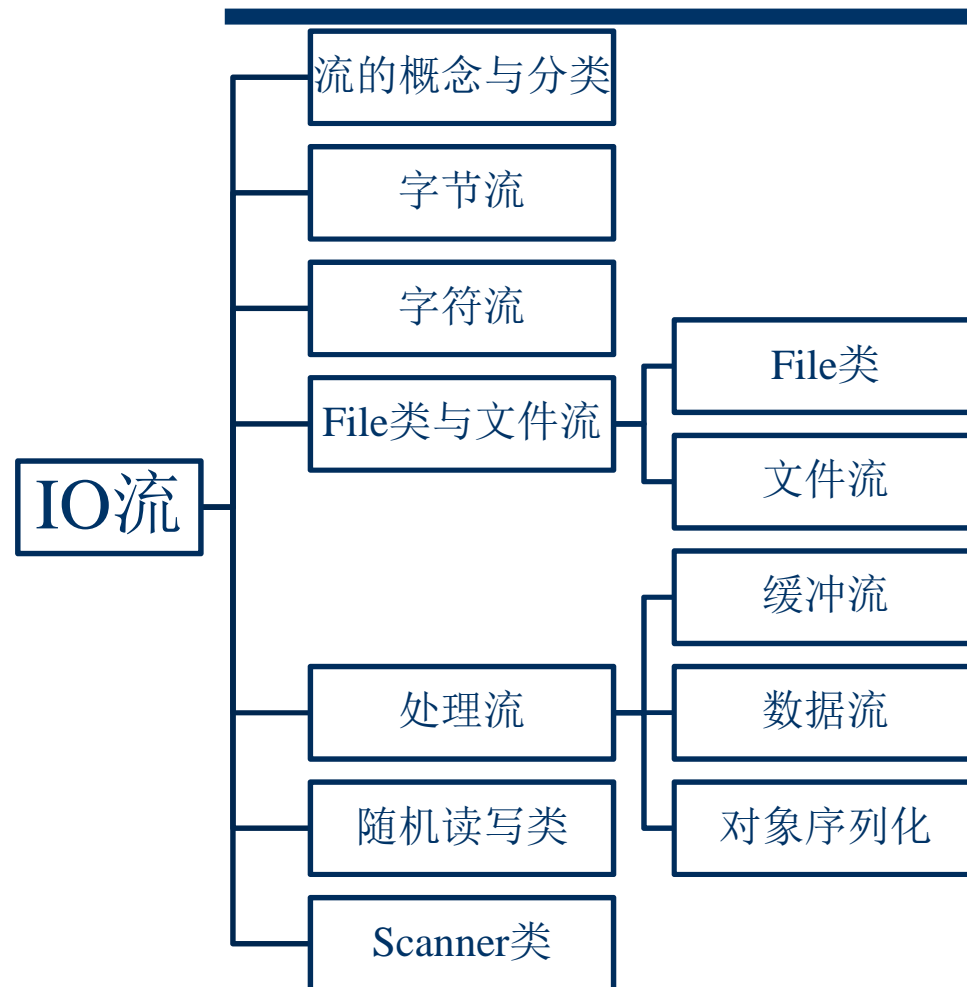




第8章 IO流

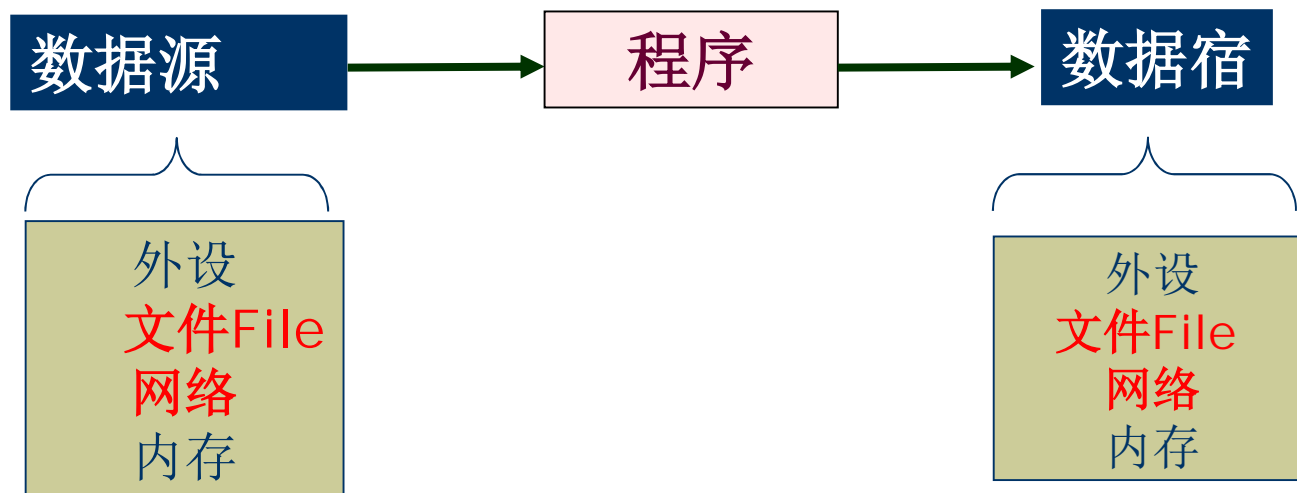
第8章 IO流



8.1 流的概念与分类

1、流的概念

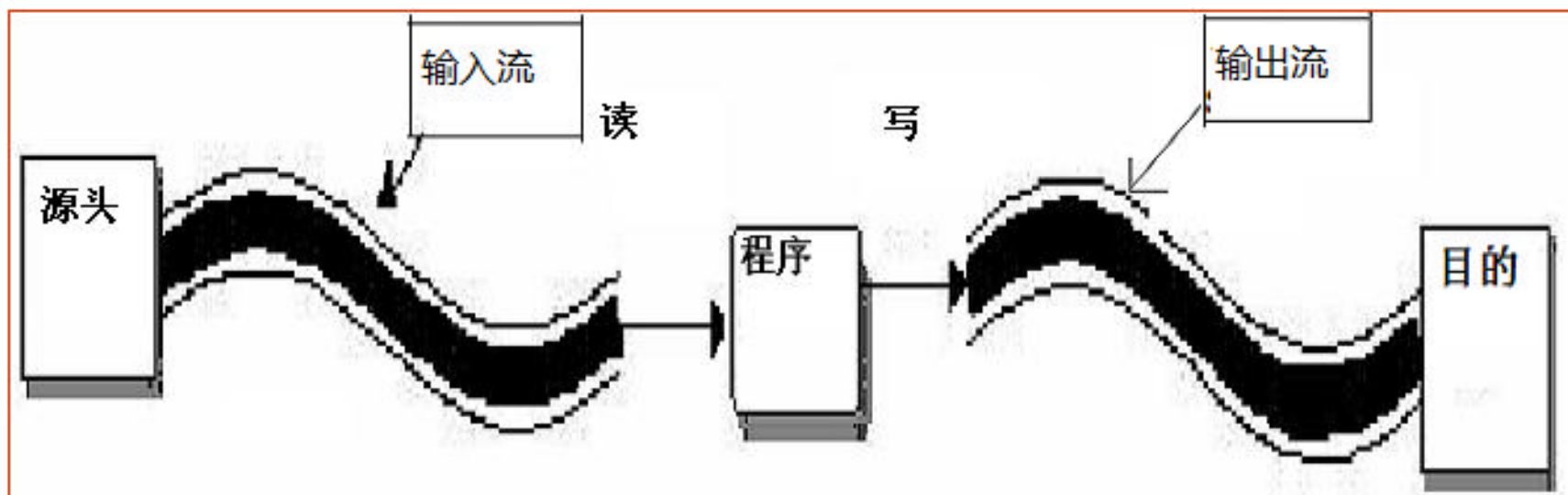
Java用流（stream）封装对数据的读写操作。流（stream）是一个抽象的概念，它提供了对数据序列读/写操作的统一接口，封装了程序与操作系统的I/O细节。



2、流的分类

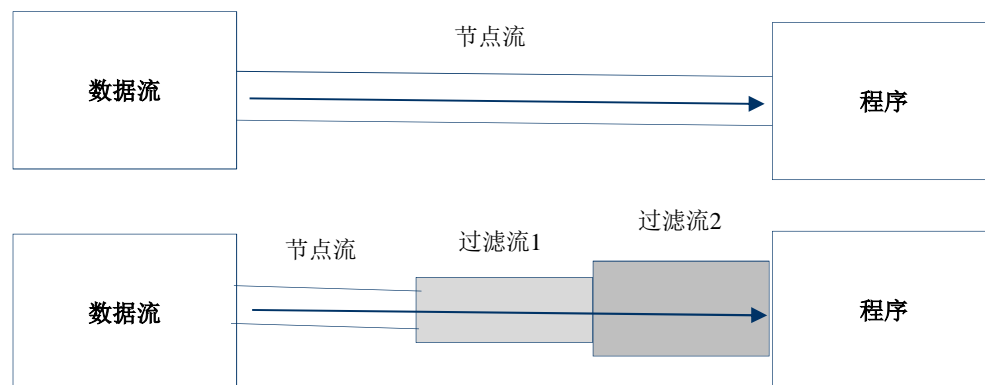
1、根据数据流流动方向的不同，把数据流分为两类：

- (1) 从数据源读数据到程序的数据流称为输入流。
- (2) 将数据从程序写到数据宿的流称为输出流。



2、流的分类

- 2、根据”流”里数据处理的单位，分为
 - ✓ 字符流(Character Streams)
 - ✓ 字节流(Byte Streams)
- 3、根据流的建立方式和功能，分为
 - ✓ 节点流：直接建立在IO媒体之上
 - ✓ 处理流：必须以某一个节点流作为流的来源



2、流的分类

Java语言中的输入/输出类(I/O类)主要定义在java.io包中，尽管I/O包下的类有很多，但它们都直接或间接继承自4个抽象类。

I/O基本类

分类	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

8.2 字节流

字节流以字节为单位来处理流上的数据，其操作的是字节或字节数组。

- 1 InputStream类
- 2 OutputStream类

字节流类

InputStream
字节输入流

StringBufferInputStream

ByteArrayInputStream

FileInputStream

FilterInputStream

PipedInputStream

SequenceInputStream

ObjectInputStream

BufferedInputStream

PushbackInputStream

