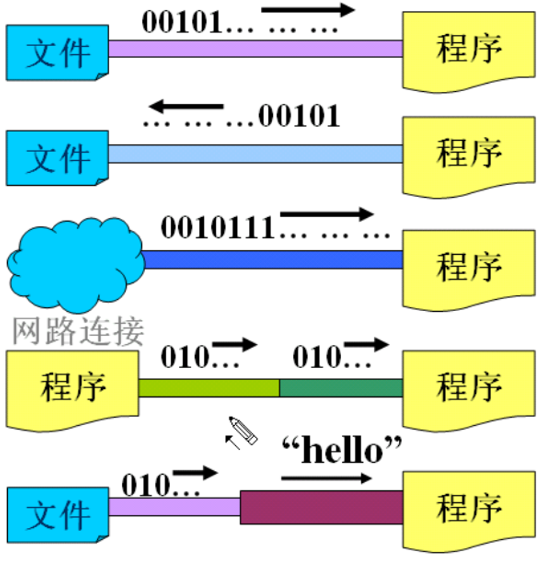
文件流处理方式简图：



按照数据流方向：输入流、输出流

按处理数据单位：字节流、字符流

按功能不同：节点流、处理流。

**节点流**：可以从一个特定的数据源（节点）读写数据（如文件、内存）。

**处理流**：“连接”已经存在的流（节点流或处理流）之上，通过对数据的处理为程序提供更为强大的读写能力。

|  |  |  |
| --- | --- | --- |
|  | 字节流 | 字符流（2字节16bit） |
| 输入流 | InputStream | Reader |
| 输出流 | OutStream | Writer |

**节点流类型**

|  |  |  |
| --- | --- | --- |
| 类型 | 字符流 | 字节流 |
| File | FileReader  FileWriter | FileInputStream  FileOutputStream |
| Memory Array | CharArrayReader  CharArrayWriter | ByteArrayInputStream  ByteArrayOutputStream |
| Memory String | StringReader  StringWriter |  |
| Pipe(管道) | PipedReader  PipedWriter | PipedInputStream  PipedOutputStream |

**处理流类型**

|  |  |  |
| --- | --- | --- |
| 处理类型 | 字符流 | 字节流 |
| Buffering | BufferedReader  BufferedWriter | BufferedInputStream  BufferedOutputStream |
| Filtering | FilterReader  FilterWriter | FilterInputStream  FilterOutputStream |
| Printing | PrintWriter | PrintStream |

**缓冲流**

缓冲流要“套接”在相应的节点流之上，对读写的数据提供了缓冲的功能，提高了读写的效率。

缓冲输入流支持其mark和reset的方法

BufferedReader提供readLine读取一行字符串（以\r或\n分隔）

BufferedWriter提供newLine写入一行

对于输出的缓冲流，写出的数据会先在内存中缓存，使用flush方法将使内存中的数据立刻写出。

I- 就是从硬盘到内存

O- 就是从内存到硬盘

**IO和NIO的区别**

|  |  |
| --- | --- |
| IO | NIO |
| 面向流 | 面向缓冲 |
| 阻塞IO | 非阻塞IO |
| 无 | 选择器 |

面向流意味着每次从流中读一个或多个字节，直至读取所有字节，它们没有被缓存在任何地方。此外，它不能前后移动流中的数据。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。 Java NIO的缓冲导向方法略有不同。数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。这就增加了处理过程中的灵活性。但是，还需要检查是否该缓冲区中包含所有您需要处理的数据。而且，需确保当更多的数据读入缓冲区时，不要覆盖缓冲区里尚未处理的数据。

当程序从硬盘往内存读取数据的时候，操作系统使用**预读来提高性能**，如果我读取了第一扇区的第三磁道的内容，那么你很有可能也会使用第二磁道和第四磁道的内容，所以操作系统会把附近磁道的内容提前读取出来，放在内存中，即**缓存**

**阻塞IO：**从硬盘读取数据，程序一直等，数据读完后，继续操作。

**非阻塞式 IO**（NIO）：从硬盘读取数据，然后程序继续向下执行，等数据读取完后，通知当前程序（对硬件来说叫**中断**，对程序来说叫**回调**），然后此程序可以立即处理数据，也可以执行完当前操作在读取数据。

**选择器（Selectors）**

Java NIO的选择器允许一个单独的线程来监视多个输入通道，可注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。

案例地址: <https://github.com/ben201708/learn_java/tree/master/src/com/io>