

Java语言程序设计

配套教材由清华大学出版社出版发行

第7章 输入输出流



中國農業大學

阚道宏

第7章 输入输出流

- Java语言将程序中数据的**输入输出**过程看作是一种**数据流动**的过程
 - 将提供输入数据的数据源称作**输入流**（input stream）
 - 将输出数据时的目的地称作**输出流**（output stream）
 - **键盘**就是一个输入流，而**显示器**则是一个输出流
 - **Java API**为数据的输入输出提供了一组**输入输出流类**
- 学习数据输入输出的基本原理
- 运用Java API提供的输入输出流类
 - **标准**输入输出和**文件**输入输出
 - 文本文件、图像文件和声音文件的基本处理方法



第7章 输入输出流

- 本章内容
 - [7.1 Java输入输出流](#)
 - [7.2 标准IO](#)
 - [7.3 文件及文件IO](#)
 - [7.4 序列化及二进制文件IO](#)
 - [7.5 文本处理](#)
 - [7.6 图像处理](#)
 - [7.7 声音处理](#)



7.1 Java输入输出流

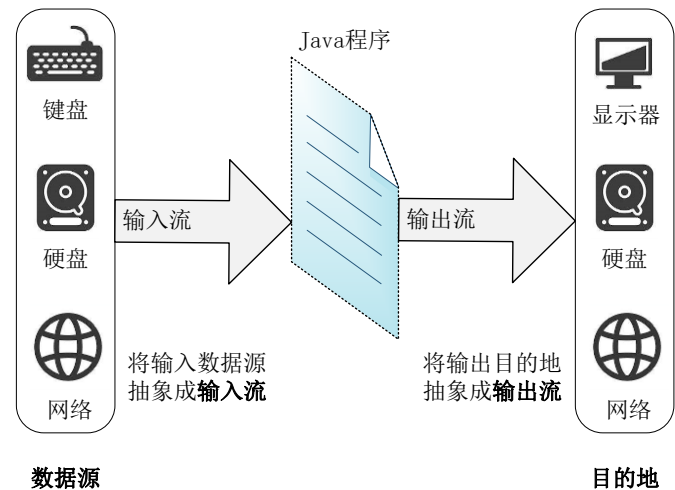
- Java程序的输入输出
 - 输入流与输出流

- 标准IO与文件IO
- 网络通信

– 字节流与字符流

- “a”：97、61、01100001
- “abc中国”：61 62 63 d6 d0 b9 fa，ANSI编码
- 字节流（byte stream）：byte
- 字符流（character stream）：char

- 字节型输入输出流：byte，字节型输入流、字节型输出流
- 字符型输入输出流：char，字符型输入流、字符型输出流



7.1 Java输入输出流

- Java API提供的输入输出流类
 - java.io包
 - 字节型输入输出流
 - 字节型输入流类族: **byte**, 根类InputStream
 - 字节型输出流类族: **byte**, 根类OutputStream
 - 字符型输入输出流
 - 字符型输入流类族: **char**、**String**, 根类Reader
 - 字符型输出流类族: **char**、**String**, 根类Writer



7.1 Java输入输出流

- Java API提供的输入输出流类
 - 输入输出流类**说明文档**
 - 输入数据也被称作“**读**”（read）数据
 - 输出数据也被称作“**写**”（write）数据
 - **对比**字节型与字符型、输入流与输出流的不同之处



7.1 Java输入输出流

java.io.InputStream类说明文档（字节型输入流类族的根类）

public abstract class **InputStream**

extends **Object**

implements **Closeable**

	修饰符	类成员（节选）	功能说明
1		InputStream()	构造方法
		int available()	返回字节输入流中可读的字节数（估计值）
2	abstract	int read()	从字节输入流中读出一个字节
3		int read(byte[] b)	根据数组b的长度读出若干字节并存放到b中
4		int read(byte[] b, int off, int len)	读出len个字节并存放到数组b中
5		long skip(long n)	跳过n个字节
6		void close()	关闭字节输入流
.....			

java.io.OutputStream类说明文档（字节型输出流类族的根类）

public abstract class **OutputStream**

extends **Object**

implements **Closeable, Flushable**

	修饰符	类成员（全部）	功能说明
1		OutputStream()	构造方法
2	abstract	void write(int b)	向字节输出流中写入一个字节
3		void write(byte[] b)	向字节输出流中写入数组b
4		void write(byte[] b, int off, int len)	写入数组b中的len个字节
5		void flush()	立即输出缓存里的内容
6		void close()	关闭字节输出流
.....			



7.1 Java输入输出流

java.io. Reader 类说明文档（字符型输入流类族的根类）			
public abstract class Reader			
extends Object			
implements Readable, Closeable			
	修饰符	类成员（节选）	功能说明
1	protected	Reader()	构造方法
2		boolean ready()	检查字符输入流是否有可读的字符
3		int read()	从字符输入流读出一个字符
4		int read(char[] cbuf)	根据字符数组cbuf的长度读出若干个字符并存放到cbuf中
5	abstract	int read(char[] cbuf, int off, int len)	读出len个字符并存放到数组cbuf中
6		long skip(long n)	跳过n个字符
7	abstract	void close()	关闭字符输入流
.....			

java.io. Writer 类说明文档（字符型输出流类族的根类）			
public abstract class Writer			
extends Object			
implements Appendable, Closeable, Flushable			
	修饰符	类成员（节选）	功能说明
1	protected	Writer()	构造方法
2		void write(int c)	向字符输出流写入一个字符
3		void write(char[] cbuf)	向字符输出流写入字符数组cbuf
4	abstract	void write(char[] cbuf, int off, int len)	写入字符数组cbuf中的len个字符
5		void write(String str)	写入字符串str
6		void write(String str, int off, int len)	写入字符串str中的len个字符
7	abstract	void flush()	立即输出缓存里的内容
8	abstract	void close()	关闭字符输出流
.....			

7.1 Java输入输出流

- Java API提供的输入输出流类
 - 字节型输入流举例

例7-1 一个使用System.in对象进行键盘输入的Java演示程序（JSystemInTest.java）

```
1 import java.io.*;           // 导入java.io包中的类
2 public class JSystemInTest { // 测试类
3     public static void main(String[] args) { // 主方法
4         byte bbuf[] = new byte[20]; // 定义字节数组保存键盘输入的字节流
5         // 按字节流方式读取键盘输入的数据
6         try { // 必须处理输入过程中可能抛出的勾选异常IOException
7             int len = System.in.read(bbuf); // 此处可能会抛出IOException异常
8             for (int n = 0; n < len; n++) {
9                 int x = Byte.toUnsignedInt( bbuf[n] ); // 将字节流先转成无符号整数
10                String hexString = Integer.toHexString(x); // 再转成十六进制字符串
11                System.out.print(hexString + " "); // 依次显示输入的字节流
12            }
13            System.out.println();
14        }
15        catch(IOException e) // 处理勾选异常IOException
16        { System.out.println( e.getMessage() ); }
17    } }
```

Problems @ Javadoc Declaration Console

<terminated> JByteCharTest [Java Application] C:\Java

abc中国

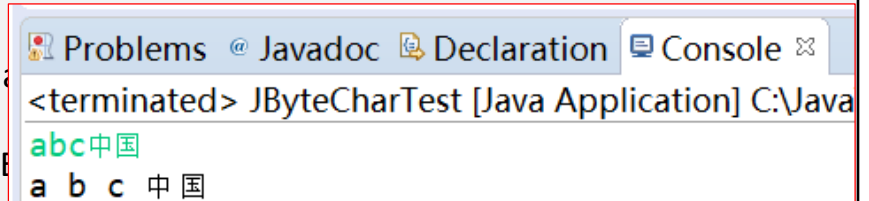
61 62 63 d6 d0 b9 fa d a

7.1 Java输入输出流

- 将字节流包装成字符流
 - 两个将字节型**包装**成字符型的类
 - 输入流包装类**InputStreamReader**

例7-2 一个使用输入流包装类InputStreamReader的Java演示程序（InputStreamReaderTest.java）

```
1 import java.io.*;           // 导入java.io包中的类
2 public class InputStreamReaderTest { // 测试类
3     public static void main(String[] args) { // 主方法
4         char cbuf[] = new char[20]; // 定义字符数组保存键盘输入的字符流
5         try { // 必须处理勾选异常IOException
6             InputStreamReader inChar = new InputStreamReader(System.in); // 包装
7             int len = inChar.read( cbuf ); // 此处可能会抛出IOException异常
8             for (int n = 0; n < len; n++) { // 显示所输入的字符数据
9                 System.out.print(cbuf[n] + " "); // 字符之间用空格隔开
10            }
11            System.out.println();
12            inChar.close(); // 关闭输出流对象inChar
13        }
14        catch(IOException e) // 处理勾选异常IOException
15        { System.out.println( e.getMessage() ); }
16    } }
```



Problems @ Javadoc Declaration Console

<terminated> JByteCharTest [Java Application] C:\Java

abc中国

a b c 中国



中國農業大學

閻道宏

7.1 Java输入输出流

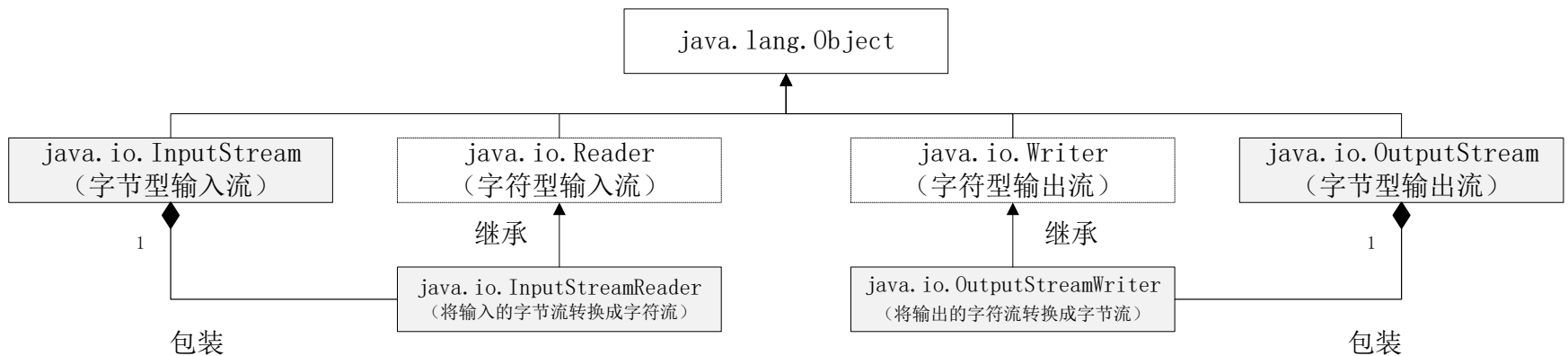
java.io. InputStreamReader 类说明文档（输入流包装类）			
public class InputStreamReader extends Reader			
	修饰符	类成员（节选）	功能说明
1		InputStreamReader (InputStream in)	构造方法（默认为系统编码）
2		InputStreamReader (InputStream in, String charsetName)	构造方法（指定字符编码）
3		String getEncoding ()	获取字符编码类型
4		boolean ready ()	检查字符输入流是否准备好
5		int read ()	从字符输入流读出一个字符
6		int read (char[] cbuf, int off, int len)	读出len个字符并存放于数组cbuf中
7		void close ()	关闭字符输入流
.....			

