IO框架

```
Java IO的学习是一件非常艰巨的任务。
```

它的挑战是来自于要覆盖所有的可能性。不仅存在各种I/O源端还有想要和他通信的接收端（文件/控制台/网络链接），而且还需要以不同的方式与他们进行通信（顺序/随机存取/缓冲/二进制/字符/行/字 等等）这些情况综合起来就给我们带来了大量的学习任务，大量的类需要学习。

我们要学会所有的这些java 的IO是很难的，因为我们没有构建一个关于IO的体系，要构建这个体系又需要深入理解IO库的演进过程，所以，我们如果缺乏历史的眼光，很快我们会对什么时候应该使用IO中的哪些类，以及什么时候不该使用它们而困惑。

所以，在开发者的眼中，IO很乱，很多类，很多方法，很迷茫。

参考资料：Java编程思想 （第4版） 第18章 Java I/O系统

IO简介

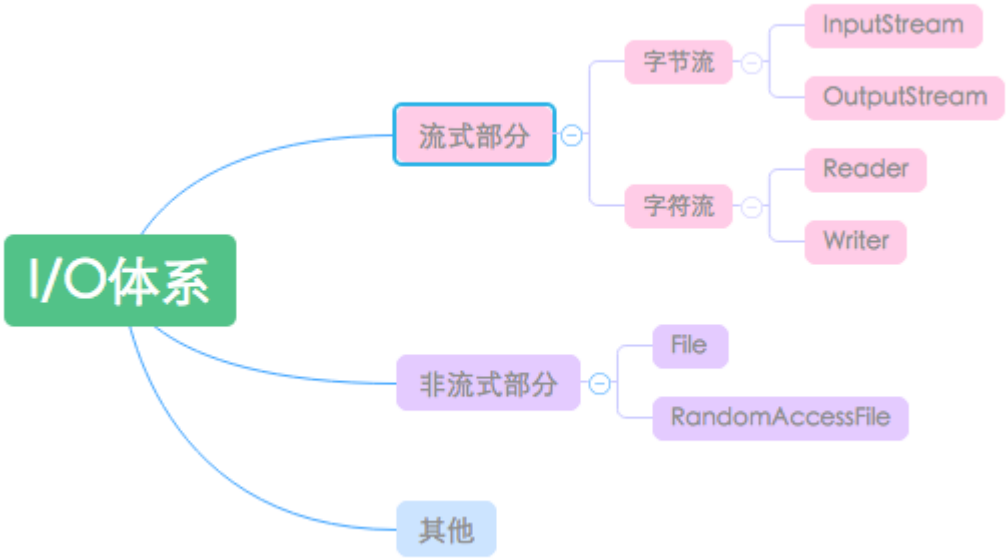
数据流是一组有序，有起点和终点的字节的数据序列。包括输入流和输出流。

流序列中的数据既可以是未经加工的原始二进制数据，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。因此Java中的流分为两种： **1) 字节流：**数据流中最小的数据单元是字节 **2) 字符流：**数据流中最小的数据单元是字符，Java中的字符是Unicode编码，一个字符占用两个字节。

Java.io包中最重要的就是5个类和一个接口。5个类指的是File、OutputStream、InputStream、Writer、Reader；一个接口指的是Serializable。掌握了这些就掌握了Java I/O的精髓了。

Java I/O主要包括如下3层次：

1. 流式部分——最主要的部分。如：OutputStream、InputStream、Writer、Reader等
2. 非流式部分——如：File类、RandomAccessFile类和FileDescriptor等类
3. 其他——文件读取部分的与安全相关的类，如：SerializablePermission类，以及与本地操作系统相关的文件系统的类，如：FileSystem类和Win32FileSystem类和WinNTFileSystem类。



IO详细介绍

在Android 平台，从应用的角度出发，我们最需要关注和研究的就是 字节流（Stream）字符流（Reader/Writer）和 File/ RandomAccessFile。当我们需要的时候再深入研究也未尝不是一件好事。关于字符和字节，例如文本文件，XML这些都是用字符流来读取和写入。而如RAR，EXE文件，图片等非文本，则用字节流来读取和写入。**面对如此复杂的类关系，有一个点是我们必须要首先掌握的，那就是设计模式中的修饰模式**，学会并理解修饰模式是搞懂流必备的前提条件哦。

