Java IO 学习笔记：概念与原理

一、概念

Java 中对文件的操作是以流的方式进行的。流是 Java 内存中的一组有序数据序列。Java 将数据从源（文件、内存、键盘、网络）读入到内存中，形成了流，然后将这些流还可以写到另外的目的地（文件、内存、控制台、网络），之所以称为流，是因为这个数据序列在不同时刻所操作的是源的不同部分。

二、分类

流的分类，Java 的流分类比较丰富，刚接触的人看了后会感觉很晕。流分类的方式很多：

1、按照输入的方向分，输入流和输出流，输入输出的参照对象是 Java 程序。

2、按照处理数据的单位不同分，字节流和字符流，字节流读取的最小单位是一个字节（1byte=8bit），而字符流一次可以读取一个字符（1char = 2byte = 16bit）。

3、按照功能的不同分，分节点流和处理流，节点流是直接从一个源读写数据的流（这个流没有经过包装和修饰），处理流是在对节点流封装的基础上的一种流，FileInputStream 是一个接点流，可以直接从文件读取数据，但是 BufferedInputStream 可以包装 FileInputStream，使得其有缓冲功能。

其实除了以上三种分类外，还有一些常常听到的一些分类比如：对象流、缓冲流、压缩流、文件流等等。其实都是节点流和处理流的子分类。当然你也可以创建新的流类型，只要你需要。

三、流分类的关系

不管流的分类是多么的丰富和复杂，其根源来自于四个基本的类。这个四个类的关系如下：

	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

四、字节流和字符流的相互转换

1、从字节流到字符流：InputStreamReader、OutputStreamWriter 类可以实现。

2、从字符流到字节流：可以从字符流中获取 char[]数组，转换为 String，然后调用 String 的 API 函数 getBytes() 获取到 byte[]，然后就可以通过 ByteArrayInputStream、ByteArrayOutputStream 来实现到字节流的转换。

以上知识是学习 Java 流的根基，对流的操作非常的容易，Java API 中提供了丰富的流处理类，API 也大差不差，看看文档即可上手。

Java IO 学习笔记：字节流

字节流是最基本的流，文件的操作、网络数据的传输等等都依赖于字节流。而字符流常常用于读取文本类型的数据或字符串流的操作等等。

关于字节流的 API，没什么好说的，看看就知道了。这里挑几个关键点：

一、InputStream 的 API

1、public int read()

从输入流读取下一个数据字节。返回 0 到 255 范围内的 int 字节值。如果因已到达流末尾而没有可用的字节，则返回值 -1。

2、public int read(byte[] b)

从输入流中读取一定数量的字节并将其存储在缓冲区数组 b 中。以整数形式返回实际读取的字节数。如果因为流位于文件末尾而没有可用的字节，则返回值 -1；否则，至少可以读取一个字节并将其存储在 b 中。此方法等同于 read(b, 0, b.length)

3、public int read(byte[] b, int off, int len)

将输入流中最多 len 个数据字节读入字节数组。尝试读取多达 len 字节，但可能读取较少数量。以整数形式返回实际读取的字节数。如果由于已到达流末尾而不再有数据，则返回 -1。

参数：

b - 读入数据的缓冲区。off - 在其处写入数据的数组 b 的初始偏移量。len - 要读取的最大字节数。

二、OutputStream 的 API

1、public void write(int b)

将指定的字节写入此输出流。write 的常规协定是：向输出流写入一个字节。要写入的字节是参数 b 的八个低位。b 的 24 个高位将被忽略。

2、public void write(byte[] b)

将 b.length 个字节从指定的字节数组写入此输出流。write(b) 的常规协定是：应该与调用 write(b, 0, b.length) 的效果完全相同。

3、public void write(byte[] b, int off, int len)

将指定字节数组中从偏移量 off 开始的 len 个字节写入此输出流。write(b, off, len) 的常规协定是：将数组 b 中的某些字节按顺序写入输出流；元素 b[off] 是此操作写入的第一个字节，b[off+len-1] 是此操作写入的最后一个字节。

参数：b - 数据。off - 数据中的初始偏移量。len - 要写入的字节数。

4、public void flush()