java.io. OutputStreamWriter 类说明文档（输出流包装类）			
public class OutputStreamWriter extends Writer			
	修饰符	类成员（节选）	功能说明
1		OutputStreamWriter (OutputStream out)	构造方法（默认为系统编码）
2		OutputStreamWriter (OutputStream out, String charsetName)	构造方法（指定字符编码）
3		String getEncoding ()	获取字符编码类型
4		void write (int c)	向字符输出流写入一个字符
5		void write (char[] cbuf, int off, int len)	写入字符数组cbuf中的len个字符
6		void write (String str, int off, int len)	写入字符串str中的len个字符
7		void flush ()	立即输出缓存里的内容
8		void close ()	关闭字符输出流
.....			



7.1 Java输入输出流

- 将字节流包装成字符流
 - 输入输出流**根类**和**包装类**之间的关系



7.2 标准IO

- 标准IO： **键盘**输入， **显示器**输出
 - 键盘输入
 - 数据之间以**空格**或**Tab**键隔开
 - 对键盘输入的数据进行扫描、分割，然后**转换**成指定的数据类型，例如转换成**int**型或**double**型，最后再将转换后的数据**赋值**给变量，这种输入方式被称作是**格式化输入**
 - 扫描器类**Scanner**
 - 显示器输出
 - 将程序中不同类型的数据统一**转换**成字符串，转换时还需要指定显示格式，例如显示宽度或保留几位小数等，然后再**输出**到显示器上，这种输出方式被称作是**格式化输出**
 - 打印流类**PrintStream**



7.2 标准IO

- 格式化输入

例7-3 一个使用扫描器类Scanner进行格式化输入的Java演示程序（JScannerTest.java）

```
1 import java.io.*;           // 导入java.io包中的类
2 import java.util.Scanner;    // 导入java.util包中定义的扫描器类Scanner
3
4 public class JScannerTest {   // 测试类
5     public static void main(String[] args) { // 主方法
6         Scanner sc = null;     // 定义一个扫描器引用变量
7         System.out.print( "从键盘输入数据: " );
8         try { // 输入时可能会抛出异常，使用扫描器类建议按此代码框架编写程序
9             sc = new Scanner( System.in ); // 将键盘对象System.in包装成扫描器对象
10            String str = sc.next();    // 读取下一个字符串
11            int x = sc.nextInt();      // 读取下一个int型整数
12            System.out.println( "输入结果为: " +str +", " +x ); // 显示输入结果
13        }
14        finally
15        { if (sc != null) sc.close(); } // 关闭扫描
16    } }
```

从键盘输入数据: **abcd 123**<回车键>
输入结果为: abcd, 123



7.2 标准IO

- 格式化输入

java.util.Scanner类说明文档			
public final class Scanner			
extends Object			
implements Iterator<String>, Closeable			
	修饰符	类成员（节选）	功能说明
1		Scanner (InputStream source)	构造方法（从字节输入流输入）
2		Scanner (InputStream source, String charsetName)	构造方法（指定字符编码）
3		Scanner (Path source)	构造方法（从文件输入）
4		Scanner (Path source, String charsetName)	构造方法（指定字符编码）
5		Scanner (File source)	构造方法（从文件输入）
6		Scanner (String source)	构造方法（从字符串输入）
7		boolean hasNext ()	检查扫描器中是否还有输入项
8		String next ()	读出下一个字符串
9		int nextInt ()	读出下一个int型整数
10		double nextDouble ()	读出下一个double型浮点数
11		Scanner useDelimiter (String pattern)	设置数据项的分隔符
12		void close ()	关闭扫描器
.....			



7.2 标准IO

- 格式化输出

java.io. PrintStream 类说明文档			
public class PrintStream			
extends FilterOutputStream			
implements Appendable, Closeable			
	修饰符	类成员（节选）	功能说明
1		PrintStream (OutputStream out)	构造方法（向字节输出流输出）
2		PrintStream (String fileName)	构造方法（向文件输出）
3		PrintStream (String fileName, String csn)	构造方法（指定字符编码）
4		PrintStream (File file)	构造方法（向文件输出）
5		void println (String s)	格式化输出一个字符串
6		void print (String s)	格式化输出（不换行）
7		void println (int x)	格式化输出一个int型整数
8		void print (int x)	格式化输出（不换行）
9		void println (double x)	格式化输出一个double型实数
10		void print (double x)	格式化输出（不换行）
11		void println (Object x)	格式化输出一个对象
12		void print (Object obj)	格式化输出（不换行）
13		void println ()	单独换一行
14		PrintStream format (String format, Object... args)	按照格式化字符串输出多个对象
15		PrintStream printf (String format, Object... args)	同 format （类似于C语言的 printf ）
16		void write (byte[] buf, int off, int len)	输出字节数组
17		void flush ()	立即输出缓存里的内容
18		void close ()	关闭打印流
.....			



7.2 标准IO

- 格式化输出
 - 打印流类 **PrintStream**

例7-4 一个使用System.out对象进行格式化输出的Java演示程序（JPrintStreamTest.java）

```
1 import java.io.*;           // 导入java.io包中的类
2 public class JPrintStreamTest { // 测试类
3     public static void main(String[] args) { // 主方法
4         int x = 10;           // 先定义几个演示数据
5         double y = 15.8;
6         String str = "abcd";
7         // 下面演示格式化输出方法
8         System.out.println(x); // 输出一个int型整数
9         System.out.println(y); // 输出一个double型实数
10        System.out.println(str); // 输出一个String字符串
11        // 下面演示同时格式化输出多个数据的方法
12        System.out.println("x=" + x + ", y=" + y + "str=" + str); // 输出一个字符串表达式
13        System.out.format("x=%d, y=%f, str=%s\n", x, y, str); // 格式化输出多个数据项
14        System.out.format("x=%h\n", x); // 显示十六进制整数
15        System.out.printf("x=%d, y=%f, str=%s\n", x, y, str); // 类似于C语言的printf函数
16    } }
17
```

Problems @ Javadoc Declaration Console

<terminated> JPrintStreamTest [Java Application] C:\Ja

```
10
15.8
abcd
x=10, y=15.8, str=abcd
x=10, y=15.800000, str=abcd
x=a
x=10, y=15.800000, str=abcd
```



中國農業大學

閻道宏

7.3 文件及文件IO

- 使用Windows“记事本”程序输入气温数据并保存到一个文本文件“temperature.txt”

日期 气温

1 30.2

2 28.7

.....

30 25.9

- 直接从硬盘文件中读取原始数据，这被称为是文件输入
- 将处理结果写入硬盘文件，这被称为是文件输出
- 文件的输入输出被简称为文件IO



中國農業大學

閻道宏

7.3 文件及文件IO

- 文件的基本概念
 - 程序文件与数据文件
 - “记事本”程序“notepad.exe”的数据文件（扩展名为.txt）
 - Word文字处理程序“WINWORD.exe”的数据文件（扩展名为.doc或.docx）
 - 文件IO指的是数据文件的输入输出

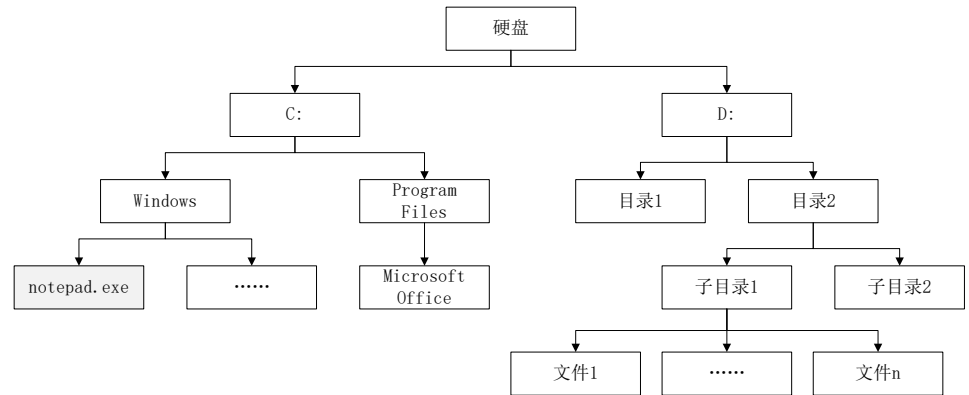


7.3 文件及文件IO

- 文件的基本概念

- 文件名 (file name)

- 目录 (directory)
 - 子目录 (subdirectory)
 - 路径 (path)
 - 文件夹 (folder)



盘符:\目录名\子目录名\.....\文件名.扩展名

C:\Windows\notepad.exe

"C:/Windows/notepad.exe" 或

"C:\\Windows\\notepad.exe"



中國農業大學

閻道宏

7.3 文件及文件IO

- 文件的基本概念

- 文件格式

- 文本文件（text file）
 - 二进制文件（binary file）

- 文本文件

- 存储字符类型的数据，主要是可见字符
 - 存储字符**编码**：ASCII编码、GBK编码
 - 具有**换行**格式：控制字符CR（13）和LF（10）
 - **通用性**强：纯文本内容、标准编码，无任何其他附加信息，使用类似于“记事本”这样的常规软件就能阅读、修改
 - 可用于**数据交换**：“程序-人”、“程序-程序”



