第 24 天 IO 流

今日内容介绍

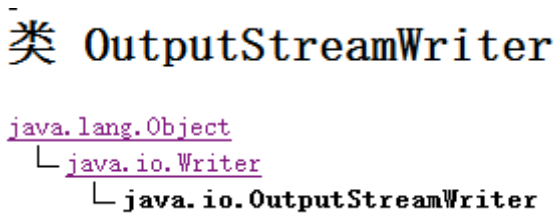
- 转换流
- 缓冲流

第1章 转换流

在学习字符流(FileReader、FileWriter)的时候，其中说如果需要指定编码和缓冲区大小时，可以在字节流的基础上，构造一个 InputStreamReader 或者 OutputStreamWriter，这又是什么意思呢？

1.1 OutputStreamWriter 类

查阅 OutputStreamWriter 的 API 介绍，OutputStreamWriter 是字符流通向字节流的桥梁：可使用指定的字符编码表，将要写入流中的字符编码成字节。它的作用的就是，将字符串按照指定的编码表转成字节，在使用字节流将这些字节写出去。



- 代码演示：

```
public static void writeCN() throws Exception {
        //创建与文件关联的字节输出流对象
        FileOutputStream fos = new FileOutputStream("c:\\cn8.txt");
        //创建可以把字符转成字节的转换流对象，并指定编码
        OutputStreamWriter osw = new OutputStreamWriter(fos,"utf-8");
        //调用转换流，把文字写出去，其实是写到转换流的缓冲区中
        osw.write("你好");//写入缓冲区。
        osw.close();
    }
```

OutputStreamWriter 流对象，它到底如何把字符转成字节输出的呢？

其实在 OutputStreamWriter 流中维护自己的缓冲区，当我们调用 OutputStreamWriter 对象的 write 方法时，会拿着字符到指定的码表中进行查询，把查到的字符编码值转成字节数存放到 OutputStreamWriter 缓冲区中。然后再调用刷新功能，或者关闭流，或者缓冲区存满后会把缓冲区中的字节数据使用字节流写到指定的文件中。

1.2InputStreamReader 类

查阅 InputStreamReader 的 API 介绍，InputStreamReader 是字节流通向字符流的桥梁：它使用指定的字符编码表读取字节并将其解码为字符。它使用的字符集可以由名称指定或显式给定，或者可以接受平台默认的字符集。

类 InputStreamReader

java.lang.Object
  └ java.io.Reader
      └ **java.io.InputStreamReader**

- 代码演示

```
public class InputStreamReaderDemo {
    public static void main(String[] args) throws IOException {
        //演示字节转字符流的转换流
        readCN();
    }
    public static void readCN() throws IOException{
        //创建读取文件的字节流对象
        InputStream in = new FileInputStream("c:\\cn8.txt");
        //创建转换流对象
        //InputStreamReader isr = new InputStreamReader(in);这样创建对象，会用本地默认码表读取，将会发生错误解码的错误
        InputStreamReader isr = new InputStreamReader(in,"utf-8");
        //使用转换流去读字节流中的字节
        int ch = 0;
        while((ch = isr.read())!=-1){
            System.out.println((char)ch);
        }
        //关闭流
        isr.close();
    }
}
```

注意：在读取指定的编码的文件时，一定要指定编码格式，否则就会发生解码错误，而发生乱码现象。

1.3转换流和子类区别

发现有如下继承关系：

OutputStreamWriter:

|--FileWriter:

InputStreamReader:

|--FileReader;

父类和子类的功能有什么区别呢？

OutputStreamWriter 和 InputStreamReader 是字符和字节的桥梁：也可以称之为字符转换流。字符转换流原理：字节流+编码表。

FileWriter 和 FileReader：作为子类，仅作为操作字符文件的便捷类存在。当操作的字符文件，使用的是默认编码表时可以不用父类，而直接用子类就完成操作了，简化了代码。

InputStreamReader isr = new InputStreamReader(new FileInputStream("a.txt"));//默认字符集。

InputStreamReader isr = new InputStreamReader(new FileInputStream("a.txt"),"GBK");//指定 GBK 字符集。

FileReader fr = new FileReader("a.txt");