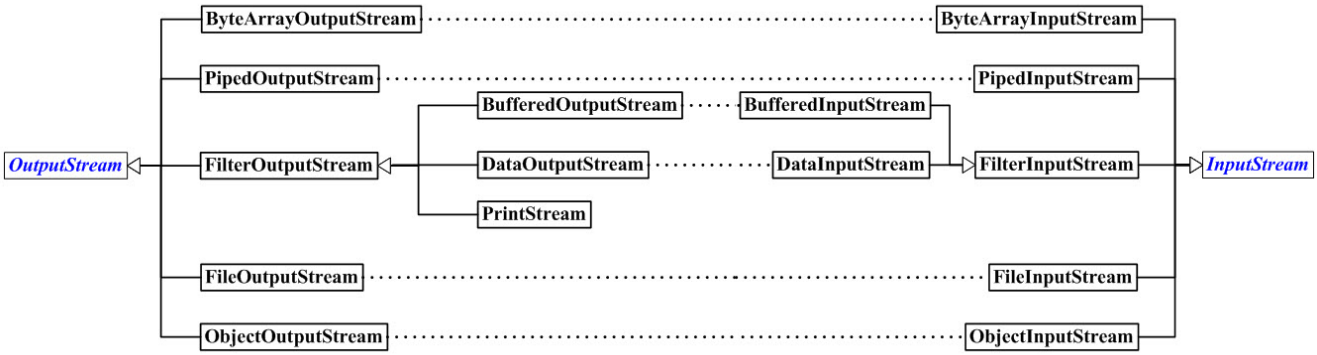
字节流的学习

在具体的学习流之前，我们必须要学的一个设计模式是装饰模式。因为从流的整个发展历史，出现的各种类之间的关系看，都是沿用了修饰模式，都是一个类的功能可以用来修饰其他类，然后组合成为一个比较复杂的流。比如说：

```
        DataOutputStream out = new DataOutputStream(
                    new BufferedOutputStream(
                    new FileOutputStream(
                    new File(file)));
```

从上面的代码块中大家不难看出这些类的关系：**为了向文件中写入数据，首先需要创建一个FileOutputStream，然后为了提升访问的效率，所以将它发送给具备缓存功能的BufferedOutput-Stream,而为了实现与机器类型无关的java基本类型数据的输出，所以，我们将缓存的流传递给了DataOutputStream。**从上面的关系，我们可以看到，其根本目的都是为outputSteam添加额外的功能。**而这种额外功能的添加就是采用了装饰模式来构建的代码。**因此，学习流，必须要学好装饰模式。

下面的图是一个关于字节流的图谱，这张图谱比较全面的概况了我们字节流中间的各个类以及他们之间的关系。



字节流的学习过程

为什么要按照一个学习路线来呢？原因是他们的功能决定的。

OutputStream -> FileOutputStream/FilterOutputStream ->DataOutputStream->bufferedOutputStream

相应的学习InputStream方法就好了。

FilterOutputStream

从学习的角度来，我们应该先掌握FilterOutputStream, 以及FileOutputStream，这两个类是基本的类，从继承关系可以不难发现他们都是对 abstract 类 OutputStream的拓展，是它的子类。然而，伴随着 对 Stream流的功能的拓展，所以就出现了 DataOutputStream，（将java中的基础数据类型写入数据字节输出流中、保存在存储介质中、然后可以用DataOutputStream从存储介质中读取到程序中还原成java基础类型）。这里多提一句、DataOutputStream、FilterOutputStream三个类的关系的这种设计既使用了装饰器模式 避免了类的爆炸式增长。