7.3 文件及文件IO

- 文件的基本概念

- 二进制文件

- 直接以**内存**的二进制存储格式在**外存**上存储数据
 - 二进制文件中数据的**存储格式**与该数据在内存中的存储格式是一致的
 - 以二进制存储数据会涉及占用字节数、整数格式或浮点格式等存储格式。
 - 二进制文件不经过任何格式转换，直接**复制**变量的内存数据
 - 可保存任意**类型**的数据
 - 存储**效率**高
 - short型整数-2100（2个字节）
 - 转成字符串“-2100”（5个字节）
 - **通用性**差：由哪个程序创建，就由哪个程序负责阅读、修改
 - 交换数据需遵循相同的**格式标准**：例如JPEG、BMP、GIF和TIFF等图象文件格式标准



7.3 文件及文件IO

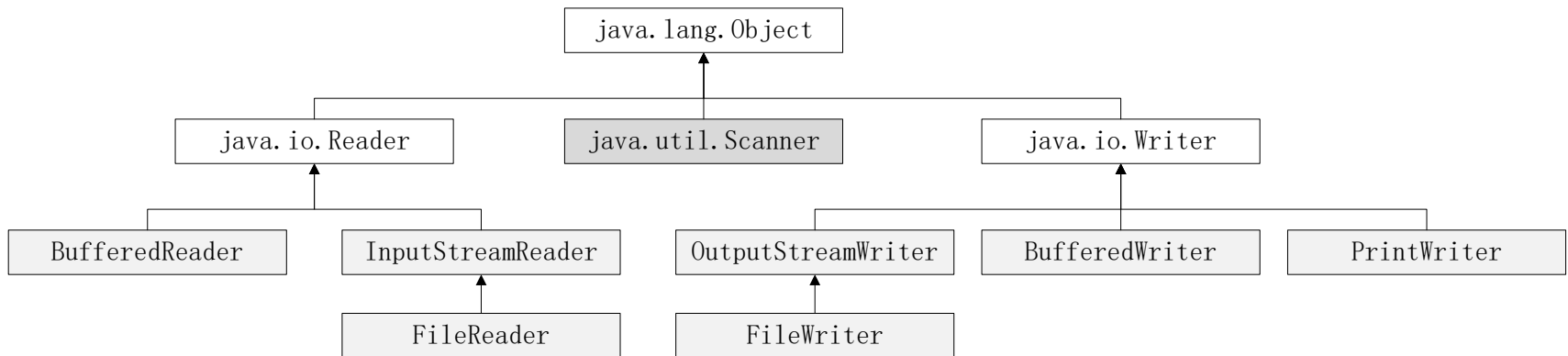
- 文件类File

java.io.File类说明文档			
public class File			
extends Object			
implements Serializable, Comparable<File>			
	修饰符	类成员（节选）	功能说明
1		File (String pathname)	构造方法
2		File (URI uri)	构造方法
3		File (File parent, String child)	构造方法
4		boolean exists ()	检查文件或目录是否存在
5		String getPath ()	获取路径
6		String getName ()	获取文件名
7		String getAbsolutePath ()	获取完整的路径（绝对路径）
8		boolean isFile ()	是否普通文件
9		boolean isDirectory ()	是否目录
10		long length ()	返回文件长度
11		boolean mkdirs ()	创建目录
12		boolean delete ()	删除文件或目录
13		boolean renameTo (File dest)	重命名
14		boolean canExecute ()	是否可执行
15		boolean canRead ()	是否可读
16		boolean canWrite ()	是否可写
17		String[] list ()	返回目录下的所有文件和子目录
18		long getFreeSpace ()	获取分区的空闲空间
.....			



7.3 文件及文件IO

- 文本文件IO



- 字符型**输入流**类族：输入char或String类型数据，根类是Reader
- 字符型**输出流**类族：输出char或String类型数据，根类是Writer
- 字符型文件输出流类**FileWriter**：**创建**文本文件并向其中**写入**文字内容
- 字符型文件输入流类**FileReader**：**读取**文本文件中的内容



例 7-5 一个完整的文本文件输入输出演示程序 (JFileWRTTest.java)

```
1      20      static void fread(String fileName) { // 输入文本文件并显示到显示器的方法
2      21          try { // 必须处理勾选异常 IOException
3      22              FileReader fr = new FileReader(fileName); // 创建字符型文件输入流
4      23              int c;
5      24              while ( (c = fr.read()) != -1 ) { // 从文件读取字符，读完为止
6      25                  System.out.print( (char)c ); // 显示所读出的字符 c
7      26              }
8      27              fr.close(); // 关闭文件
9      28              System.out.println();
10     29              System.out.println(fileName);
11     30          }
12     31          catch(IOException e) // 处理 IOException
13     32          { System.out.println( e.getMessage() );
14     33      } }

15     System.out.println();
16     }
17     catch(IOException e) // 处理 IOException
18     { System.out.println( e.getMessage() );
19 }
```

Problems @ Javadoc Declaration Console

<terminated> JFileWRTTest [Java Application] C:\Java\bin\java.exe d:/fwr.txt 输出成功。

中国农业大学作为教育部直属高校，其历史起自于1905年成立的京师大学堂农科大学。1949年9月，由三所大学下属的农学院合并而成..... d:/fwr.txt 输入成功。

fwr.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

中国农业大学作为教育部直属高校，其历史起自于1905年成立的京师大学堂农科大学。1949年9月，由三所大学下属的农学院合并而成.....

第 1 行, 第 1 列



中国农业大学

网络中心

7.3 文件及文件IO

- 文本文件IO的编程要点
 - 字符型文件输出流类 **FileWriter**
 - 字符型文件输入流类 **FileReader**
 - 异常处理。所有对文件的输入输出操作都可能会抛出 **IOException** 类或其子类的异常。它们属于勾选异常，Java程序必须捕获并处理这些异常
 - 关闭输入输出流。文件IO操作结束后，程序员应当调用文件输入输出流类的方法成员 **close()**，关闭文件输入输出流。通常情况下，文件只能被一个程序操作，这样可以避免读写冲突



7.3 文件及文件IO

- 带缓冲区（buffer）的文本文件IO
 - 100个数据
 - 每次写入一个数据，总共写100次
 - 先将100个数据写入内存缓冲区，再一次性写入文件
 - 对FileReader、FileWriter进行包装
 - 带缓冲区的字符型输入流类BufferedReader
 - 带缓冲区的字符型输出流类BufferedWriter

```
BufferedReader br = new BufferedReader( new FileReader(fileName) );  
BufferedWriter bw = new BufferedWriter( new FileWriter(fileName) );
```

- 使用包装后新的输入输出流对象，对文件进行读写操作时会自动使用内存缓冲区



例 7-6 一个使用缓冲区进行文本文件 IO 的 Java 演示程序 (JBufferedWRTTest.java)

```
1      24      static void fread(String fileName) { // 输入文本文件并显示到显示器的方法
2      import java.io.*; // 必须处理勾选异常 IOException
3      public class JBufferedWRTTest {
4          public static void main(String[] args) {
5              // 将字符型文件输入流对象包装成带缓冲区的字符型文件输入流对象
6              // 27      BufferedReader br = new BufferedReader( new FileReader(fileName) );
7              String s;
8              // 28      while ( (s = br.readLine()) != null ) { // BufferedReader 增加了读一行的方法
9                  System.out.print( s ); // 所读出的字符串中去掉了回车和换行符
10                 System.out.println();
11             }
12             br.close(); // 关闭文件输入流
13             System.out.println(fileName+"输入成功。");
14         }
15         catch(IOException e) // 处理 IOException 异常 (勾选异常)
16         { System.out.println( e.getMessage() ); }
17     } }
18
19     bw.close(); // 关闭文件输出流
20     System.out.println(fileName+"输出成功。");
21     System.out.println();
22 }
```

类 **BufferedWriter** 增加了一个换行方法 **newLine()**
类 **BufferedReader** 增加了一个读一行的方法 **readLine()**

7.3 文件及文件IO

- **格式化**文本文件IO
 - 文本文件只能存储字符类型数据，即**char**或**String**类型数据
 - 任何其他类型数据，例如**int**型或**double**型等，都需要经过**格式化**，转换成字符串形式才能写入文本文件
 - 字符型打印流类**PrintWriter**
 - **字符型**打印流类**PrintWriter**与**字节型**打印流类**PrintStream**的使用方法基本相同
 - 常用的显示器对象**System.out**是**字节型**打印流类**PrintStream**的对象
 - 扫描器类**Scanner**

Java语言的类**PrintWriter/PrintStream** → C语言的**printf()**函数

Java语言的类**Scanner** → C语言的**scanf()**函数



例 7-7 一个对文本文件进行格式化输入输出的 Java 演示程序 (JFormatWSTest.java)

```
1  import java.util.Scanner;
2  import java.io.*;
3  public class JFormatWSTest {
4      static void fscan(String fileName) { // 格式化输入文本文件并显示到显示器的方法
5          Scanner sc = null; // 定义一个扫描器引用变量
6          try { // 必须处理勾选异常 IOException
7              sc = new Scanner( new File(fileName) ); // 创建文件扫描器
8              while ( sc.hasNext() ) { // 检查扫描器中是否还有可输入的数据
9                  int x = sc.nextInt(); // 读取下一个 int 型整数
10                 double y = sc.nextDouble(); // 读取下一个 double 型实数
11                 String str = sc.next(); // 读取下一个字符串
12                 System.out.println( x + ", " + y + ", " + str ); // 显示输入结果
13             }
14             sc.close(); // 关闭文件扫描器
15             System.out.println(fileName + "输入成功。");
16         }
17         catch(IOException e) // 处理 IOException 异常 (勾选异常)
18         { System.out.pr
19         }
20     }
21     public static void main(String[] args) {
22         pw.close(); // 关闭文件打
23         System.out.println(fileName + "输入成功。");
24     }
25     catch(IOException e) // 处理 IOE
26     { System.out.println( e.getMessag
27     }
```

