# 第10天【IO流】

## 主要内容

1. IO体系
2. File类
3. FileInputStream和FileOutputStream
4. FileReader和FileWriter
5. BufferedInputStream和BufferedOutputStream
6. BufferedReader和BufferedWriter
7. DataInputStream和DataOutputStream
8. ObjectInputStream和ObjectOutputStream
9. 序列化和反序列化
10. 其他流
11. 复制文件夹

## 学习目标

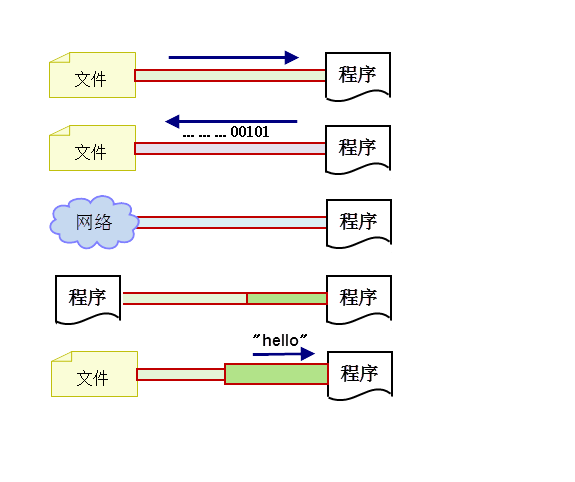
|  |  |
| --- | --- |
| 知识点 | 要求 |
| IO流体系 | 理解 |
| File | 理解 |
| FileInputStream | 掌握 |
| FileOutputStream | 掌握 |
| FileReader | 了解 |
| FileWriter | 了解 |
| BufferedInputStream | 掌握 |
| BufferedOutputStream | 掌握 |
| BufferedReader | 理解 |
| BufferedWriter | 理解 |
| DataInputStream | 了解 |
| DataOutputStream | 了解 |
| ObjectInputStream | 了解 |
| ObjectOutputStream | 了解 |
| 序列化和反序列化 | 理解 |
| 其他流 | 了解 |
| 复制文件夹 | 理解 |

## 一、IO流概述

### 1.1 IO流概述 Input Output Stream

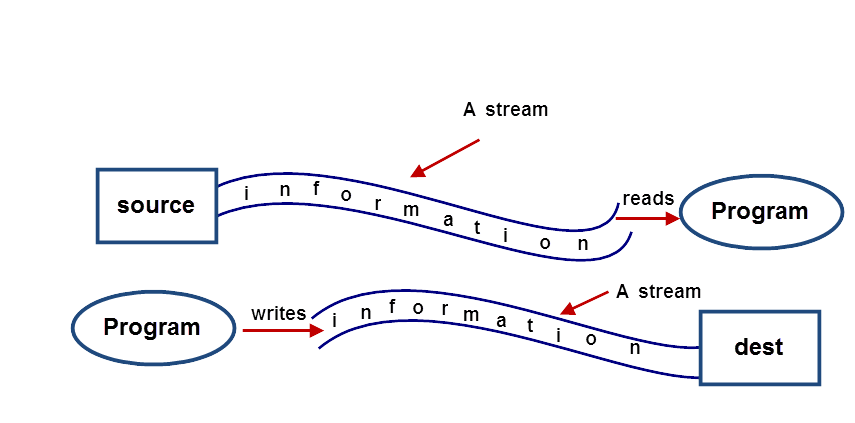
在Java程序中，对于数据的输入/输出操作以“流” (stream) 方式进行；Java提供了各种各样的“流”类，用以获取不同种类的数据；程序中通过标准的方法输入或输出数据。Java的流类型一般位于java.io包中。

数据源data source，提供原始数据的原始媒介。常见的：数据库、文件、其他程序、内存、网络连接、IO设备。



流是一个抽象、动态的概念，是一连串连续动态的数据集合。

数据源就像水箱，流就像水管中流着的水流，程序就是我们最终的用户。 流是一个抽象、动态的概念，是一连串连续动态的数据集合。



### 1.2 IO流分类

**按流的方向分类：**

* + - 输入流：数据流向是数据源到程序（以InputStream、Reader结尾的流）。
    - 输出流：数据流向是程序到目的地（以OutPutStream、Writer结尾的流）。

|  |  |
| --- | --- |
| 图14.6　文件输入输出原理图.jpg   |  | | --- | | **菜鸟雷区**  输入/输出流的划分是相对程序而言的，并不是相对数据源。 | |

**按处理的数据单元分类：**

* + - 字节流：以字节为单位获取数据，命名上以Stream结尾的流一般是字节流，顶级类InputStream、OutputStream。
    - 字符流：以字符为单位获取数据，命名上以Reader/Writer结尾的流一般是字符流，顶级类Reader、Writer。

**按处理对象不同分类：**

* + - 节点流：可以直接从数据源或目的地读写数据，如FileInputStream、FileReader等。
    - 处理流：不直接连接到数据源或目的地，是”处理流的流”。通过对其他流的处理提高程序的性能，如BufferedInputStream、BufferedReader等。处理流也叫包装流。

