输入/输出

孙聪

网络与信息安全学院

2019-11-04

课程内容

- Java概述
- 面向对象程序设计概念
- Java语言基础
- Java面向对象特性
- Java高级特征
- 容器类
- 常用预定义类
- 异常处理
- 输入输出
- 线程

提要

提要

java.io.File

- File类是外存文件和目录的路径名（pathname）的抽象表示
- File类用来操作文件和获得文件的信息，但不提供对文件数据读取的方法（由文件流提供）
- 通过File类的方法，可以
  - 生成新的目录、临时文件
  - 改变文件名
  - 删除文件
  - 得到文件或目录的描述信息（文件名、路径、可读写性、长度等）
  - 列出一个目录中的所有文件或满足某种模式的文件
  - 检查一个File对象代表的是文件还是目录

抽象路径名（abstract pathname）

- 抽象路径名包括两部分：
  - 系统相关的前缀字符串（可选）
    - Unix的root: “/”
    - Windows: “盘符:\\”
    - Windows UNC路径: “\\\\”
  - 名字字符串（零个或多个）
- 路径名可以是绝对的或者相对的
  - 绝对路径名能够唯一确定文件位置
  - 相对路径名必须解释为从其他路径名获得的一部分信息
  - 默认情况下，相对路径名会被解析为相对于当前用户目录
  - 相对路径名没有前缀
- 抽象路径名的parent部分：前缀字符串 + 名字字符串序列中除最后一个名字之外的字符串序列

创建File对象

- public File(String pathname)

```
File myFile = new File(“myfile.txt”);
```

- public File(String parent, String child)

```
File myFile = new File(“d:\works”, “source\myfile.txt”);
```

- public File(File parent, String child)

```
File myFileParent = new File(“d:\works”);
File myFile = new File(myFileParent, “source\myfile.txt”);
```

- 注意:
  - 为保证程序的可移植性，应尽量使用相对路径
  - parent: 父路径字符串或对象，绝对或相对路径
  - child: 相对于parent的目录或文件名字符串

java.io.File主要接口方法

文件名操作接口方法	用法
String getName()	返回当前抽象路径名对应的文件或目录的名字
String getParent()	返回当前抽象路径名的parent部分的路径名字符串
File getParentFile()	将当前抽象路径名的parent部分作为File对象返回
String getPath()	将当前抽象路径名转化为一个路径名字符串
File getAbsoluteFile()	返回当前抽象路径名的绝对形式
String getAbsolutePath()	返回当前抽象路径名的绝对路径名字符串
File getCanonicalFile()	返回当前抽象路径名的规范形式
String getCanonicalPath()	返回当前抽象路径名的规范路径名字符串
boolean renameTo(File dest)	对当前抽象路径名对应文件进行重命名

```
public class FilePathOps{
    public static void main(String[] args){
        File parent=new File(“E:\\Dir1\\Dir2”);
        File absFile=new File(parent, “Dir3\\Dir4\\FileName.java”);
        System.out.println(absFile.getName());
        System.out.println(absFile.getParent());
        System.out.println(absFile.getParentFile().toString());
        System.out.println(absFile.getPath());
        System.out.println(absFile.getAbsolutePath());
        System.out.println(absFile.getAbsoluteFile().toString());

        File relFile=new File(“subdir\\FileName.java”);
        System.out.println(relFile.getName());
        System.out.println(relFile.getParent());
        System.out.println(relFile.getParentFile().toString());
        System.out.println(relFile.getPath());
        System.out.println(relFile.getAbsolutePath());
        System.out.println(relFile.getAbsoluteFile().toString());
    }
}
```

java.io.File主要接口方法

文件信息测试获取接口方法	用法
boolean canExecute()	测试能否执行当前抽象路径名对应的文件
boolean canRead()	测试能否读取当前抽象路径名对应的文件
boolean canWrite()	测试能否修改当前抽象路径名对应的文件
boolean exists()	测试当前抽象路径名对应的文件或目录是否存在
boolean isAbsolute()	测试当前抽象路径名是否为绝对的
boolean isDirectory()	测试当前抽象路径名对应的是否为目录
boolean isFile()	测试当前抽象路径名对应的是否为一般文件
boolean isHidden()	测试当前抽象路径名对应的文件是否是隐藏的
long lastModified()	获得当前抽象路径名对应文件的最后修改时间
long length()	返回当前抽象路径名对应文件的长度

java.io.File主要接口方法

目录操作接口方法	用法
boolean mkdir()	创建由当前抽象路径名对应的目录
boolean mkdirs()	创建当前抽象路径名对应的目录（包括任何必需但尚不存在的parent目录）
String[] list(...)	返回当前抽象路径名对应目录下的所有文件和目录的字符串
File[] listFiles(...)	返回当前抽象路径名对应目录下所有文件对应的File对象组成的数组