LineNumberInputStream

DataInputStream

OutputStream
字节输出流

ByteArrayOutputStream
字节数组输出流

FileOutputStream
文件输出流

FilterOutputStream
过滤输出流

PipedOutputStream
管道输出流

ObjectOutputStream
对象输出流 (序列化)

BufferdOutputStream
缓冲输出流

PrintStream
打印流

DataOutputStream
数据输出流

InputStream (抽象类)

- ◆ *public int read()*
- ◆ *public int read(byte b[])*
- ◆ *public int read(byte b[],int off,int len)*
- ◆ *public int available() throws IOException*
- ◆ *public void close() throws IOException*

其他方法（用在包装流类中）

- ❑ **long skip(long n)** //跳过此输入流中数据的 n 个字节。
- ❑ **void mark(int readlimit)** //在此输入流中标记当前的位置
- ❑ **void reset()** //将此流重新定位到最后一次对此输入流调用 mark 方法时的位置。
- ❑ **boolean markSupported()** //测试此输入流是否支持 mark 和 reset 方法。

OutputStream(抽象类)

- ◆ *public void write(int b)* (*b*有大小限制)
- ◆ *public void write(byte b[])*
- ◆ *public void write(byte b[],int off,int len)*
- ◆ *public void flush()*
- ◆ *public void close()*

写完后一定要关闭输出流，数据才真正地写到了文件中



例:

