第8章 输入、输出流及文件处理

部分内容摘自

《Java面向对象编程》，孙卫琴

《Java 程序设计》，唐大仕

1 文件及目录

- 文件与目录管理

1.1 File类

- File类提供了若干处理文件、目录和获取它们基本信息的方法。
- java中目录也是文件（File）,java中没有Directory类。

- File 类的构造方法有三个：
    1. File(String pathname)
    2. File(String parent, String child)
    3. File(File parent, String child)

1.2文件类例子

```
//在Java中，将目录也当作文件处理
//
        File f;
        f = new File("Test.java");
        f = new File("E:\\ex\\","Test.java");

        File path = new File("E:\\ex\\");
        File f = new File(path, "Test.java");
```

1.3 File类方法介绍

- 关于文件/目录名操作

  String getName()
  String getPath()
  String getAbsolutePath()
  String getParent()
  boolean renameTo(File newName)

- File 测试操作

  boolean exists()
  boolean canWrite()
  boolean canRead()
  boolean isFile()
  boolean isDirectory()
  boolean isAbsolute();

- 获取常规文件信息操作

  long lastModified()
  long length()
  boolean delete()

- 目录操作

  boolean mkdir()
  String[] list()

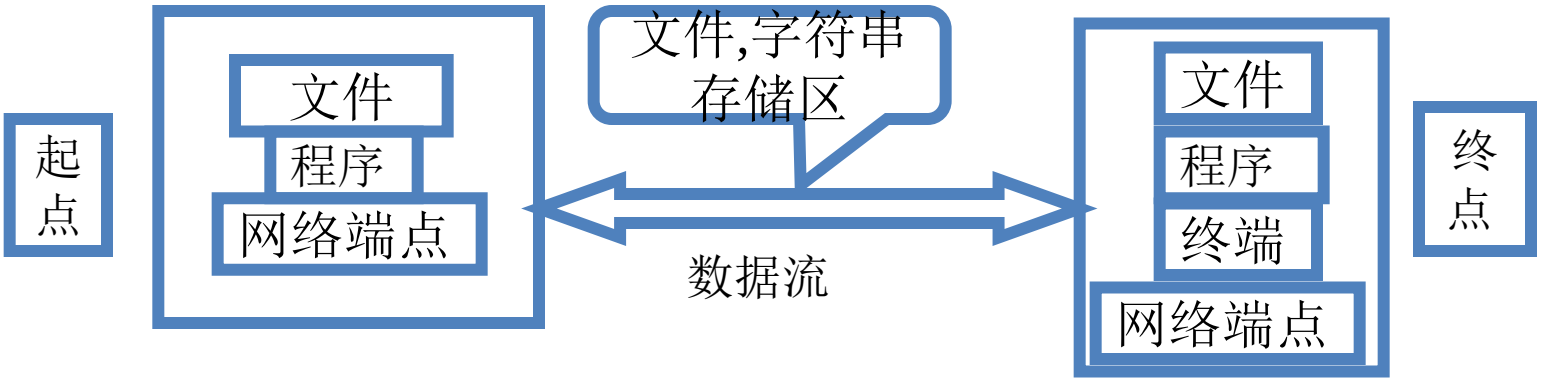
示例:列出所有文件

1.4 File类的作用（例子： ch08. UseFile ）

创建目录： **ch08. UseFile.main()**
递归列目录： **ch08. UseFile. listDir()**
递归删目录： **ch08. UseFile. deleteDir()**
递归列目录： **ch08.** ListAllFiles.java

2 输入流/输出流

- 大部分程序都需要输入/输出处理，比如从键盘读取数据、向屏幕中输出数据、从文件中读或者向文件中写数据、在一个网络连接上进行读写操作等。在Java中，把这些不同类型的输入、输出源抽象为流Stream。
- 按流的方向，可分为输入流与输出流。



2.1字节流与字符流

	字 节 流(byte)	字 符 流(char)
输　入	InputStream	Reader
输　出	OutputStream	Writer

2.2. InputStream类

- InputStream类最重要的方法是读数据的read()方法。read()方法功能是逐字节地以二进制的原始方式读取数据，它有三种形式：
- public int read()；
- public int read(byte b[])；
- public int read(byte[] b, int off, int len)；

2.3．OutputStream类

- OutputStream类的重要方法是write()，它的功能是将字节写入流中，write()方法有三种形式：

  public void write (int b)；// 将参数b的低位字节写入到输出流

  public void write (byte b[])；// 将字节数组b[]中的全部字节顺序写入到输出流

  public void write(byte[] b, int off, int len)；// 将字节数组b[]中从off开始的len个字节写入到流中
- Output的另外两个方法是flush()及close()。
- public void flush ()；
- public void close()；