文件创建/删除接口方法	用法
boolean createNewFile()	自动地创建一个新的空文件，该文件由当前抽象路径名命名（当且仅当该文件原先不存在）
boolean delete()	删除当前抽象路径名对应的文件或目录
void deleteOnExit()	要求在当前虚拟机终止时，删除当前抽象路径名对应的文件或目录

```
public class RenameFile {
    private static void fileData(File f) {
        System.out.println(“ Absolute path: ” + f.getAbsolutePath());
        System.out.println(“ Can read: ” + f.canRead());
        System.out.println(“ Can write: ” + f.canWrite());
        System.out.println(“ getName: ” + f.getName());
        System.out.println(“ getParent: ” + f.getParent());
        System.out.println(“ getPath: ” + f.getPath());
        System.out.println(“ length: ” + f.length());
        System.out.println(“ lastModified: ” + new Date(f.lastModified()));
        if (f.isFile())
            System.out.println(“It's a file”);
        else
            System.out.println(“It's a directory”);
    }
    public static void main(String[] args) {
        File old = new File(args[0]);   File rname = new File(args[1]);
        System.out.println(“The original file's information: ”);
        fileData(old);
        old.renameTo(rname);
        System.out.println(“\nThe file information after rename: ”);
        fileData(rname);
        if (!old.exists())
            System.out.println(“\nThe original file never exists”);
    }
```

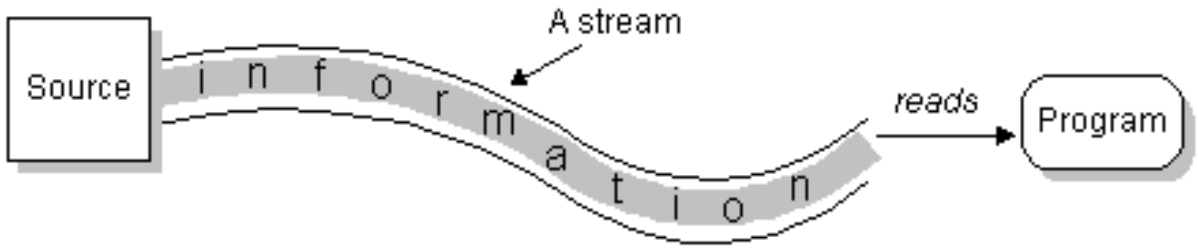
提要

流式输入/输出

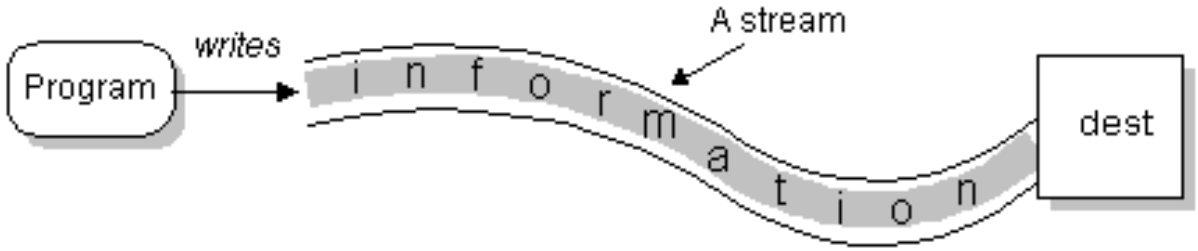
- 流(stream)：能够输出数据的数据源对象，或能够接收数据的接收端对象
- 流中的字节依据先进先出，具有严格的顺序

流式输入/输出

- 流(stream)：能够输出数据的数据源对象，或能够接收数据的接收端对象
- 流中的字节依据先进先出，具有严格的顺序
- 两种流
  - 输入流：从某种数据源（如文件、内存等）到程序的流



