# 尚马教育 JAVA 基础课程

# IO

文档编号：A15

创建日期： 2017-04-12

最后修改日期：2019-09-23

版 本 号：V3.0

电子版文件名：尚马教育-第一阶段-15.IO专题课程.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Java基础版本更新 | V2.0 |
| 2019-08-09 | 徐丽莎 | Java基础版本更新 | V3.0 |

**主讲人：**

**徐丽莎**

目录

[尚马教育 JAVA 基础课程 1](#_Toc14525)

[IO 1](#_Toc30937)

[1. File 3](#_Toc28862)

[1.1. 常用构造 3](#_Toc29906)

[1.2. 常用属性 3](#_Toc10243)

[1.3. 路径 4](#_Toc2072)

[1.4. 常用方法 4](#_Toc4878)

[1.5. 递归 5](#_Toc26616)

[1.6. 过滤文件 6](#_Toc24971)

[2. IO流 7](#_Toc16081)

[3. 字节流 8](#_Toc28720)

[3.1. 字节输入流(InputStream) 9](#_Toc19186)

[3.1.1. 方法 9](#_Toc20155)

[3.1.2. 案例 10](#_Toc9507)

[3.2. 字节输出流(OutputStream) 12](#_Toc25295)

[3.2.1. 方法 13](#_Toc29838)

[3.2.2. 案例 13](#_Toc30394)

[3.3. 字节流复制文件 14](#_Toc17132)

[3.4. 高效字节流复制文件 16](#_Toc15211)

[4. 字符流 18](#_Toc31518)

[4.1. 字符输入流(Reader) 19](#_Toc16164)

[4.1.1. 方法 19](#_Toc11682)

[4.1.2. 案例 20](#_Toc9899)

[4.2. 字符输出流(Writer) 22](#_Toc20857)

[4.2.1. 方法 22](#_Toc9474)

[4.2.2. 案例 22](#_Toc11646)

[4.2.3. 字符流复制文件 23](#_Toc19956)

[4.2.4. 高效字符流复制文件 25](#_Toc13634)

[5. 其他流 25](#_Toc505)

[5.1. 数据流 25](#_Toc15840)

[5.1.1. 方法 26](#_Toc26170)

[5.1.2. 案例 26](#_Toc21101)

[5.2. 序列化流 28](#_Toc12737)

[5.2.1. 案例 28](#_Toc27887)

[5.3. 转换流 30](#_Toc8470)

[5.3.1. 案例 31](#_Toc16382)

[5.4. Properties 32](#_Toc11225)

[5.4.1. 方法 33](#_Toc24191)

[5.4.2. 案例 33](#_Toc13337)

[6. 作业 34](#_Toc14868)

## File

* 存储在变量,数组和对象中的数据是暂时的,当程序终止时他们就会丢失.为了能够永久的保存程序中创建的数据,需要将他们存储到硬盘或光盘的文件中.这些文件可以移动,传送,亦可以被其他程序使用.由于数据存储在文件中,所以我们需要学习一个和文件有密切关系的类,叫做File类,将要掌握获取文件的属性以及删除和重命名文件.最终如何向文件中写入数据和从文件中读取数据.
* File类描述的是一个存在或者不存在的文件或文件夹。（文件夹也可以称为目录）

### 常用构造

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **构 造 方 法** | **说    明** | | **File(String pathname)** | **指定文件（或目录）名创建File类对象** | | **[File](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/io/File.html" \l "File-java.lang.String-java.lang.String-)([String](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/lang/String.html" \o "class in java.lang) parent, [String](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/lang/String.html" \o "class in java.lang) child)** | **指定父级目录和子级文件创建File类对象** | | **[File](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/io/File.html" \l "File-java.io.File-java.lang.String-)([File](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/io/File.html" \o "class in java.io) parent, [String](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/lang/String.html" \o "class in java.lang) child)** | **指定父级目录和子级文件创建File类对象** | |
| // 在当前目录下创建一个与aaa.txt文件名相关联的文件对象  File f1 = **new** File("aaa.txt");  // 指明详细的路径以及文件名，请注意双斜线  File f2 = **new** File("D:\\Java\\Hello.java"); |