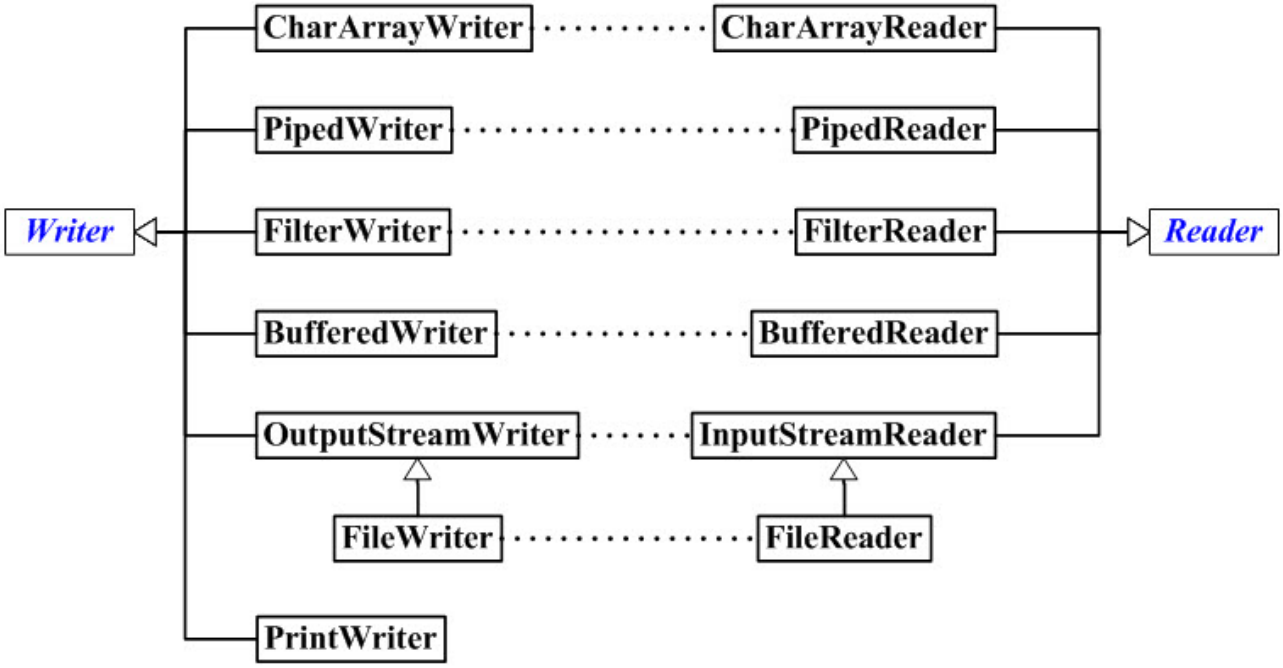
BufferedOutputStream

为了提升Stream的执行效率，所以出现了bufferedOutputStream。bufferedOutputStream就是将本地添加了一个缓存的数组。在使用bufferedOutputStream之前每次从磁盘读入数据的时候都是需要访问多少byte数据就向磁盘中读多少个byte的数据，而出现bufferedOutputSteam之后，策略就改了，会先读取整个缓存空间相应大小的数据，这样就是从磁盘读取了一块比较大的数据，然后缓存起来，从而减少了对磁盘的访问的次数以达到提升性能的目的。

另外一方面，我们知道了outputStream（输出流）的发展历史后，我们便可以知道如何使用outpuSteam了，同样的方法，我们可以运用到inputStream中来，这样对称的解释就出现到了inputStream相关的中来了，于是，我们对整个字节流就有了全方位的理解，所以这样子我们就不会感觉到流的复杂了。这个时候对于其他的一些字节流的使用(byteArrayOutputStream/PipeOutputStream/ObjectOutputStream)的学习就自需要在使用的时候看看API即可。

字符流的学习

下图则是一个关于字符流的图谱，这张图谱比较全面的概况了我们字符流中间的各个类以及他们之间的关系。



字符流的学习和字节流的学习是一样的，它和字节流有着同样的发展过程，只是，字节流面向的是我们未知或者即使知道了他们的编码格式也意义不大的文件（png，exe, zip）的时候是采用字节，而面对一些我们知道文件构造我们就能够搞懂它的意义的文件（json，xml）等文件的时候我们还是需要以字符的形式来读取，所以就出现了字符流。reader 和 Stream最大的区别我认为是它包含了一个readline（）接口，这个接口标明了，一行数据的意义，这也是可以理解的，因为自有字符才具备行的概念，相反字节流中的行也就是一个字节符号。

字符流的学习历程：

Writer- >FilterWriter->BufferedWriter->OutputStreamWriter->FileWriter->其他

同时类比着学习Reader相关的类。

FilterWriter/FilterReader

字符过滤输出流、与FilterOutputStream功能一样、只是简单重写了父类的方法、目的是为所有装饰类提供标准和基本的方法、要求子类必须实现核心方法、和拥有自己的特色。这里FilterWriter没有子类、可能其意义只是提供一个接口、留着以后的扩展。。。本身是一个抽象类。

BufferedWriter/BufferedReader

BufferedWriter是 Writer类的一个子类。他的功能是为传入的底层字符输出流提供缓存功能、同样当使用底层字符输出流向目的地中写入字符或者字符数组时、每写入一次就要打开一次到目的地的连接、这样频繁的访问不断效率底下、也有可能会对存储介质造成一定的破坏、比如当我们向磁盘中不断的写入字节时、夸张一点、将一个非常大单位是G的字节数据写入到磁盘的指定文件中的、没写入一个字节就要打开一次到这个磁盘的通道、这个结果无疑是恐怖的、而当我们使用BufferedWriter将底层字符输出流、比如FileReader包装一下之后、我们可以在程序中先将要写入到文件中的字符写入到BufferedWriter的内置缓存空间中、然后当达到一定数量时、一次性写入FileReader流中、此时、FileReader就可以打开一次通道、将这个数据块写入到文件中、这样做虽然不可能达到一次访问就将所有数据写入磁盘中的效果、但也大大提高了效率和减少了磁盘的访问量！