  - 输出流：从程序到外界某种目的地（存储介质等）的用于将程序数据顺序写出到外界的流



流式输入/输出

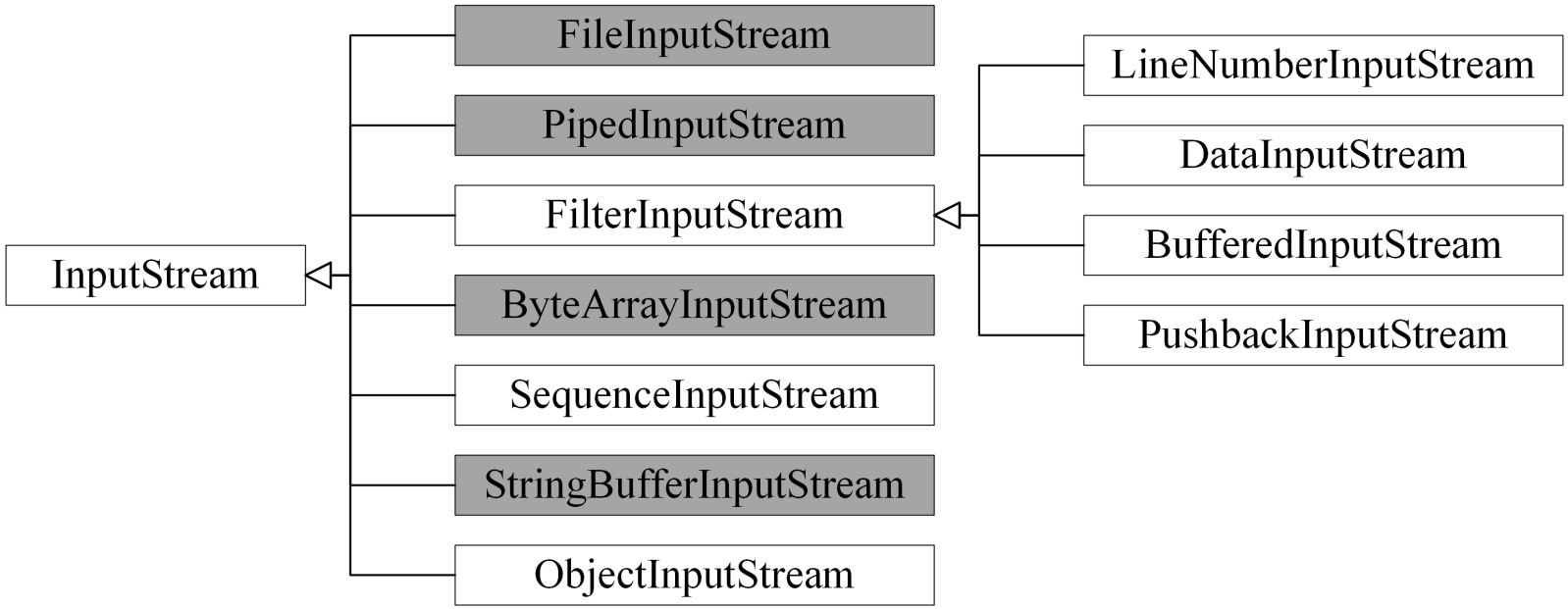
- 由于流是有向的，输入流只能被读，输出流只能被写
  - 读过程（输入流）
    - 打开流→若流中还有数据，执行读操作→关闭流
  - 写过程（输出流）
    - 打开流→当有数据需要输出时执行写操作→关闭流
  - 打开
    - 所有的流在创建时自动打开
  - 关闭
    - 程序可调用close()方法关闭流
    - 否则Java运行环境的垃圾收集器将隐含将流关闭

流的分类

- 依据流相对于程序的另一个端点的不同：
  - 节点流：以特定源（如磁盘文件、内存某区域或线程之间的管道）为端点构造的输入/输出流
  - 过滤流：以其他已存在的流为端点构造的输入/输出流
- 依据流中的数据单位不同：
  - 字节流：流中的数据以8位字节为单位进行读写，以InputStream和OutputStream为共同父类
  - 字符流：流中的数据以16位字符为单位进行读写，以Reader和Writer为共同父类

输入字节流的类层次

- 所有输入字节流都是InputStream的子类
  - 灰色为节点流，其他为过滤流



输入字节流的用法

类	功能与用法
ByteArrayInputStream	允许将内存的缓冲区当作InputStream使用
StringBufferInputStream	将String转换成InputStream（已弃用）
FileInputStream	用于从文件中读取信息
PipedInputStream	产生用于写入相关PipedOutputStream的数据，实现管道化。多线程中的数据源
SequenceInputStream	将两个或多个InputStream对象转换成单一InputStream
FilterInputStream	抽象类，作为某些过滤流的接口