`System.in`: `InputStream` 的一个子类对象,

`System.out`: `PrintStream`的一个子类对象

`System.err`: `PrintStream`的一个子类对象

8.3 字符流

▪ **Reader ,Writer**是所有字符流类的抽象基类，以字符为单位处理流上的数据，其操作的是字符、字符数组或字符串。

- ◆ Reader

- ◆ Writer

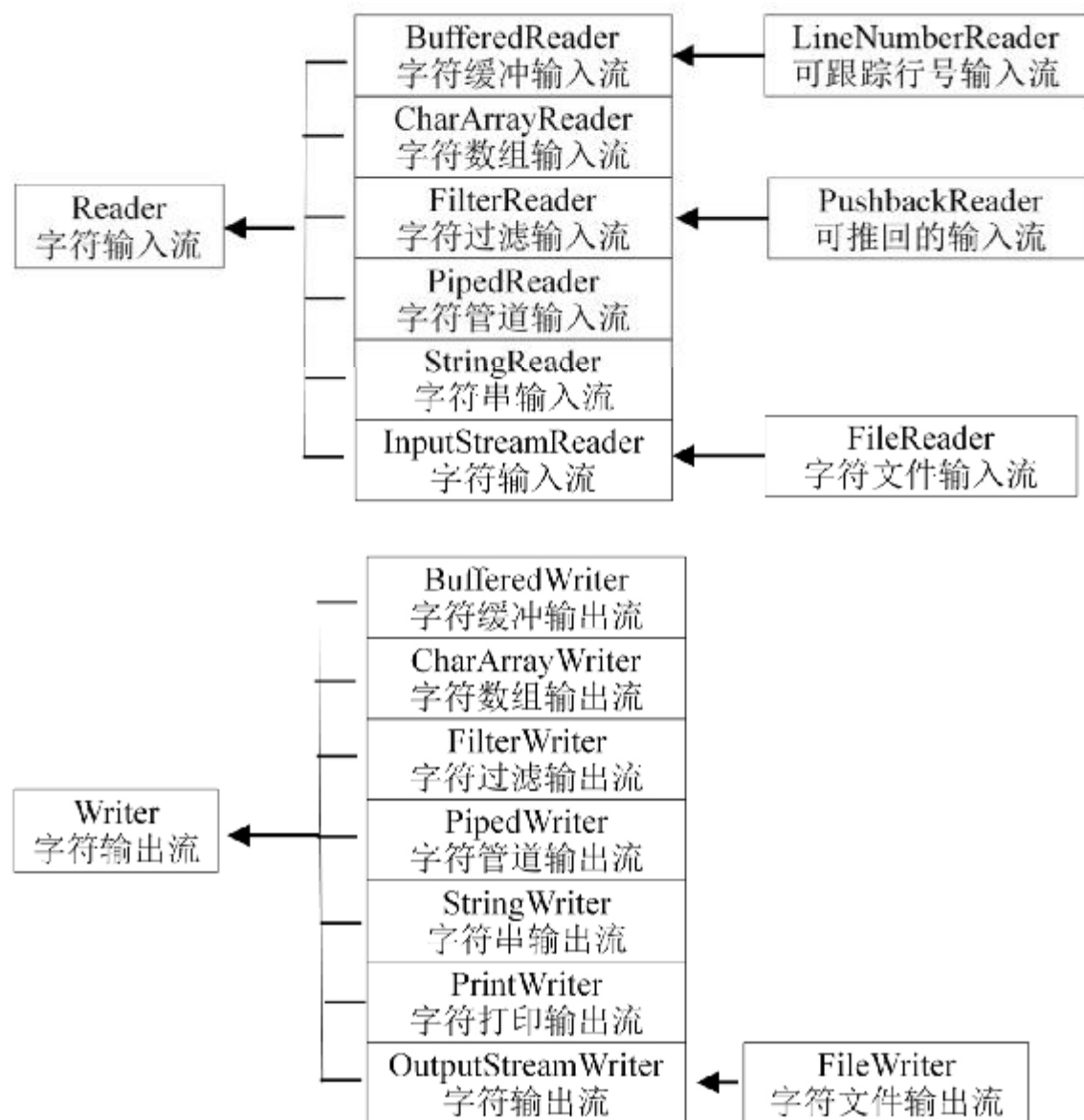


图 8-4 字符输入输出流常用子类

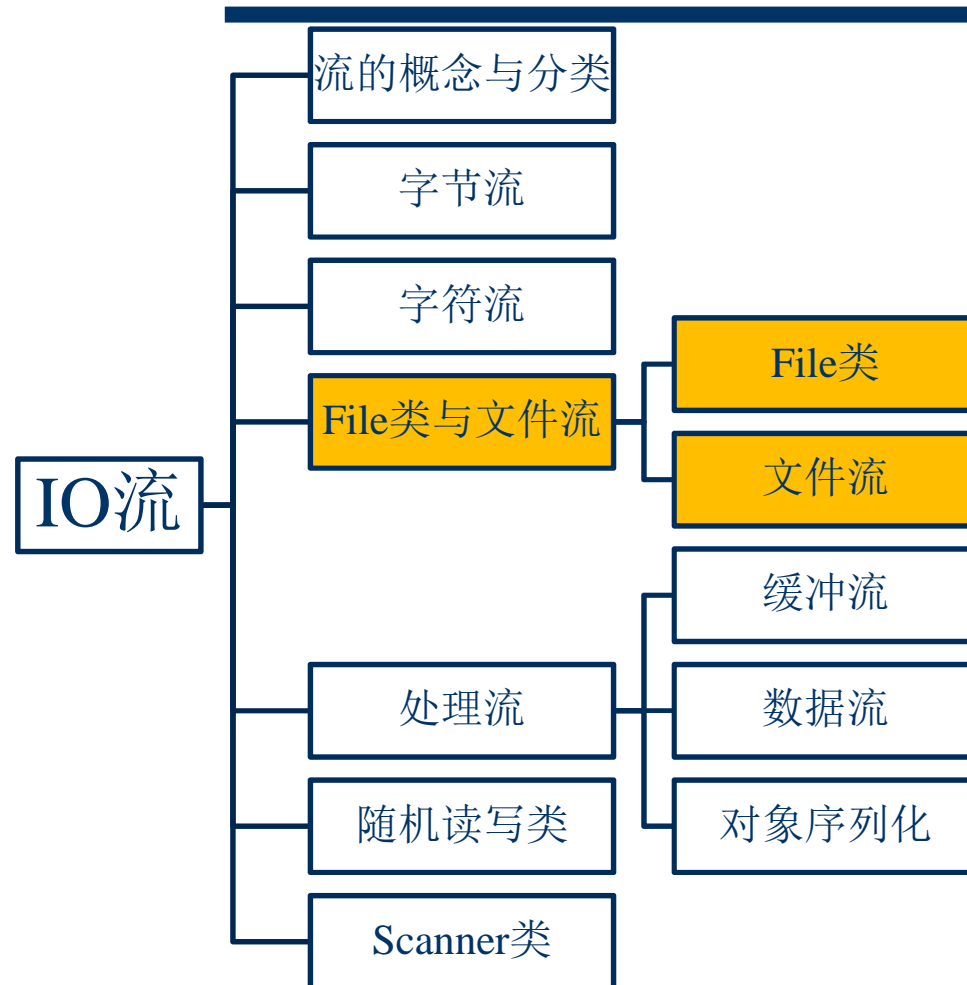
Reader class

- ◆ *public int read()*
- ◆ *public int read(char b[])*
- ◆ *public int read(char[] b, int off, int len)*
- ◆ *public int available()*
- ◆ *public void close()*

Writer class

- ◆ *public void write(int b)*
- ◆ *public void write(char b[])*
- ◆ *public void write(char b[],int off,int len)*
- ◆ *Public void write(String str)*
- ◆ *public void flush()*
- ◆ *public void close()*

第8章 IO流



File (class)

代表磁盘文件本身属性的类，而不是文件内容。提供方法，用于检查，操作主机平台上的文件，目录，但不包括文件读写。

- 构造方法:

File fx = new File("c:\\abc\\myfile.dat")

File fx = new File("c:/abc/myfile.dat")