Problems @ Javadoc Declaration Console

<terminated> JPrintWSTest [Java Application] C:\Java
d:/pws.txt输出成功。

10, 15.8, abcd
10, 15.8, abcd
d:/pws.txt输入成功。

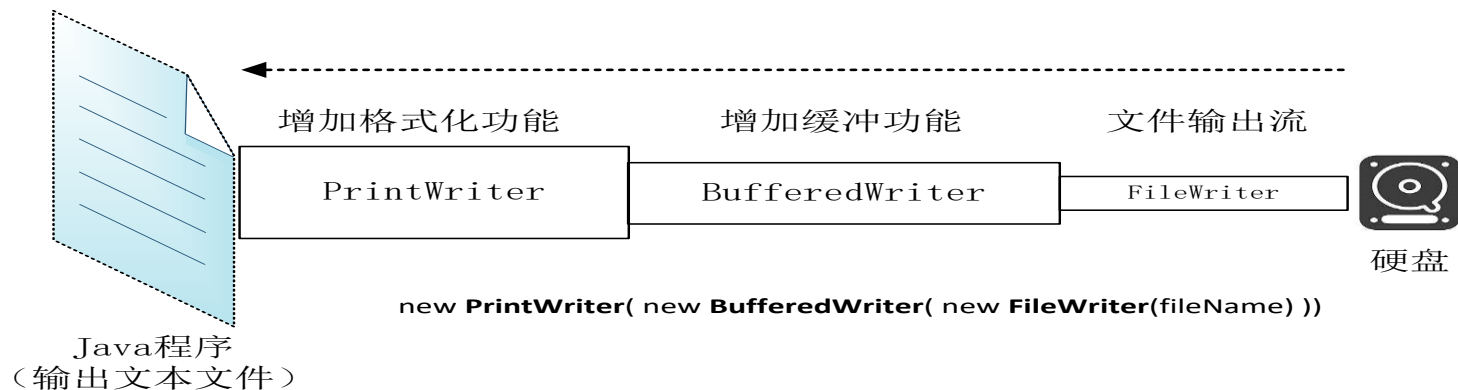


7.3 文件及文件IO

- **格式化**文本文件IO
 - 使用类PrintWriter创建一个字符型文件打印流类对象

```
PrintWriter pw = new PrintWriter( fileName );
```

```
PrintWriter pw =  
    new PrintWriter( new BufferedWriter( new FileWriter(fileName) ));
```



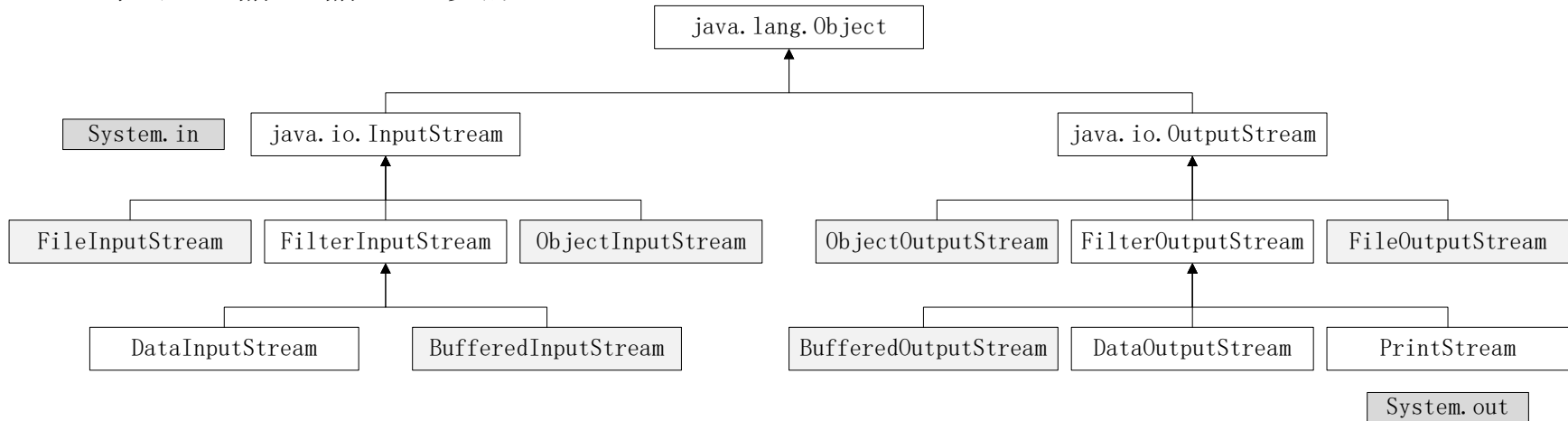
7.4 序列化及二进制文件IO

- 程序需要将所处理变量或对象中的数据**序列化**（serialization）成一个字节流
 - 将数据**保存**到外存文件
 - 通过序列化，可以将内存变量或对象中的数据序列化成字节流，然后保存到外存文件中去，这被称为是数据的**持久化**（persistence）
 - 再次执行程序时，可以将外存文件中的数据迅速恢复到内存变量或对象中，这被称为是数据的**反序列化**（deserialization）
 - 通过网络**传输**数据
- 将数据**序列化**成字节流而不是**格式化**成字符流，这是因为序列化的**处理速度**相对较快，所得到的字节流**数据量**也更小
- 将序列化后的字节流数据保存到外存文件，必须使用**二进制**文件格式



7.4 序列化及二进制文件IO

- 字节型输入输出流类族



- 字节型输入流类族：输入byte型数据，根类是InputStream
- 字节型输出流类族：输出byte型数据，根类是OutputStream
- 二进制文件IO类：FileInputStream、FileOutputStream
- 带缓冲区的字节型输入输出流类：BufferedInputStream、BufferedOutputStream
- 带反序列化/序列化功能的字节型输入输出流类：ObjectInputStream（反序列化）、ObjectOutputStream（序列化）



例 7-8 一个简单数据序列化及二进制文件 IO 的 Java 演示程序 (JObjectIO.java)

```
22     static void fread(String fileName) { // 输入二进制文件并反序列化的方法
23         try { // 必须处理勾选异常 IOException
24             // 先创建字节型文件输入流对象，再包装成带反序列化功能的输入流对象
25             FileInputStream fis = new FileInputStream(fileName);
26             ObjectInputStream ois = new ObjectInputStream( fis );
27             int x = ois.readInt();          // 输入并反序列化一个 int 型整数（4 字节）
28             double y = ois.readDouble(); // 输入并反序列化一个 double 型实数（8 字节）
29             String str = ois.readUTF(); // 输入并反序列化一个字符串（UTF-8 转 UTF-16）
30             System.out.println( x + ", " + y + ", " + str ); // 显示输入结果
31             ois.close(); // 关闭数据输入流
32             System.out.println(fileName + "输入成功。");
33         }
34         catch(IOException e) // 处理 IOException 异常（勾选异常）
35         { System.out.println( e.getMessage() ); }
36     } }
```

16 oos.close(); // 关闭输出流

Problems @ Javadoc Declaration Console

<terminated> JObjectIO [Java Application] C:\Java\jre

d:/sim-io.dat输出成功。

10, 15.8, abcd中国

d:/sim-io.dat输入成功。

sim-io.dat - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

— □w□ @/櫃櫃標 abcd滑球

第 1 行,

7.4 序列化及二进制文件IO

- 简单数据序列化及二进制文件IO的编程要点
 - 对象输出流类**ObjectOutputStream**
FileOutputStream fos = new FileOutputStream(fileName);
BufferedOutputStream bos = new BufferedOutputStream(fos);
ObjectOutputStream oos = new ObjectOutputStream(bos);
 - 对象输入流类**ObjectInputStream**
FileInputStream fis = new FileInputStream(fileName);
BufferedInputStream bis = new BufferedInputStream(fis);
ObjectInputStream ois = new ObjectInputStream(bis);
 - **基本数据类型**数据的序列化
 - **int**型整数可以直接看作是一个4字节的字节流
 - **double**型实数则是一个8字节的字节流
 - **字符串**数据的序列化
 - 方法**writeUTF()**会：UTF-16编码转换成UTF-8编码，然后再序列化成字节流
 - 方法**readUTF()**：UTF-8编码转成UTF-16编码，然后再进行反序列化



7.4 序列化及二进制文件IO

- 对象序列化

- 程序也需要将类类型的**对象数据**序列化成字节流，然后保存到二进制文件中，或通过网络进行传输
- 在Java语言中，只有实现“可序列化”接口**Serializable**的类才能被序列化
`public interface Serializable; // 接口Serializable的定义代码`
- 接口**Serializable**是一个**标记接口**，即不包含任何成员，是一个空接口。实现**Serializable**接口的目的是为类**激活**（或称启用）序列化功能
- 对象输出流类**ObjectOutputStream**：**writeObject()**用于序列化并输出对象数据
- 对象输入流类**ObjectInputStream**：**readObject()**用于输入并反序列化对象



例 7-9 一个对钟表对象进行序列化和反序列化的 Java 演示程序 (JObjectIO.java)

```
1      21      static void fread(String fileName) { // 输入二进制文件并反序列化的方法
38      class Clock implements Serializable { // 定义钟表类时激活序列化功能
39          // 必须添加一个 long 型字段 serialVersionUID, 为类指定一个序列化编号
40          private static final long serialVersionUID = 2018L; // 本例将编号指定为 2018
41          private int hour, minute, second; // 字段: 时分秒
42          public void show() // 显示时间
43          { System.out.println( hour + ":" + minute + ":" + second ); }
44          public Clock(int h, int m, int s) // 构造方法
45          { hour = h; minute = m; second = s; }
46      }
11      31      }
12      32      catch(IOException e) // 处理 IOException 异常 (勾选异常)
13      33      { System.out.println( e.getMessage() ); }
14      34      catch(ClassNotFoundException e) // 处理与类相关的异常
15      35      { System.out.println( e.getMessage() ); }
16      36      } }
17      37
18      catch(IOException e) // 处理 IOException
19      { System.out.println( e.getMessage() ); }
20      }
```

Problems @ Javadoc Declaration Console

<terminated> JObjectIO [Java Application] C:\Java\jre
d:/obj-io.dat输出成功。

8:30:15
10:30:15
d:/obj-io.dat输入成功。

7.4 序列化及二进制文件IO

- 对象序列化
 - 将对象中**字段成员**所保存的数据序列化成字节流
 - 如果某些字段成员不希望被序列化（例如保存银行**账号**或**密码**的字段），则应当在定义类时为这些字段添加修饰符“**transient**”，将它们定义成“**非持久化**”字段
 - Java语言中，类的**非持久化**字段和**静态**字段都不参与今后的序列化或反序列化处理



7.4 序列化及二进制文件IO

- 对象输入输出流类说明文档

java.io.ObjectInputStream类说明文档			
public class ObjectInputStream extends InputStream implements ObjectInput , ObjectStreamConstants			
	修饰符	类成员（节选）	功能说明
1		ObjectInputStream (InputStream in)	构造方法
2		int read (byte[] b, int off, int len)	按字节流读出len个字节
3		byte readByte ()	读出一个byte型整数（1字节）
4		int readUnsignedByte ()	读出一个无符号单字节整数（1字节）
5		int readInt ()	读出一个int型整数（4字节）
6		float readFloat ()	读出一个float型实数（4字节）
7		double readDouble ()	读出一个double型实数（8字节）
8		char readChar ()	读出一个char型字符（2字节）
9		String readUTF ()	读出UTF-8编码的字符串
10		Object readObject ()	读出一个对象然后反序列化
11		void close ()	关闭对象输入流
.....			