### 常用属性

* [pathSeparator](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/io/File.html" \l "pathSeparator)/[pathSeparatorChar](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/io/File.html" \l "pathSeparatorChar)与系统相关的路径分隔符。
  + Windows \
  + Uinx /
* [separatorChar](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/io/File.html" \l "separatorChar)/[separator](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\jdk%20api%201.8_中文.CHM::/java/io/../../java/io/File.html" \l "separator) 与系统相关的默认名称分隔符
  + Windows ;
  + Unix :

### 路径

* 相对路径
  + 相对路径是指相对于某位置的路径，是指相对于当前目录。
  + 在执行Java程序时，相对路径为执行java命令时当前所在的目录
* 绝对路径
  + 对于UNIX平台，绝对路径名的前缀是"/"。相对路径名没有前缀。
  + 对于Windows平台，绝对路径名的前缀由驱动器号和一个":"组成，例"c:\\..."。相对路径没有盘符前缀。

### 常用方法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **方 法 原 型** | **说    明** | | **boolean exists()** | **判断文件是否存在，存在返回true，否则返回false** | | **boolean isFile()** | **判断是否为文件，是文件返回true，否则返回false** | | **boolean isDirectory()** | **判断是否为目录，是目录返回true，否则返回false** | | **String getName()** | **获得文件的名称** | | **String getAbsolutePath()** | **获得文件的绝对路径** | | **long length()** | **获得文件的长度（字节数）** | | **boolean createNewFile()**  **throws IOException** | **创建新文件，创建成功返回true，否则返回false，有可能抛出IOException异常，必须捕捉** | | **boolean delete()** | **删除文件，删除成功返回true，否则返回false** | | **File[] listFiles()** | **返回文件夹内的子文件与子文件夹的数组** | |
| **public** **class** FileDemo {  **public** **static** **void** main(String[] args) {  // 创建一个文件对象，使之与一个文件关联  File file = **new** File("test.txt");  // 显示与文件有关的属性信息  System.***out***.println("文件或目录是否存在：" + file.exists());  System.***out***.println("是文件吗：" + file.isFile());  System.***out***.println("是目录吗：" + file.isDirectory());  System.***out***.println("名称：" + file.getName());  System.***out***.println("绝对路径：" + file.getAbsolutePath());  System.***out***.println("文件大小：" + file.length());  }  } |

### 递归

* 查看当前目录下所有的子目录和子文件

|  |
| --- |
| **private** **static** **void** testFile(File file,String pathStr) {  File[] files = file.listFiles();//一级资源  **for**(File fi:files) {  **if**(fi.isDirectory()) {//判断是否是目录  System.***out***.println(pathStr+fi.getName());  *testFile*(fi,"| "+pathStr);  }**else** {  //文件  System.***out***.println(pathStr+fi.getName());  }  }  } |
| **public** **static** **void** main(String[] args) {  testFile(**new** File("src"),"|-");  } |

### 过滤文件

查看当前目录下所有的java文件

|  |
| --- |
| **private** **static** **void** testFile(File file, String pathStr) {  File[] files = file.listFiles();//一级资源  **for**(File fi:files) {  **if**(fi.isDirectory()) {//判断是否是目录  System.***out***.println(pathStr+fi.getName());  *testFile*(fi,"| "+pathStr);  }**else** {  //文件  **if**(fi.getName().endsWith("java")) {  System.***out***.println(pathStr+fi.getName());  }}}} |
| **private** **static** **void** testFile(File file, String pathStr) {  File[] files = file.listFiles(**new** FileFilter() {  @Override  **public** **boolean** accept(File child) {// child 子目录 子文件  **if** (child.isDirectory()) {  **return** **true**;  }  **return** child.getName().endsWith("txt");  }  });  // files = file.listFiles(child -> {  // if (child.isDirectory()) {  // return true;  // }  // return child.getName().endsWith("txt");  // });  //.......  } |
| **private** **static** **void** testFile(File file, String pathStr) {  // FilenameFilter  File[] files = file.listFiles((parentFolder, childName) -> {  **if** (**new** File(parentFolder, childName).isDirectory()) {  **return** **true**;  }  **return** childName.endsWith("txt");  });  **for** (File fi : files) {  **if** (fi.isDirectory()) {  // System.out.println(pathStr+fi.getName());  *testFile*(fi, "| " + pathStr);  } **else** {  System.***out***.println(pathStr + fi.getName());  }  }  } |