OutputStreamWriter/InputStreamReader

输入字符转换流、是输入字节流转向输入字符流的桥梁、用于将输入字节流转换成输入字符流、通过指定的或者默认的编码将从底层读取的字节转换成字符返回到程序中、与OutputStreamWriter一样、本质也是使用其内部的一个类来完成所有工作：StreamDecoder、使用默认或者指定的编码将字节转换成字符；OutputStreamWriter/InputStreamReader只是对StreamDecoder进行了封装、isr内部所有方法核心都是调用StreamDecoder来完成的、InputStreamReader只是对StreamDecoder进行了封装、使得我们可以直接使用读取方法、而不用关心内部实现。

> `OutputStreamWriter`、`InputStreamReader`分别为`InputStream`、`OutputStream`的低级输入输出流提供将字节转换成字符的桥梁、他们只是外边的一个门面、真正的核心：

OutputStreamWriter中的StreamEncoder：

> 1、使用指定的或者默认的编码集将字符转码为字节
>
> 2、调用`StreamEncoder`自身实现的写入方法将转码后的字节写入到底层字节输出流中。

InputStreamReader中的StreamDecoder：

> 1、使用指定的或者默认的编码集将字节解码为字符
>
> 2、调用`StreamDecoder`自身实现的读取方法将解码后的字符读取到程序中。

在理解这两个流的时候要注意：java——io中只有将字节转换成字符的类、没有将字符转换成字节的类、原因很简单——字符流的存在本来就像对字节流进行了装饰、加工处理以便更方便的去使用、在使用这两个流的时候要注意：由于这两个流要频繁的对读取或者写入的字节或者字符进行转码、解码和与底层流的源和目的地进行交互、所以使用的时候要使用BufferedWriter、BufferedReader进行包装、以达到最高效率、和保护存储介质。

FileReader/FileWriter

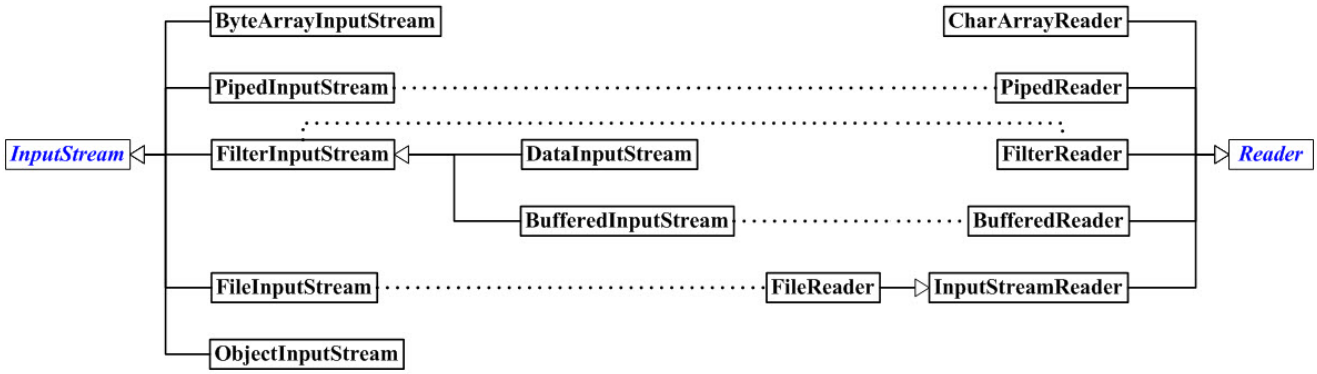
FileReader和FileWriter 继承于InputStreamReader/OutputStreamWriter。

从源码可以发现FileWriter 文件字符输出流、主要用于将字符写入到指定的打开的文件中、其本质是通过传入的文件名、文件、或者文件描述符来创建FileOutputStream、然后使用OutputStreamWriter使用默认编码将FileOutputStream转换成Writer（这个Writer就是FileWriter）。如果使用这个类的话、最好使用BufferedWriter包装一下、高端大气上档次、低调奢华有内涵！