节点流处于IO操作的第一线，所有操作必须通过它们进行；处理流可以对节点流进行包装，提高性能或提高程序的灵活性。

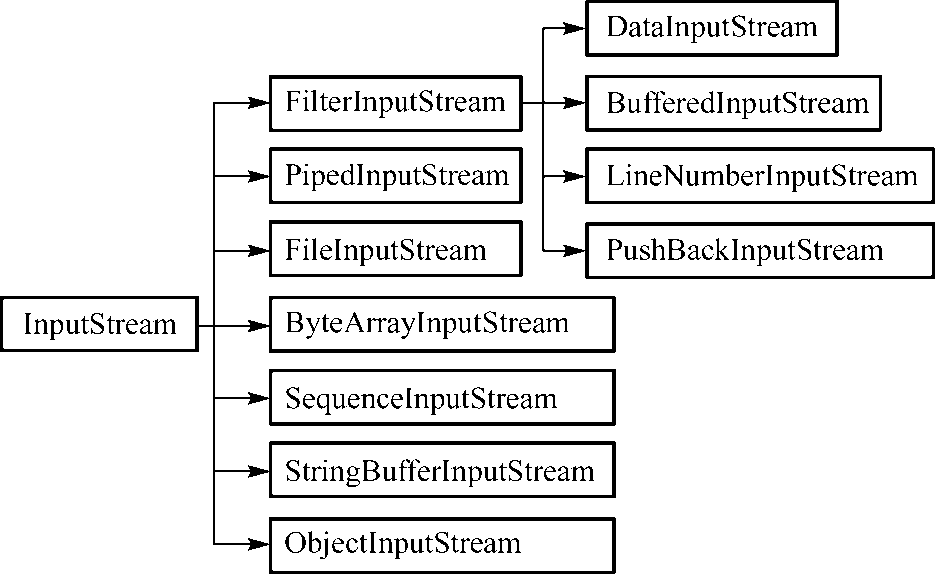
生活案例：节点流就好比水井、水库，处理流就好比自来水厂、水塔等，目的是为了过滤和方便。但是如果没有水井、水库，只有自来水厂和水塔肯定不行。