这三句代码的功能是一样的，其中第三句最为便捷。

注意：一旦要指定其他编码时，绝对不能用子类，必须使用字符转换流。什么时候用子类呢？

条件：

1、操作的是文件。2、使用默认编码。

总结：

字节--->字符 ： 看不懂的--->看的懂的。 需要读。输入流。 InputStreamReader

字符--->字节 ： 看的懂的--->看不懂的。 需要写。输出流。 OutputStreamWriter

第2章 缓冲流

在我们学习字节流与字符流的时候，大家都进行过读取文件中数据的操作，读取数据量大的文件时，读取的速度会很慢，很影响我们程序的效率，那么，我想提高速度，怎么办？

Java 中提高了一套缓冲流，它的存在，可提高 IO 流的读写速度

缓冲流，根据流的分类分类字节缓冲流与字符缓冲流。

2.1 字节缓冲流

字节缓冲流根据流的方向，共有 2 个

- 写入数据到流中，字节缓冲输出流 BufferedOutputStream
- 读取流中的数据，字节缓冲输入流 BufferedInputStream

它们的内部都包含了一个缓冲区，通过缓冲区读写，就可以提高了 IO 流的读写速度

2.1.1 字节缓冲输出流 BufferedOutputStream

通过字节缓冲流，进行文件的读写操作 写数据到文件的操作

- 构造方法

  public BufferedOutputStream(OutputStream out)创建一个新的缓冲输出流，以将数据写入指定的底层输出流。

```
public class BufferedOutputStreamDemo01 {
    public static void main(String[] args) throws IOException {

        //写数据到文件的方法
        write();
    }

    /*
     * 写数据到文件的方法
     * 1,创建流
     * 2，写数据
     * 3，关闭流
     */
    private static void write() throws IOException {
        //创建基本的字节输出流
        FileOutputStream fileOut = new FileOutputStream("abc.txt");
        //使用高效的流，把基本的流进行封装，实现速度的提升
        BufferedOutputStream out = new BufferedOutputStream(fileOut);
        //2，写数据
        out.write("hello".getBytes());
        //3，关闭流
        out.close();
    }
}
```

2.1.2 字节缓冲输入流 BufferedInputStream

刚刚我们学习了输出流实现了向文件中写数据的操作，那么，现在我们完成读取文件中数据的操作

- 构造方法

  public BufferedInputStream(InputStream in)

```
    /*
     * 从文件中读取数据
     * 1,创建缓冲流对象
     * 2，读数据，打印
     * 3，关闭
     */
    private static void read() throws IOException {
        //1,创建缓冲流对象
        FileInputStream fileIn = new FileInputStream("abc.txt");
        //把基本的流包装成高效的流
        BufferedInputStream in = new BufferedInputStream(fileIn);
```

```
        //2，读数据
        int ch = -1;
        while ( (ch = in.read()) != -1 ) {
            //打印
            System.out.print((char)ch);
        }
        //3，关闭
        in.close();
    }
```