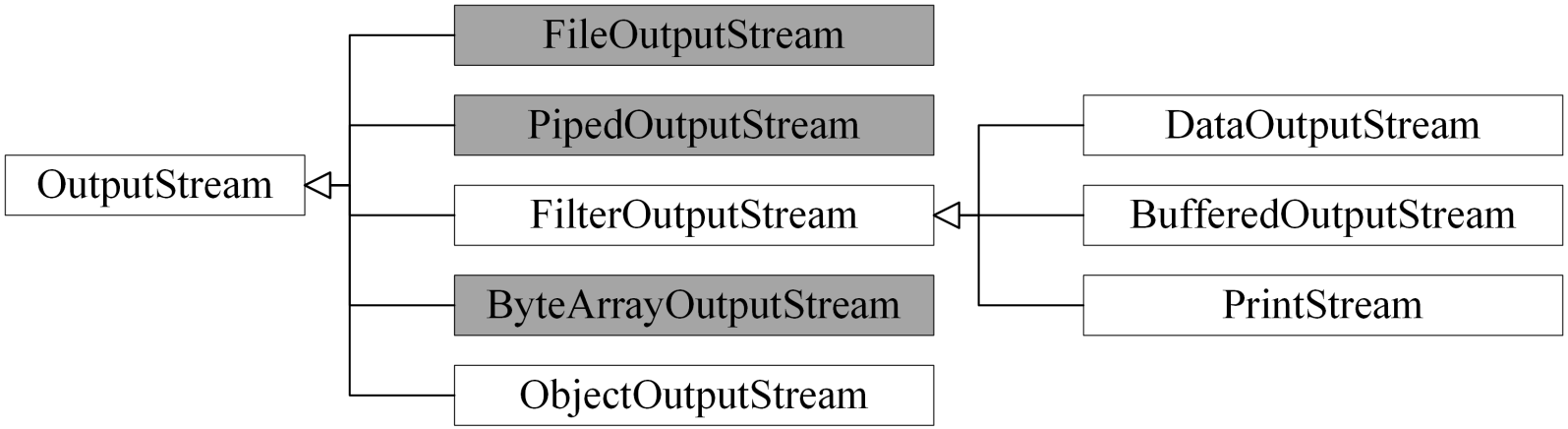
FilterInputStream的子类	功能与用法
DataInputStream	包含用于读取基本类型数据的全部接口
BufferedInputStream	向进程中添加缓冲区，避免每次从流中读取时都进行实际的读操作
LineNumberInputStream	跟踪输入流中的行号，可get/set行号
PushbackInputStream	Java编译器使用，通常作编译器的扫描类

InputStream接口方法

接口方法	功能与用法
int read()	读入一个字节作为方法的返回值，若返回-1，则表示文件结束
int read(byte[] b) int read(byte[] b, int off, int len)	将读入的数据放在一个字节数组中，并返回所读的字节数，两个整型变量用以指定数据在数组中的存放位置
void close()	当输入流中的数据读完时，关闭流
int available()	返回输入流中未读的字节数
long skip(long n)	跳过n个字节
boolean markSupported()	测试打开的流是否支持标记
void mark(int readlimt)	标记流中的当前位置，并建立readlimt大小缓冲区，readlimt指定了将来通过reset()方法能够重复读取的字节数上限
void reset()	返回到mark()方法对流所做的标记处

输出字节流的类层次

- 所有输出字节流都是OutputStream的子类
  - 灰色为节点流，其他为过滤流



输出字节流的用法

类	功能与用法
ByteArrayOutputStream	在内存中创建缓冲区，所有送往“流”的数据都要放置在此缓冲区中
FileOutputStream	用于将信息写入到文件
PipedOutputStream	任何写入其中的信息都会自动作为相关PipedInputStream的输出，实现管道化。多线程中的数据目的地
FilterOutputStream	抽象类，作为某些过滤流的接口

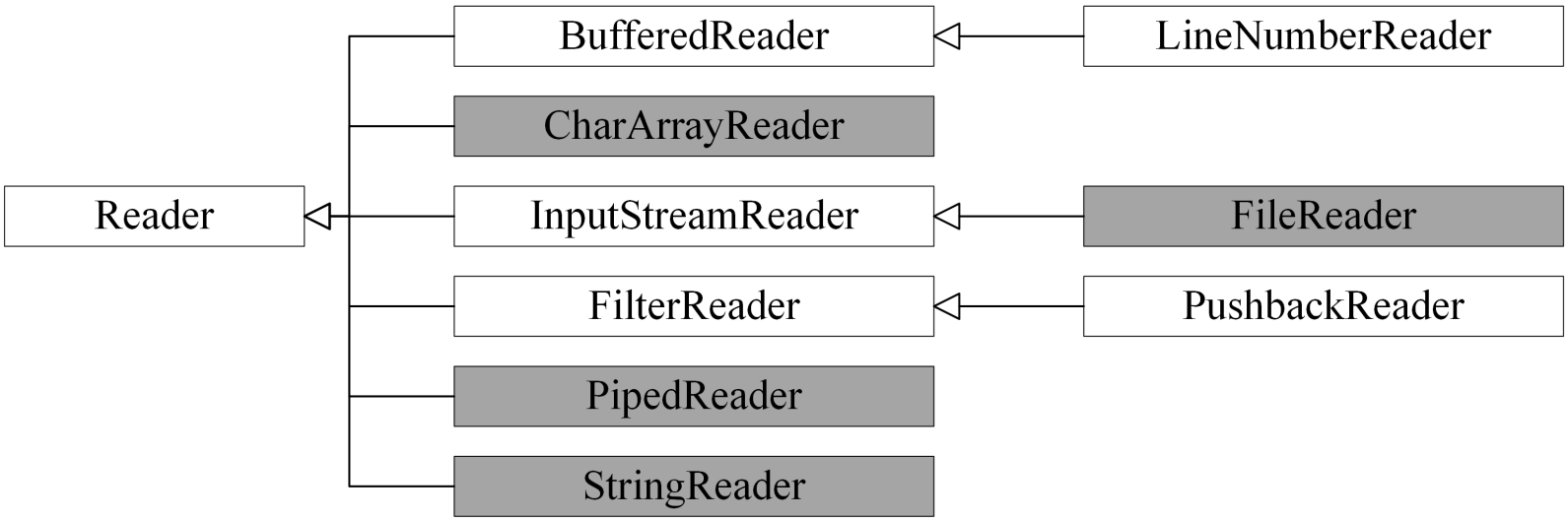
FilterOutputStream的子类	功能与用法
DataOutputStream	包含用于写入基本类型数据的全部接口
BufferedOutputStream	向进程中添加缓冲区，避免每次向流中写入时都进行实际的写操作
PrintStream	用于产生格式化输出，处理数据的显示。 两个重要方法：print()，println()

