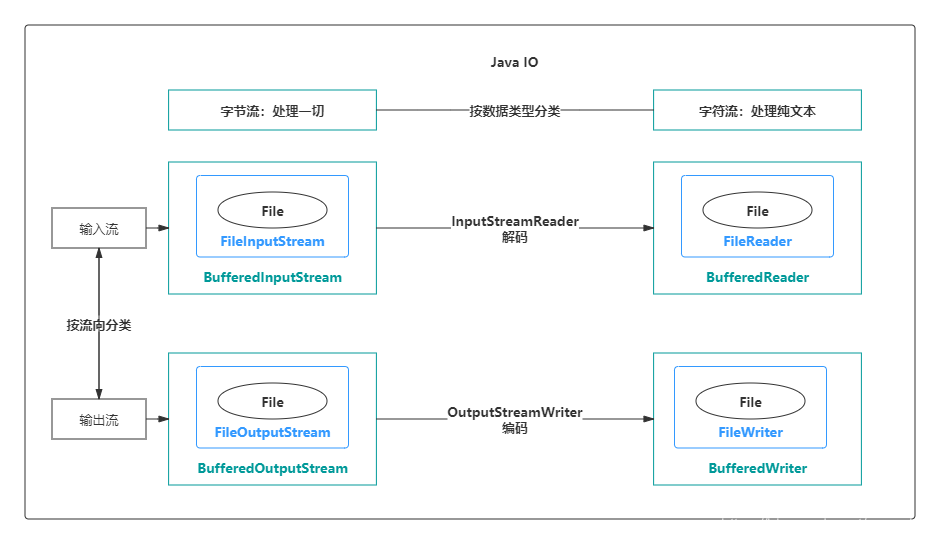
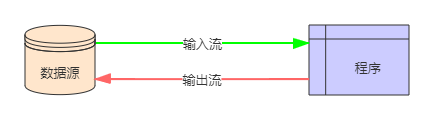
IO流主要的分类方式有以下3种：

1. 按数据流的方向：输入流、输出流
2. 按处理数据单位：字节流、字符流
3. 按功能：节点流、处理流

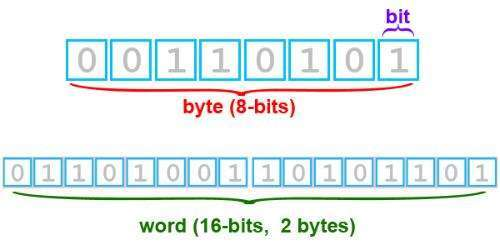


**1、输入流与输出流**

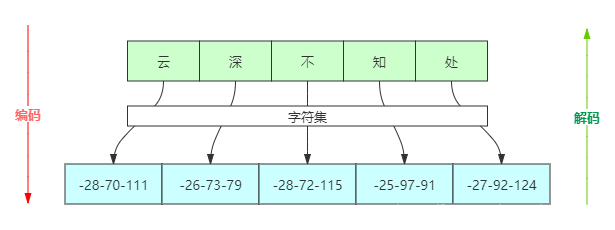


**2、字节流与字符流**

Java中字符是采用Unicode标准，Unicode 编码中，一个英文为一个字节，一个中文为两个字节。



而在UTF-8编码中，一个中文字符是3个字节。例如下面图中，“云深不知处”5个中文对应的是15个字节：-28-70-111-26-73-79-28-72-115-25-97-91-27-92-124

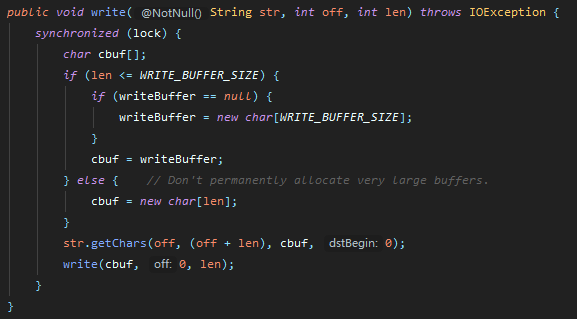


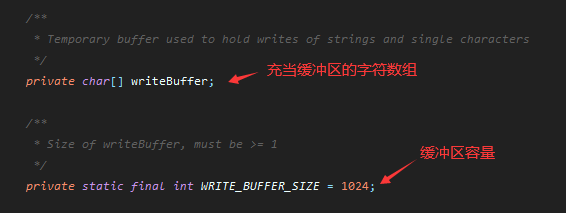
那么问题来了，如果使用字节流处理中文，如果一次读写一个字符对应的字节数就不会有问题，一旦将一个字符对应的字节分裂开来，就会出现乱码了。为了更方便地处理中文这些字符，Java就推出了字符流。