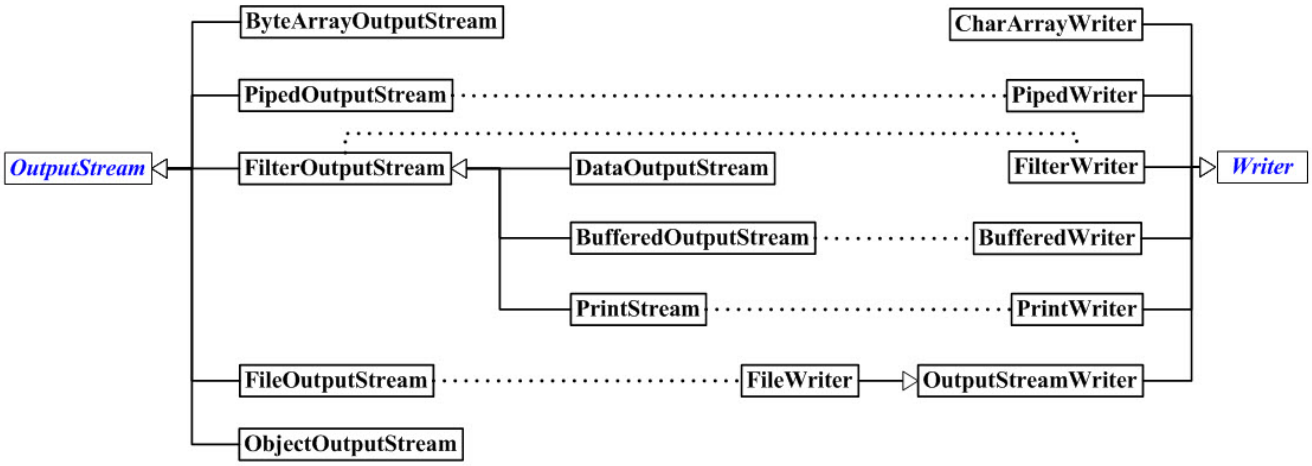
FileReader 文件字符输入流、用于将文件内容以字符形式读取出来、一般用于读取字符形式的文件内容、也可以读取字节形式、但是因为FileReader内部也是通过传入的参数构造InputStreamReader、并且只能使用默认编码、所以我们无法控制编码问题、这样的话就很容易造成乱码。所以读取字节形式的文件还是使用字节流来操作的好、同样在使用此流的时候用BufferedReader包装一下、就算冲着BufferedReader的readLine()方法去的也要使用这个包装类、不说他还能提高效率、保护存储介质。

字节流与字符流的关系

那么字节输入流和字符输入流之间的关系是怎样的呢？请看下图



同样的字节与字符输出流字节的关系也如下图所示



字节流与字符流的区别

字节流和字符流使用是非常相似的，那么除了操作代码的不同之外，还有哪些不同呢?

字节流在操作的时候本身是不会用到缓冲区（内存）的，是与文件本身直接操作的，而字符流在操作的时候是使用到缓冲区的字节流在操作文件时，即使不关闭资源（close方法），文件也能输出，但是如果字符流不使用close方法的话，则不会输出任何内容，说明字符流用的是缓冲区，并且可以使用flush方法强制进行刷新缓冲区，这时才能在不close的情况下输出内容

那开发中究竟用字节流好还是用字符流好呢?

在所有的硬盘上保存文件或进行传输的时候都是以字节的方法进行的，包括图片也是按字节完成，而字符是只有在内存中才会形成的，所以使用字节的操作是最多的。

如果要java程序实现一个拷贝功能，应该选用字节流进行操作（可能拷贝的是图片），并且采用边读边写的方式（节省内存）。

字节流与字符流的转换

虽然Java支持字节流和字符流，但有时需要在字节流和字符流两者之间转换。InputStreamReader和OutputStreamWriter，这两个为类是字节流和字符流之间相互转换的类。

InputSreamReader用于将一个字节流中的字节解码成字符:

有两个构造方法:

```
InputStreamReader(InputStream in);
```

功能：用默认字符集创建一个InputStreamReader对象

```
InputStreamReader(InputStream in,String CharsetName);
```

功能：接收已指定字符集名的字符串，并用该字符创建对象

OutputStream用于将写入的字符编码成字节后写入一个字节流。

同样有两个构造方法：

```
OutputStreamWriter(OutputStream out);
```

功能：用默认字符集创建一个OutputStreamWriter对象；

```
OutputStreamWriter(OutputStream out,String  CharSetName);
```

功能：接收已指定字符集名的字符串，并用该字符集创建OutputStreamWrite对象

为了避免频繁的转换字节流和字符流，对以上两个类进行了封装。

BufferedWriter类封装了OutputStreamWriter类；

BufferedReader类封装了InputStreamReader类；

封装格式：

```
BufferedWriter out=new BufferedWriter(new OutputStreamWriter(System.out));
BufferedReader in= new BufferedReader(new InputStreamReader(System.in);
```

利用下面的语句，可以从控制台读取一行字符串：

```
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
String line=in.readLine();
```