OutputStream接口方法

接口方法	功能与用法
void write(int)	向输出流写一个字节
void write(byte[] b) void write(byte[] b, int offset, int length)	向输出流写一个字节数组（由offset和length指示的数据块）
void close()	当完成输出流的写操作后关闭流
void flush()	强制将缓存的输出数据写出

输入字符流的类层次

- 所有输入字符流都是Reader类的子类
  - 灰色为节点流，其他为过滤流

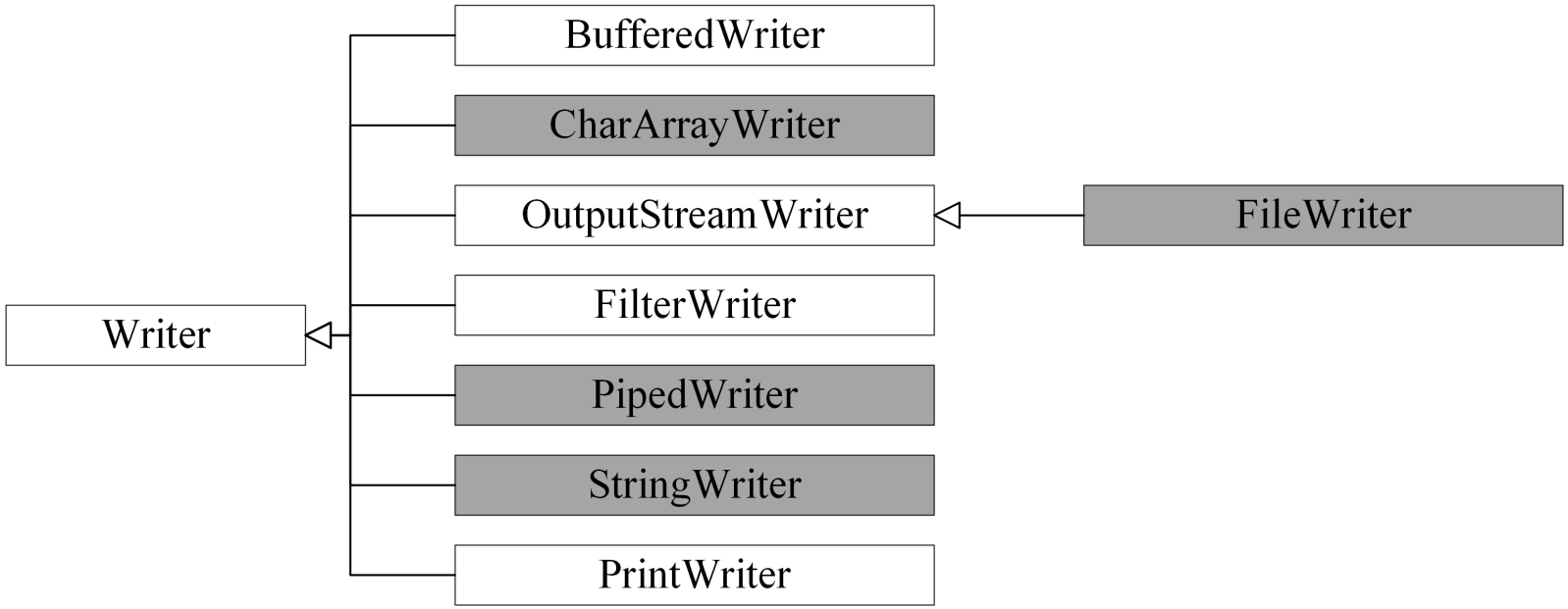


Reader接口方法

接口方法	功能与用法
int read()	读入一个字符作为方法的返回值，若返回-1，则表示文件结束
int read(char[] cbuf) int read(char[] cbuf, int offset, int length)	将读入的数据放在一个字符数组中，并返回所读的字符数，两个整型变量用以指定数据在数组中的存放位置
void close()	当输入流中的数据读完时，关闭流
long skip(long n)	跳过n个字符
boolean markSupported()	测试打开的流是否支持标记
void mark(int readlimt)	标记流中的当前位置，并建立readlimt大小缓冲区，readlimt指定了将来通过reset()方法能够重复读取的字符数上限
void reset()	返回到mark()方法对流所做的标记处
boolean ready()	测试当前流是否准备好进行读

输出字符流的类层次

- 所有输出字符流都是Writer类的子类
  - 灰色为节点流，其他为过滤流



Writer接口方法

接口方法	功能与用法
void write(int c)	向输出流写一个字符
void write(char[] cbuf) void write(char[] cbuf, int offset, int length)	向输出流写一个字符数组（由offset和length指示的数据块）
int write(String str) int write(String str, int offset, int length)	向输出流写一个字符串（由offset和length指示的子串）