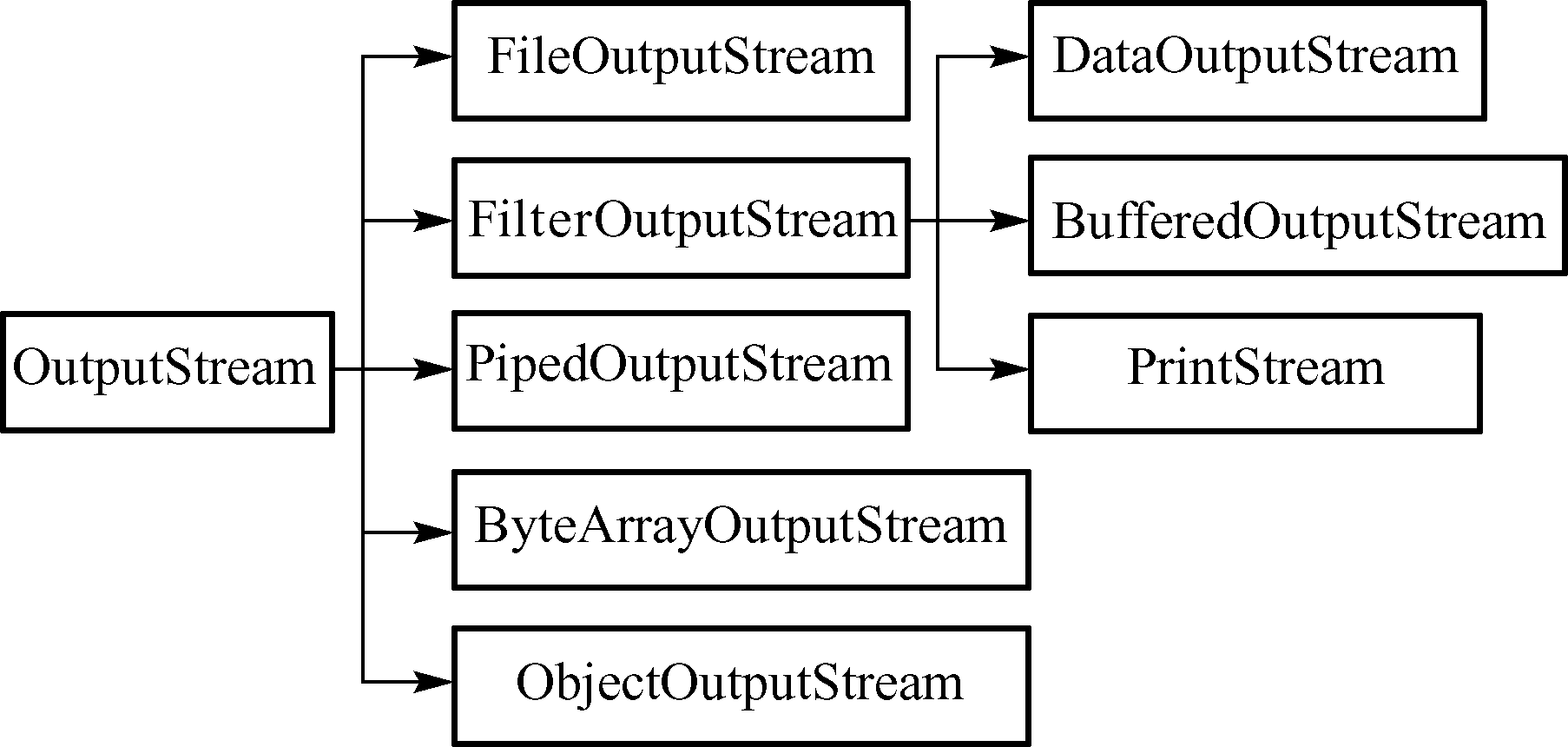
|  |
| --- |
| 10-6 |

### 1.3 IO流体系结构

**字节流**

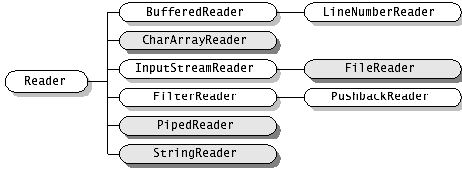
* InputStream和OutputStream是Java语言中最基本的两个字节输入输出类。其他所有字节输入输出流类都继承自这两个基类。
* 这两个类都是抽象类，不能创建它们的实例，只能使用它们的子类.
* FilterInputStream和FilterOutputStream是所有包装流的父类

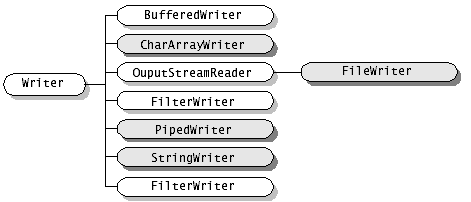




**Reader和Writer**

* Java语言中最基本的两个字符输入输出类。
* 其他所有字符输入输出流类都继承自这两个基类。
* 这两个类都是抽象类，不能创建它们的实例，只能使用它们的子类.





### 1.4 File类的使用

File类用来代表文件和文件夹。主要作用有两个：获取文件或者文件夹的属性； 实现对文件、文件夹的创建和删除。 文件夹：file folder 目录：directory

#### 【示例1】使用File获取文件或文件夹属性

|  |
| --- |
| **public class** TestFile1 {  **public static void** main(String[] args) {  //1.创建一个File对象，指向一个文件或者文件夹  //File file = new File("c:\\bjsxt\\readme.txt");  //File file = new File("c:/bjsxt/readme.txt");  //File file = new File("readme.txt");  File file = **new** File(**"c:/"**);  //2.使用这个File对象，获取文件或者文件夹的属性  System.**out**.println(file.getName());//文件或者文件夹的名称  System.**out**.println(file.length());//长度  System.**out**.println(file.exists());//是否存在  System.**out**.println(file.getPath());//  System.**out**.println(file.getAbsolutePath());//绝对路径  System.**out**.println(file.isDirectory());//判断file是否指向一个目录  System.**out**.println(file.isFile());//判断file是否指向一个文件   System.**out**.println(file.isHidden());  System.**out**.println(file.canWrite());  System.**out**.println(file.canRead());  System.**out**.println(file.canExecute());   //String [] fileNameArr = file.list();  //某个文件夹下有哪些子文件夹和文件  System.**out**.println(**"============"**);  File fileArr[] = file.listFiles();  **for**(File f :fileArr){  //System.out.println(f.toString());  System.**out**.print(**new** Date(f.lastModified()).toLocaleString());  **if**(f.isFile()){  System.**out**.print(**" "**+f.length()+**" "**);  }**else**{  System.**out**.print(**" <DIR> "**);  }  System.**out**.println(f.getName());  }  } } |

#### 【示例2】使用File类新建、删除文件和文件夹

|  |
| --- |
| **public class** TestFile2 {  **public static void** main(String[] args) {  //创建一个File对象  //File file = new File("c:/bjsxt/readme.txt");  File file = **new** File(**"c:/bjsxt/abc/cba/acb/bac/readme.txt"**);  //如果文件存在就删除，如果不存在就创建  **if**(file.exists()){  file.delete();  }**else**{  **try** {  //判断所在文件夹是否存在，不存在，要先创建文件夹  File dir = file.getParentFile();  **if**(!dir.exists()){  //dir.mkdir();//make directory  dir.mkdirs();  }  //创建文件  file.createNewFile();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } } |

|  |
| --- |
| **注意**   * File不仅可以指向一个文件，也可以指向一个文件夹（作为一个文件对待） * File不能对文件的内容进行操作，需要借助IO流实现 |