2.1.3 使用基本的流与高效的流完成复制文件

我们一直在说，高效的流速度快并高效，怎么体现呢？需要通过一个复制文件耗时的比较过程，来体验一下高效流带来的快感。

```
/*
 * 需求：将 d:\\test.avi 文件进行复制
 *       采用 4 种方式复制
 *       方式 1：  采用基本的流，一次一个字节的方式复制  共耗时 224613 毫秒
 *       方式 2：  采用基本的流，一个多个字节的方式赋值  共耗时     327 毫秒
 *       方式 3：  采用高效的流，一次一个字节的方式复制  共耗时    2047 毫秒
 *       方式 4：  采用高效的流，一个多个字节的方式赋值  共耗时      96 毫秒
 *
 * 数据源： d:\\test.avi
 * 目的地 1：d:\\copy1.avi
 * 目的地 2：d:\\copy2.avi
 * 目的地 3：d:\\copy3.avi
 * 目的地 4：d:\\copy4.avi
 *
 * 实现的步骤：
 *  1，指定数据源
 *  2，指定目的地
 *  3，读数据
 *  4，写数据
 *  5，关闭流
 *
 */
public class CopyAVI {
    public static void main(String[] args) throws IOException {
        //开始计时
        long start = System.currentTimeMillis();
        //方式 1：  采用基本的流，一次一个字节的方式复制
        //method1("d:\\test.avi", "d:\\copy1.avi");
        //方式 2：  采用基本的流，一个多个字节的方式赋值
        //method2("d:\\test.avi", "d:\\copy2.avi");
```

```
            //方式 3： 采用高效的流，一次一个字节的方式复制
            //method3("d:\\test.avi", "d:\\copy3.avi");
            //方式 4： 采用高效的流，一个多个字节的方式赋值
            method4("d:\\test.avi", "d:\\copy4.avi");

            //结束计时
            long end = System.currentTimeMillis();
            //打印耗时多少毫秒
            System.out.println("共耗时 " +(end - start)+ "毫秒");
        }

        //方式 4： 采用高效的流，一个多个字节的方式赋值
        private static void method4(String src, String dest) throws IOException {
            //1，指定数据源
            BufferedInputStream in = new BufferedInputStream(new FileInputStream(src));
             //2，指定目的地
            BufferedOutputStream out = new BufferedOutputStream(new
FileOutputStream(dest));
             //3，读数据
            byte[] buffer = new byte[1024];
            int len = -1;
            while ( (len = in.read(buffer)) != -1) {
                //4,写数据
                out.write(buffer, 0, len);
            }
             //5，关闭流
            in.close();
            out.close();
        }

        //方式 3： 采用高效的流，一次一个字节的方式复制
        private static void method3(String src, String dest) throws IOException {
            //1，指定数据源
            BufferedInputStream in = new BufferedInputStream(new FileInputStream(src));
             //2，指定目的地
            BufferedOutputStream out = new BufferedOutputStream(new
FileOutputStream(dest));
             //3，读数据
            int ch = -1;
            while ((ch=in.read()) != -1) {
                //4，写数据
                out.write(ch);
            }
             //5，关闭流
```

```
            in.close();
            out.close();
        }

        //方式 2： 采用基本的流，一个多个字节的方式赋值
        private static void method2(String src, String dest) throws IOException {
            //1，指定数据源
            FileInputStream in = new FileInputStream(src);
            //2，指定目的地
            FileOutputStream out = new FileOutputStream(dest);
            //3，读数据
            byte[] buffer = new byte[1024];
            int len = -1;
            while ( (len=in.read(buffer)) != -1) {
                //4，写数据
                out.write(buffer, 0, len);
            }
            //5，关闭流
            in.close();
            out.close();
        }

        //方式 1： 采用基本的流，一次一个字节的方式复制
        private static void method1(String src, String dest) throws IOException {
            //1，指定数据源
            FileInputStream in = new FileInputStream(src);
            //2，指定目的地
            FileOutputStream out = new FileOutputStream(dest);
            //3，读数据
            int ch = -1;
            while (( ch=in.read()) != -1) {
                //4，写数据
                out.write(ch);
            }
            //5，关闭流
            in.close();
            out.close();
        }
    }
```

2.2 字符缓冲流

- 字符缓冲输入流 BufferedReader
- 字符缓冲输出流 BufferedWriter

完成文本数据的高效的写入与读取的操作

2.2.1 字符缓冲输出流 BufferedWriter

将文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入。

- 方法：

void newLine() 根据当前的系统，写入一个换行符

```
/*
 * BufferedWriter 字符缓冲输出流
 * 方法
 *  public void newLine()写入一个行分隔符
 *
 * 需求： 通过缓冲输出流写入数据到文件
 * 分析：
 *  1，创建流对象
 *  2，写数据
 *  3，关闭流
 *
 */
public class BufferedWriterDemo {
    public static void main(String[] args) throws IOException {
        //创建流
        //基本字符输出流
        FileWriter fileOut = new FileWriter("file.txt");
        //把基本的流进行包装
        BufferedWriter out = new BufferedWriter(fileOut);
        //2，写数据
        for (int i=0; i<5; i++) {
            out.write("hello");
            out.newLine();
        }
        //3,关闭流
        out.close();
    }
}
```

2.2.2 字符缓冲输入流 BufferedReader

从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取。

- 方法

public String readLine() 读取一个文本行，包含该行内容的字符串，不包含任何行终止符，如果已到达流末尾，则返回 null

```
/*
 * BufferedReader 字符缓冲输入流
 *
 * 方法：
 *  String readLine()
 * 需求：从文件中读取数据，并显示数据
 */
public class BufferedReaderDemo {
    public static void main(String[] args) throws IOException {
        //1,创建流
        BufferedReader in = new BufferedReader(new FileReader("file.txt"));
        //2，读数据
        //一次一个字符
        //一次一个字符数组
        //一次读取文本中一行的字符串内容
        String line = null;
        while( (line = in.readLine()) != null ){
            System.out.println(line);
        }

        //3,关闭流
        in.close();
    }
}
```

