原文参考：

[https://blog.csdn.net/mu\_wind/article/details/108674284?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522168977314816800184124407%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=168977314816800184124407&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~top\_positive~default-1-108674284-null-null.142^v90^koosearch\_v1,239^v2^insert\_chatgpt&utm\_term=IO%E6%B5%81&spm=1018.2226.3001.4187](https://blog.csdn.net/mu_wind/article/details/108674284?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522168977314816800184124407%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=168977314816800184124407&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-108674284-null-null.142%5ev90%5ekoosearch_v1,239%5ev2%5einsert_chatgpt&utm_term=IO%E6%B5%81&spm=1018.2226.3001.4187)

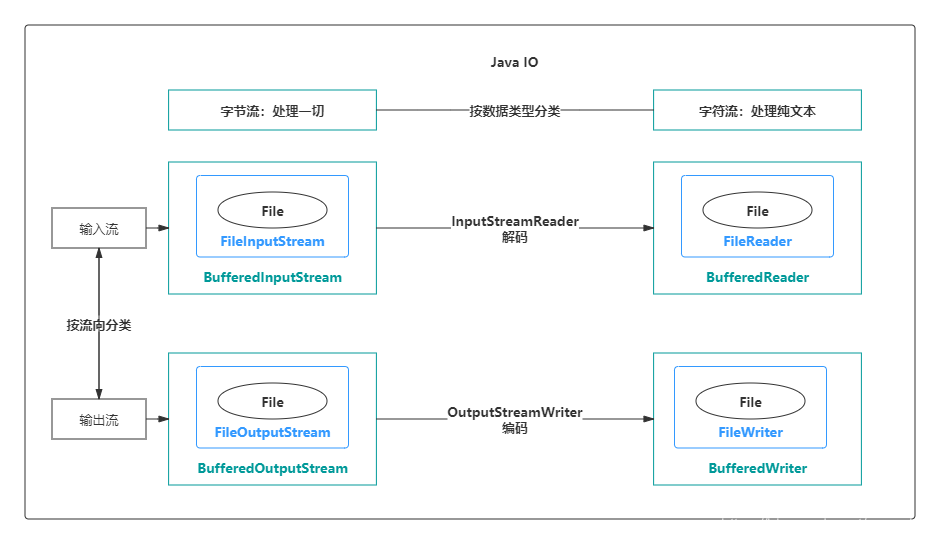
# 1.1 IO流分类

IO流主要的分类方式有以下3种：

按数据流的方向：输入流、输出流