## IO流

* File类表示存在的文件或文件夹，只能操作文件/目录的基本属性信息。
* 如果是操作文件内容的话，就必须使用IO来完成。
* I/O类库中使用“流”这个抽象概念。Java对设备中数据的操作是通过流的方式。表示任何有能力产出数据的数据源对象，或者是有能力接受数据的接收端对象。“流”屏蔽了实际的I/O设备中处理数据的细节。IO流用来处理设备之间的数据传输。设备是指硬盘、内存、键盘录入、网络等。
* Java用于操作流的对象都在IO包中。IO流技术主要用来处理设备之间的数据传输。
* IO流的分类
  + 流按操作数据类型的不同分为两种：字节流与字符流。
  + 流按流向分为：输入流，输出流（以程序为参照物，输入到程序，或是从程序输出）

|  |
| --- |
|  |

## 字节流

* 计算机中都是二进制数据,一个字节是8个2进制位.字节可以表示所有的数据,比如文本,音频,视频.图片,都是作为字节存在的.也就是说字节流处理的数据非常多。
* 在文本文件中存储的数据是以我们能读懂的方式表示的。而在二进制文件中存储的数据是用二进制形式表示的。我们是读不懂二进制文件的，因为二进制文件是为了让程序来读取而设计的。例如，Java的源程序(.java源文件)存储在文本文件中,可以使用文本编辑器阅读,但是Java的类(字节码文件)存储在二进制文件中,可以被Java虚拟机阅读。二进制文件的优势在于它的处理效率比文本文件高。

### **字节输入流(InputStream)**

|  |
| --- |
|  |

* FileInputStream类称为文件输入流，继承于InputStream类，是进行文件读操作的最基本类；它的作用是将文件中的数据输入到内存中，我们可以利用它来读文件；由于它属于字节流，因此在读取Unicode字符（如中文）的文件时可能会出现乱码问题。

#### 方法

|  |
| --- |
|  |
|  |

#### 案例

* 读取指定文件内容。

|  |
| --- |
| **private** **static** **void** test1(File file) {  // 读取文件内容  Objects.*requireNonNull*(file);  **if** (!file.exists()) {  System.***out***.println("文件不存在");  **return**;  }  InputStream inputStream = **null**;  **try** {  // 1.构建字节输入流对象  inputStream = **new** FileInputStream(file);// file文件的内容 目前都已经存储在inputStream对象里面  // 2.结合输入流对象 读取内容  **int** result = inputStream.read();// 一次读取1个字节 读到文件的末尾 返回值 -1  System.***out***.println((**char**) result);  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (Objects.*nonNull*(inputStream)) {  inputStream.close();  }  } **catch** (IOException e) {  e.printStackTrace();  }}} |
| **private** **static** **void** test3(File file) {  **try** (InputStream inputStream = **new** FileInputStream(file);){  // 结合输入流对象 循环读取内容  // byte[] by = new byte[(int) file.length()];//设定数组的空间大小与文件的内容大小一致  **byte**[] by = **new** **byte**[3];//设定数组的空间大小与文件的内容大小一致  **int** result = 0;//最多读取b.length个字节内容 存储到字节数组里面 (有效的字节内容个数)    **while**((result = inputStream.read(by))!=-1) {  //将字符数组转换成字符串  // System.out.println(Arrays.toString(by));  System.***out***.print(**new** String(by, 0, result));  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **private** **static** **void** test4(File file) {  **try** (InputStream inputStream = **new** FileInputStream(file);){  **byte**[] by = **new** **byte**[1024];  **int** result = 0;//读取的有效字节个数 -1  //off: 从数组第几个 索引开始写入  //len: 写len个元素到字节数组 <=by.length  **while**((result = inputStream.read(by, 0, by.length))!=-1) {  System.***out***.print(**new** String(by, 0, result));  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