### 本节作业

1. 流有哪些不同的分类。
2. 流的体系结构
3. 使用File类获取文件和文件夹的属性
4. 使用File类新建删除文件和文件夹

## 二、文件流

### 2.1 文件字节流 FileInputStream和FileOutputStream

* FileInputStream和FileOutputStream是字节流，是节点流，数据源和目的地是文件。
* 复制文件需要分别创建一个输入流和输出流完成文件读写
* 需要创建一个中转站，借助循环和中转站完成复制
* 流使用完毕一定要关闭，这和垃圾回收没有关系

#### 【示例3】复制文件（中转站是一个字节）

|  |
| --- |
| **public class** TestFileStream {  **public static void** main(String[] args) **throws** IOException {  //1.创建流  File file1 = **new** File(**"e:/readme.txt"**);  File file2 = **new** File(**"e:/readme2.txt"**);  InputStream fis = **new** FileInputStream(file1);  OutputStream fos = **new** FileOutputStream(file2);  //2.使用流  //2.1 准备一个中转站(一个字节)  **int** n;  //2.2 读取一个字节到中转站  n = fis.read();  **while**(n!=-1){//读到了文件的末尾  //2.3 写一个字节到目的文件  fos.write(n);  //2.4 再读一个  n = fis.read();  }  //3.关闭流  fis.close();  fos.close();  } } |

* 缺点：中转站太小，速度慢，效率低；复制更大文件时效果更明显；可以将中转站由一个字节变为一个字节数组，减少读写硬盘的次数。
* 问题：如果不是覆盖文件，而是追加内容，如何实现。

#### 【示例4】复制文件（中转站是一个字节数组）

|  |
| --- |
| **public class** TestFileStream2 {  **public static void** main(String[] args) **throws** IOException {  //1.创建流 // File file1 = new File("e:/readme.txt"); // File file2 = new File("e:/readme2.txt"); // InputStream fis = new FileInputStream(file1); // //OutputStream fos = new FileOutputStream(file2);//默认是覆盖文件 // OutputStream fos = new FileOutputStream(file2,true);//这是追加内容 // InputStream fis = new FileInputStream(new File("e:/readme.txt")); // OutputStream fos = new FileOutputStream(new File("e:/readme2.txt"),true);  InputStream fis = **new** FileInputStream(**"e:/readme.txt"**);  OutputStream fos = **new** FileOutputStream(**"e:/readme2.txt"**,**true**);  //2.使用流  //2.1 准备一个中转站(一个字节数组)  **byte** [] buf = **new byte**[1024];  //2.2 读取一些字节到中转站  **int** len = fis.read(buf);//读取文件的数据到buf数组，返回真实读取的字节个数赋给len  **while**(len!=-1){//读到了文件的末尾  //2.3 写一个字节字节数组到目的文件  //fos.write(buf);  fos.write(buf,0,len);  //2.4 再读一些字节到字节数组  len = fis.read(buf);  }  //3.关闭流  fis.close();  fos.close();  } } |

#### 【示例5】进行异常处理

|  |
| --- |
| **public class** TestFileStream3 {  **public static void** main(String[] args) {  InputStream fis = **null**;  OutputStream fos = **null**;  **try**{  //1.创建流  fis = **new** FileInputStream(**"e:/readme.txt"**);  fos = **new** FileOutputStream(**"e:/readme2.txt"**);  //2.使用流  //2.1 准备一个中转站(一个字节数组)  **byte** [] buf = **new byte**[1024];  //2.2 读取一些字节到中转站  **int** len = fis.read(buf);  **while**(len!=-1){//读到了文件的末尾  //2.3 写一个字节字节数组到目的文件  //fos.write(buf);  fos.write(buf,0,len);  //2.4 再读一些字节到字节数组  len = fis.read(buf);  }  }**catch** (FileNotFoundException e){  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  //3.关闭流  **try** {  **if**(fis!= **null**){  fis.close();  }  } **catch** (IOException e) {  e.printStackTrace();  }  **try** {  **if**(fos!= **null**){  fos.close();  }  } **catch** (IOException e) {  e.printStackTrace();  }  }  } } |

* 异常处理的分析：创建流、使用流要使用一次try-catch语句，关闭流要分开进行异常处理
* Java7异常处理新特征：try-with-resources：不用显示的进行资源的关闭，只要将资源的实例化对象放入try后面的()中，作用范围是当前try语句，执行完毕（正常完成或是发生意外）后就会自动进行关闭，可省略finally语句，更加简单实用。

#### 【示例6】JDK7异常处理新特征

|  |
| --- |
| **public class** TestFileStream4 {  **public static void** main(String[] args) {  **try**(InputStream fis = **new** FileInputStream(**"e:/readme.txt"**);  OutputStream fos = **new** FileOutputStream(**"e:/readme2.txt"**)){  //1.创建流  //2.使用流  //2.1 准备一个中转站(一个字节数组)  **byte** [] buf = **new byte**[1024];  //2.2 读取一些字节到中转站  **int** len = fis.read(buf);  **while**(len!=-1){//读到了文件的末尾  //2.3 写一个字节字节数组到目的文件  //fos.write(buf);  fos.write(buf,0,len);  //2.4 再读一些字节到字节数组  len = fis.read(buf);  }  }**catch** (FileNotFoundException e){  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } } |

