Java IO系列课程02

------ Java IO流使用（一）

本文档包含以下内容：

1. 学习目标
2. IO流的概念
3. Java IO 的分类
4. 字符与字节
5. FileInputStream\FileOutputStream
6. BufferedInputStream\BufferedOutputStream
7. FileReader\FileWriter
8. BufferedReader \BufferedWriter
9. 练习题

**前言：**

1. 学习目标

掌握对文件本身的操作。增、删、改、查。

1. IO流概念

**流：**代表任何有能力产出数据的数据源对象或者是有能力接受数据的接收端对象

**流的本质:**数据传输，根据数据传输特性将流抽象为各种类，方便更直观的进行数据操作。

**流的作用：**为数据源和目的地建立一个输送通道。

1. IO流的分类

按照不同的分类方式，可以把流分为不同的类型。常用的分类有三种

**按照流的流向分，可以分为输入流和输出流。**相对于程序而言的输入输出

**输入流：**只能从中读取数据，而不能向其写入数据。 把文件中的信息读取到程序中

**输出流：**只能向其写入数据，而不能向其读取数据。把程序中的信息输出到文件中

**2、按照操作单元划分，可以划分为字节流和字符流**。

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。

字节流主要是由InputStream和outPutStream作为基类，而字符流则主要有Reader和Writer作为基类。

**3、节点流和处理流**



1. InputStream 是所有的输入字节流的父类，它是一个抽象类，主要包含三个方法：

|  |
| --- |
| //读取一个字节并以整数的形式返回(0~255),如果返回-1已到输入流的末尾。   int read() ； 字节数转成int   //读取一系列字节并存储到一个数组buffer，返回实际读取的字节数，如果读取前已到输入流的末尾返回-1。   int read(byte[] buffer) ；   //读取length个字节并存储到一个字节数组buffer，从off位置开始存,最多len， 返回实际读取的字节数，如果读取前以到输入流的末尾返回-1。   int read(byte[] buffer, int off, int len) ； |

2.Reader

|  |
| --- |
| //读取一个字符并以整数的形式返回(0~255),如果返回-1已到输入流的末尾。  返回的int值的含义，每一个二进制位（bit）有0和1两种状态，八个二进制位（bit）被称为一个字节（Byte），一个字节可以表示出2^8=256种状态，从0000 0000到1111 1111。每个状态可以对应一个符号。   int read() ；   //读取一系列字符并存储到一个数组buffer，返回实际读取的字符数，如果读取前已到输入流的末尾返回-1。   int read(char[] cbuf) ；   //读取length个字符,并存储到一个数组buffer，从off位置开始存,最多读取len，返回实际读取的字符数，如果读取前以到输入流的末尾返回-1。   int read(char[] cbuf, int off, int len)  ，如果读取前以到输入流的末尾返回NUll   String readLine() null  //对比InputStream和Reader所提供的方法，就不难发现两个基类的功能基本一样的，  //只不过读取的数据单元不同。在执行完流操作后，要调用close()方法来关系输入流，  //因为程序里打开的IO资源不属于内存资源，垃圾回收机制无法回收该资源，所以应该显式关闭文件IO资源。  //除此之外，InputStream和Reader还支持如下方法来移动流中的指针位置：  //在此输入流中标记当前的位置   //readlimit - 在标记位置失效前可以读取字节的最大限制。   void mark(int readlimit)   // 测试此输入流是否支持 mark 方法   boolean markSupported()   // 跳过和丢弃此输入流中数据的 n 个字节/字符   long skip(long n)   //将此流重新定位到最后一次对此输入流调用 mark 方法时的位置 |

  3.OutputStream

   OutputStream 是所有的输出字节流的父类，它是一个抽象类，主要包含如下四个方法：

|  |
| --- |
| //向输出流中写入一个字节数据,该字节数据为参数b的低8位。   void write(int b) ; a ===> 97 read()   //将一个字节类型的数组中的数据写入输出流。   void write(byte[] b);   //将一个字节类型的数组中的从指定位置（off）开始的,len个字节写入到输出流。   void write(byte[] b, int off, int len);   //将输出流中缓冲的数据全部写出到目的地。   void flush(); |

4.Writer

 Writer 是所有的输出字符流的父类，它是一个抽象类,主要包含如下六个方法：

|  |
| --- |
| //向输出流中写入一个字符数据,该字节数据为参数b的低16位。   void write(int c);   //将一个字符类型的数组中的数据写入输出流，   void write(char[] cbuf)   //将一个字符类型的数组中的从指定位置（offset）开始的,length个字符写入到输出流。   void write(char[] cbuf, int offset, int length);   //将一个字符串中的字符写入到输出流。   void write(String string);   //将一个字符串从offset开始的length个字符写入到输出流。   void write(String string, int offset, int length);   //将输出流中缓冲的数据全部写出到目的地。   void flush() //将缓冲区中的数据刷新出去 |
|  |