### 字节输出流(OutputStream)

|  |
| --- |
|  |

* OutputStream该类是抽象类，需要使用具体的实现类来创建对象查看API文档，找到了OutputStream的实现类FileOutputStream ,主要用于将数据写入文件。

#### 方法

|  |
| --- |
|  |
|  |

#### 案例

* 将数据写入文件(磁盘)

|  |
| --- |
| **private** **static** **void** test1(File file) {  **try**(  //1.创建输出流对象  OutputStream outputStream = **new** FileOutputStream(file,**true**);//文件不存在 可以自动创建文件  ){  //2.将内容写入文件里面  // outputStream.write(97);  // outputStream.write('c');// char 转换 int  outputStream.write('\n');  // outputStream.write('b');  // outputStream.write('1');  // byte[] by = {97,98,'a'};  // outputStream.write("我们".getBytes());  **byte**[] by = "我们".getBytes();  System.***out***.println(Arrays.*toString*(by));  outputStream.write(by, 1, 3);  //off 从第几个索引位置开始读  //len 写多少个  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

### 字节流复制文件

* 什么是文件拷贝？很显然，先开一个输入流，将文件加载到流中，再开一个输出流，将流中数据写到文件中。就实现了文件的拷贝。

|  |
| --- |
| /\*\*  \* 文件复制  \* **@param** sourceFile 源文件  \* **@param** targetFile 目标文件  \*/  **private** **static** **void** testCopy1(File sourceFile, File targetFile) {  // 原理:将读到的源文件的内容 写入 目标文件里面  **try** (  // 1.创建输入流对象  InputStream inputStream = **new** FileInputStream(sourceFile);  // 3.创建输出流对象  OutputStream outputStream = **new** FileOutputStream(targetFile);) {  // 2.循环读取文件内容  **int** result = 0;  **while** ((result = inputStream.read()) != -1) {  // 循环的写入目标文件  outputStream.write(result);// 读一个字节 写一个字节  }  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **private** **static** **void** testCopy2(File sourceFile, File targetFile) {  **long** start = System.*currentTimeMillis*();  **try** (  // 1.创建输入流对象  InputStream inputStream = **new** FileInputStream(sourceFile);  // 3.创建输出流对象  OutputStream outputStream = **new** FileOutputStream(targetFile);) {  // 2.循环读取文件内容  **int** result = 0;  **byte**[] by = **new** **byte**[1024 \* 10];// 创建一个缓冲区  **while** ((result = inputStream.read(by)) != -1) {  // 循环的写入目标文件  outputStream.write(by, 0, result);// 读一个字节 写一个字节  }  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  **long** end = System.*currentTimeMillis*();  System.***out***.println(end-start);//2035  } |

### 高效字节流复制文件

* 上述程序中我们为了提高流的使用效率,自定义了字节数组,作为缓冲区.Java其实提供了专门的字节流缓冲来提高效率.
* BufferedOutputStream和BufferedOutputStream类可以通过减少读写次数来提高输入和输出的速度。它们内部有一个缓冲区，用来提高处理效率。查看API文档，发现可以指定缓冲区的大小。其实内部也是封装了字节数组，默认的字节是8192。
* 显然缓冲区输入流和缓冲区输出流要配合使用。首先缓冲区输入流会将读取到的数据读入缓冲区，当缓冲区满时，或者调用flush方法，缓冲输出流会将数据写出。
* 注意：当然使用缓冲流来进行提高效率时，对于小文件可能看不到性能的提升。但是文件稍微大一些的话，就可以看到实质的性能提升了。