按处理数据单位：字节流、字符流

按功能：节点流、处理流



1、输入流与输出流

输入与输出是相对于应用程序而言的，比如文件读写，读取文件是输入流，写文件是输出流，这点很容易搞反。

2、字节流与字符流

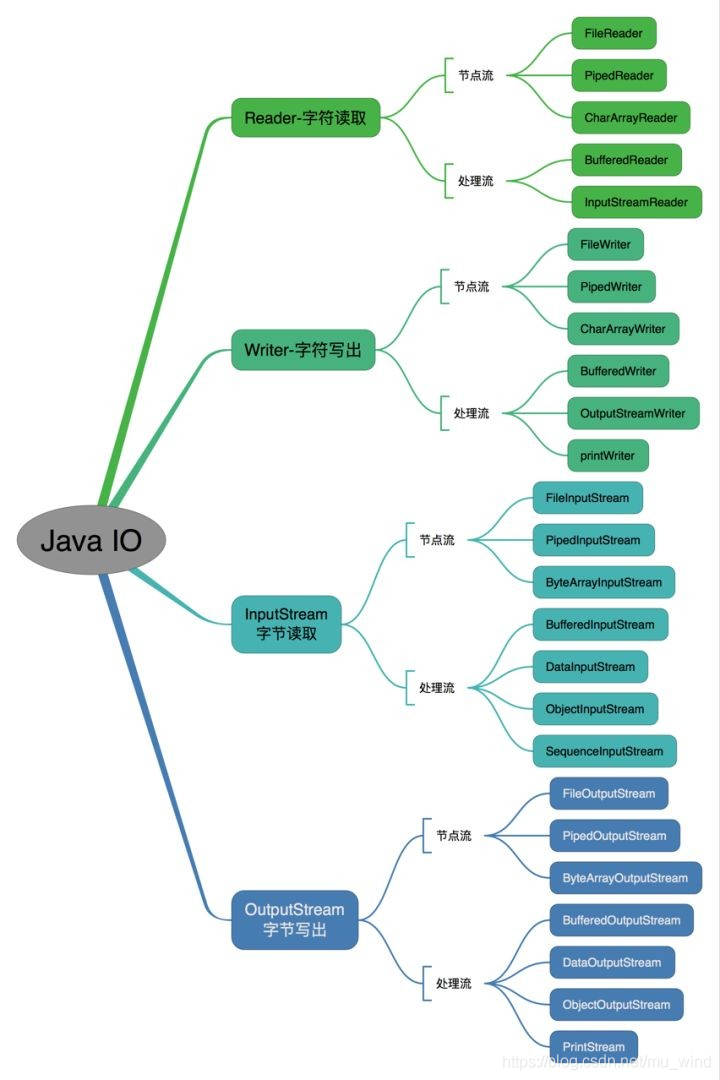
字节可以用于所有东西，字符（文字符号）只能用来处理纯文本

字节流和字符流的用法几乎完成全一样，区别在于字节流和字符流所操作的数据单元不同，字节流操作的单元是数据单元是8位的字节，字符流操作的是数据单元为16位的字符。

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。



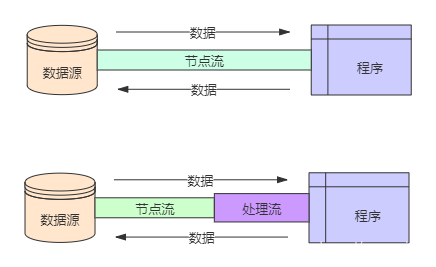
3、节点流和处理流

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。

# 使用方式

1、FileInputStream、FileOutputStream（字节流）

字节流的方式效率较低，不建议使用

2、BufferedInputStream、BufferedOutputStream（缓冲字节流）

缓冲字节流是为高效率而设计的，真正的读写操作还是靠FileOutputStream和FileInputStream，所以其构造方法入参是这两个类的对象也就不奇怪了。

3、InputStreamReader、OutputStreamWriter（字符流）

字符流适用于文本文件的读写，OutputStreamWriter类其实也是借助FileOutputStream类实现的，故其构造方法是FileOutputStream的对象

4、字符流便捷类

Java提供了FileWriter和FileReader简化字符流的读写，new FileWriter等同于new OutputStreamWriter(new FileOutputStream(file, true))