FileInputStream

|  |
| --- |
| FileInputStream(File file) // 创建“File对象”对应的“文件输入流”  FileInputStream(FileDescriptor fd) // 创建“文件描述符”对应的“文件输入流”    可以看出，FileDescriptor可以看做一种指向文件引用的抽象化概念。它能表示一个开放文件，一个开放的socket或者一个字节的源。它最主要的用途就是去创建FileInputStream或者FileOutputStream。并且也说了不应该创建应用自己的文件描述符。  FileInputStream(String path) // 创建“文件(路径为path)”对应的“文件输入流”  int available() // 返回“剩余的可读取的字节数”或者“skip的字节数”  void close() // 关闭“文件输入流”  FileChannel getChannel() // 返回“FileChannel”  final FileDescriptor getFD() // 返回“文件描述符”  int read() // 返回“文件输入流”的下一个字节  int read(byte[] buffer, int off, int len) // 读取“文件输入流”的数据并存在到buffer，从off开始存储，存储长度是len。  long skip(long n) // 跳过n个字节 |

FileOutputStream

|  |
| --- |
| FileOutputStream(File file);//如果文件不存在则创建文件，如果文件存在则删除文件后重新创建文件。  FileOutputStream(File file, boolean append);//第二个参数如果为true的话在文件后面追加写入的内容。  FileOutputStream(String name);//和第1个一样  FileOutputStream(String name, boolean append)  FileOutputStream(FileDescriptor fdObj);//通过文件描述符创建文件输出字节流  //文件描述符表示一个到文件系统中的某个实际文件的现有连接。    void write(byte[] b);//把字节数组写入文件  void write(byte[] b, int off, int len);//把从字节数组b中的off下标开始，把字节写入文件，写入len个字节。  void write(int b);把int低八位写入文件，则写一个int(32位)要写四次: 每次右移八位. |

BufferedWriter

|  |
| --- |
| void close() 关闭此流、释放与此流有关的资源。  void flushBuffer() 将cb中缓存的字符flush到底层out中、  void flush() 刷新此流、同时刷新底层out流  void newLine() 写入一个换行符。  void write(int c) 将一个单个字符写入到cb中。  void write(char cbuf[], int off, int len) 将一个从下标off开始长度为len个字符写入cb中  void write(String s, int off, int len) 将一个字符串的一部分写入cb中  从流里面读取文本，通过缓存的方式提高效率，读取的内容包括字符、数组和行。  缓存的大小可以指定，也可以用默认的大小。大部分情况下，默认大小就够了 |

BufferedReader

|  |
| --- |
| void close() 关闭此流、释放与此流有关的所有资源  void mark(int readAheadLimit) 标记此流此时的位置  boolean markSupported() 判断此流是否支持标记  void reset() 重置in被最后一次mark的位置  需要注意的是这里提供了mark和reset功能，也就是可以记住读取的位置，将来可以回滚，重新读取。这需要在读取数据时避免对缓冲区中的这部分数据覆盖掉，需要保存起来，同时保存的长度可以在mark的时候指定。调用reset可以回滚，但是必须mark过，而且mark过后读取的数据不能超过mark的时候设置的大小，否则会失效，调用reset会抛出异常。  boolean ready() 判断此流是否可以读取字符  int read() 读取单个字符、以整数形式返回。如果读到in的结尾则返回-1。  int read(char[] cbuf, int off, int len) 将in中len个字符读取到cbuf从下标off开始长度len中  String readLine() 读取一行  long skip(long n) 丢弃in中n个字符！ |

1. 带Buffered的处理流

**1.为什么需要带Buffered处理流？**

为了提高字符读取的效率。

BufferedInputStream和BufferedOutputStream这两个类分别是FilterInputStream和FilterOutputStream的子类，作为装饰器子类。并且构造函数需要FilterInputStream/FilterOutputStream的子类入参。

BufferedReader和BufferedWriter这两个类分别是Reader和Writer的子类，作为装饰器子类。并且构造函数需要Reader/Writer的子类入参。

使用它们可以防止每次读取/发送数据时进行实际的写操作，代表着使用缓冲区。我们有必要知道不带缓冲的操作，每读一个字节就要写入一个字节，由于涉及磁盘的IO操作相比内存的操作要慢很多，所以不带缓冲的流效率很低。带缓冲的流，可以一次读很多字节，但不向磁盘中写入，只是先放到内存里。等凑够了缓冲区大小的时候一次性写入磁盘，这种方式可以减少磁盘操作次数，速度就会提高很多！

也就是说如果当前缓冲区没有数据，则调用底层reader去读取数据到缓冲区；如果有数据则直接读取。

默认的缓冲大小是8k，也就是每次读取都是8k为单位。

BufferedReader中还提供了一行一行读取的功能readLine函数，这不是Reader中的方法，会把换行符（\r、\n、\r\n）去掉。

**2.flush的作用**

BufferedOutputStream在close()时会自动flush   
 BufferedOutputStream或者Bufferedwriter在不调用close()的情况下，缓冲区不满，又需要把缓冲区的内容写入到文件或通过网络发送到别的机器时，才需要调用flush。