File (File dir, String name),

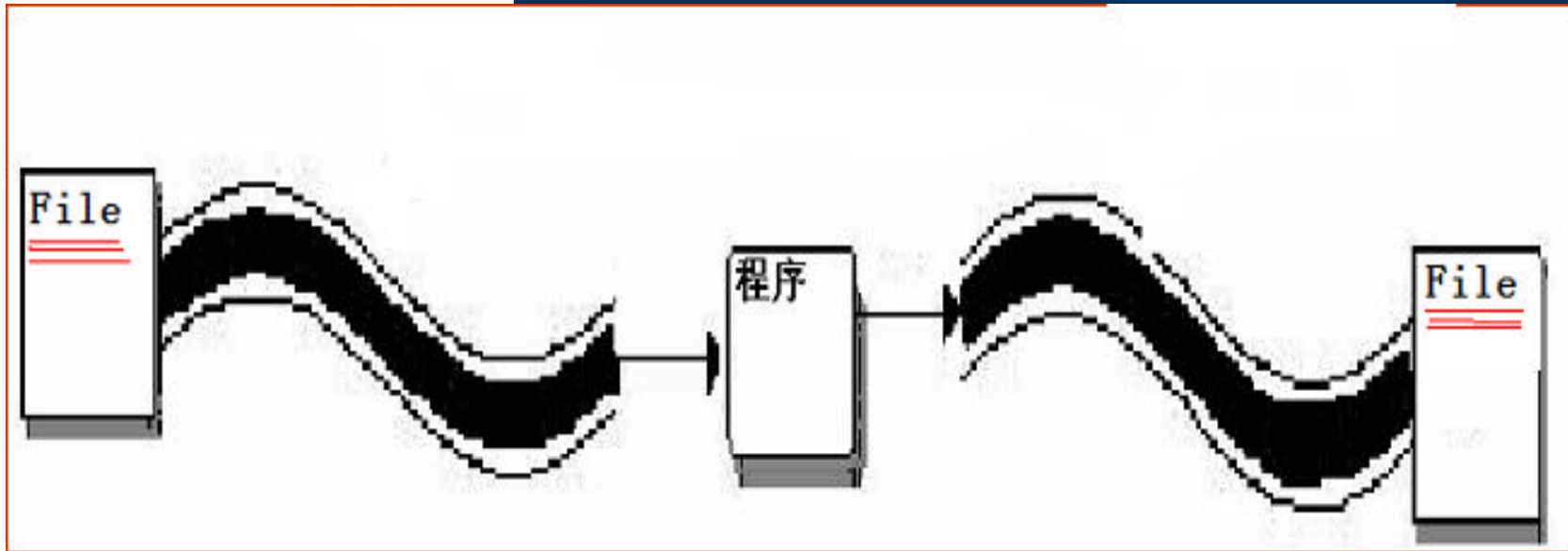
- 方法:

File fx=new File("myfile.dat");

- *String getName(), String getPath(), String getParent()*
- *boolean canRead(), boolean canWrite(), boolean canExecute(), long length()*
- *boolean isDirectory(); boolean isFile(); boolean isHidden(), long lastModified()*
- *boolean createNewFile(), boolean delete(), boolean mkdir()*
- *String[] list() , String[] list(FilenameFilter filter)*
- *boolean renameTo(File dest) , File[] listRoots()*

FileInputStream/FileOutputStream

FileReader&FileWriter



◆ 创建FileInputStream对象的两种方式:

1)FileInputStream inOne=new FileInputStream("he.test");

2)File f=new File("he.test")

FileInputStream inTwo=new FileInputStream(f)

- 创建FileInputStream实例对象时，指定的文件应当是存在和可读的。
- 创建FileOutputStream实例对象时，如果指定的文件已经存在，这个文件中原来内容将被覆盖清除，可以指定还不存在的文件名