7.4 序列化及二进制文件IO

- 对象输入输出流类说明文档

java.io.ObjectOutputStream类说明文档			
public class ObjectOutputStream			
extends OutputStream			
implements ObjectOutput, ObjectOutputStreamConstants			
	修饰符	类成员（节选）	功能说明
1		ObjectOutputStream (OutputStream out)	构造方法
2		void write (byte[] b, int off, int len)	按字节流写入len个字节
3		void writeByte (int v)	写入一个字节（v的低字节）
4		void writeInt (int v)	写入一个int型整数（4字节）
5		void writeFloat (float v)	写入一个float型实数（4字节）
6		void writeDouble (double v)	写入一个double型实数（8字节）
7		void writeChar (int v)	写入一个字符（v的2个低字节）
8		void writeUTF (String str)	转成UTF-8编码字节流后再写入
9		void writeObject (Object obj)	序列化对象然后写入其字节流
10		void flush ()	立即输出缓存里的内容
11		void close ()	关闭数据输出流
.....			



7.5 文本处理

- 文本编辑
 - 文本**编辑**通常使用图形用户界面
 - 用户在图形界面的文本编辑框（例如Java API的文本区域类**JTextArea**）中编辑文本
 - 将编辑好的内容保存到**文本文件**中
- 文本处理
 - 文本处理通常包括**分词**、**查找**、**替换**等字符串操作
 - 文本处理中非常重要的技术——**正则表达式**
（regular expression）



7.5 文本处理

- 文本编辑
 - 文本编辑器演示程序
 - 使用图形界面，方便用户操作
 - 在图形界面中使用多行文本编辑框（**JTextArea**）来**输入**、**编辑**文字
 - 添加**保存**按钮（**JButton**），其功能是将多行文本编辑框中的内容保存到文本文件
 - 添加**打开**按钮（**JButton**），其功能是将文本文件中的内容读入多行文本编辑框
 - 打开或保存文件时使用文件选择对话框（**JFileChooser**）来选择文件



中國農業大學

閻道宏

例 7-10 一个使用 Java API 编写的简单的文本编辑器演示程序 (JNotepad.java)

```

1      26      50      // 为打开按钮添加监听器, 打开已有的文本文件
2      27      51      bOpen.addActionListener( new ActionListener() {    // 匿名类
3      28      52          public void actionPerformed(ActionEvent e) {
4      29      53              JFileChooser fc = new JFileChooser("d:/"); // 创建文件选择对话框
5      30      54              fc.showOpenDialog(null);                // 弹出打开对话框
6      31      55              File f = fc.getSelectedFile();            // 获得所选择的文件
7      32      56              try { // 必须处理异常 IOException (勾选异常)
8      33      57                  // 创建文本文件输入流, 然后包装成带缓冲区的输入流
9      34      58                  BufferedReader in = new BufferedReader( new FileReader(f) );
10     35     59                  text.setText("");                // 清空多行文本编辑框
11     36     60                  String s;
12     37     61                  while ((s = in.readLine()) != null) // 读出文本文件中的一行
13     38     62                      { text.append(s + "\n"); }    // 添加到多行文本编辑框里
14     39     63                  in.close();                        // 关闭文本文件输入流
15     40     64              }
16     41     65              catch(IOException eIO) // 捕获并处理 IOException 异常 (勾选异常)
17     42     66                  { System.out.println( eIO.getMessage() ); }
18     43     67          } });
19     44     68      } }
20     45      49      out.close(); // 关
21     46      49      }
22     47      49      catch(IOException eIO)
23     48      49      { System.out.println( eIO.getMessage() ); }
24     49      49      } });
25     cp.validate(); // 检查

```



文本编辑器演示程序



打开

保存

张三, A公司, 1391234567, zhangsan@cau.edu.cn
 李四, B公司, 1387654321, li@163.com
 王五, C公司, 1890000001, wuWang@sina.com

7.5 文本处理

- 文本分词
 - 字符串类**String**: 分词方法**split()**

例7-11 一个使用字符串类String进行分词的Java演示程序（JTextSplit.java）

```
1 public class JTextSplit { // 测试类
2     public static void main(String[] args) { // 主方法
3         String str = "I am in Beijing"; // 待分词的字符串
4         String words[]; // 字符串数组，用于保存分词结果
5         // 对存储在str中的字符串"I am in Beijing"进行分词
6         words = str.split(" "); // 将空格作为分隔符进行分词
7         System.out.println("\n" + str + "\n的分词结果如下: ");
8         for (String w: words) // 显示分词结果
9             { System.out.print "[" + w + " ] "; }
10        System.out.println(); System.out.println(); // 换行
11        // 再对字符串"one,two,three,four,five"进行分词
12        str = "one,two,three,four,five"; // 待分词的字符串
13        words = str.split(","); // 将逗号作为分隔符进行分词
14        System.out.println("\n" + str + "\n的分词结果如下: ");
15        for (String w: words) // 显示分词结果
16            { System.out.print "[" + w + " ] "; }
17        System.out.println();
18    } }
```

Problems @ Javadoc Declaration Console

<terminated> JTextSplit [Java Application] C:\Java\jre

"I am in Beijing"的分词结果如下:
[I] [am] [in] [Beijing]

"one,two,three,four,five"的分词结果如下:
[one] [two] [three] [four] [five]

7.5 文本处理

- 正则表达式
 - 一个字符序列可以包含哪些字符，各字符之间又有怎样的排列规律，这被称为是字符序列的**模式**（pattern）
 - 计算机语言使用**正则表达式**来描述字符序列的模式
 - 正则表达式具有专门的**语法规则**，可以描述各种复杂的字符序列模式
 - 应用正则表达式
 - 先按照语法**编写**描述某种字符序列模式的正则表达式
 - 用正则表达式去**匹配**（match）字符串，检查字符串是否符合规定的模式，或是在字符串中查找符合规定模式的子字符串
 - 正则表达式语法
 - 单个字符
 - 单词（多个字符）
 - 词组（多个单词）
 - 句子（或称整行）



7.5 文本处理

表7-1 描述单个字符模式的正则表达式

模式说明（单个字符模式）	正则表达式语法	正例与反例
某个特定的字符。例如字母a	"a"	正例: "a" 反例: "b"、"A"、"1"、",".....
任意字符（除了换行、回车）。 注: "."被称为通配符	"."	正例: "a"、"A"、"1"、","..... 反例: "\n"、"\r"
某几个特定字符中的一个。 例如，字母a、b、d中的一个	"[abd]"	正例: "a"、"b"、"d" 反例: "c"、"e"、"1"、",".....
某几个连续字符中的一个。 例如，字母a~d中的一个	"[a-d]"	正例: "a"、"b"、"c"、"d" 反例: "e"、"A"、"1"、",".....
除去几个特定字符之外的其他字符。 例如，除去字母a、b、d之外的其他字符	"[^abd]"	正例: "c"、"e"、"1"、","..... 反例: "a"、"b"、"d"
除去某几个连续字符之外的其他字符。 例如，除去字母a~d之外的其他字符	"[^a-d]"	正例: "e"、"A"、"1"、","..... 反例: "a"、"b"、"c"、"d"
数字字符，即0~9之间的字符	"[0-9]"或简写成"[\d]"	正例: "0"、"1"、"2"、"3"..... 反例: "a"、"b"、"A"、",".....
非数字字符	"^[0-9]"或简写成"[\D]"	正、反例与上一格相反
空白字符，即空格、Tab键、换行等	"[\n\r\t\x0B\f]" 或简写成"[\s]"	正例: " "、"\n"、"\r"、"\t"..... 反例: "a"、"A"、"1"、",".....
非空白字符	"[^\\s]"或简写成"[\\S]"	正、反例与上一格相反
单词字符，即字母或数字字符	"[a-zA-Z_0-9]" 或简写成"[\w]"	正例: "a"、"z"、"A"、"1"..... 反例: " "、","、"_", "\$".....
非单词字符	"[^\\w]"或简写成"[\\W]"	正、反例与上一格相反



7.5 文本处理

- 正则表达式

表7-2 描述单词（多个字符）模式的正则表达式

模式说明（单词模式）	正则表达式语法	正例与反例
某个特定的单词。例如abc	"abc"	正例: "abc" 反例: "a"、"ab"、"Abc"、"ab2".....
任意多个一样的字符，不能是0个。 例如3个字母a，即aaa	"a+"	正例: "a"、"aa"、"aaa"..... 反例: ""、"b"、"Ab".....
任意多个一样的字符，可以是0个。 例如3个字母a，即aaa	"a*"	正例: ""、"a"、"aa"、"aaa"..... 反例: "b"、"Ab".....
某个可以省略的字符。 例如，字母a可以省略	"a?"	正例: ""、"a" 反例: "b"、"A"、"Ab".....
n个一样的字符。 例如 3个字母a，即aaa	"a{3}"	正例: "aaa" 反例: "a"、"aa"、"Aaa".....
至少n个一样的字符。 例如，至少3个字母a	"a{3,}"	正例: "aaa"、"aaaa"、"aaaaa"..... 反例: "a"、"aa"、"Aaa".....
m~n个一样的字符。 例如，1~3个字母a	"a{1, 3}"	正例: "a"、"aa"、"aaa"..... 反例: "aaaa"、"Aa"、"Aaa".....
含有某个或某几个字符的单词 例如a0、a1、a2	"a[\\d]"	正例: "a0"、"a1"、"a2"..... 反例: "a"、"ab"、"b1".....
带可重复字符的单词。例如abc、abbc、ac，其中的字母b可重复	"ab*c"	正例: "ac"、"abc"、"abbc"..... 反例: "a"、"c"、"acb".....