|  |
| --- |
| **private** **static** **void** testCopy3(File sourceFile, File targetFile) {  **try** (  // 1.创建高效输入流对象  BufferedInputStream bufferedInputStream = **new** BufferedInputStream(**new** FileInputStream(sourceFile));  // 创建高效输出流对象  BufferedOutputStream bufferedOutputStream = **new** BufferedOutputStream(  **new** FileOutputStream(targetFile));) {  **int** result = 0;  **while** ((result = bufferedInputStream.read()) != -1) {  bufferedOutputStream.write(result);// 读一个字节 写一个字节  // 将读到的字节内容写入缓冲区buf[]  // 当缓冲区满了 会自动刷出到文件里面  // 或者当调用close方法的时候 也会将缓冲区的内容刷出到文件里面  }  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **private** **static** **void** testCopy4(File sourceFile, File targetFile) {  **long** start = System.*currentTimeMillis*();  **try** (  // 1.创建高效输入流对象  BufferedInputStream bufferedInputStream = **new** BufferedInputStream(**new** FileInputStream(sourceFile));  // 创建高效输出流对象  BufferedOutputStream bufferedOutputStream = **new** BufferedOutputStream(  **new** FileOutputStream(targetFile));) {  **byte**[] by = **new** **byte**[1024\*10];//8192  **int** result = 0;  **while** ((result = bufferedInputStream.read(by)) != -1) {  bufferedOutputStream.write(result);// 读一个字节 写一个字节  // 将读到的字节内容写入缓冲区buf[]  // 当缓冲区满了 会自动刷出到文件里面  // 或者当调用close方法的时候 也会将缓冲区的内容刷出到文件里面  }  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  **long** end = System.*currentTimeMillis*();  System.***out***.println(end-start);//540  } |

## 字符流

* 计算机并不区分二进制文件与文本文件。所有的文件都是以二进制形式来存储的，因此，从本质上说，所有的文件都是二进制文件。所以字符流是建立在字节流之上的，它能够提供字符层次的编码和解码。例如，在写入一个字符时，Java虚拟机会将字符转为文件指定的编码（默认是系统默认编码），在读取字符时，再将文件指定的编码转化为字符。
* 字符流就是：字节流 + 编码表，为了更便于操作文字数据。
* 常见的码表：
* ASCII：美国标准信息交换码。用一个字节的7位可以表示
* ISO8859-1：拉丁码表。欧洲码表，用一个字节的8位表示。又称Latin-1
* GB2312：英文占一个字节，中文占两个字节.中国的中文编码表
* GBK：中国的中文编码表升级，融合了更多的中文文字符号
* Unicode：国际标准码规范，融合了多种文字。所有文字都用两个字节来表示，Java语言使用的就是unicode。
* UTF-8：最多用三个字节来表示一个字符

### 字符输入流(Reader)

|  |
| --- |
|  |

* 由于Reader也是抽象类，所以想要使用字符输入流需要使用Reader的子类。查看API文档，找到了FileReader。

#### 方法

|  |
| --- |
|  |
|  |

#### 案例

* 使用字符流读取文件内容

|  |
| --- |
| **private** **static** **void** test1(File file) {  **try** (  // 1.创建字 符输入流对象  Reader reader = **new** FileReader(file);) {  // 2.循环读取文件内容  **int** result = 0 ;//一次读了一个字符  **while**((result = reader.read())!=-1) {  System.***out***.print((**char**)result);  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **private** **static** **void** test2(File file) {  **try** (  // 1.创建字 符输入流对象  Reader reader = **new** FileReader(file);) {  // 2.循环读取文件内容  **int** result = 0 ;//一次读了一个字符  **char**[] ch = **new** **char**[1024];//1024  **while**((result=reader.read(ch))!=-1) {  System.***out***.print(**new** String(ch,0,result));  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **private** **static** **void** test3(File file) {  **try** (  // 1.创建字 符输入流对象  Reader reader = **new** FileReader(file);) {  // 2.循环读取文件内容  **int** result = 0 ;  **char**[] ch = **new** **char**[1024];//1024  **while**((result=reader.read(ch))!=-1) {  System.***out***.print(**new** String(ch,0,result));  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

### 字符输出流(Writer)

|  |
| --- |
|  |

#### 方法

|  |
| --- |
|  |
|  |

#### 案例

* 使用字符输入流将数据写入文件

|  |
| --- |
| **private** **static** **void** test(String path) {  **try** (  // 创建字符输出流对象  Writer writer = **new** FileWriter(path, **true**);) {  //写入数据  writer.write('\n');  writer.write('我');  writer.write('\n');  writer.write("生活的".toCharArray());  writer.write("abcd".toCharArray(), 0, 2);  writer.write('\n');  writer.write("sdjhurh还以为语文");  writer.append("abc");  System.***out***.println("success");  } **catch** (IOException e) {  e.printStackTrace();  }  } |