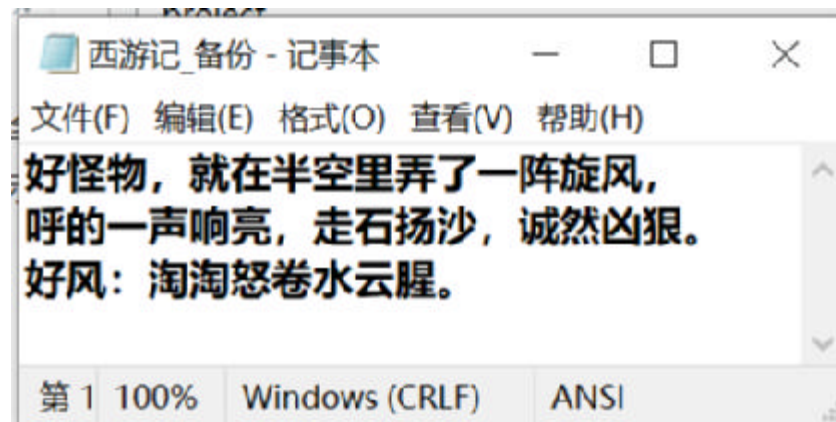
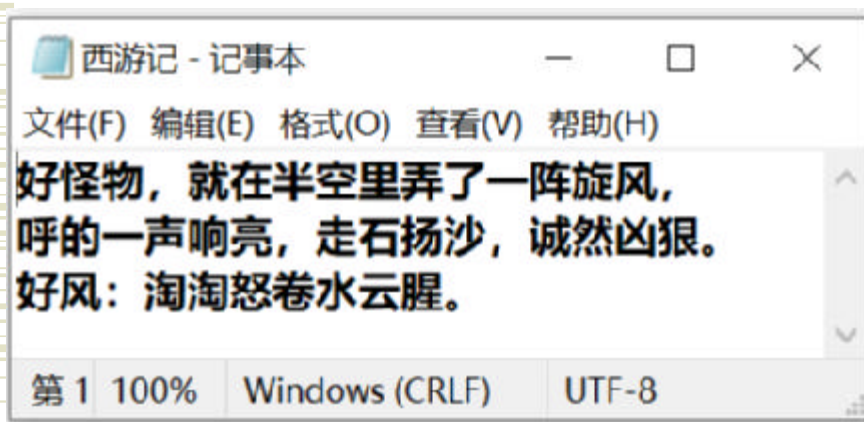
- 
- ◆ 拷贝一个文件的内容到另一个文件中

```
import java.io.*;
public class UseFileIO {
public static void main(String[] args) {
try (FileInputStream fs=new
FileInputStream("UseFileIO.java");FileOutputStream
fo=new FileOutputStream("UseFileIO.java.bak")){
    byte[] b=new byte[fs.available()];
    fs.read(b);
    fo.write(b);
}catch(IOException e){
    System.out.println("无法打开文件");
}}//end class
```

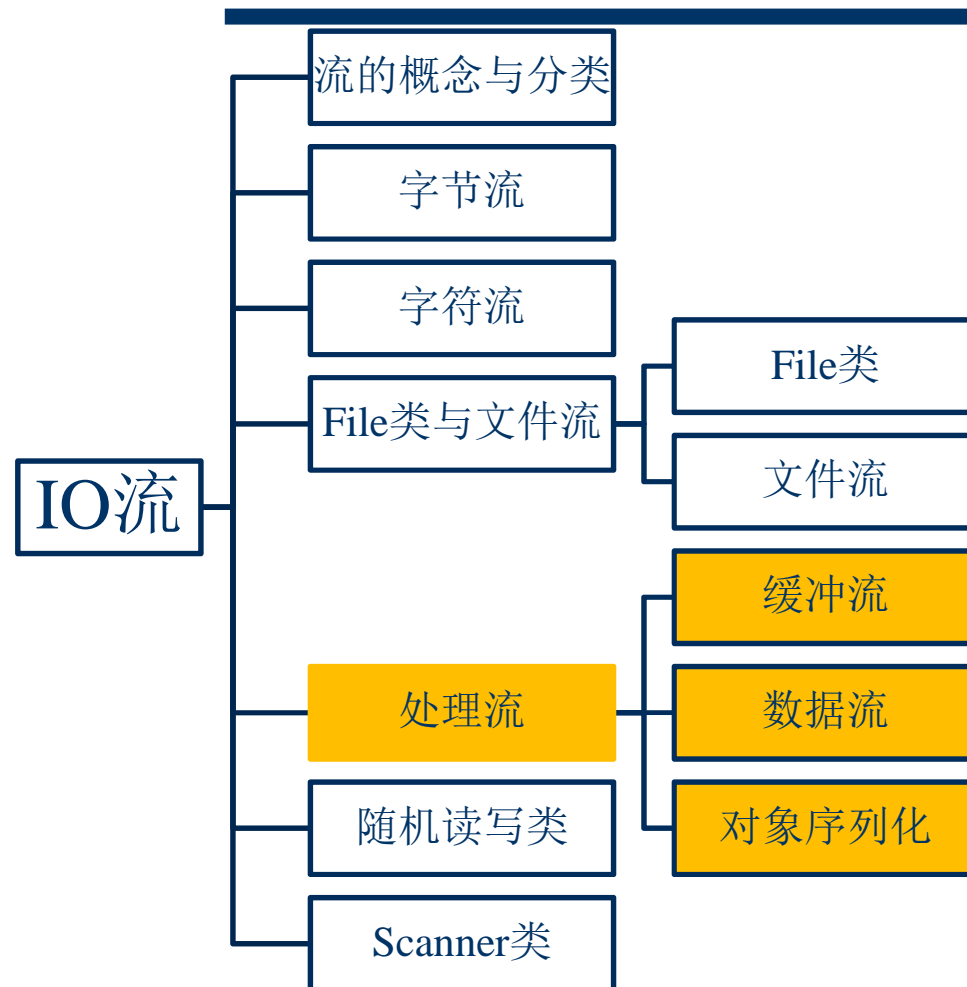
```
import java.io.*;
import java.nio.charset.Charset;
public class UseFileIO {
    public static void main(String[] args) throws IOException {
        long size = 0;
        Charset cs=Charset.forName("UTF-8");
        try(FileReader fr=new FileReader("西游记.txt", cs);
            FileWriter fw = new FileWriter("西游记_备份.txt")) {

            char[] buf = new char[1024];
            int len = 0;
            //读一个数组大小，写一个数组大小方法。
            while((len = fr.read(buf)) != -1) {
                fw.write(buf, 0, len);
                size += len;
            }
            System.out.println("复制完成，共复制了 " + size + " 个字符。");
        } catch(FileNotFoundException e) {
            System.out.println("找不到要复制的文件!");
        } catch(IOException e) {
            System.out.println("复制过程中出现I/O错误!");
        }
    }
}
```