2.4. Reader类

- Reader类与InputStream类相似，都是输入流，但差别在于Reader类读取的是字符（char），而不是字节。
- Reader的重要方法是read()，有三种形式：
- □ public int read()；
- □ public int read(char b[])；
- □ public int read(char[] b, int off, int len)；

2.5．Writer类

- Writer类与OutputStream类相似，都是输出流，但差别在于Writer类写入的是字符（char），而不是字节。Writer的方法有：

  public void write (int b)；// 将参数b的低两字节写入到输出流

  public void write (char b[])；// 将字符数组b[]中的全部字节顺序写入到输出流

  public void write(char[] b, int off, int len)；// 将字节数组b[]中从off开始的len个字节写入到流中

  public void write(String s)；// 将字符串写入流中

  public void write(String s, int off, int len)；// 将字符串写入流中, off为位置，len为长度

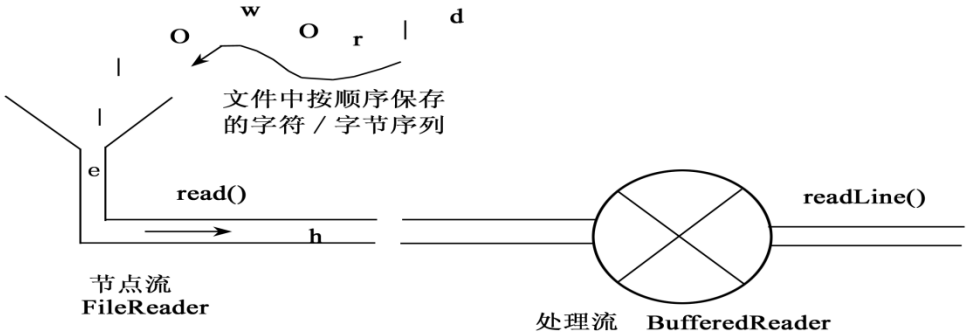
  public void flush ()；// 刷新流

  public void close()；// 关闭流

2.6 节点流和处理流

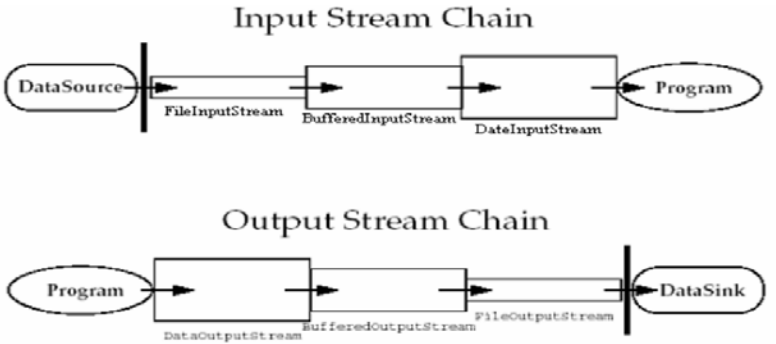
- 按照流是否直接与特定的地方（如磁盘、内存、设备等）相连，分为节点流与处理流两类。
- （1）节点流（Node Stream）
  - 可以从或向一个特定的地方（节点）读写数据。如文件流FileReader。
- （2）处理流（Processing Stream）
  - 是对一个已存在的流的连接和封装，通过所封装的流的功能调用实现数据读、写功能。处理流又称为过滤流，如缓冲处理流BufferedReader。

- 节点流与处理流的关系，如图所示。节点流直接与节点（如文件）相连，而处理流对节点流或其他处理流进一步进行处理（如缓冲、组装成对象，等等）。



- 处理流的构造方法总是要带一个其他的流对象作参数。如：
- BufferedReader in = new BufferedReader(new FileReader(file));
- BufferedReader in2 =
- new BufferedReader(
- new (InputstReamreader(
- new FileInputStream(file)));
-

一个流对象经过其他流的多次包装，称为流的链接



2.7 常用的节点流

节点类型	字节流	字符流
File 文件	FileInputStream FileOutputStream	FileReader FileWriter
Memory Array 内存数组	ByteArrayInputStream ByteArrayOutputStream	CharArrayReader CharArrayWriter
Memory String 字符串		StringReader StringWriter
Pipe 管道	PipedInputStream PipedOutputStream	PipedReader PipedWriter

