Java程序设计

第9章 流、文件及基于文本的应用

第9章 流、文件及基于文本的应用

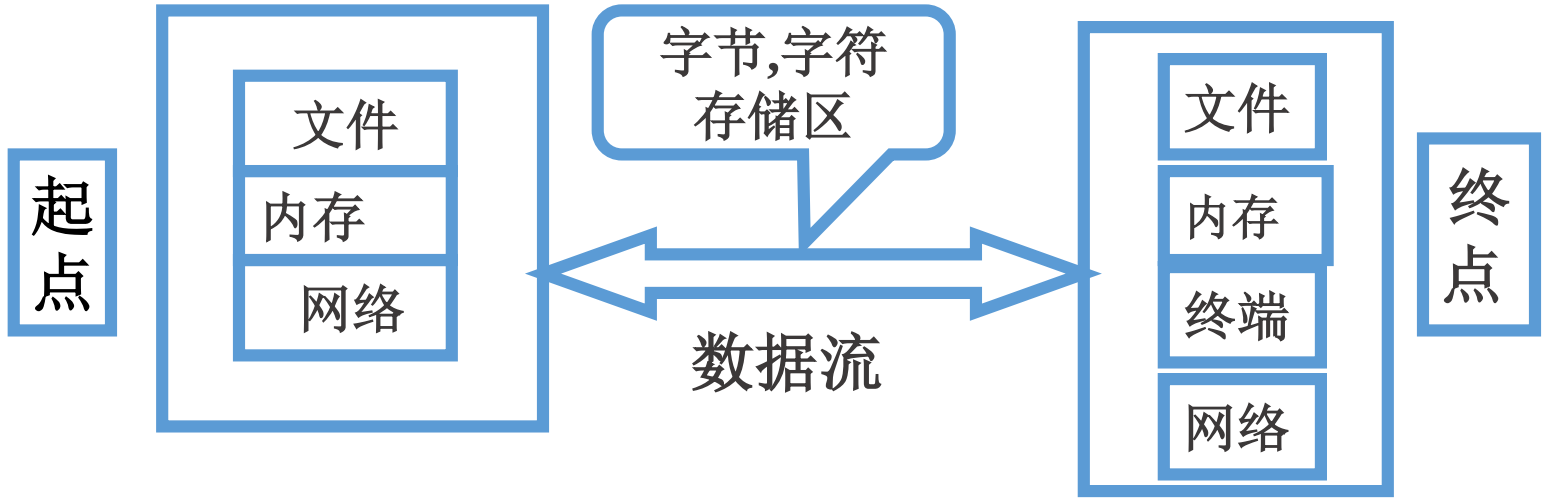
- 9.1 输入输出流
- 9.2 文件及目录
- 9.3 正则表达式

9.1 输入输出流

输入输出流

流

- 把不同类型的输入、输出都抽象为流（Stream）
  - 按流的方向，可分为输入流与输出流
- java.io包
  - 从JDK1.4起，加了 java.nio 包， JDK1.7 作了改进，称nio2



字节流与字符流

	字 节 流	字 符 流
输　入	InputStream	Reader
输　出	OutputStream	Writer

InputStream类

- InputStream类
- read()方法
  - 逐字节地以二进制的原始方式读取数据
    - public int read() ;　　读入一个字节，-1表示无
    - public int read(byte b[]) ;　　返回读入的字节数
    - public int read(byte[] b, int off, int len) ;

OutputStream类

- OutputStream类
- write()方法
  - 它的功能是将字节写入流中
    - 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 ()；　刷新缓存，实际写入到文件、网络
    - public void close()；　关闭流

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()
  - public int read()； //需要将int转成char
  - public int read(char b[])；
  - public int read(char[] b, int off, int len)；