* 关于Java9中异常处理的新变化：try 之前定义好对象，try () 括号中引入创建好的对象。如果多个对象，使用；分隔。如果try之前定义的对象会抛出异常，就不推荐使用该方式，因为需要在方法签名中throws异常。

#### 【示例7】JDK9异常处理新特征

|  |
| --- |
| **public class** TestFileStream5 {  **public static void** main(String[] args) **throws** IOException {  //1.创建流  InputStream fis = **new** FileInputStream(**"e:/readme.txt"**);;  OutputStream fos = **new** FileOutputStream(**"e:/readme2.txt"**);  **try**(fis;fos){  //2.使用流  //2.1 准备一个中转站(一个字节数组)  **byte** [] buf = **new byte**[1024];  //2.2 读取一些字节到中转站  **int** len = fis.read(buf);  **while**(len!=-1){//读到了文件的末尾  //2.3 写一个字节字节数组到目的文件  //fos.write(buf);  fos.write(buf,0,len);  //2.4 再读一些字节到字节数组  len = fis.read(buf);  }  }**catch** (FileNotFoundException e){  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } } |

### 2.2 文件字符流FileReader和FileWriter

* FileReader和FileWriter是字符流，节点流，数据源和目的地是文件。

#### 【示例8】复制文件（中转站是一个字符）

|  |
| --- |
| **public class** TestFileReaderWriter {  **public static void** main(String[] args) **throws** IOException {  //1.创建字符流  Reader fr = **new** FileReader(**new** File(**"e:/readme.txt"**));  Writer fw = **new** FileWriter(**"e:/readme2.txt"**);  //2.使用字符流  /\*  int n = fr.read();//一次读一个字符，不是一个字节。一个汉字一次搞定  while(n!= -1){  //System.out.println((char)n);  fw.write(n);  n = fr.read();  }  \*/  **int** n=0;  **while**((n = fr.read())!=-1){  fw.write(n);  }   //3.关闭字符流  fr.close();  fw.close();  } } |

#### 【示例9】复制文件（中转站是一个字符数组，并进行异常处理）

|  |
| --- |
| **public class** TestFileReaderWriter2 {  **public static void** main(String[] args) {  //1.创建字符流  **try**( Reader fr = **new** FileReader(**new** File(**"e:/readme.txt"**));  Writer fw = **new** FileWriter(**"e:/readm22.txt"**);){  //2.使用字符流  **char** cbuf=new char[1024];  **int** len = fr.read(cbuf);  **while**(len!= -1){  //System.out.println(cbuf);  fw.write(cbuf,0,len);  len = fr.read(cbuf);  }  }**catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } } |

* 其实只有字节流，没有字符流。字符流的底层还是字节流，进行了封装转换，是开发者可以更简单的来处理非英文字符
* 字节流可以完成所有类型文件的复制（文本、音频、视频、图片、chm）；字符流只可以完成文本文件的复制（txt、java） doc不是文本文件；字符流一般用来处理包含中文的文本文件

### 本节作业

1. 使用FileInputStream和FileOutputStream实现文件复制
2. 使用FileReader和FileWriter实现文件复制

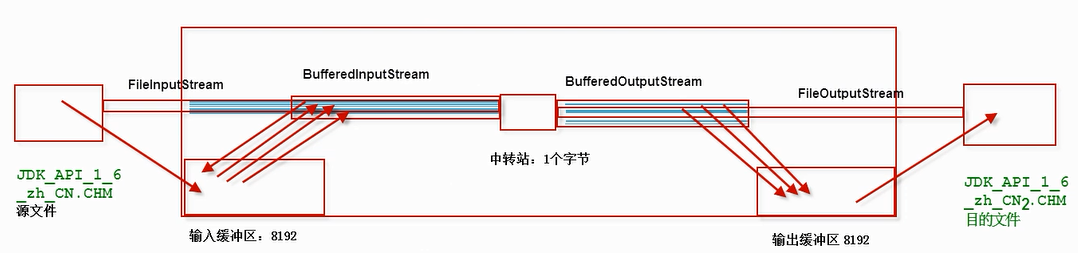
## 三、缓冲流

### 3.1缓冲字节流BufferedInputStream和BufferedOutputStream

|  |
| --- |
| **public class** TestCopy5 {  **public static void** main(String[] args) **throws** IOException {  //1.创建一个输入流和输出流  InputStream fis = **new** FileInputStream(**new** File(**"e:/JDK\_API.CHM"**));  OutputStream fos = **new** FileOutputStream(**new** File(**"e:/JDK\_API2.CHM"**));  //默认输入缓冲区大小8192  BufferedInputStream bis = **new** BufferedInputStream(fis);  //默认输出缓冲区大小8192  BufferedOutputStream bos = **new** BufferedOutputStream(fos);   //2.使用输入流和输出流完成文件复制  //2.1准备一个中转站（水杯）  **int** n;  //2.2先读一个字节  n = bis.read();//读取一个字节，赋给n  **while**(n != -1){  //2.3再写一个字节  bos.write(n);  //2.4在读一个字节  n = bis.read();  }  //3.关闭输入流和输出流  bis.close();  bos.close();  } } |