void close()	当完成输出流的写操作后关闭流
void flush()	强制将缓存的输出数据写出

Reader和Writer

- Reader和Writer提供对Unicode的完整兼容
- 适配器类
  - InputStreamReader：将InputStream转换为Reader
  - OutputStreamWriter：将OutputStream转换为Writer

输入/输出流的演进（节点流）

Java 1.0类	Java 1.1类
InputStream	Reader 适配器：InputStreamReader
OutputStream	Writer 适配器：OutputStreamWriter
FileInputStream	FileReader
FileOutputStream	FileWriter
StringBufferInputStream（已弃用）	StringReader
（无相应的类）	StringWriter
ByteArrayInputStream	CharArrayReader
ByteArrayOutputStream	CharArrayWriter
PipedInputStream	PipedReader
PipedOutputStream	PipedWriter

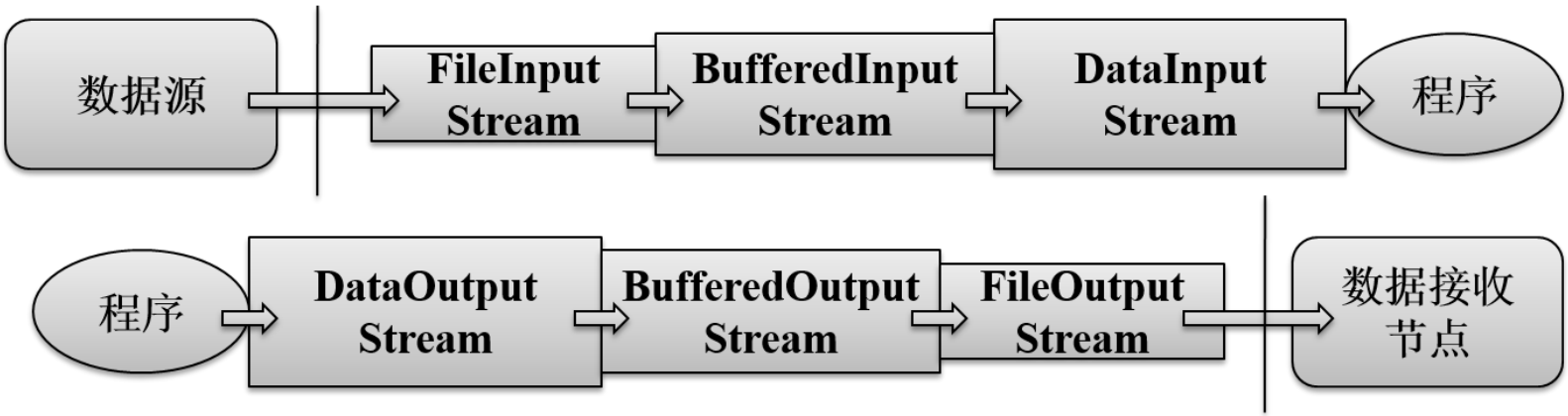
输入/输出流的演进（过滤流）

Java 1.0类	Java 1.1类
FilterInputStream	FilterReader
FilterOutputStream	FilterWriter（抽象类，没有子类）
BufferedInputStream	BufferedReader（有readLine()）
BufferedOutputStream	BufferedWriter
DataInputStream/DataOutputStream	仍使用DataInputStream/DataOutputStream（在需使用readLine()时，使用BufferedReader，除此之外应优先考虑DataInputStream/DataOutputStream）
PrintStream	PrintWriter
LineNumberInputStream(已弃用)	LineNumberReader
PushbackInputStream	PushbackReader