【运行结果】



第8章 IO流



8.5 处理流

- 处理流简化了输入/输出过程中的一些数据处理需求（如缓冲，解析基本类型数据、对象等）。实现了在读/写数据的同时对数据进行处理。
- 处理流不能独立使用，它最终必须连接具体的节点流对象



8.5 处理流

表 8-9 常用的处理流

类型	字节流	字符流
缓冲	BufferedInputStream BufferedInputStream	BufferedReader BufferedWriter
字节字符转换		InputStreamReader OutputStreamWriter
对象序列化	ObjectInputStream ObjectOutputStream	
解析基本数据类型	DataInputStream DataOutputStream	
带行号	LineNumberInputStream	LineNumberReader
可回退	PushbackInputStream	PushbackReader
打印显示	PrintStream	PrintWriter

8.5.1 缓冲流

缓冲流:

-----提高读写速度,

-----有了缓冲区, 在流上执行skip, mark和reset方法都成为可能

- ◆ BufferedInputStream的两个构造函数

BufferedInputStream(InputStream in,[int size])

- ◆ BufferedOutputStream类的两个构造函数

BufferedOutputStream(OutputStream out,[int size])

- ◆ BufferedReader和BufferedWriter类

BufferedReader

read(char[])

readLine可以一次读取一行文本

BufferedWriter

write()

newLine方法可以向字符流中写入换行符

example

- ◆ 1 创建一个文件输入的过滤流

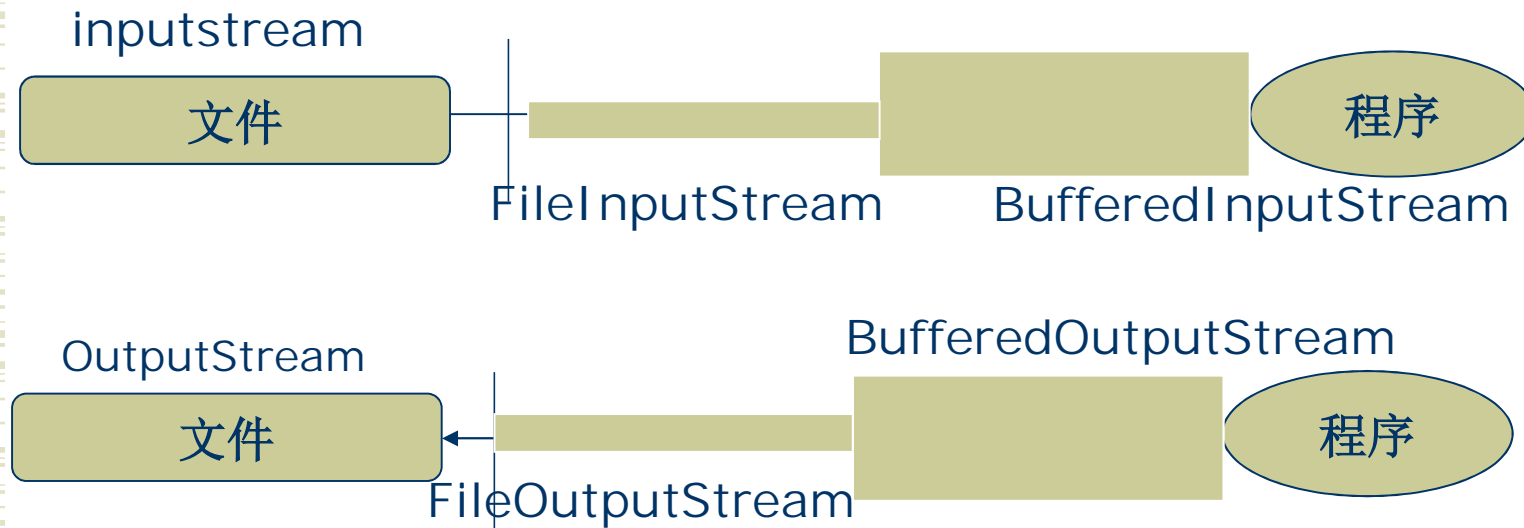
```
FileInputStream fis=new FileInputStream("a.txt");
```

```
BufferedInputStream bis=new BufferedInputStream(fis);
```

- ◆ 2 创建一个文件输出过滤流

```
FileOutputStream fos=new FileOutputStream("b.txt");
```

```
BufferedOutputStream bos=new BufferedOutputStream(fos);
```



```

import java.io.*;
public class UseBuffer {
    public static void main(String[] args) {
        System.out.println("请输入多行文本");
        try(BufferedReader bufR=new BufferedReader(new InputStreamReader(System.in));
            BufferedWriter bos=new BufferedWriter(new FileWriter("a.txt")); ){

            String str=null;
            while((str=bufR.readLine())!=null && str.length()>0) {
                bos.write(str);
                bos.newLine();
            }
            bos.flush();
            System.out.println("输入结束");
        }catch(IOException e){
            System.out.println(e.getMessage());
        }
    }
}