#### 【示例10】复制文件（使用缓冲流字节流提高效率）

* 缓冲流的原理



* 只要关闭高层流即可，底层流不用手动关闭；因为高层的关闭方法就是把底层流关闭
* 如何刷新输出缓冲区（让缓冲区内容写入硬盘，保证一致）
  + 满了就自动刷新
  + bos.close() 先flush，再关闭
  + 手动刷新 flush()

### 3.2缓冲字符流BufferedReader和BufferedWriter

* 问题：之前的文件读写都是按照字节、字符或者数组来实现的，对于文本文件而言，能否按照行，一行行读写呢。
* 提供了BufferedReader和BufferedWriter实现按行读写

#### 【示例11】复制文件（按行读写）

|  |
| --- |
| **public class** TestCopy6 {  **public static void** main(String[] args) **throws** IOException {  //创建两个流  BufferedReader br =   **new** BufferedReader(**new** FileReader(**new** File(**"e:/sqlnet.log"**)));  BufferedWriter bw =   **new** BufferedWriter(**new** FileWriter(**"e:/sqlnet2.log"**));   //使用两个流完成按行读取的功能  //中转站就是一个字符串，存储一行数据  //先读一行  String str = br.readLine();  **while**(str != **null** ){  //再写一行  bw.write(str);  bw.newLine(); //bw.write("\r\n");不同操作系统中换行符是不同的  //再读一行  str = br.readLine();//!!!  }  //关闭两个流  br.close();  bw.close();  } } |

**总结1：BufferedReader和BufferedWriter的优点**

1.速度快

2.简化编程

**总结2：readLine()底层的原理**

底层还是一个一个字符的读取，append()放入到StringBuilder（或者char[] ）中,遇到换行符 ，将StringBuilder(char[])转换成String并返回

**总结3：不同的操作系统中换行符是不同的**

Unix系统里，每行结尾只有“<换行>”，即“\n”；

Windows系统里面，每行结尾是“<回车><换行>”，即“\r\n”；

Mac系统里，每行结尾是“<回车>”,即“\r”。

### 本节作业

1. 缓冲流的原理
2. BufferedReader的readLine()方法的原理
3. 使用缓冲字节流复制文件
4. 使用缓冲字符流按行复制文件

## 四、数据流和对象流

### 4.1数据流DataInputStream和DataOutputStream

之前使用文件流、缓冲流读取文件只能按照字节、数组方式读取，最方便的也是按行读取，能否很方便的实现对各种基本类型和引用类型数据的读写，并保留其本身的类型。

数据流DataInputStream和DataOutputStream和对象流ObjectInputStream和ObjectOutputStream可以解决这个问题，最大的优势就是提供了方便操作各种数据类型的方法，直接调用，简单方便。

注意

* 只有字节流，没有字符流
* 都是处理流，不是节点流
* 数据流只能操作基本数据类型和字符串，对象流还可以操作对象
* 写入的是二进制数据，无法直接通过记事本等查看
* 写入的数据需要使用对应的输入流来读取

#### 【示例12】使用数据流读写文件

|  |
| --- |
| **public class** TestDataStream {  **public static void** main(String[] args) **throws** Exception {  //write();  read();  }   **public static void** write() **throws** Exception{  //创建输出流  OutputStream fos = **new** FileOutputStream(**"e:/readme2.txt"**);  BufferedOutputStream bos = **new** BufferedOutputStream(fos);  DataOutputStream dos = **new** DataOutputStream(bos);   //使用输出流  dos.writeInt(123);  dos.writeDouble(3.14);  dos.writeChar(**'A'**);  dos.writeBoolean(**true**);  dos.writeUTF(**"bjsxt"**);   //关闭输出流  dos.close();  }   **public static void** read() **throws** Exception{  //创建输入流  DataInputStream dis =   **new** DataInputStream(**new** BufferedInputStream(  **new** FileInputStream(  **new** File(**"e:/readme2.txt"**))));   //使用输入流  System.**out**.println(dis.readInt());  **double** d = dis.readDouble();  System.**out**.println(d);  System.**out**.println(dis.readChar());  System.**out**.println(dis.readBoolean());  System.**out**.println(dis.readUTF());  //System.out.println(dis.readUTF());   //关闭输入流  dis.close();  } } |