字节流和字符流的其他区别：

字节流一般用来处理图像、视频、音频、PPT、Word等类型的文件。字符流一般用于处理纯文本类型的文件，如TXT文件等，但不能处理图像视频等非文本文件。用一句话说就是：字节流可以处理一切文件，而字符流只能处理纯文本文件。

字节流本身没有缓冲区，缓冲字节流相对于字节流，效率提升非常高。而字符流本身就带有缓冲区，缓冲字符流相对于字符流效率提升就不是那么大了。详见文末效率对比。





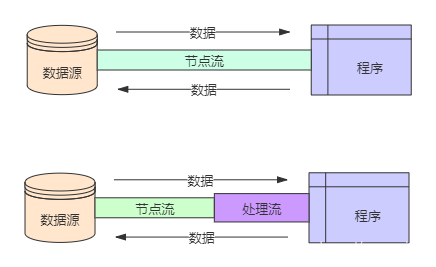
3、节点流和处理流

节点流：直接操作数据读写的流类，比如FileInputStream

处理流：对一个已存在的流的链接和封装，通过对数据进行处理为程序提供功能强大、灵活的读写功能，例如BufferedInputStream（缓冲字节流）

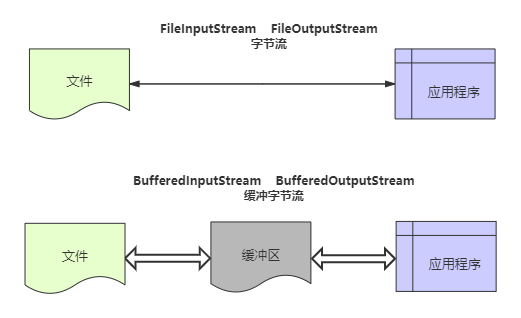
处理流和节点流应用了Java的装饰者设计模式。

下图就很形象地描绘了节点流和处理流，处理流是对节点流的封装，最终的数据处理还是由节点流完成的。

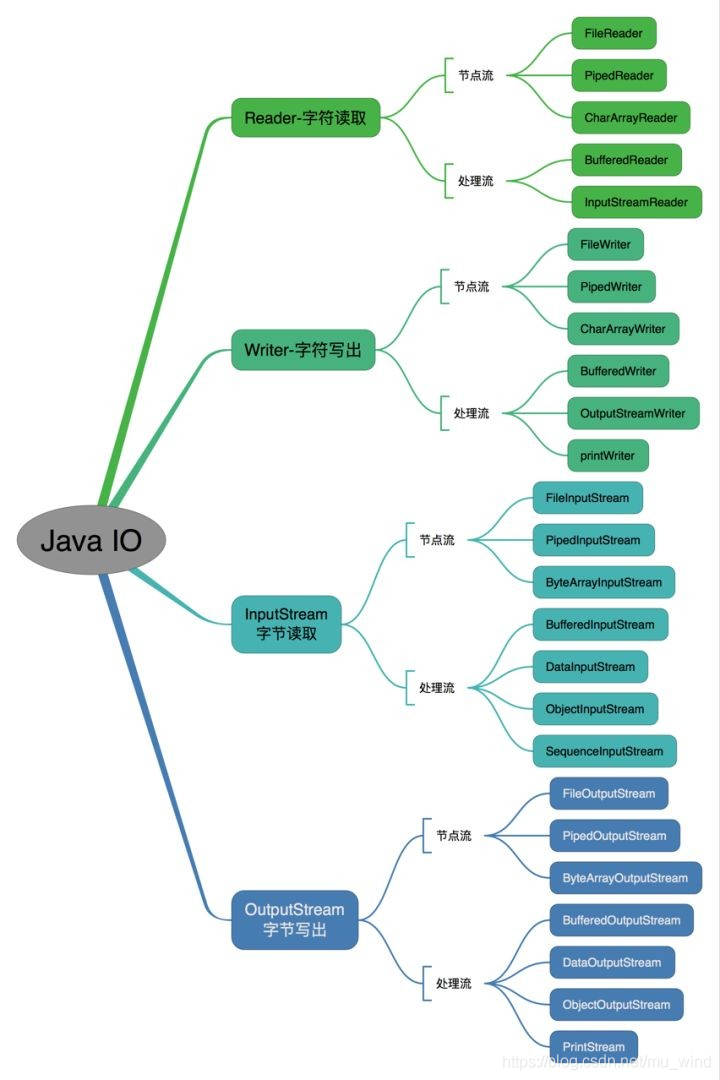


在诸多处理流中，有一个非常重要，那就是缓冲流。

我们知道，程序与磁盘的交互相对于内存运算是很慢的，容易成为程序的性能瓶颈。减少程序与磁盘的交互，是提升程序效率一种有效手段。缓冲流，就应用这种思路：普通流每次读写一个字节，而缓冲流在内存中设置一个缓存区，缓冲区先存储足够的待操作数据后，再与内存或磁盘进行交互。这样，在总数据量不变的情况下，通过提高每次交互的数据量，减少了交互次数。



完整的IO分类图如下：



**2 IO流对象**

第一节中，我们大致了解了IO，并完成了几个案例，但对IO还缺乏更详细的认知，那么接下来我们就对Java IO细细分解，梳理出完整的知识体系来。

Java种提供了40多个类，我们只需要详细了解一下其中比较重要的就可以满足日常应用了

2.1 File类

File类是用来操作文件的类，但它不能操作文件中的数据。

public class File extends Object implements Serializable, Comparable<File>

1

File类实现了Serializable、 Comparable<File>，说明它是支持序列化和排序的。

File类的构造方法

方法名 说明