2.8 常用的处理流

处理类型	字节流	字符流
Buffering 缓冲	BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
Filtering 过滤	FilterInputStream FilterOutputStream	FilterReader FilterWriter
Converting between bytes and character 字节流转为字符流		InputStreamReader OutputStreamWriter
Object Serialization 对象序列化	ObjectInputStream ObjectOutputStream	
Data conversion 基本数据类型转化	DataInputStream DataOutputStream	
Counting 行号处理	LineNumberInputStream	LineNumberReader
Peeking ahead 可回退流	PushbackInputStream	PushbackReader
Pinting 可显示处理	PrintStream	PrintWriter

2.9 输入输出流例子

- InputStream/OutputStream例子：

  ch08. ReadWriteFileByteStreamDemo

```
public static void copyIO(InputStream in, OutputStream out)
        throws IOException {
    byte[] buf = new byte[CHUNK_SIZE];
    /**
     * 从输入流读取内容并写入到另外一个流的典型方法
     */
    int len = in.read(buf);
    while (len != -1) {
        out.write(buf, 0, len);
        len = in.read(buf);
    }
}
```

2.10 输入输出流例子 Reader/Writer例子 ch08. ReadWriteFileCharDemo

```
public static String readFile(String fsrc) throws IOException {
    //这句话是否可以写成Reader reader ;为什么?
    Reader reader = null;
    try {
        reader = new FileReader(fsrc);
        StringBuffer buf = new StringBuffer();
        char[] chars = new char[CHUNK_SIZE];
        int readed = reader.read(chars);
        // 从一个流里面读取内容的经典写法
        while (readed != -1) {
            // 文件是size不能被CHUNK_SIZE整除，所以要记录每次读到的长度readed
            // 写入到buf的时候，不是用的 buf.append(chars);
            // 而是用buf.append(chars, 0, readed);
            buf.append(chars, 0, readed);
            readed = reader.read(chars);
        }
        return buf.toString();
    } finally {
        //reader!=null的判断是否可以取消，为什么?
        if (reader != null) {
            reader.close();
        }
    }
}
```

try{}finall{}中如果有impments Closable的对象要申请并关闭的时候，ReadWriteFileCharDemo中如下两种写法等效。

```
public static String readFile(String fsrc)
        throws IOException {
    Reader reader = null;
    try {
        reader = new FileReader(fsrc);
        StringBuffer buf = new StringBuffer();
        char[] chars = new char[CHUNK_SIZE];
        int readed = reader.read(chars);
        // 从一个流里面读取内容的经典写法
        while (readed != -1) {
            buf.append(chars, 0, readed);
            readed = reader.read(chars);
        }
        return buf.toString();
    } finally {
        if (reader != null) {
            reader.close();
        }
    }
}
```

```
public static String readFile2(String fsrc)
        throws IOException {
    try (Reader reader = new FileReader(fsrc);) {
        StringBuffer buf = new StringBuffer();
        char[] chars = new char[CHUNK_SIZE];
        int readed = reader.read(chars);
        // 从一个流里面读取内容的经典写法
        while (readed != -1) {
            buf.append(chars, 0, readed);
            readed = reader.read(chars);
        }
        return buf.toString();
    }
}
```

1 reader必须impments Closable。

2 必须在try()里面申请。

2.11 再谈节点流和处理流

```
public static void main(String[] args) {
    try {
        // 节点流 可以直接根据一个目标构造
        FileInputStream fin = new FileInputStream("D:/temp/1.txt");
        FileOutputStream fout = new FileOutputStream("D:/temp/1.txt");
        ByteArrayInputStream bin = new ByteArrayInputStream("hellow".getBytes());
        FileReader freader = new FileReader("D:/temp/1.txt");
        FileWriter fwriter = new FileWriter("D:/temp/1.txt");
        // 处理流必须依托已经存在的流构造，一般作用是对已有流做一些功能上的增强
        ObjectInputStream oin = new ObjectInputStream(fin);
        ObjectOutputStream oout = new ObjectOutputStream(fout);
        LineNumberReader lin=new LineNumberReader(freader);
        //这里的代码不规范，流打开了没有关闭
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

2.12 使用处理流的例子

- ch08.ReadFileByProcessingStream.java