### 4.2 对象流ObjectInputStream和ObjectOutputStream

#### 【示例13】使用对象流读写文件

|  |
| --- |
| **public class** TestObjectStream {  **public static void** main(String[] args) **throws** Exception {  //write();  read();  }   **public static void** write() **throws** Exception{  //创建输出流  OutputStream fos = **new** FileOutputStream(**"e:/readme2.txt"**);  BufferedOutputStream bos = **new** BufferedOutputStream(fos);  ObjectOutputStream oos = **new** ObjectOutputStream(bos);   //使用输出流  oos.writeInt(123);  oos.writeDouble(3.14);  oos.writeChar(**'A'**);  oos.writeBoolean(**true**);  oos.writeUTF(**"bjsxt"**);  oos.writeObject(**new** Date());  oos.writeObject(**new** Student(1, **"111"**, 22, 333.3));   //关闭输出流  oos.close();  }   **public static void** read() **throws** Exception{  //创建输入流  ObjectInputStream ois =   **new** ObjectInputStream(**new** BufferedInputStream(  **new** FileInputStream(  **new** File(**"e:/readme2.txt"**))));   //使用输入流  System.**out**.println(ois.readInt());  **double** d = ois.readDouble();  System.**out**.println(d);  System.**out**.println(ois.readChar());  System.**out**.println(ois.readBoolean());  System.**out**.println(ois.readUTF());  //System.out.println(dis.readUTF());  Object date = (Date)ois.readObject();  System.**out**.println(date);  System.**out**.println(ois.readObject());   //关闭输入流  ois.close();  } } |

注意：使用对象流读写引用类型的数据，需要相应类实现Serializable接口，否则会提示异常，提示没有序列化，比如：java.io.NotSerializableException: com.bjsxt.entity.Student。

### 4.3序列化和反序列化

1. 什么是序列化和反序列化

序列化 ：Serialization  将对象的状态信息转换为可以存储或传输的形式的过程。对象(内存)------->字节数组  字节序列（外存、网络）

内存---》硬盘 序列化

硬盘---》内存 反序列化

反序列化：DeSerialization

字节数组  字节序列（外存、网络）----------->对象）

1. 什么时候需要序列化和反序列化

  存储或传输 比如存储到外存（硬盘）中   传输到网络

1. 如何实现序列化和反序列化

  相应的类要实现Serializable接口

|  |
| --- |
| **public class** Student **implements** Serializable {  }  ObjectOutputStream oos = **new** ObjectOutputStream(bos);  oos.writeObject(**new** Student(1, **"111"**, 22, 333.3));  ObjectInputStream ois = **new** ObjectInputStream（bis）;  Student stu = (Student)ois.readObject(); |

1. 序列化的细节

* 为什么序列化接口没有任何方法，哪还有什么用

(查看ObjectOutputStream源码)

* static属性不参与序列化
* 如果不希望某个属性参与序列化，要使用transient修饰
* Exception in thread "main" java.io.InvalidClassException:

com.bjsxt.entity.Student; local class incompatible:

stream classdesc serialVersionUID = 5954363181006202290,

local class serialVersionUID = -1877375566195009060

解决方案:给出一个固定的序列化版本号

* 使用对象流把一个对象写到文件时不仅保证该对象是序列化的，而且该对象的成员对象也必须是可序列化的。
* <https://jingyan.baidu.com/article/656db918c36534e381249c83.html>

### 本节作业

1. 使用数据流读写各种基本类型的数据
2. 使用对象流读些各种基本数据类型和引用数据类型的数据
3. 序列化和反序列化及其关键技能点

## 五、其他流

### 5.1其他流

1. 打印流：PrintStream和PrintWriter

只有输出流，没有输入流

System.out、System.err是PrintStream的实例变量

1. 转换流：InputStreamReader和OutputStreamWriter

实现字节流到字符流的转换，是适配器设计模式的应用

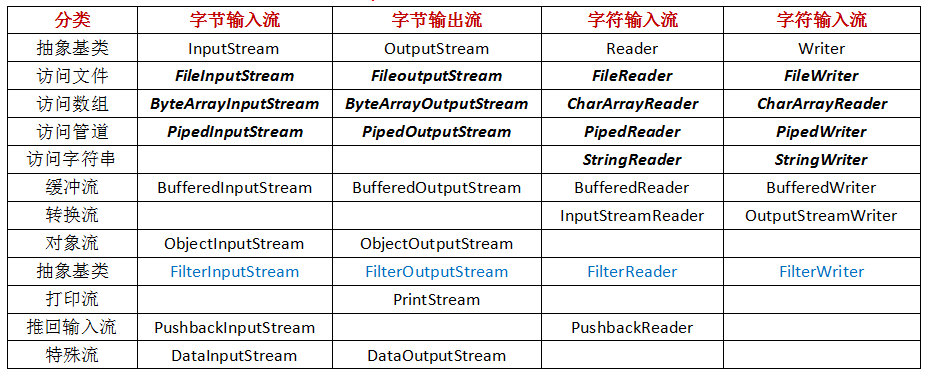
只能从字节流转换成字符流，可以带来处理字符的便利。没有字符流转换成字节流的转换流，因为没有这种需求。

1. Java IO流的设计使用了装饰模式，动态组装流，可以减少子类的数量，是继承的一种替代方案。

|  |
| --- |
| OutputStream fos = **new** FileOutputStream(**"e:/readme.txt"**);  *//提高速度* BufferedOutputStream bos = **new** BufferedOutputStream(fos);  *//简化操作*  DataOutputStream dos = **new** DataOutputStream(bos); |