File(File parent, String child) 根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。

File(String pathname) 通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例。

File(String parent, String child) 根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例。

File(URI uri) 通过将给定的 file: URI 转换为一个抽象路径名来创建一个新的 File 实例。

File类的常用方法

方法 说明

createNewFile() 当且仅当不存在具有此抽象路径名指定名称的文件时，不可分地创建一个新的空文件。

delete() 删除此抽象路径名表示的文件或目录。

exists() 测试此抽象路径名表示的文件或目录是否存在。

getAbsoluteFile() 返回此抽象路径名的绝对路径名形式。

getAbsolutePath() 返回此抽象路径名的绝对路径名字符串。

length() 返回由此抽象路径名表示的文件的长度。

mkdir() 创建此抽象路径名指定的目录。

2.2 字节流

InputStream与OutputStream是两个抽象类，是字节流的基类，所有具体的字节流实现类都是分别继承了这两个类。

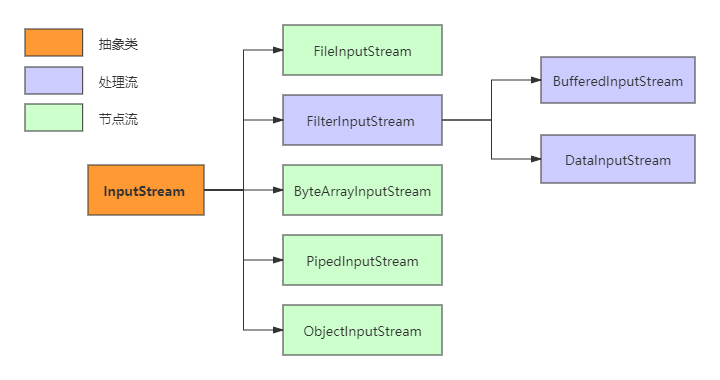
以InputStream为例，它继承了Object，实现了Closeable

public abstract class InputStream

extends Object

implements Closeable

InputStream类有很多的实现子类，下面列举了一些比较常用的：



详细说明一下上图中的类：

InputStream：InputStream是所有字节输入流的抽象基类，前面说过抽象类不能被实例化，实际上是作为模板而存在的，为所有实现类定义了处理输入流的方法。

FileInputSream：文件输入流，一个非常重要的字节输入流，用于对文件进行读取操作。

PipedInputStream：管道字节输入流，能实现多线程间的管道通信。

ByteArrayInputStream：字节数组输入流，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的形式存入到该类中的字节数组中去。