- 注意：
  - 以下Java 1.0类在Java 1.1中没有相应类：DataInputStream, DataOutputStream, File, RandomAccessFile, SequenceInputStream
  - 演进的目的不是替代，而是国际化和提升效率
  - 优先选择新类，新类解决不了则使用旧类

流的嵌套

- 单独使用节点流的情况在实际程序中较少出现
- 一般常通过过滤流将多个流套接在一起，利用各种流的特性共同处理数据，套接的多个流构成了一个流链
- 优点：方便数据的处理并提高处理的效率
- 例：



提要

常用的输入/输出流

- 文件流
- 缓存流
- 数据流
- 标准I/O

文件流

- 文件流包括：
  - FileReader/FileWriter类
  - FileInputStream/FileOutputStream类
- 创建文件流：以文件名或File类型的对象作为参数，例如
  - public FileInputStream(String name)
  - public FileInputStream(File file)

```
public class FileCopy{
    public static void main(String[] args) throws IOException{
        FileInputStream in=new FileInputStream("FileCopy.java");
        FileOutputStream out=new FileOutputStream("FileCopy.txt");
        int c;
        while( (c=in.read())!=-1)
            out.write(c);
        in.close();
        out.close();
    }
}
```

缓存流

- 缓存流包括：
  - BufferedInputStream/BufferedOutputStream类
  - BufferedReader/BufferedWriter类
- 把数据从原始流成块读入或把数据积累到一个大数据块后再成批写出，通过减少系统资源的读写次数来加快程序的执行
- BufferedOutputStream和BufferedWriter仅在缓冲区满或调用flush()时才写数据
- 缓存流是过滤流，必须以其他流为前端流
  - 典型地以InputStream/OutputStream为前端流，并可指定缓冲区大小，如：
    - public BufferedInputStream(InputStream in)
    - public BufferedInputStream(InputStream in, int size)
- 特有方法
  - BufferedReader增加readLine()
  - BufferedWriter增加newLine()
  - 一旦需使用readLine()方法逐行读，就应使用BufferedReader，否则应优先使用DataInputStream

```
public class BufferedIO{
    public static void main(String[] args) throws IOException{
     BufferedReader in=new BufferedReader( new FileReader( “BufferedIO.java” ));
        PrintWriter out=new PrintWriter( new BufferedWriter(
                        new FileWriter( “BufferedIO.txt” ))); //流链
        String s;
        int linecnt=1;
        StringBuilder sb=new StringBuilder();
        while((s=in.readLine())!=null){
            sb.append(linecnt+ “:” +s+ “\n” );
            out.println(linecnt+ “:” +s);
            linecnt++;
        }
        in.close();
        out.close();
        System.out.print(sb.toString());
    }
}
```

数据流

- 数据流包括DataInputStream/DataOutputStream类
- 读写基本数据类型的接口方法：

DataInputStream接口方法	DataOutputStream接口方法
byte readByte()	void writeByte(byte)
(无对应方法)	void writeBytes(String)
boolean readBoolean()	void writeBoolean(boolean)
char readChar()	void writeChar(char)
(无对应方法)	void writeChars(String)
short readShort()	void writeShort(short)
int readInt()	void writeInt (int)
long readLong()	void writeLong(long)
float readFloat()	void writeFloat(float)
double readDouble()	void writeDouble(double)
String readUTF()	void writeUTF(String)

数据流

- readUTF()与writeUTF()
  - writeBytes(String)和writeChars(String)方法在DataInputStream中没有对应的方法恢复出String
  - 用DataOutputStream写字符串并使得DataInputStream能恢复出字符串的方法是使用writeUTF()和readUTF()

```
public class DataIO{
    public static void main(String[] args) throws IOException{
        DataOutputStream out=new DataOutputStream(
                new BufferedOutputStream(new FileOutputStream(“data.txt”)));
        out.writeBoolean(false); out.writeChar( ‘c’ );
        out.writeByte(1); out.writeShort(2);
        out.writeInt(3); out.writeLong(4L);
        out.writeFloat(5.0f); out.writeDouble(6.0);
        out.writeUTF( “hello world!” );
        out.close();

        DataInputStream in=new DataInputStream(
                new BufferedInputStream( new FileInputStream( “data.txt” )));
        System.out.println(in.readBoolean()+ “;” +in.readChar()+ “;” );
        System.out.println(in.readByte()+ “;” +in.readShort()+ “;” );
        System.out.println(+in.readInt()+ “;” +in.readLong());
        System.out.println(in.readFloat()+ “;” +in.readDouble()+ “;” );
        System.out.println(in.readUTF());
        in.close();
    }
}
```