#### 字符流复制文件

* 一个文本文件中有中文有英文字母，有数字。想要把这个文件拷贝到别的目录中。我们可以使用字节流进行拷贝，使用字符流呢？肯定也是可以的。

|  |
| --- |
| **private** **static** **void** testCopy1(File sourceFile, File targetFile) {  // 读取sourceFile内容  // 写入targetFile  **try** (  Reader reader = **new** FileReader(sourceFile);  Writer writer = **new** FileWriter(targetFile);) {  **int** result = 0;  **while** ((result = reader.read()) != -1) {  writer.write(result);  }  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **private** **static** **void** testCopy2(File sourceFile, File targetFile) {  **try** (  Reader reader = **new** FileReader(sourceFile);  Writer writer = **new** FileWriter(targetFile);) {  **char**[] ch = **new** **char**[1024];  **int** result = 0;  **while**((result= reader.read(ch))!=-1) {  writer.write(ch, 0, result);  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

* 字节流可以拷贝视频和音频等文件，那么字符流可以拷贝这些吗？
* 经过验证拷贝图片是不行的。发现丢失了信息，为什么呢？
* 计算机中的所有信息都是以二进制形式进行的存储（1010）图片中的也都是二进制

在读取文件的时候字符流自动对这些二进制按照码表进行了编码处理，但是图片本来就是二进制文件，不需要进行编码。有一些巧合在码表中有对应，就可以处理，并不是所有的二进制都可以找到对应的。信息就会丢失。所以字符流只能拷贝以字符为单位的文本文件

（以ASCII码为例是127个，并不是所有的二进制都可以找到对应的ASCII，有些对不上的，就会丢失信息。）

#### 高效字符流复制文件

|  |
| --- |
| **private** **static** **void** testCopy3(File sourceFile, File targetFile) {  **try**(  BufferedReader bufferedReader = **new** BufferedReader(**new** FileReader(sourceFile));  BufferedWriter bufferedWriter = **new** BufferedWriter(**new** FileWriter(targetFile));  ){  String content = "";  **while**((content = bufferedReader.readLine())!=**null**) {  bufferedWriter.write(content);  bufferedWriter.newLine();  }  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

## 其他流

### 数据流

* 操作基本数据类型的流对象。
* DataInputStream：从底层输入流中读取基本 Java 数据类型。
* DataOutputStream:将基本类型的值或字符串转换为字节，并且将字节输出到数据流。

#### 方法

* DataInputStream数据输入流：

|  |
| --- |
|  |
|  |

* DataOutputStream数据输出流：

|  |
| --- |
|  |
|  |

#### 案例

* 对于数据流而言，需要遵循前提: 先写再读。

|  |
| --- |
| **private** **static** **void** test2(String string) {  **try** (DataOutputStream dataOutputStream = **new** DataOutputStream(**new** FileOutputStream(**new** File(string)));  ) {  dataOutputStream.writeInt(1001);  dataOutputStream.writeUTF("张三");  dataOutputStream.writeUTF("1234");  dataOutputStream.writeDouble(6000.0);  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |
| **private** **static** **void** test1(String path) {  **try** (DataInputStream dataInput = **new** DataInputStream(**new** FileInputStream(**new** File(path)))) {  System.***out***.println("id:" + dataInput.readInt());// 4个字节  System.***out***.println("name:" + dataInput.readUTF());//  System.***out***.println("pass:" + dataInput.readUTF());//  System.***out***.println("sal：" + dataInput.readDouble());  // java.io.EOFException end of File  System.***out***.println("success");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