```

请输入多行文本

this is a test

这是一次测试

结束。

输入结束



8.5.2 数据流

- ◆ **DataInputStream**

从节点输入流中读取Java基本数据类型和字符串类型。

- ◆ **DataOutputStream**

用来将Java基本数据类型数据和字符串类型写到数据输出流中。

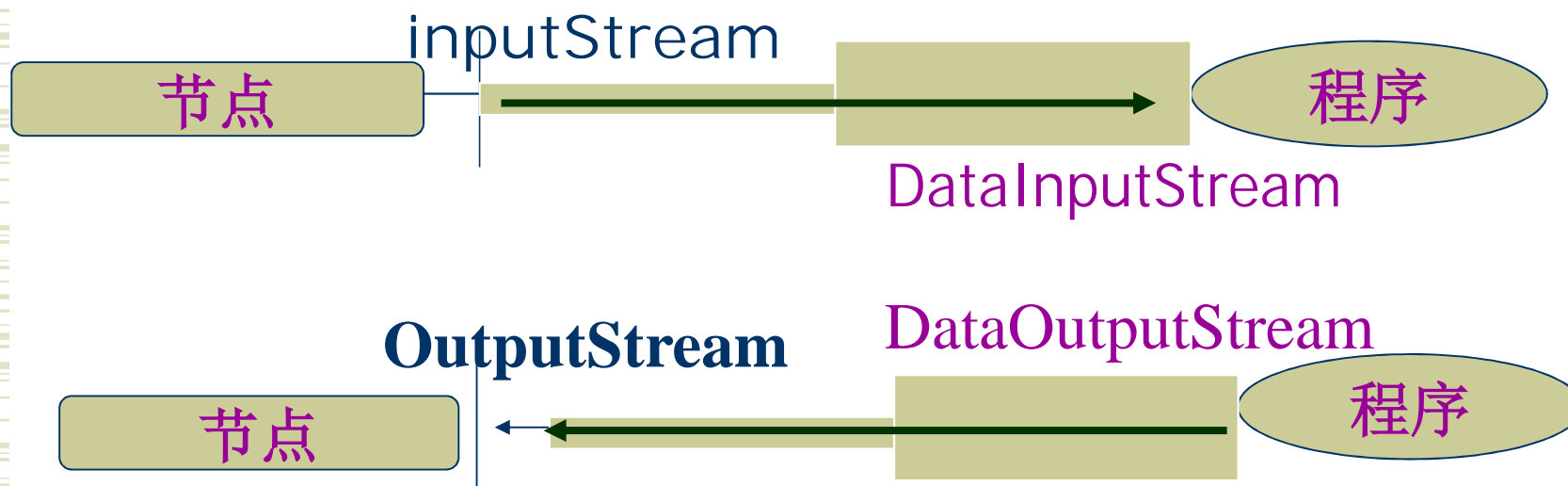
8.5.2 数据流

◆ DataInputStream

从节点输入流中读取Java基本数据类型和字符串类型。

◆ DataOutputStream

将Java基本数据类型数据和字符串类型写到数据输出流中。



例：使用DataOutputStream将数据写入a.data文件中，然后
再用DataInputStream从该文件中读取并显示出来。

```
import java.io.*;
public class DataIOStream {
    public static void main(String arg[]) {
        double r=4.0d;
        String str="this is a test";
        try(DataOutputStream dos=new DataOutputStream(new FileOutputStream("a.data"))
            DataInputStream dis=new DataInputStream(new FileInputStream("a.data")))
        {
            dos.writeDouble(r);
            dos.writeUTF(str);
            dos.close(); //写入文件中。

            double tempR=dis.readDouble();
            String tempStr=dis.readUTF();
            System.out.printf("r=%f, str=%s", tempR, tempStr);

        } catch (EOFException e) {
            System.out.println("已达到数据流的末尾，读取未完成!");
        } catch (IOException e) {
            System.out.println("读取过程中发生了I/O错误!");
            e.printStackTrace();
        }
    }
}
```

【运行结果】

r=4.000000, str=this is a test

8.5.3 对象序列化

- ◆ ObjectInputStream和ObjectOutputStream这两个包装类，用于从底层流读取\写入对象类型的数据
- ◆ ObjectInputStream与ObjectOutputStream类所写的对象必须实现了**Serializable**接口或者**Externalizable**接口。对象中的**transient**和**static**类型成员变量不会被读取和写入。

8.5.3 对象序列化

◆ 1、隐式序列化

(1) 对对象的要求：实现**Serializable** 接口

Serializable是一个空接口，其中并没有定义任何方法，该接口只是为实现了该接口的类做一个允许被序列化的标志。

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

(2) 对象流的操作

ObjectOutputStream提供了**writeObject(Object o)**方法进行序列化，**ObjectInputStream** 提供了**Object readObject()** 方法进行反序列化操作。

```
import java.io.*;
class SePerson implements Serializable { //定义可序列化类型SePerson,
    private static final long serialVersionUID = -4830331550637537793L;
    private transient String pin; //不参与序列化字段
    private static int count=0; //同上
    private String name;
    private int age;
    public SePerson() { }
    public SePerson(String name, String pass, int i) {
        password=pass;
        this.name=name;
        age=i;
        count++;
    }
    public String getInfo() {
        return "name: "+name+" pin: "+pin+" age=: "+age+" count="+count; }
}
```

序列化对象

```
class SeI0 { //并将其序列化存放于文件se.data中
    public static void main(String arg[]) {
        SePerson so=new SePerson("阿sa", "*_3412", 78);
        SePerson so2=new SePerson("张三", "!@7890", 40);
        try( FileOutputStream fos=new FileOutputStream("se.data");
            ObjectOutputStream oos=new ObjectOutputStream(fos)) {

            oos.writeObject(so);
            oos.writeObject(so2);
        }
        catch(IOException e){
            System.out.print(e.getMessage());}
    }}
}
```

反序列化对象

```
class DeseIO { //将存放于文件se.data中数据，在内存中反序列化成对象。  
    public static void main(String arg[]) {  
        try(FileInputStream fis=new FileInputStream("se.data");  
            ObjectInputStream ois=new ObjectInputStream(fis)) {  
  
            SePerson ex= (SePerson)ois.readObject();  
            SePerson ex2= (SePerson)ois.readObject();  
            System.out.println(ex.getInfo());  
            System.out.print(ex2.getInfo());}  
        catch(Exception e2){  
            System.out.print(e2.getMessage());}}}
```