表7-3 描述词组（多个单词）模式的正则表达式

模式说明（词组模式）	正则表达式语法	正例与反例
某个特定的词组。 例如，某个文件目录c:/java/src可认为是由3个单词组成的词组。正则表达式使用小括号"()"进行分组	"c:/java/src" 或定义成词组： "(c:)(/java)(/src)" 注：3个单词的序号依次为0、1、2	正例："c:/java/src" 反例："c:/java/bin"、"d:/java/src".....
含有某个或某几个特定字符的词组。 例如，硬盘分区c或d上的目录/java/src	"([cd:])(/java)(/src)"	正例："c:/java/src"、"d:/java/src"..... 反例："c:/java/bin"、"e:/java/src".....
用" "指定含有某几个特定单词的词组。 例如，指定目录c:/java下的/src或/bin子目录	"(c:)(/java)(/src bin)"	正例："c:/java/src"、"c:/java/bin" 反例："c:/java/doc"、"c:/java/".....
用"+"表示含有重复字符的词组。 例如，目录c:/java下的所有子目录。 注："."是通配符	"(c:)(/java)(/.+)"	正例："c:/java/src"、"c:/java/bin"..... 反例："c:/cpp/src"、"c:/cpp/hpp".....
用"+"表示含有重复单词的词组。 例如，目录c:/java/java/src	"(c:)(/java)+(/src)"	正例："c:/java/src"、"c:/java/java/src"..... 反例："c:/cpp/src"、"c:/cpp/hpp".....

表7-4 描述句子（整行）模式的正则表达式

模式说明（句子模式）	正则表达式语法	正例与反例
用"^"指定以某个字符或单词开头的句子。 例如，指定位于硬盘分区"c:"上的文件目录/java或/java/src	"^(c:)(/java)(/src)?"	正例："c:/java"、"c:/java/src" 反例："c:/; c:/java"、"d:/java".....
用"\$"指定以某个字符或单词结尾的句子。 例如，指定以"src"结尾的文件目录c:/src或c:/java/src	"(c:)(/java)?(/src)\$"	正例："c:/src"、"c:/java/src" 反例："c:/src/img"、"c:/src; c:/".....
用"^"和用"\$"指定完整的句子（整行）。 例如c:/java/src	"^(c:)(/java)(/src)\$"	正例："c:/java/src" 反例："c:/; c:/java/src"、"c:/java/src/img".....
用"\b"指定句子中某个单词前面或后面的字符必须是非单词字符，即不能是字母或数字字符	".*\bdog\b.*"	正例："I love dog." 反例："I love doggie.".....
用"\B"指定句子中某个单词前面或后面的字符必须是单词字符，即只能是字母或数字字符	".*\Bdog\B.*"	正例："I love doggie." 反例："I love dog.".....



7.5 文本处理

- 正则表达式
 - 几个例子

- 中国的**邮政编码**: `"\b[1-9]\d{5}\b"`
- HTTP**网址**: `"^http://www.(.\w+([-_]\w+)*)+$"`
- 电子邮件**地址: `"\w+([-_]\w+)*@\w+([-_.]\w+)*.\w+"`

— 以字符串常量形式书写，其中的反斜杠“****”需使用转义形式，将其写成“****”



7.5 文本处理

- 正则表达式
 - 模式类**Pattern**与匹配器类**Matcher**

java.util.regex. Pattern 类说明文档			
public final class Pattern extends Object implements Serializable			
	修饰符	类成员（节选）	功能说明
1	static	Pattern compile (String regex)	编译字符串形式的正则表达式，将其转成模式对象的形式
2	static	Pattern compile (String regex, int flags)	编译字符串形式的正则表达式
3	static	boolean matches (String regex, CharSequence input)	检查字符串input是否符合正则表达式regex规定的模式
4		String[] split (CharSequence input)	根据正则表达式指定的分隔符对字符串进行分词
5		String pattern ()	返回描述正则表达式的字符串
6		Matcher matcher (CharSequence input)	使用正则表达式对字符串input进行处理，返回一个匹配器对象（将用于下一步词法分析或处理）



7.5 文本处理

- 正则表达式
 - 模式类 **Pattern**

例7-12 给出一个使用模式类Pattern进行分词的Java演示程序（JPatternTest.java）

```
1 import java.util.regex.*; // 导入java.util.regex包中与正则表达式相关的类
2 public class JPatternTest {           // 测试类
3     public static void main(String[] args) { // 主方法
4         String str = "one,two three, four  five"; // 格式比较随意的字符串
5         Pattern p = Pattern.compile("[, ]+");      // 描述由任意多个逗号或空格组成的分隔符
6         String words[] = p.split(str);             // 使用正则表达式进行分词
7         System.out.println("\\" + str + "\"的分词结果如下: ");
8         for (String s: words) // 显示分词结果, 用“[]”将每个单词括起来
9             { System.out.print("[" + s + "]); }
10        System.out.println();
11    } }
```

"one,two three, four five"的分词结果如下:
[one][two][three][four][five]



中國農業大學

閻道宏

7.5 文本处理

- 正则表达式
 - 模式类 **Pattern**

例7-13 一个使用模式类Pattern检查email邮箱格式的Java演示程序（JMailTest.java）

```
1 import java.util.regex.Pattern; // 导入java.util.regex包中的类Pattern
2 public class JMailTest {           // 测试类
3     public static void main(String[] args) { // 主方法
4         String mFormat = "^\\w+([-_]\\w+)*@\\w+([-_]\\w+)*\\.\\w+$"; // email格式
5         String mail1 = "kan-daohong@cau.edu.cn"; // 符合email格式的邮箱
6         String mail2 = "kan-daohong.cau.edu.cn"; // 不符合email格式的邮箱
7         // 下面调用模式类Pattern的静态方法matches()检查mail1和mail2的邮箱格式
8         if (Pattern.matches(mFormat, mail1) == true) // 检查mail1是否符合邮箱格式
9             System.out.println(mail1 + ": 合法邮箱");
10        else System.out.println(mail1 + ": 非法邮箱");
11        if (Pattern.matches(mFormat, mail2) == true) // 检查mail2是否符合邮箱格式
12            System.out.println(mail2 + ": 合法邮箱");
13        else System.out.println(mail2 + ": 非法邮箱");
14    } }
```

kan-daohong@cau.edu.cn: 合法邮箱

kan-daohong.cau.edu.cn: 非法邮箱



中國農業大學

閻道宏

7.5 文本处理

- 正则表达式

- 匹配器类**Matcher**

- 使用正则表达式对某个文本字符串进行词法分析或处理的过程可分为如下三步

- 创建描述正则表达式的**模式对象**

- ```
Pattern p = Pattern.compile("[^@]+@([\\w]+)(\\.([\\w]+)+)");
```

- 使用模式对象**创建**处理某个文本字符串的**匹配器对象**

- ```
Matcher m = p.matcher( str );
```

- 使用匹配器对象m对**文本字符串**str进行词法分析或处理，例如查找或替换与正则表达式匹配的子串



7.5 文本处理

- 正则表达式
 - 匹配器类 **Matcher**

```
Problems @ Javadoc Declaration Console
<terminated> JMatcherFind [Java Application] C:\Java
张三,A公司,1391234567,zhangsan@cau.edu.cn
李四,B公司,1397654321,li@163.com
王五,C公司,1890000001,wuwang@sina.com

上述文本内容中的email地址如下:
18-37: zhangsan@cau.edu.cn
56-66: li@163.com
85-100: wuwang@sina.com
```

例7-14 一个在通讯录文本字符串中查找email地址的Java演示程序（JMatcherFind.java）

```
1 import java.util.regex.*; // 导入java.util.regex包中与正则表达式相关的类
2 public class JMatcherFind { // 测试类
3     public static void main(String[] args) { // 主方法
4         String str = "张三,A公司,1391234567,zhangsan@cau.edu.cn\n" + // 通讯录字符串
5             "李四,B公司,1397654321,li@163.com\n" +
6             "王五,C公司,1890000001,wuwang@sina.com\n";
7         String mf = "\\w+([-_]\\w+)*@[\\w+([-_]\\w+)*\\.\\w+"; // email格式的正则表达式
8         Pattern p = Pattern.compile(mf); // 将正则表达式编译成模式对象
9         Matcher m = p.matcher(str); // 取得处理字符串str的匹配器对象
10        System.out.println( str + "\n上述文本内容中的email地址如下: ");
11        while (m.find() == true) { // 显示查找结果: 检查是否还有下一个匹配项
12            int p1 = m.start(); int p2 = m.end(); // 取出匹配项的起始和结束下标
13            System.out.println( p1 + "-" + p2 + ": " + str.substring(p1, p2) ); // 显示匹配子串
14        }
15    } }
```



7.5 文本处理

- 正则表达式
 - 匹配器类 **Matcher**

```
Problems @ Javadoc Declaration Console
<terminated> JMatcherReplace [Java Application] C:\Java\jre1.8.0_152\b
搜索以下文本中的dog（不区分大小写）：I love small dog and big DOG.
13-16: dog
25-28: DOG

将dog替换成cat: I love small cat and big cat.
```