5、BufferedReader、BufferedWriter（字符缓冲流）

# 2 IO流对象

接下来我们就对Java IO细细分解，梳理出完整的知识体系来

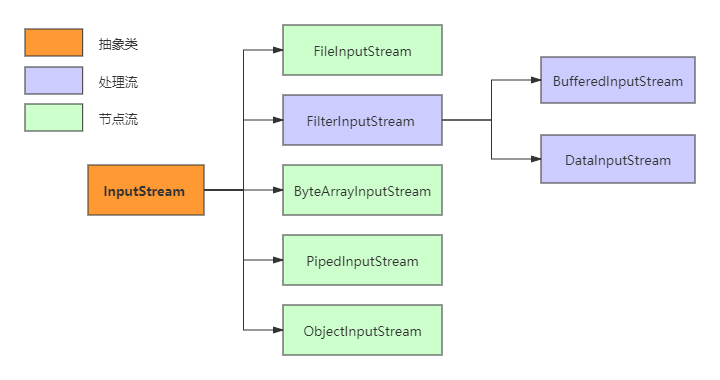
# 2.1 File类

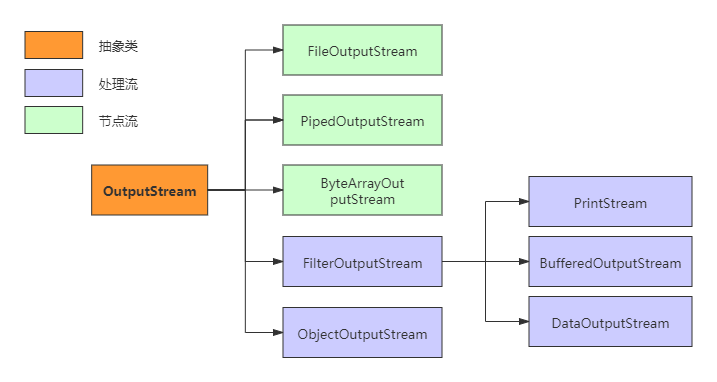
File类是用来操作文件的类，但它不能操作文件中的数据。



# 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。

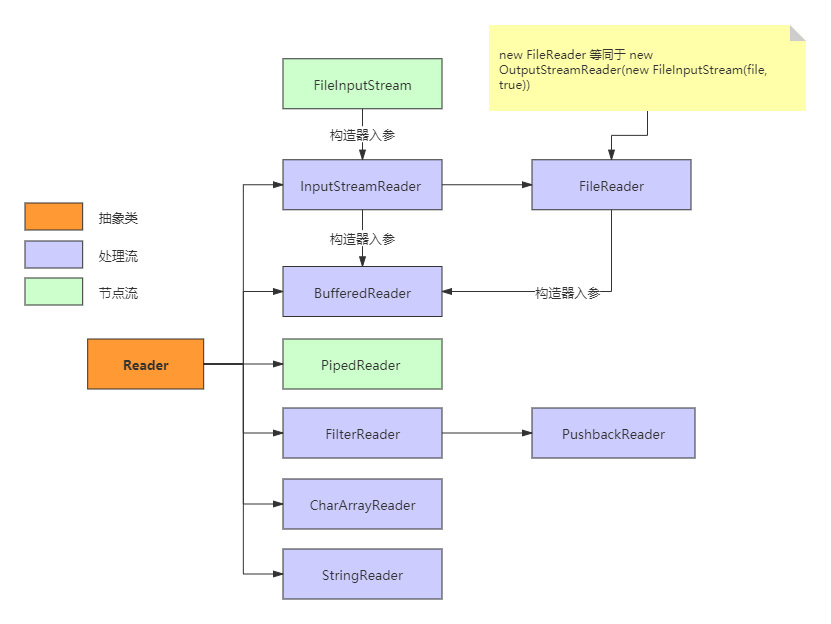




具体解析见原文

# 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。



# 3 IO流方法

# 3.1 字节流方法

字节输入流InputStream主要方法：

1. read() ：从此输入流中读取一个数据字节。
2. read(byte[] b) ：从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中。
3. read(byte[] b, int off, int len) ：从此输入流中将最多 len 个字节的数据读入一个 byte 数组中。
4. close()：关闭此输入流并释放与该流关联的所有系统资源。
5. 字节输出流OutputStream主要方法：
6. write(byte[] b) ：将 b.length 个字节从指定 byte 数组写入此文件输出流中。
7. write(byte[] b, int off, int len) ：将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此文件输出流。
8. write(int b) ：将指定字节写入此文件输出流。
9. close() ：关闭此输入流并释放与该流关联的所有系统资源。

# 3.2 字符流方法

字符输入流Reader主要方法：

1. read()：读取单个字符。
2. read(char[] cbuf) ：将字符读入数组。
3. read(char[] cbuf, int off, int len) ： 将字符读入数组的某一部分。
4. read(CharBuffer target) ：试图将字符读入指定的字符缓冲区。
5. flush() ：刷新该流的缓冲。
6. close() ：关闭此流，但要先刷新它。
7. 字符输出流Writer主要方法：
8. write(char[] cbuf) ：写入字符数组。
9. write(char[] cbuf, int off, int len) ：写入字符数组的某一部分。
10. write(int c) ：写入单个字符。
11. write(String str) ：写入字符串。
12. write(String str, int off, int len) ：写入字符串的某一部分。
13. flush() ：刷新该流的缓冲。
14. close() ：关闭此流，但要先刷新它。

另外，字符缓冲流还有两个独特的方法：

BufferedWriter类newLine() ：写入一个行分隔符。这个方法会自动适配所在系统的行分隔符。

BufferedReader类readLine() ：读取一个文本行。