刷新此输出流并强制写出所有缓冲的输出字节。flush 的常规协定是：如果此输出流的实现已经缓冲了以前写入的任何字节，则调用此方法指示应将这些字节立即写入它们预期的目标。

三、几点原则

1、不管是输入还是输出流，使用完毕后要 close()，如果是带有缓冲区的输出流，应在关闭前调用 flush()。

2、应该尽可能使用缓冲区，来减少 IO 次数，以提高性能。

3、能用字符流处理的不用字节流。

四、例子

下面是一个操作字节流的例子：

要操作的文本文件 x.txt

```
白日依山尽，黄河入海流。
欲穷千里目，更上一层楼。

—— 王之涣《登鹳雀楼》登
```

```
import java.io.*;

/**
* Created by IntelliJ IDEA.
*
* @author leizhimin 2008-8-27 22:16:44
*/
public class TestIOStream {
    public static void main(String[] args) {
        testStream();
        testBufferedStream();
        testSelectStream();
    }

    /**
     * 字节流测试
     */
    public static void testStream() {
        InputStream fis = null;
        OutputStream fos = null;
        try {
            fis = new FileInputStream("C:\\x.txt");
            fos = new FileOutputStream("C:\\xcopy.txt");
            long num = 0;   //读取字节计数
            int bt = 0;     //每次读入字节内容
            //当读入文件末尾时，读入数据的值为-1
            //每次读入一个字节，存放到变量 bt 中，直到读完整个文件
            while ((bt = fis.read()) != -1) {
//                System.out.print(bt);   //以数字的形式逐个输出文件的每个字节
                System.out.print((char) bt);   //以字母的形式逐个输出文件的每个字节
                fos.write(bt); //将字节写入输出流中，实现文件的 copy 功能
                num++;
            }
            System.out.println("读取的字节数为" + num);
            fis.close();
            fos.close();
```

```
        } catch (FileNotFoundException e) {
            System.out.println("找不到指定的文件！");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("文件读取时发生 IO 异常！");
            e.printStackTrace();
        }
    }

    /**
     * 缓冲的字节流测试
     */
    public static void testBufferedStream() {
        int buffer = 10; //缓冲大小
        try {
            BufferedInputStream bis = new BufferedInputStream(new FileInputStrea
m("C:\\x.txt"));
            BufferedOutputStream bos = new BufferedOutputStream(new FileOutput
Stream("C:\\bf2.txt"));
            int bench = 0;
            byte bts[] = new byte[buffer];      //创建字节流缓存
            while ((bis.read(bts)) != -1) {
                bos.write(bts);  //将字节写入输出流中，实现文件的 copy 功能
                bench++;
            }
            System.out.println("bench=" + bench);
            //将输入流缓冲区中的数据全部写出（千万记住)
            bos.flush();
            bis.close();
            bos.close();
        } catch (FileNotFoundException e) {
            System.out.println("找不到指定的文件！");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("文件读取时发生 IO 异常！");
            e.printStackTrace();
        }
    }

    /**
     * 字节流的选择读取测试
     */
    public static void testSelectStream() {
        OutputStream fos = null;
```

```
        int buffer = 25;
        try {
            BufferedInputStream bis = new BufferedInputStream(new FileInputStrea
m("C:\\x.txt"));
            fos = new FileOutputStream("C:\\testSelectStream.txt");

            byte bts[] = new byte[buffer];      //创建缓存
            //从输入流的第 5 个字节开始，往后读取 10 个字节，存放到缓存 bts 中
            //这个方法有个陷阱，缓存 buffer 的大小最小为“偏移量+要读取字节数”，在次最小
应该为 15，否则抛 IndexOutOfBoundsException 异常
            bis.read(bts, 5, 10);
            //将字节写入输出流中，实现文件的 copy 功能
            fos.write(bts);

            bis.close();
            fos.close();
        } catch (FileNotFoundException e) {
            System.out.println("找不到指定的文件！");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("文件读取时发生 IO 异常！");
            e.printStackTrace();
        }
    }
}
```