2.2.3 使用字符缓冲流完成文本文件的复制

刚刚我们学习完了缓冲流，现在我们就使用字符缓冲流的特有功能，完成文本文件的复制

```
/*
 * 采用高效的字符缓冲流，完成文本文件的赋值
 *
 * 数据源： file.txt
 * 目的地： copyFile.txt
 *
 * 分析：
 *  1，指定数据源， 是数据源中读数据，采用输入流
 *  2,指定目的地，是把数据写入目的地，采用输出流
 *  3,读数据
 *  4，写数据
 *  5，关闭流
 */
public class CopyTextFile {
    public static void main(String[] args) throws IOException {
```

```
            //1，指定数据源，  是数据源中读数据，采用输入流
            BufferedReader in = new BufferedReader(new FileReader("file.txt"));
            //2,指定目的地，是把数据写入目的地，采用输出流
            BufferedWriter out = new BufferedWriter(new FileWriter("copyFile.txt"));
            //3,读数据
            String line = null;
            while ( (line = in.readLine()) != null ) {
                //4，写数据
                out.write(line);
                //写入换行符号
                out.newLine();
            }
            //5，关闭流
            out.close();
            in.close();
        }
    }
```

第3章 流的操作规律

IO 流中对象很多，解决问题(处理设备上的数据时)到底该用哪个对象呢？
把 IO 流进行了规律的总结(四个明确)：

- 明确一：要操作的数据是数据源还是数据目的。
  源：InputStream　　Reader
  目的：OutputStream Writer
  先根据需求明确要读，还是要写。

- 明确二：要操作的数据是字节还是文本呢？
  源：
    字节：InputStream
    文本：Reader
  目的：
    字节：OutputStream
    文本：Writer
    已经明确到了具体的体系上。

- 明确三：明确数据所在的具体设备。
  源设备：
    硬盘：文件　File 开头。
    内存：数组，字符串。
    键盘：System.in;
    网络：Socket

目的设备：

硬盘：文件　File 开头。

内存：数组，字符串。

屏幕：System.out

网络：Socket

完全可以明确具体要使用哪个流对象。

- 明确四：是否需要额外功能呢？

  额外功能：

  转换吗？转换流。InputStreamReader OutputStreamWriter

  高效吗？缓冲区对象。BufferedXXX

InputStream

FileInputStream

BufferedInputStream

OuputStream

FileOutputStream

BufferedOuputStream

Writer

OutputStreamWriter

FileWriter

BufferedWriter

Reader

InputStreamReader

FileReader

BufferedReader

第4章 总结

4.1知识点总结

- 字节流
  - 字节输入流 InputStream
    - FileInputStream 操作文件的字节输入流
    - BufferedInputStream 高效的字节输入流
  - 字节输出流 OutputStream
    - FileOutputStream 操作文件的字节输出流
    - BufferedOutputStream 高效的字节输出流

- 字符流
  - 字符输入流 Reader
    - FileReader 操作文件的字符输入流
    - BufferedReader 高效的字符输入流
    - InputStreamReader 输入操作的转换流(把字节流封装成字符流)
  - 字符输出流 Writer
    - FileWriter 操作文件的字符输出流
    - BufferedWriter 高效的字符输出流
    - OutputStreamWriter 输出操作的转换流(把字节流封装成字符流)

- 方法：
  - 读数据方法：
    - read() 一次读一个字节或字符的方法
    - read(byte[] char[]) 一次读一个数组数据的方法
    - readLine() 一次读一行字符串的方法(BufferedReader 类特有方法)
    - readObject() 从流中读取对象(ObjectInputStream 特有方法)
  - 写数据方法：
    - write(int) 一次写一个字节或字符到文件中
    - write(byte[] char[]) 一次写一个数组数据到文件中
    - write(String) 一次写一个字符串内容到文件中
    - writeObject(Object) 写对象到流中(ObjectOutputStream 类特有方法)
    - newLine() 写一个换行符号(BufferedWriter 类特有方法)

- 向文件中写入数据的过程
  1，创建输出流对象
  2，写数据到文件
  3，关闭输出流
- 从文件中读数据的过程
  1， 创建输入流对象
  2， 从文件中读数据
  3， 关闭输入流

- 文件复制的过程
  1， 创建输入流（数据源）
  2， 创建输出流（目的地）
  3， 从输入流中读数据
  4， 通过输出流，把数据写入目的地
  5， 关闭流