Writer类

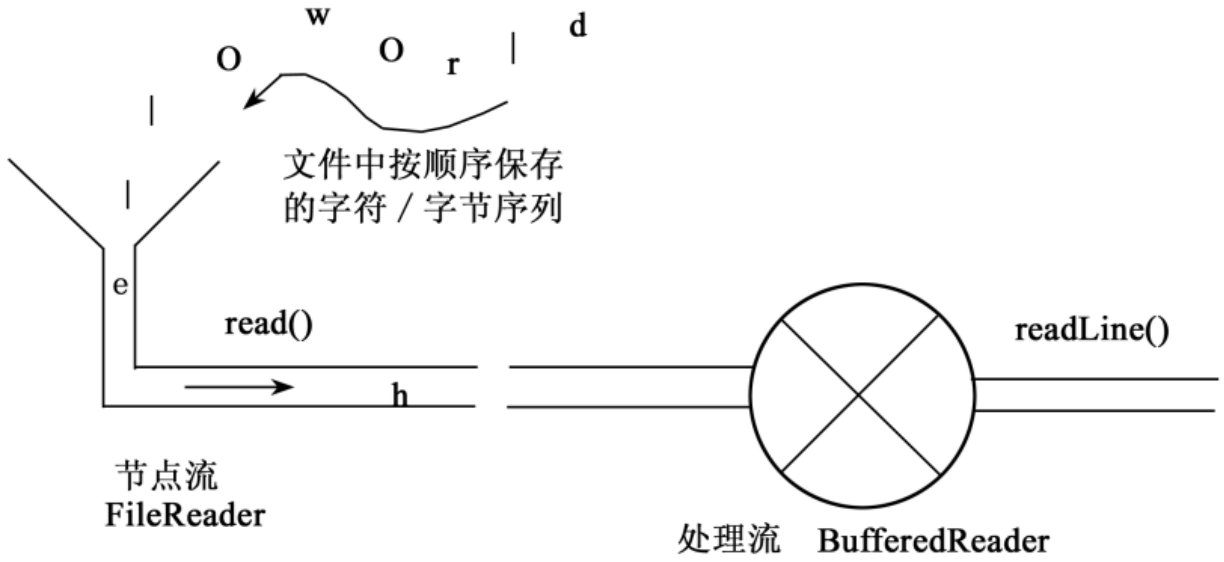
- Writer类
  - 与OutputStream类相似，都是输出流
  - 但差别在于Writer类写入的是字符（char），而不是字节。
- Writer的方法有：
  - public void write (int b)；// 将参数b的低两字节写入到输出流
  - public void write (char b[])；// 将字符数组b[]中的全部字节顺序写入到输出流
  - public void write(char[] b, int off, int len)；// 将字节数组b[]中从off开始的len个字节写入到流中
  - public void write(String 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

节点流和处理流

- 流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 FileInputStream，内存流 ByteArrayInputStream
- （2）处理流（Processing Stream）
  - 是对一个已存在的流的连接和封装，处理流又称为过滤流（Filter)
  - 如缓冲处理流 BufferedReader

节点流与处理流

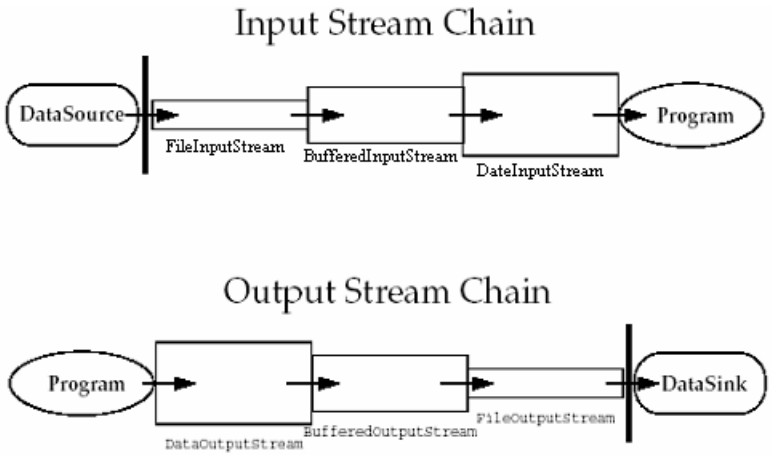
- 节点流（Node Stream）
  - 直接与节点（如文件）相连
- 处理流（Processing Stream）
  - 对节点流或其他处理流进一步进行处理
  - （如缓冲、组装成对象，等等）



流的包装（链接）

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

- BufferedReader in2 =
- new BufferedReader(
- new InputStreamReader(
- new FileInputStream(file), “utf-8”));
- s = in2.readLine();



- 一个流对象经过其他流的多次包装，称为流的链接

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

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 与字符编码有关
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

标准输入和标准输出

- System.in
- 　　System.in 为InputStream类型.
- System.out
- 　　System.out为 PrintStream类型.
- System.err
- 　　System.err为 PrintStream类型.