例7-15 一个使用匹配器类Matcher实现字符串查找与替换的Java演示程序（JMatcherReplace.java）

```
1 import java.util.regex.*; // 导入java.util.regex包中与正则表达式相关的类
2 public class JMatcherReplace { // 测试类
3     public static void main(String[] args) { // 主方法
4         // 查找以下文本字符串text中的单词dog（不区分大小写）
5         String text = "I love small dog and big DOG.";
6         Pattern p = Pattern.compile("dog", Pattern.CASE_INSENSITIVE); // 对大小写不敏感
7         Matcher m = p.matcher(text); // 取得处理字符串text的匹配器对象
8         // 显示字符串中查找到的所有dog（不区分大小写）
9         System.out.println("搜索以下文本中的dog（不区分大小写）：" + text);
10        while (m.find() == true) { // 显示查找结果：检查是否还有下一个匹配项
11            int s = m.start(); int e = m.end(); // 取得匹配子串的起始和结束下标
12            String msg = String.format("%d-%d: %s", s, e, text.substring(s, e));
13            System.out.println(msg);
14        }
15        // 将字符串中的dog全部替换成cat
16        System.out.print("\n将dog替换成cat: ");
17        String str = m.replaceAll("cat"); // 将查找到的dog全部替换成cat
18        System.out.println(str);
19    } }
```



7.5 文本处理

- 正则表达式
 - 匹配器类**Matcher**

java.util.regex.Matcher类说明文档			
public final class Matcher extends Object implements MatchResult			
	修饰符	类成员（节选）	功能说明
1		boolean find()	查找下一个匹配项
2		int start()	取出匹配项的起始下标
3		int end()	取出匹配项的结束下标
4		boolean matches()	检查匹配器中的字符串是否符合正则表达式规定的模式
5		String replaceAll (String replacement)	替换所有的匹配项
6		Matcher appendReplacement (StringBuffer sb, String replacement)	取出匹配项及其前面的未匹配子串，替换匹配项，然后一起追加到sb中
7		StringBuffer appendTail (StringBuffer sb)	将匹配项后面的字符串追加到sb中
8		Matcher usePattern (Pattern newPattern)	重新指定正则表达式
9		boolean lookingAt()	匹配查找
10		int groupCount()	获得分组个数
11		String group (int group)	取出分组
.....			

7.6 图像处理

- 使用Java API进行图像处理
 - 打开图像文件
 - 图像有不同的文件格式，常用的有JPEG、BMP、PNG、GIF和TIFF等
 - 打开图像文件就是将文件中的图像数据读入内存，以便后续显示或处理
 - Java API提供了两个存储图像数据的类
 - 不可修改的图标类`javax.swing.ImageIcon`，主要用于显示
 - 可以修改的带缓存图像类`java.awt.image.BufferedImage`，主要用图像编辑或处理
 - 显示图像
 - 可以使用标签组件`JLabel`来显示图标类图像
 - 显示带缓存的图像一般使用画布类`Canvas`，通过重写绘图方法`paint()`来显示图像
 - 修改图像。可以修改带缓存的图像，例如修改像素值，或在图像上绘图
 - 保存图像文件。通常需要将修改后的带缓存图像保存回图像文件。Java API提供了一个图像输入输出类`javax.imageio.ImageIO`



7.6 图像处理

- 图标类 **Imagelcon**

例7-16 一个图标类Imagelcon的Java演示程序（JImagelconTest

```
1 import java.awt.*;           // 导入java.awt包中的类
2 import java.awt.event.*;     // 导入java.awt.event包中的类
3 import javax.swing.*;        // 导入javax.swing包中的类
4
5 public class JImagelconTest {           // 测试类
6     public static void main(String[] args) { // 主方法
7         JFrame w = new JFrame();         // 创建框架窗口类JFrame的对象
8         w.setTitle("图像演示程序");     // 初始化窗口
9         w.setSize(420, 320); w.setLocation(100, 100); w.setVisible(true);
10        w.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        // 在窗口内容面板里添加一个显示图像的标签组件
12        Container cp = w.getContentPane(); // 获得窗口的内容面板（默认边框布局）
13        JLabel box = new JLabel();
14        cp.add(box, BorderLayout.CENTER); // 将标签放在内容面板的中间
15        cp.validate();                   // 检查并自动布局容器里的组件
16        // 从图像文件加载图像，创建一个图标对象
17        Imagelcon ii = new Imagelcon("d:/1.jpg");
18        box.setIcon(ii); // 在标签组件中显示图像
19    } }
```



7.6 图像处理

- 图标类**ImageIcon**

- 说明文档

javax.swing.ImageIcon类说明文档			
public class ImageIcon			
extends Object			
implements Icon , Serializable , Accessible			
	修饰符	类成员（节选）	功能说明
1		ImageIcon()	构造方法
2		ImageIcon (String filename)	构造方法（从文件加载）
3		ImageIcon (URL location)	构造方法（从网络加载）
4		int getIconWidth()	返回图标的宽度
5		int getIconHeight()	返回图标的高度
6		Image getImage()	读出图标里的图像
7		void setImage (Image image)	设置图标的图像
.....			



例 7-17 一个带缓存图像类 BufferedImage 的 Java 演示程序 (JBufferedImageTest.java)

```

1  import java.awt.*;           // 导入 java
2  import java.awt.event.*;     // 导入 java
3  import javax.swing.*;       // 导入 javax
4  import java.io.*;           // 导入 java
5  import java.awt.image.BufferedImage;
6  import javax.imageio.ImageIO;
7
8  public class JBufferedImageTest {
9      public static void main(String[] args) {
10         JFrame w = new JFrame();
11         w.setTitle("图像演示程序");
12         w.setSize(420, 320); w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         // 从文件加载图像, 创建 BufferedImage 对象
14         BufferedImage bi = null;
15         try { // 必须处理勾选异常 IOException
16             File fin = new File("d:/1-new.jpg");
17             bi = ImageIO.read(fin);
18         }
19         catch(IOException e) { // 必须处理勾选异常 IOException
20             System.out.println( e.getMessage() ); // 输出异常信息
21         }
22         // 修改图像: 将图像中各像素的颜色值设为其补色
23         Color c1, c2;
24         for (int y = 0; y < bi.getHeight(); y++) { // 行循环
25             for (int x = 0; x < bi.getWidth(); x++) { // 列循环
26                 c1 = new Color( bi.getRGB(x, y) ); // 取出颜色值
27                 c2 = new Color( 255-c1.getRed(), 255-c1.getGreen(), 255-c1.getBlue() ); // 计算补色
28                 bi.setRGB(x, y, c2.getRGB()); // 设为补色 c2
29             }
30         }
31         // 在窗口内容面板里添加一个显示图像的画布组件
32         Container cp = w.getContentPane(); // 获得窗口的内容面板 (默认边框布局)
33         MyCanvas cv = new MyCanvas(bi); // 创建用于显示图像的画布对象
34         cp.add(cv, BorderLayout.CENTER); // 将画布放在内容面板的中间
35         cp.validate(); // 检查并自动布局容器里的组件
36         // 保存图像: 将修改后的图像保存到一个新的图像文件
37         try { // 必须处理勾选异常 IOException
38             File fout = new File("d:/1-new.jpg");
39             ImageIO.write(bi, "jpg", fout); // 图像输入输出类 ImageIO
40         }
41         catch(IOException e) { System.out.println( e.getMessage() ); }
42     }
43
44     class MyCanvas extends Canvas { // 定义一个新的画布类, 重写 paint()方法
45         private BufferedImage bi = null;
46         public MyCanvas(BufferedImage bi) {
47             this.bi = bi;
48         }
49         public void paint(Graphics g) {
50             if (bi != null) {
51                 g.drawImage(bi, 0, 0, bi.getWidth(), bi.getHeight(), this);
52             }
53         }
54     }
55 }

```



7.6 图像处理

- 带缓存图像类 **BufferedImage**

java.awt.image. BufferedImage 类说明文档			
public class BufferedImage			
extends Image			
implements WritableRenderedImage, Transparency			
	修饰符	类成员（节选）	功能说明
1	static	int TYPE_INT_RGB	图像常量，8位RGB存储
2	static	int TYPE_INT_BGR	图像常量，8位BGR存储
3		BufferedImage (int width, int height, int imageType)	构造方法
4		int getWidth ()	获取图像宽度
5		int getHeight ()	获取图像高度
6		int getType ()	获取图像类型
7		int getRGB (int x, int y)	读取某个像素的RGB值
8		void setRGB (int x, int y, int rgb)	设置某个像素的RGB值
9		BufferedImage getSubimage (int x, int y, int w, int h)	取出一幅子图像，即裁剪
10		Graphics getGraphics ()	取出图像的绘图对象
.....			



7.6 图像处理

- 带缓存图像类 **BufferedImage**
 - 图像输入输出类 **ImageIO** 定义了一组静态方法

javax.imageio. ImageIO 类说明文档			
public final class ImageIO extends Object			
	修饰符	类成员（节选）	功能说明
1	static	BufferedImage read (File input)	从文件读取图像
2	static	BufferedImage read (URL input)	从网址读取图像
3	static	BufferedImage read (InputStream input)	从输入流读取图像
4	static	boolean write (RenderedImage im, String formatName, File output)	将图像写入文件
6	static	boolean write (RenderedImage im, String formatName, OutputStream output)	将图像写入输出流
7	static	String[] getReaderFormatNames ()	以字符串数组的形式返回系统 可以读取的图像格式
8	static	String[] getWriterFormatNames ()	以字符串数组的形式返回系统 可以输出的图像格式
9	static	ImageWriter getImageWriter (ImageReader reader)	返回对应的编码器
10	static	ImageReader getImageReader (ImageWriter writer)	返回对应的解码器
.....			