#### 【示例14】认识其他IO流

|  |
| --- |
| **public class** Test {  **public static void** main(String[] args) **throws** IOException {  //1.打印流 只有输出流，没有输入流  PrintStream ps; //字节流 System.out System.err  PrintWriter pw; //字符流 后面讲解Servlet时会使用  //System.out就是PrintStream的一个引用变量  System.**out**.println();  //System.err也是PrintStream的一个引用变量  System.**err**.println();  //println()强大作用：不管什么类型数据，都给你变成字符串，并输出  //2.转换流  //接收键盘的输入一行数据，并输出  //接收一行数据，需要使用BufferedReader(或者Scanner)。接收键盘的输入，  //需要使用System.in;@2 // InputStream is = System.in; //三相插头 //将字节输入流InputStream转换为字符输入流Reader 三相转两相的转换头 // Reader reader = new InputStreamReader(is); // BufferedReader br = new BufferedReader(reader);//两相的插座  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.**in**));  BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(**"e:/bjsxt.txt"**));  //使用两个流完成按行读取的功能  //中转站就是一个字符串，存储一行数据  //先读一行  String str = br.readLine();  **while**(!**"bye"**.equals(str) ){  //再写一行  bw.write(str);  //bw.write("\r\n");不同操作系统中换行符是不同的  bw.newLine();  //再读一行  str = br.readLine();//!!!  }  //关闭两个流  br.close();  bw.close();  FileInputStream fis; //节点流 数据源是文件  FileOutputStream fos;//节点流 目的地是文件  //数组流 节点流 数据源和目的地都是数组  ByteArrayInputStream bais;  ByteArrayOutputStream baos;  } } |



### 5.2 复制文件夹

**问题1：使用字节流还是字符流**

 使用字节流  可能有图片、视频、音频.....等二进制文件

**问题2：如何提高复制速度**

 BufferedInputStream和BufferedOutputStream

 byte [] buf = new byte[1024];



**问题3：涉及的技能点**

 1.IO流 :文件的复制

   2.递归：各级文件夹和文件的递归复制

  3.File类：文件夹的定义和创建

**问题4：问题的迭代**

 1.复制一个文件

  2.复制一个文件夹下所有的文件（不包括子文件夹）

  3.复制一个文件夹下所有的文件和子文件夹，从而完成文件夹的复制

#### 【示例15】复制一个文件

|  |
| --- |
| **public class** TestDirCopy {  **public static void** main(String[] args) {  copyFile(**"e:/readme.txt"**,**"d:/readme2.txt"**);  }  **public static void** copyFile(String sourceFileName,  String targetFileName) {  //1.创建一个输入流和输出流  BufferedInputStream bis = **null**;  BufferedOutputStream bos = **null**;  **try** {  bis = **new** BufferedInputStream(**new** FileInputStream(  **new** File(sourceFileName)));  bos = **new** BufferedOutputStream(  **new** FileOutputStream(targetFileName));  //2.使用输入流和输出流完成文件复制  //2.1准备一个中转站（水杯）  **byte** [] buf = **new byte**[1024];  **int** len = bis.read(buf);  **while**(len !=-1){  bos.write(buf, 0, len);  len = bis.read(buf);  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }**finally**{  //3.关闭输入流和输出流  **try** {  **if**(bis != **null**){  bis.close();  }   } **catch** (IOException e) {  e.printStackTrace();  }  **try** {  **if**(bos != **null**){  bos.close();  }   } **catch** (IOException e) {  e.printStackTrace();  }  }  } } |

#### 【示例16】复制一个文件文件夹（含子文件夹）

|  |
| --- |
| **public class** TestDirCopy3 {  **public static void** main(String[] args) {  //copyFile("e:/readme.txt","d:/readme2.txt");  copyDir(**"e:/402视频"**,**"e:/zzz视频"**);  }  **public static void** copyDir(String sourceDirName,  String targetDirName){  //创建一个目的文件夹  //File dir = new File("e:/403");  File targetDir = **new** File(targetDirName);  **if**(!targetDir.exists()){  targetDir.mkdir();  }   //复制源文件夹下的所有文件到目的文件夹  File sourceDir = **new** File(sourceDirName);  File [] files = sourceDir.listFiles();//文件夹下所有的文件和子文件夹  **for**(File file:files){  //如果是文件就复制  **if**(file.isFile()){  //copyFile("e:/402/402授课笔记.nyf","e:/zzz/402授课笔记.nyf");  copyFile(sourceDirName+**"/"**+file.getName(),  targetDirName+**"/"**+file.getName());  }  //如果是文件夹，就递归  **if**(file.isDirectory()){  //copyDir("e:/402/20180226-常用类","e:/zzz/20180226-常用类");  copyDir(sourceDirName+**"/"**+file.getName(),  targetDirName+**"/"**+file.getName());  }  }  }  } |

### 本节作业

1. 查询资料，理解装饰模式和适配器模式
2. 接收键盘输入的一行行数据，写入到文件中
3. 复制文件夹