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();
- JDK1.5以后增加了java.util.Scanner类

常见内容的读写

- 常见的内容
  - 二进制
  - 文本
  - 对象
- 二进制流的读写
  - 示例：Dump.java

字符的读写

- 字符的读写
  - 常见的编码
    - UTF-8, ASCII，GB2312， 默认编码
  - 示例： CopyFileAddLineNumber.java
- 使用java.nio.file.Files的readAllLines()方法
  - 请参考JDK的源码
  - 示例： ReadAllLines.java

对象的读写

- 对象的读写
  - ObjectInputStream， ObjectOutputStream，
- 基本数据的读写
  - DataInputStream, DataOutputStream

- 序列化（serialize）与反序列化（ deserialize）
  - 要求对象实现 Serializable 接口
    - （该接口没有方法，只是一个标记）
  - 示例 SerializeDemo.java

网络流

- 网络流
  - URLGetContent.java
- 要点：
  - URL
  - 网络流
  - 线程、invokeLater

应用示例

- 背单词
- Recite.java
- 要点：
  - 文本文件读取
  - 字符串处理
  - 集合
  - 线程、Timer

补充两个示例

- 读取进程的输出
  - ProcessAndStream.java
- 读取Jar文件中的资源
  - ResourseFromJar.java

9.2 文件及目录

文件及目录

文件类

- java.io包中定义与数据输入、输出功能有关的类，包括提供文件操作功能的File类
- 创建File类对象

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

- 在Java中，将目录（directory, 文件夹)也当作文件处理
- File类中提供了实现目录管理功能的方法

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

File类方法

- 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()

RandomAccessFile类

- 类似于C语言的文件操作
- RandomAccessFile，可以实现对文件的随机读写操作
- 构造方法
  - RandomAccessFile(String name，String mode)；
  - RandomAccessFile(File f，String mode)；
- 定位方法
  - public void seek(long pos)；
- 读写方法
  - readBealoon()，readChar()，readInt()，readLong()，readFloat()，readDouble()，readLine()，readUTF()等
  - writeBealoon()，writeChar()，writeInt()，writeLong()，writeFloat()，writeDouble()，writeLine()，writeUTF()等

示例

- 示例:列出所有文件 ListAllFiles.java
  - String[] list() 是关键
  - 使用递归

9.3 正则表达式

正则表达式

正则表达式

- 正则表达式（Regular Expressions）
- 它实际上是用来匹配字符串的一种模式。
- 是文本处理中常用的工具
- 主要的应用包括：匹配验证、分割、查找、替换
-

正则表达式

- 正则表达式的写法
  - 字符{数量}位置
  - 如　[0-9]{2,4}\b　可以匹配 123　1988　2015　16

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
\b	单词边界	
\B	非单词边界	
()	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|