7.6 图像处理

- 修改图像

- 图像处理

- 图像滤波、图像分割、 γ 校正、几何变换等
 - 编写图像处理程序需具备**数字图像处理**的专业知识

- 图像编辑

- 图像裁剪、为图像添加文字或在图像上绘图等
 - 使用带缓存图像类**BufferedImage**的方法成员**getSubimage()**取出指定区域的子图像，就可以实现图像裁剪的功能
 - 使用图形类**Graphics**则可以实现在图像上**添加文字**或**绘制图形**的功能。带缓存图像包含一个Graphics类的子类绘图对象



7.6 图像处理

- 修改图像

- 图像编辑

- 在带缓存图像上**添加文字**
 - 首先取出带缓存图像的
 - 然后使用绘图对象在图



```
Graphics g = bi.getGraphics(); // 获取图像bi的绘图对象
```

```
g.setColor( Color.YELLOW ); // 设置绘图颜色
```

```
Font ef = new Font("TimesRoman", Font.PLAIN, 32); // 选择字体
```

```
g.setFont( ef ); // 设置字体
```

```
g.drawString("Hello, World!", 20, 40); // 显示文字信息
```

```
g.drawOval(20, 80, 180, 60); // 画一个椭圆
```



中國農業大學

閻道宏

7.7 声音处理

- 编写简单的**录音**及**播放**程序
 - **录音**就是对**麦克风**输入的声音进行数字化采样，然后将其**保存**成音频文件
 - **播放**就是播放**音频文件**里的声音
- 数字化采样所得到的声音数据可以做很多后续处理
 - **语音识别**。编写语音识别程序还需要语音信号处理、人工智能方面的专业知识
 - 数字化采样是后续声音信号处理的第一步



7.7 声音处理

- 相关概念与术语

- 音频格式

- 编码算法（PCM或MP3）、采样率（8KHz或16KHz）、采样位数（8位或16位）、声道数（单声道或双声道）、帧率、每帧字节数等参数

- Java API主要支持PCM音频编码算法，并提供了一个音频格式类**AudioFormat**来描述音频格式
 - 创建音频格式**对象**时需指定相关的参数

AudioFormat af = **new** AudioFormat(8000f, 16, 1, true, true);



中國農業大學

閻道宏

7.7 声音处理

- 相关概念与术语
 - 音频格式

javax.sound.sampled.AudioFormat类说明文档			
public class AudioFormat extends Object			
	修饰符	类成员（节选）	功能说明
1		AudioFormat (float sampleRate, int sampleSizeInBits, int channels, boolean signed, boolean bigEndian)	构造方法
2		AudioFormat (AudioFormat.Encoding encoding, float sampleRate, int sampleSizeInBits, int channels, int frameSize, float frameRate, boolean bigEndian)	构造方法
3		int getChannels ()	获取声道数
4		AudioFormat.Encoding getEncoding ()	获取编码格式
5		float getFrameRate ()	获取帧率
6		int getFrameSize ()	获取每帧的字节数
7		float getSampleRate ()	获取采样率
8		int getSampleSizeInBits ()	获取采样位数
.....			



7.7 声音处理

- 相关概念与术语

- 音频文件格式

- WAVE、AIFF、AU、MP3、WMA等
 - Java API支持**WAVE**、**AIFF**和**AU**文件格式

- 音频系统

- 本地**计算机**的音频系统
 - Java API提供一个音频系统类**AudioSystem**



7.7 声音处理

- 相关概念与术语

javax.sound.sampled. AudioSystem 类说明文档			
public class AudioSystem extends Object			
	修饰符	类成员（节选）	功能说明
1	static	boolean isLineSupported (Line.Info info)	系统是否支持某种数据线
2	static	boolean isFileTypeSupported (AudioFormat.Type fileType)	系统是否支持某种音频文件格式
3	static	boolean isFileTypeSupported (AudioFormat.Type fileType, AudioInputStream stream)	系统是否支持某种音频文件格式和某种音频输入流格式
4	static	TargetDataLine getTargetDataLine (AudioFormat format)	创建输入音频的目标数据线对象
5	static	SourceDataLine getSourceDataLine (AudioFormat format)	创建输出音频的源数据线对象
6	static	Clip getClip ()	创建播放音频的片段对象
7	static	AudioInputStream getAudioInputStream (File file)	创建音频文件输入流对象
8	static	AudioInputStream getAudioInputStream (URL url)	创建网络音频文件输入流对象
9	static	int write (AudioInputStream stream, AudioFormat.Type fileType, File out)	保存音频文件
10	static	Mixer getMixer (Mixer.Info info)	创建混音器对象
11	static	Mixer.Info[] getMixerInfo ()	获取混音器信息
.....			

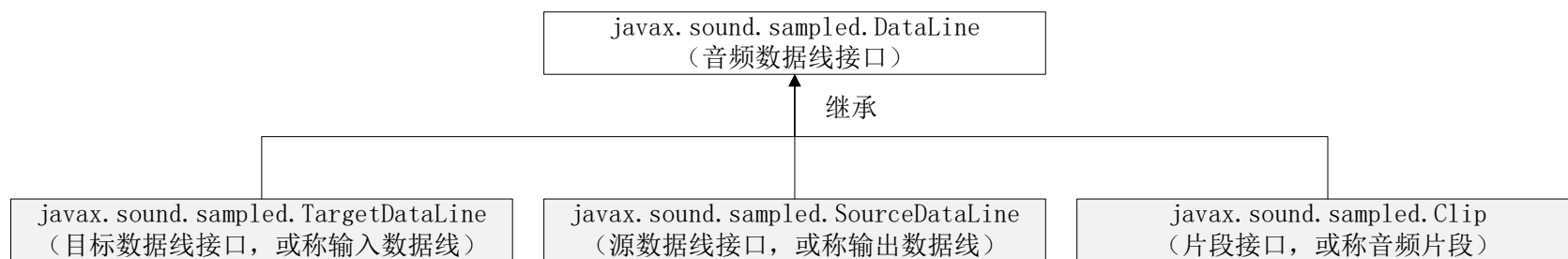


7.7 声音处理

- 相关概念与术语

- 数据线

- Java API将连接音频设备的物理线路抽象成若干个**数据线**（data line）接口



- 目标数据线接口**TargetDataLine**表示连接音频输入设备（例如**麦克风**）的数据线
 - 源数据线接口**SourceDataLine**接口表示连接音频输出设备（例如**音箱**）的数据线
 - 片段接口**Clip**表示可以播放的音频数据（即**音频片段**）
 - **javax.sound.sampled**包

- 音频输入流：音频输入流类**AudioInputStream**



例 7-18 一个实现音频“录音-回放-保存”功能的 Java 演示程序 (JRecordWavTest.java)

```

1  import java.io.*;
2  import javax.sound.sampled.*;
3
4  public class JRecordWavTest {
5      public static void main(String[] args) {
6          AudioFormat format = new AudioFormat(
7              TargetDataLine.TargetDataLine.class,
8              SourceDataLine.SourceDataLine.class,
9              44100, 2, 16, true);
10         // 先创建源数据线的源数据流
11         AudioFormat af = new AudioFormat(
12             // 录音-回放-保存
13             try { // 创建源数据线的源数据流
14                 // 创建源数据线的源数据流
15                 DataLine.Info info = new DataLine.Info(
16                     if (SourceDataLine.class.isAssignableFrom(
17                         {
18                             tLine = (SourceDataLine) AudioSystem.getLine(
19                                 // 创建源数据线的源数据流
20                                 Sys
21                                 tLine = (SourceDataLine) AudioSystem.getLine(
22                                     tLine.start(), // 启动源数据流，开始录音
23                                     int dataLen = tLine.read(buf, 0, buf.length); // 读音频数据，读满缓冲区
24                                     tLine.stop(); tLine.close(); // 停止录音，关闭目标数据线
25                                     System.out.println("Line Stop");
26                                     // 回放：创建输出音频的的源数据线，然后打开回放，结束后关闭
27                                     info = new DataLine.Info(SourceDataLine.class, af);
28                                     sLine = (SourceDataLine) AudioSystem.getLine(info); // 获取源数据线
29                                     System.out.println("Play");
30                                     sLine.open(af); // 按照指定的音频格式 af 打开源数据线
31                                     sLine.start(); // 启动源数据线，开始播放
32                                     sLine.write(buf, 0, dataLen); // 将音频数据写入源数据线
33                                     sLine.drain(); sLine.close(); // 播放完音频数据后关闭源数据线
34                                     // 保存：先将音频数据包装成一个字节数组输入流，
35                                     // 然后再包装成一个音频输入流，最后保存到音频文件 (.wav)
36                                     ByteArrayInputStream inBuf = new ByteArrayInputStream(buf);
37                                     AudiInputStream ais = new AudiInputStream(
38                                         inBuf, af, dataLen / af.getFrameSize() );
39                                     System.out.println("Save");
40                                     File fout = new File("d:/1.wav"); // 音频文件
41                                     AudioSystem.write(ais, AudioFileFormat.Type.WAVE, fout); // 输出音频文件
42                                     ais.close(); inBuf.close(); // 关闭音频输入流和字节数组输入流
43                                 }
44                                 catch (LineUnavailableException e) { System.out.println( e.getMessage() ); }
45                                 catch (IOException e) { System.out.println( e.getMessage() ); }
46                             } }

```



7.7 声音处理

- 播放音频文件

例7-19 一个播放音频文件的Java演示程序（JPlayWavTest.java）

```
1 import java.io.*;          // 导入java.io包中的输入输出流类
2 import javax.sound.sampled.*; // 导入javax.sound.sampled包中的音频类
3
4 public class JPlayWavTest {    // 主类
5     public static void main(String[] args) { // 主方法
6         try { // 播放过程中可能会抛出勾选异常
7             // 为音频文件建立起音频输入流
8             File f = new File("d:/1.wav"); // 音频文件
9             AudioInputStream ais = AudioSystem.getAudioInputStream(f);
10            System.out.println( ais.getFormat() ); // 显示音频格式
11            Clip c = AudioSystem.getClip(); // 获取播放音频的片段对象
12            c.open(ais); // 在片段对象中打开音频输入流ais
13            c.setFramePosition(0); // 设置播放起始位置
14            System.out.println("Start");
15            c.start(); // 开始播放音频
16            Thread.sleep(10000); // 休眠等待10秒，否则音频未播放完程序就结束了
17            c.close(); // 关闭片段对象
18            System.out.println("Close");
19        }
20        catch (Exception e) { System.out.println( e.getMessage() ); }
21    } }
```



中國農業大學

閻道宏

第7章 输入输出流

- 本章学习要点

- Java API中的类往往经历了**多级抽象**和**多层包装**，例如输入输出流类族中的类。读者在学习过程中要注意及时总结并梳理出类与类之间的继承或包装关系
- 初学者可以从**常用类**开始，先学习使用，然后再追溯其超类。逐步从**微观**到**宏观**，最终实现从整体上把握Java API类库的目标
- 学习并掌握**标准IO**和**文件IO**的常规编程方法和代码框架
- 学习并掌握基本的**文本处理**方法，并能运用简单的正则表达式进行文本分析和处理
- 学习并了解基本的**图像**及**声音**处理方法