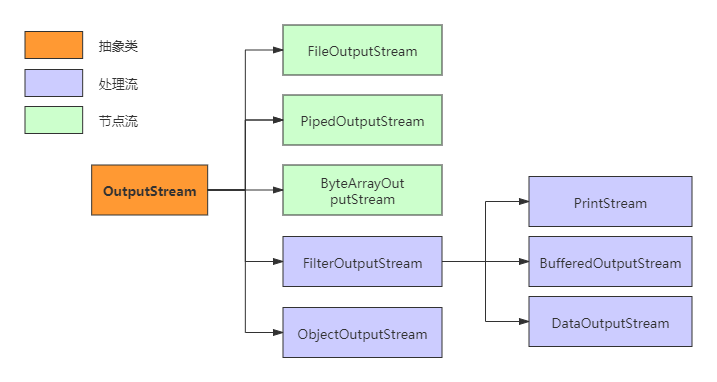
FilterInputStream：装饰者类，具体的装饰者继承该类，这些类都是处理类，作用是对节点类进行封装，实现一些特殊功能。

DataInputStream：数据输入流，它是用来装饰其它输入流，作用是“允许应用程序以与机器无关方式从底层输入流中读取基本 Java 数据类型”。

BufferedInputStream：缓冲流，对节点流进行装饰，内部会有一个缓存区，用来存放字节，每次都是将缓存区存满然后发送，而不是一个字节或两个字节这样发送，效率更高。

ObjectInputStream：对象输入流，用来提供对基本数据或对象的持久存储。通俗点说，也就是能直接传输对象，通常应用在反序列化中。它也是一种处理流，构造器的入参是一个InputStream的实例对象。

OutputStream类继承关系图：

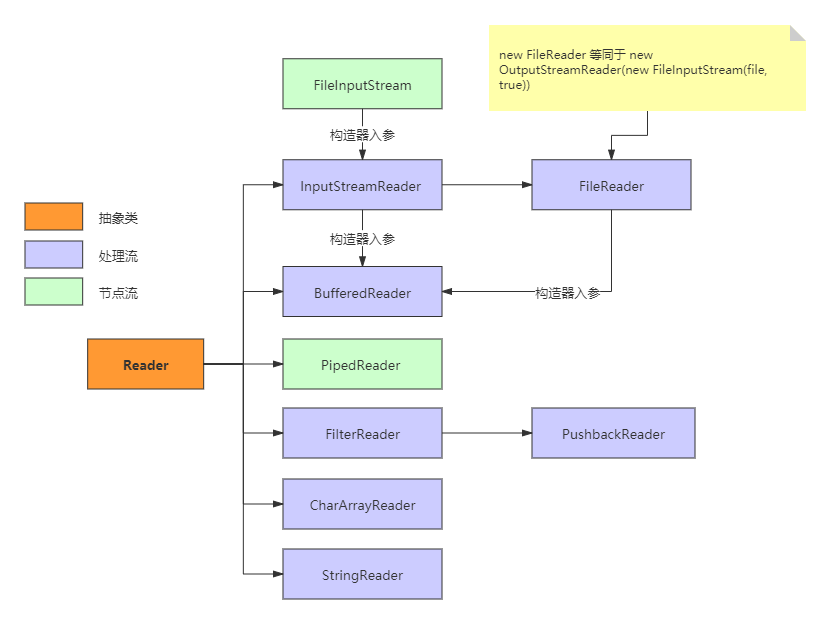


OutputStream类继承关系与InputStream类似，需要注意的是PrintStream.

## 2.3 字符流

与字节流类似，字符流也有两个抽象基类，分别是Reader和Writer。其他的字符流实现类都是继承了这两个类。

以Reader为例，它的主要实现子类如下图：



各个类的详细说明：

InputStreamReader：从字节流到字符流的桥梁（InputStreamReader构造器入参是FileInputStream的实例对象），它读取字节并使用指定的字符集将其解码为字符。它使用的字符集可以通过名称指定，也可以显式给定，或者可以接受平台的默认字符集。

BufferedReader：从字符输入流中读取文本，设置一个缓冲区来提高效率。BufferedReader是对InputStreamReader的封装，前者构造器的入参就是后者的一个实例对象。

FileReader：用于读取字符文件的便利类，new FileReader(File file)等同于new InputStreamReader(new FileInputStream(file, true),"UTF-8")，但FileReader不能指定字符编码和默认字节缓冲区大小。

PipedReader ：管道字符输入流。实现多线程间的管道通信。

CharArrayReader：从Char数组中读取数据的介质流。

StringReader ：从String中读取数据的介质流。

Writer与Reader结构类似，方向相反，不再赘述。唯一有区别的是，Writer的子类PrintWriter。

————————————————

版权声明：本文为CSDN博主「云深i不知处」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/mu\_wind/article/details/108674284