关键字 transient 来让属性不序列化
5. 用 Externalizable 来定义自己的序列化逻辑。

类： RandomAccessFile 它实现了 DataInput, DataOutput 接口； 可读/可写
此类可以访问一个文件中的随机位置（并不一定是从头到尾）

```
方法： getFilePoint() //获得此文件中的当前偏移量
       seek(long pos); //定位当前指针的偏移量
       read(), read(byte[] buf), read(byte[] buf, int off, int len);
       write(int b), write(byte[] buf), write(byte[] buf, int off, int len);
```

至此，我们的总结就写到这儿了，希望能对大家有所帮助！

作者：叶加飞 （Steven Ye）
Mailto: leton.ye@gmail.com