```
public static List<String> readLines(String fsrc) throws IOException {
    try (Reader reader = new FileReader(fsrc);
            //LineNumberReader 必须在其他流基础上构建
            LineNumberReader lineReader = new LineNumberReader(reader);) {
        String line = "";
        List<String> lines = new ArrayList<String>();
        while (line != null) {
            lines.add(line);
            //每次读取一行
            line = lineReader.readLine();
        }
        return lines;
    }
}
```

2.13 System中的标准输入和标准输出

- System.out 提供向“标准输出”写出数据的功能

  System.out为 PrintStream类型.
- System.in 提供从“标准输入”读入数据的功能

  System.in 为InputStream类型.
- System.err提供向“标准错误输出”写出数据的功能

  System.err为 PrintStream类型.

向标准输出写出数据

System.out/System.err的println/print方法

- println方法可将方法参数输出并换行
- print方法将方法参数输出但不换行
- print和println方法针对多数数据类型进行了重写 (boolean, char, int, long, float, double以及char[], Object和 String).
- print(Object)和println(Object)方法中调用了参数的toString()方法，再将生成的字符串输出

从标准输入读取数据

- 为了使用方便，经常将System.in用各种处理流进行封装处理，如：

```
BufferedReader br = new BufferedReader( new
InputStreamReader(System.in));
br.readLine();
```

2.14 文件的随机访问RandomAccessFile类

- RandomAccessFile，可以实现对文件的随机读写操作
- RandomAccessFile(String name，String mode)；
- RandomAccessFile(File f，String mode)；
- **public void seek(1ong pos)；**
- readBealoon()，readChar()，readInt()，readLong()，readFloat()，readDouble()，readLine()，readUTF()等
- writeBealoon()，writeChar()，writeInt()，writeLong()，writeFloat()，writeDouble()，writeLine()，writeUTF()等

3 基于文本的应用的几个问题

3.1 命令行参数

- 在启动Java应用程序时可以一次性地向应用程序中传递0~多个参数----命令行参数
- 命令行参数使用格式：

  java ClassName lisa "bily" "Mr Brown"

- 命令行参数被系统以String数组的方式传递给应用程序中的main方法，由参数args接收

  public static void main(String[] args)

例子： ch08.TestCommandLine

命令行参数用法举例

```
1   public class Test9_1 {
2       public static void main(String[] args) {
3           for ( int i = 0; i < args.length; i++ ) {
4               System.out.println("args[" + i + "] = " + args[i]);
5           }
6       }
7   }
```

//运行程序Test9_1.java

```
java Test9_1  lisa  "bily"  "Mr Brown"
```

//输出结果：

```
args[0] = lisa
args[1] = bily
args[2] = Mr Brown
```

3.2 系统属性(System Properties)

- 在Java中，系统属性起到替代环境变量的作用(环境变量是平台相关的)
- 可使用System.getProperties()方法获得一个 Properties类的对象，其中包含了所有可用的系统属性信息
- 可使用System.getProperty(String name)方法获得特定系统属性的属性值
- 在命令行运行Java程序时可使用-D选项添加新的系统属性

例子：ch08.TestProperties

3.4 正则表达式

- 正则表达式是文本处理中常用的工具，它实际上是用来匹配字符串的一种模式。在Java中有一个正则表达式引擎(在java.util.regex包中)，可以用正则表达式来验证和处理文本字符。

正则表达式的基本元素

字 符	含 义	描 述
.	代表一个字符的通配符	能和回车符之外的任何字符相匹配
[]	字符集	能和括号内的任何一个字符相匹配。方括号内也可以表示一个范围，用“—”符号将起始和末尾字符区分开来，例如[0-9]
[^]	排斥性字符集	和集合之外的任意字符匹配
^	起始位置	定位到一行的起始处并向后匹配
$	结束位置	定位到一行的结尾处并向前匹配
()	组	按照子表达式进行分组
|	或	或关系的逻辑选择，通常和组结合使用
\	转义	匹配反斜线符号之后的字符，所以可以匹配一些特殊符号，例如$和|

符 号	含 义	描 述
*	零个或多个	匹配表达式首项字符的零个或多个副本
+	一个或多个	匹配表达式首项字符的一个或多个副本
?	零个或一个	匹配表达式首项字符的一个或零个副本
n	重复	匹配表达式首项字符的n个副本

- \d 表示数字，相当于[0-9]
- \D 表示非数字，相当于 [^0-9]
- \s 表示空白符，相当于 [\t\n\x0B\f\r]
- \S 表示非空白符，相当于[^\s]
- \w 表示单词字符，相当于 [a-zA-Z_0-9]
- \W 表示非单词字符，相当于 [^\w]

正则表达式例子：

- ch08.RegExpDemo