关于数量

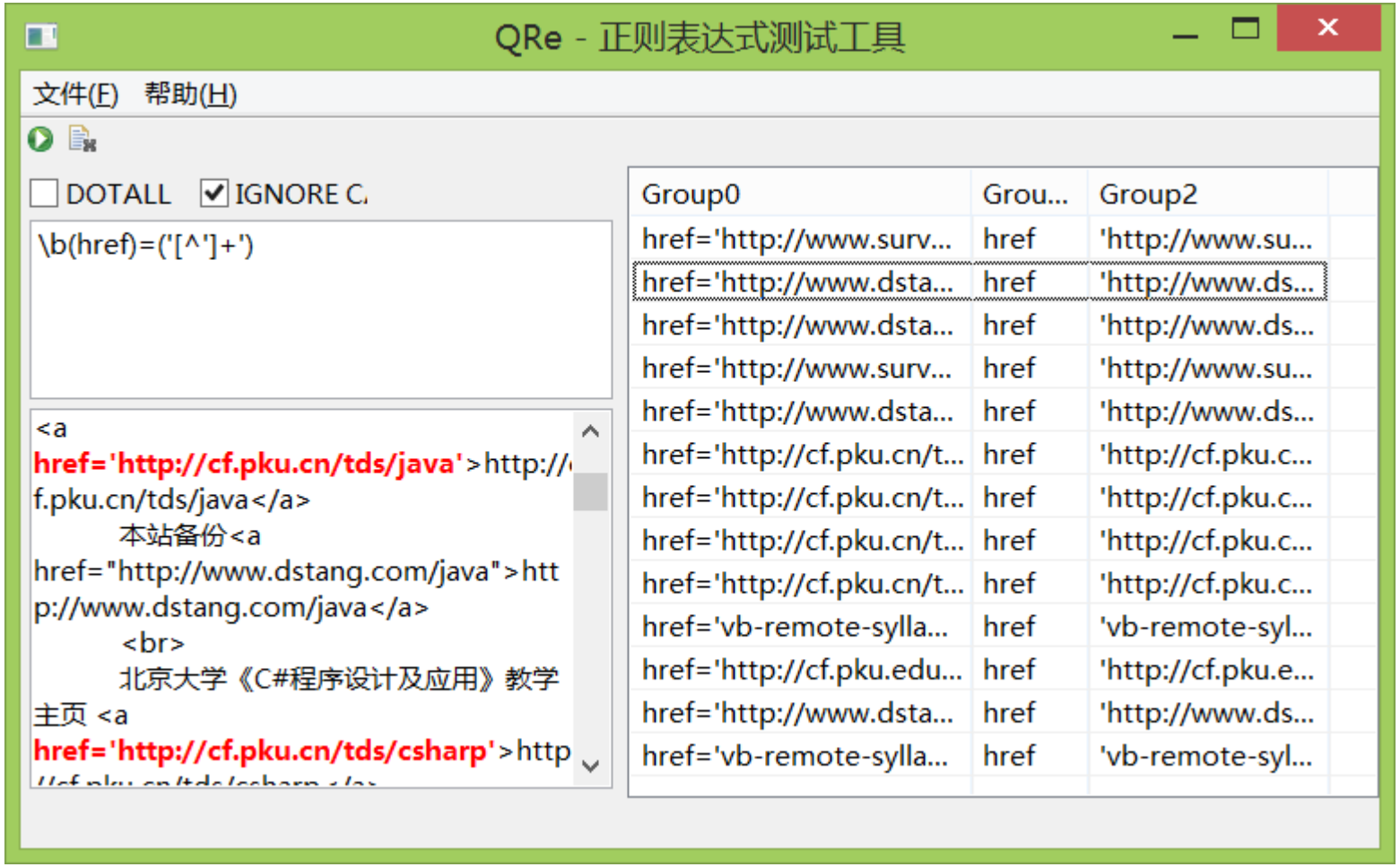
符 号	含 义	描 述
*	零个或多个	匹配表达式首项字符的零个或多个副本
+	一个或多个	匹配表达式首项字符的一个或多个副本
?	零个或一个	匹配表达式首项字符的一个或零个副本
n	重复	匹配表达式首项字符的n个副本

关于字符

- \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]

使用一些工具

- \b(href)=('[^']+')
- 匹配网址



Pattern类

- java.util.regex包
  - 主要的类 Pattern类， Matcher类
- 应用之一：分割 RegexSplitter.java
  - 如： 对以逗号和/或空格分隔的输入字符串进行切分

```
Pattern p = Pattern.compile( "[, \\s]+");
String[] result =
        p.split( "one,two, three   four ,  five ");
for (int i=0; i<result.length; i++)
        System.out.println(result[i]);
```

匹配

- 应用之二：匹配验证 RegexEmailValidate.java
- 判断一个email地址是否合法
  - String pattern = "^[^@]+@[\\w]+(\\.[\\w]+)*$";
  - String email = "dstang2000@263.net";
  - boolean ok = Pattern.matches(pattern, email);

  - 例中的模式要求email地址在@的前面有多个非@的字符，在@之后，需要一些由点（.）隔开的一些单词字符（\w）。要注意\在java的源程序的字符串中要写成\\

Matcher类

- 应用之三：查找替换 RegexReplacement.java
- Matcher类
  - 通过调用某个模式（Pattern对象）的matcher方法可以得到Matcher对象
- Matcher类的方法
  - find方法将扫描输入序列，寻找下一个与模式匹配的地方。
  - appendReplacement方法
- Matcher中的group
  - 所谓group（分组），是指正则表达式中一对圆括号括起来的一部分
  - group(0)或group()表示整个匹配项，group(1)、group(2)…表示各个分组
  - 替换时，$0表示整个匹配项，$1、$2…表示各个分组（圆括号）

应用举例

- 可参考
  - http://lzjold3.blog.163.com/blog/static/106138120112259583295 8/
- 示例: 从网页内容中找到链接的网址
  - RegexHref.java

应用示例

- 简单的网络爬虫
- URLCrawler.java
- 要点：
  - 流
  - 正则表达式
  - 集合
  - 线程