### 序列化流

* 当创建对象时,程序运行时它就会存在,但是程序停止时,对象也就消失了.但是如果希望对象在程序不运行的情况下仍能存在并保存其信息，将会非常有用，对象将被重建并且拥有与程序上次运行时拥有的信息相同。可以使用对象的序列化。
* ObjectOutput，对象的序列化： 将内存中的对象直接写入到文件设备中
* ObjectInput，对象的反序列化： 将文件设备中持久化的数据转换为内存对象
* Java的对象序列化（Object Serialization）将那些实现了 Serializable接口的对象转换成一个字节序列，并可以在以后将这个字节序列完全恢复为原来的对象。这一过程甚至可通过网络进行。这意味着序列化机制能自动弥补不同操作系统之间的差异。
* 只要对象实现了 Serializable接口（该接口仅是一个标记接口，不包括任何方法），对象的序列化处理就会非常简单。当序列化的概念被加入到语言中时，许多标准库类都发生了改变，以便能够使之序列化——其中包括所有原始数据类型的包装器、所有容器类以及许多其他的东西。甚至 Class 对象也可以被序列化。
* 由于上述ObjectOutput和ObjectInput是接口，所以需要使用具体实现类
* ObjectOutputStream：被写入的对象必须实现一个接口:Serializable
* ObjectInputStream：读取序列化对象的数据。该方法抛出异常：ClassNotFountException。

#### 案例

|  |
| --- |
| **private** **static** **void** testObjectOutPut(File file) {  Scanner input = **new** Scanner(System.***in***);  System.***out***.println("请录入用户名:");  String name = input.nextLine();  System.***out***.println("请录入密码:");  String pass = input.nextLine();  **try** (  // 创建ObjectOutPut对象  ObjectOutput objectOutput = **new** ObjectOutputStream(**new** FileOutputStream(file));) {  User user = **new** User(name, pass);  user.setUid(1001);  user.setMoney(10000.0);  objectOutput.writeObject(user);  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  System.***out***.println("注册成功，数据已经保存");  input.close();  } |
| **private** **static** **void** testObjectInput(File file,String name,String pass) {  //反序列化处理  **try**(  ObjectInput objectInput = **new** ObjectInputStream(**new** FileInputStream(file));  ){  User user = (User)objectInput.readObject();  System.***out***.println(user);  **if**(!name.equals(user.getName()) || !pass.equals(user.getPwd())) {  System.***out***.println("用户名或者密码数据有误，登陆失败");  **return**;  }  System.***out***.println("登陆成功");  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  }  } |
| **public** **class** User **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -5182703526695977624L;  **private** Integer uid;  **private** String name;  **private** String pwd;  **private** **transient** Double money;//transient 序列化的时候起作用的  } |

### 转换流

* Reader和Writer最重要的子类是InputStreamReader和OutputStreamWriter类。
* InputStreamReader类包含了一个底层输入流，可以从中读取原始字节。它根据指定的编码方式，将这些字节转换为Unicode字符。
* OutputStreamWriter从运行的程序中接收Unicode字符，然后使用指定的编码方式将这些字符转换为字节，再将这些字节写入底层输出流中
* 源或者目的对应的设备是字节流，但是操作的却是文本数据，可以使用转换作为桥梁。提高对文本操作的便捷
* 一旦操作文本涉及到具体的指定编码表时，必须使用转换流。
* 其实就是：
  + 字符流和字节流之间的桥梁
  + 可对读取到的字节数据经过指定编码转换成字符
  + 可对读取到的字符数据经过指定编码转换成字节

#### 案例

* 读取文件内容在控制台进行打印输出

|  |
| --- |
| **private** **static** **void** test2() {  //读取文件内容在控制台进行打印输出  **try** (BufferedReader bufferedReader = **new** BufferedReader(**new** FileReader(**new** File("src/c.txt")));  BufferedWriter bufferedWriter = **new** BufferedWriter(**new** OutputStreamWriter(System.***out***));  ){  //字符流转换成字节流对象 PrintStream对象  //打印控制台 PrintStream  String content = "";  **while**((content = bufferedReader.readLine())!=**null**) {  bufferedWriter.write(content);  bufferedWriter.newLine();  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  } |