使用DataInputStream的readByte()逐字节读取时，任何字节值均为合法，因而通过返回值难以检测输入是否结束，应使用available()方法检查还有多少可供存取的字节

```
public class DataInputEOF{
    public static void main(String[] args) throws IOException{
        DataInputStream in = new DataInputStream(
                             new BufferedInputStream(
                             new FileInputStream("DataInputEOF.java")));
        while(in.available()!=0)
            System.out.print((char)in.readByte());
        in.close();
    }
}
```

标准I/O

- 标准输入：键盘
- 标准输出：加载Java程序的命令窗口
- Java在System类中定义了三个标准I/O流（是System类的三个静态变量）：
  - System.in：标准输入流
  - System.out：标准输出流
  - System.err：标准错误输出流

标准I/O

- System.in
  - 完整定义：public static final InputStream in
  - 在程序运行时一直打开并准备好提供输入数据
- System.out
  - 完整定义：public static final PrintStream out
  - 在程序运行时一直打开并准备好接收输出的数据
- System.err
  - 完整定义：public static final PrintStream err
  - 在程序运行时一直打开并准备好接收输出的数据
  - 显示错误消息或其他能够马上引起用户注意的信息

标准I/O

- 程序从键盘读入数据：使用System.in的read()方法
  - 如：int ch=System.in.read();
  - System.in.read()的执行使得整个程序被挂起，直到用户从键盘输入数据才继续运行
  - System.in.read()从键盘缓冲区读入一个字节的数据，返回的是整型值（低位字节为输入数据，高位字节全为零）
  - 从键盘逐行读入：嵌套BufferedReader和InputStreamReader
- PrintStream定义了在命令窗口显示不同类型数据的方法print()和println()

```
public class StandardIO {
    public static void main(String[] args) {
        String s;
        BufferedReader in = new BufferedReader(
                            new InputStreamReader(System.in));
        System.out.println("Please input: ");
        try {
            s = in.readLine();
            while (!s.equals("exit")) {
                System.out.println(" read: " + s);
                s = in.readLine();
            }
            System.out.println("End of Inputting");
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

输入输出的重定向

- 输入/输出重定向可分别使用以下命令
  - java [ClassName] < input.txt
  - java [ClassName] < input.txt > output.txt
- 也可使用以下静态方法
  - System.setIn(InputStream)
  - System.setOut(PrintStream)
  - System.setErr(PrintStream)

提要

对象的串行化

- 将对象的状态转换成一个字节序列，这个字节序列能够存储在外存中，能够在网络上传送，并能够在以后被完全恢复为原来的对象
- 对象串行化机制的典型应用场景
  - Java远程方法调用(Remote Method Invocation, RMI)
  - Java Bean / EJB

对象串行化的方法

- 通过ObjectOutputStream的writeObject方法将一个对象写入到流中
  - public final void writeObject(Object obj) throws IOException
- 通过ObjectInputStream的readObject方法将一个对象从对象流中读出
  - public final Object readObject() throws IOException, ClassNotFoundException
- ObjectOutputStream实现了java.io.DataOutput接口
- ObjectInputStream实现了java.io.DataInput接口

输出对象

```
public class SerializeDate {
  public static void main(String args[]) throws IOException{
    Date d = new Date();
    ObjectOutputStream out = new ObjectOutputStream(
                             new FileOutputStream("date.dat"));
    out.writeObject(d);
    out.close();
  }
}
```

输入对象

```
public class UnSerializeDate {
  public static void main(String[] args)
        throws IOException, ClassNotFoundException{
    ObjectInputStream in = new ObjectInputStream(
                           new FileInputStream("date.dat"));
    Object o = in.readObject();
    in.close();
    if(o instanceof Date)
        System.out.println(((Date)o).toString());
  }
}
```

- 一个类只有实现了Serializable接口，其对象才是可串行化的
  - 在指定对象不可串行化时，writeObject将抛出NotSerializableException类型的异常
- 如何构造可串行化对象的类
  - Serializable接口的定义：

```
package java.io;
public interface Serializable{};
```

  - 只需在类声明中implements Serializable，如：

```
public class MySerializableClass implements Serializable { ... }
```

串行化的要求

- 对象串行化时，只有对象的数据被保存，而对象所属类的成员方法和构造方法不保存
- 静态变量和使用transient关键字的变量不被串行化
  - 若被保存对象引用了不可串行化的对象，则该对象引用所对应的成员变量应使用transient关键字，以保证串行化正常进行
  - 保存某些瞬时的状态没有意义，如Thread对象或 FileInput-Stream对象，对于这些对象，必须用transient标明，否则编译器报错