注意了：

1、缓冲的功能应该通过相应的缓冲流来包装原始流来实现，而不是自己连续多次数据，最后写到一个数组中，这是很愚昧的做法（但是还有很多人在用）。

2、read(byte[] b, int off, int len)这个方法要好好体会了，往往和你想象的不一样。

3、将读取的一个字节强制转换为 char 是不合适的，除非你想看看能输出什么。

Java IO 学习笔记：字符流

字符流的处理和字节流差不多，API 基本上完全一样，就是计量单位不同。另外字符流还提供一些其他的处理流，比如按行读取流、字符串流等等。

下面给个例子看看：

```
import java.io.*;

/**
* 字符流测试 *
* @author leizhimin 2008-8-27 22:16:44
*/
public class TestIOStream {
    public static void main(String[] args) {
        testReaderWriter();
        testLineNumberReader();
    }

    /**
     * 带缓冲的字符流
     */
    public static void testReaderWriter() {
        int bufsize = 25;
        try {
            BufferedReader bufferedReader = new BufferedReader(new FileReader(n
ew File("C:\\x.txt")));
            BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(ne
w File("C:\\xb.txt")));

            char buf[] = new char[bufsize];    //字符缓冲区
            while (bufferedReader.read(buf) != -1) {
                bufferedWriter.write(buf);
            }
            bufferedWriter.flush();
            bufferedReader.close();
            bufferedWriter.close();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
```

```
    * 按行读取的字符流
    */
   public static void testLineNumberReader() {
      try {
         LineNumberReader lineNumberReader = new LineNumberReader(new Bu
fferedReader(new FileReader(new File("C:\\x.txt"))));

         String lineString;     //行字符串变量
         int x = 0;           //行号
         while ((lineString = lineNumberReader.readLine()) != null) {
            x++;
            System.out.println("行号：" + x + " >>>" + lineString);
         }
         lineNumberReader.close();
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

运行结果：

```
行号：1 >>>白日依山尽，黄河入海流。
行号：2 >>>欲穷千里目，更上一层楼。
行号：3 >>>
行号：4 >>>—— 王之涣《登鹳雀楼》
```

Java IO 的一般使用原则

一、按数据来源（去向）分类：

1 、是文件： FileInputStream, FileOutputStream (字节流)FileReader, FileWriter(字符)

2 、是 byte[] ： ByteArrayInputStream, ByteArrayOutputStream(字节流)

3 、是 Char[]: CharArrayReader, CharArrayWriter(字符流)

4 、是 String: StringBufferInputStream, StringBufferOuputStream (字节流)StringReader, StringWriter(字符流)

5 、网络数据流： InputStream, OutputStream(字节流) Reader, Writer(字符流)

二、按是否格式化输出分：

要格式化输出： PrintStream, PrintWriter

三、按是否要缓冲分：

要缓冲： BufferedInputStream, BufferedOutputStream(字节流) BufferedReader, BufferedWriter(字符流)

四、按数据格式分：

1 、二进制格式（只要不能确定是纯文本的）：InputStream, OutputStream 及其所有带 Stream 结束的子类

2 、纯文本格式（含纯英文与汉字或其他编码方式）； Reader, Writer 及其所有带 Reader, Writer 的子类

五、按输入输出分：

1 、输入： Reader, InputStream 类型的子类

2 、输出： Writer, OutputStream 类型的子类

六、特殊需要：

1 、从 Stream 到 Reader,Writer 的转换类： InputStreamReader, OutputStreamWriter

2 、对象输入输出： ObjectInputStream, ObjectOutputStream

3 、进程间通信： PipeInputStream, PipeOutputStream, PipeReader, PipeWriter

4 、合并输入： SequenceInputStream

5 、更特殊的需要： PushbackInputStream, PushbackReader, LineNumberInputStream, LineNumberReader

决定使用哪个类以及它的构造进程的一般准则如下（不考虑特殊需要）：

首先，考虑最原始的数据格式是什么：原则四

第二，是输入还是输出：原则五

第三，是否需要转换流：原则六第 1 点

第四，数据来源（去向）是什么：原则一

第五，是否要缓冲：原则三（特别注明：一定要注意的是 readLine() 是否有定义，有什么比 read, write 更特殊的输入或输出方法）

第六，是否要格式化输出：原则二