* 把用户在控制台录入的数据保存到文件里

|  |
| --- |
| **private** **static** **void** test1() {  // 把用户在控制台录入的数据保存到文件里  // System.in 录入  // System.out 输出  // BufferedReader类构造里面： 需要一个字符输入流对象  // 需要System.in对象转换成字符流对象  **try** (BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));  BufferedWriter bufferedWriter = **new** BufferedWriter(**new** FileWriter(**new** File("src/c.txt")));) {  System.***out***.println("请录入数据:");  **while** (**true**) {  String str = bufferedReader.readLine();  **if** ("bye".equalsIgnoreCase(str)) {  **break**;  }  bufferedWriter.write(str);  bufferedWriter.newLine();  }  System.***out***.println("success");  } **catch** (IOException e) {  e.printStackTrace();  }  } |

### Properties

* 该类是位于util包里的一个工具类，与IO的结合能十分有效地读写文件，特别在准确快速读取方面效率十分高！
* Properties只加载key=value这样的键值对，与文件名无关，注释使用#
* 该类通常用于对键值对形式的配置文件进行操作.提高代码的维护性。

#### 方法

|  |
| --- |
|  |
|  |

#### 案例

* 用户登录：加载配置文件用户信息，判断用户是否能够登录成功。

|  |
| --- |
| **private** **static** **void** test() {  Properties properties = **new** Properties();  **try** {  //读取配置文件的内容(只对java项目有效)  properties.load(**new** FileInputStream(**new** File("resources/user.properties")));  } **catch** (IOException e) {  e.printStackTrace();  }  System.***out***.println(properties.getProperty("user.pass"));  //手动解码  String name = properties.getProperty("user.uname");  // try {  // name = new String(name.getBytes("ISO8859-1"),"utf-8");  // } catch (UnsupportedEncodingException e) {  // e.printStackTrace();  // }  System.***out***.println(name);  } |

## 作业

1. 使用IO技术，创建一个目录，然后复制一个文件到该目录！
2. 使用IO技术，开发出一个控制台的资源管理器！要求：从命令行输入一个路径！如果存在,将该目录下所有的文件和文件夹列举出来，如果不存在则输出不存在该路径。
3. 基于转换流，从控制台输入一些字符串，并将该类信息保存到日志文件”log.txt”中去。
4. 从控制台进行输入用户名以及用户密码，判断是否登录成功！要求准确的用户名和密码存在配置文件中！
5. 创建一个学生类，包含属性：学号、姓名、性别，包含show（）方法用于显示学生的详细信息。

创建测试类，在控制台上显示添加学生信息，要求程序循环运行，并依次提示接收学生类的所有属性值，保存到学生对象中，再将学生对象保存到集合对象中，并提示“是否继续添加（y/n）：”,如果选择“y”则继续添加，否则退出循环，并将保存学生数据的集合对象通过序列化保存到“student.dat”文件中。

实现从“student.dat”文件中反序列化保存学生数据的集合对象，并遍历打印输出学生信息。

1. 已知文件a.txt文件中的内容为“AAbcdea22dferwplkCC321ou1”,请编写程序读取该文件内容，要求去掉重复字母(区分大小写字母)并按照自然排序顺序后输出到b.txt文件中。即b.txt文件内容应为”123ACabcdefklopruw”这样的顺序输出。
2. 读取任意txt文件内容,并统计出这个文本中每个字符以及每个字符出现的次数，并以以下格式: 字符=次数 持久化保存文件中。
3. 使用集合相关的功能,存储10个1-50(含50)的随机偶数元素,要求数字不能重复,添加完成后从大到小倒序遍历输出到控制台并使用IO流将集合中的元素按指定格式输出到当指定文件中, 例如: 48,44,40,38,34,30,26……
4. 已知student\_info.txt文件中有如下数据：（姓名-年龄-总分)  
   张三-21-98  
   李四-23-97  
   王五-25-100  
   赵六-15-100  
   孙七-19-93  
   运用IO技术获取将该文件中的数据分别封装成5个Student(姓名为String类型,年龄为int类型,总分为int类型 )对象存入集合中（需要自己定义Student类）  
   要求：根据学生的总分进行排序（降序），如果分数相同则比较年龄，年龄较大的排在前面。并显示排序之后的结果。