【运行结果】

```
name: 阿sa pin: null age=: 78 count=0  
name: 张三 pin: null age=: 40 count=0
```

8.5.3 对象序列化（显式序列化，自学，自己阅读教材示例）

◆ 2、显式序列化

（1）对对象的要求：实现**Externalizable** 接口

Externalizable继承自**Serializable**，它定义**writeExternal(ObjectOutput out)**和**void readExternal(ObjectInput in)**两个抽象方法，分别用于指定序列化/反序列化哪些属性，从而对序列化的细节加以控制。

```
public class myClass implements Externalizable
{ ..... }
```

（2）对象流的操作

ObjectOutputStream提供了**writeObject(Object o)**方法进行序列化，**ObjectInputStream** 提供了**Object readObject()** 方法进行反序列化操作。

利用Externalizable接口序列化SePerson2对象, 且只序列化它的name属性。

```
import java.io.*;
class SePerson2 implements Externalizable {
    private transient String password;
    private String name;
    private int age;
    public SePerson2() { //必须有一个public的不带参数的构造方法,
    }
    public SePerson2(String name, String pass, int i){
        password=pass;
        this.name=name;
        age=i;
    }
    public String getInfo(){
        return "name: "+name+", password: "+password+", age="+age; }
    @Override
    public void writeExternal (ObjectOutput out) throws IOException {
        out.writeObject(name); //只序列化name
    }
    @Override
    public void readExternal (ObjectInput in) throws IOException,
    ClassNotFoundException {
        name = (String)in.readObject(); //只反序列化name
    }
}
```



```
public class ExternalizeDemo {
    public static void main(String arg[]) {
        SePerson2 so2=new SePerson2("张三", "!@7890", 40);

        try(FileOutputStream fos=new FileOutputStream("se.data");
            ObjectOutputStream oos=new ObjectOutputStream(fos);
            FileInputStream fis=new FileInputStream("se.data");
            ObjectInputStream ois=new ObjectInputStream(fis)){

            oos.writeObject(so2);
            oos.close();
            SePerson2 ex=(SePerson2)ois.readObject();

            System.out.println(ex.getInfo());
        }catch(Exception e2){
            System.out.print(e2.getMessage());}

    }
}
```

【运行结果】

name: 张三, password: null, age=: 0

8.6 随机读写类（参考教材，自己阅读示例）

RandomAccessFile类

- ◆ 当我们需要在程序中对文件内容进行随机读写时，可以使用Java提供的一个更方便的类：RandomAccessFile。
- ◆ RandomAccessFile直接继承Object类，并实现了DataOutput和DataInput接口，因此，RandomAccessFile同时实现读和写功能，并允许自由定位文件指针实现随机读写。
- ◆ 但它只能读写文件，不能读写其他I/O节点。

```
import java.io.*;
public class RandomAccessFileDemo{
    public static void main(String arg[]){
        try{
            RandomAccessFile rf=new RandomAccessFile("1. result", "rw");
            for(int i=0;i<10;i++) rf.writeDouble(i*1.2);
            rf.close();
            rf=new RandomAccessFile("1. result", "rw");
            rf.seek(5*8);
            rf.writeDouble(47.001);
            rf.close();
            rf=new RandomAccessFile("1. result", "r");
            for(int i=0;i<10;i++)
                System.out.println("value "+i+": "+rf.readDouble());
            rf.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

8.7 Scanner类（参考教材，自己阅读示例）

- ◆ 从JDK 5.0版本开始，Java增加了一个专门用于处理数据输入的Scanner类，该类位于java.util包中，用户利用它可以方便地实现各种数据类型的数据输入。
- ◆ Scanner也称为读取器，它支持以较为简单的方式从输入流中获取基本类型的的数据或字符串，同时提供了对带有分隔符（用以分隔多个数据的字符，如空格、逗号等）的文本的解析能力。

Scanner

Scanner的构造函数

- `Scanner(File source)` //生成的值是从指定文件扫描的。
- `Scanner(File source, String charsetName)`
- `Scanner(InputStream source)` //生成值从指定的输入流扫描
- `Scanner(InputStream source, String charsetName)`
- `Scanner(String source)` // 构造一个新的 `Scanner`，它生成的值是从指定字符串扫描的。

Scanner

◆ Scanner的构造函数

- `Scanner(Readable source)` //生成值是从指定源扫描的
- `Scanner(ReadableByteChannel source)` // 生成值从指定信道扫描的。
- `Scanner(ReadableByteChannel source, String charsetName)` // 构造一个新的 Scanner，它生成的值是从指定信道扫描的

使用Scanner解析文件






- ◆ java.util.*
- ◆ 创建Scanner对象，并指向要解析的文件，或字节流。

例1: `File file=new File("hello.java");`

`Scanner sc=new Scanner(file);`

例2: `Scanner sc = new Scanner(System.in);`

`int i = sc.nextInt();`

- 
- 
- 
- 
- 
- ◆ 利用Scanner从data.txt中读入数据，并计算每人的平均成绩，
 - ◆ 记录的格式为：
成绩1 成绩2 成绩3 姓名


```
public class useScanner {
    public static void main(String arg[]) {
        File f=new File("./data.txt");
        int count=0;
        double sum=0;
        String name;
        try(Scanner sc=new Scanner(f)){
            while(sc.hasNext()) {
                int score=0;
                for(int i=0;i<4;i++) {
                    score=sc.nextInt();
                    count++;
                    sum +=score;
                }//end for
                name=sc.next();
                double aver=sum/count;
                System.out.println(name+" 的平均成绩: "+aver);
            }//end while
        }catch(FileNotFoundException e) {
            System.out.println("没有找到文件");
        }catch(InputMismatchException e) {
            e.printStackTrace();
        }
    }
}
```

【运行结果】



(1) 源文件 data.txt

张三 的平均成绩:39.5
李四 的平均成绩:60.25

(2) 运行后的结果图