# IO流

## File和IO流概述

IO流是什么:

1.可以将数据从本地文件中读取出来

2.可以将数据从内存保存到本地文件

File类是什么:

1.在读写数据时告诉虚拟机要操作的(文件/文件夹)在哪

2.对(文件/文件夹)本身进行操作.包括创建,删除等.

## File类的构造方法

File它是文件和目录路径名的抽象表示:

1.文件和目录可以通过File封装成对象

2.File封装的对象仅仅时一个路径,它可以是存在的,也可以是不存在的.

常用构造方法:

File(String pathname) : 通过将给定的路径名字符串转换为抽象路径名来创建File对象

File(String parent, String child) :从父路径名字符串+子路径名字符串创建新的File对象

File(File parent, String child) ;从父File对象+子路径名字符串创建新的File对象

**特别的:**

三个构造方法都是创建了直到文件名的File对象

第一个就是在构造括号里面给出指定的路径直到精确到文件名

第二个是将路径和文件名拆开,先再构造方法括号里传入路径,再传入该路径下的一个文件名

第三个是在构造方法的括号里面将一个表示路径的File对象(在前),和该路径下的一个文件(在后)传进去

## File的绝对路径和相对路径

绝对路径:从盘符开始

File file1 = new File("E:\\a.txt");

相对路径:相对当前项目下的路径

File file2 = new File("a.txt");

## File类的创建功能

File类的创建功能:

public boolean createNewFile() : 创建一个新的空的文件

**注意点:**

1.如果文件存在,那么创建失败,返回false;

2.如果文件不存在,创建成功,返回true;

3. createNewFile()方法不管调用者有没有后缀,只能创建文件;

public boolean mkdir() : 创建一个单极文件夹

**注意点:**

1.只能创建单级文件夹,不能创建多级文件夹;

2.不管调用者有没有后缀,只能创建单级文件夹;

public boolean mkdirs() : 创建一个多级文件夹;

**注意点:**

1.可以创建单级文件夹,也可以创建多级文件夹;

2. 不管调用者有没有后缀,只能创建文件夹;

**一般创建文件夹使mkdirs()方法**

## File的删除功能

public boolean delete() : 删除由此抽象路径名表示的文件或者目录

**注意点: 只能删文件或者空文件夹**

1.不走回收站;

2.如果删除的是文件,那么直接删除,如果删除的是文件夹,那么只能删除空文件夹

3.如果一定要删除由内容的文件夹,只能先进入到这个文件夹,把内容删完才能再删文件夹

## File类的获取和判断功能

public boolean isDirectory() 测试此抽象路径名表示的File是否为目录

public boolean isFile() 测试此抽象路径名表示的File是否为文件

public boolean exists() 测试此抽象路径名表示的File是否存在

public String getName() 返回由此抽象路径名表示的文件或目录的名称

**注意点:**

1.如果调用者是文件,那么获取的是文件名和后缀名

2.如果调用者是一个文件夹,那么获取的是文件夹的名字

**高级获取功能:**

Public File[] listFiles() 返回此抽象路径名表示的目录下的文件和文件夹的File对象数组

listFiles方法的注意事项:

1.当调用者布顿在,返回null;

2.当调用者是一个文件时,返回null;

3.当调用者是一个空文件夹时,返回一个长度为0的数组;

4.当调用者是一个有内容的文件夹时,将里面所有文件和文件夹的路径放在File数组中返回;

5.当调用者是一个由隐藏文件的文件夹时, 将里面所有文件和文件夹的路径放在File数组中返回,包含隐藏内容;

6. 当调用者是一个需要权限才能进入的文件夹时,返回null;

## IO流概述

解决内存中数据不能永久化存储的问题.

I表示intput,是内存从硬盘上读取数据的过程,称之为读数据;

O表示output,是内存将数据写入硬盘的过程,称之为写数据.

## IO流的分类

**按照流向:**

输入流:硬盘→内存 读

输出流:内存→硬盘 写

**按照数据类型分:**

字节流:可以操作所有类型的文件;

字符流:只能操作纯文本文件.

<纯文本文件:可以用电脑自带的记事本能打开能看懂的文件>

## 字节流

### 字节输出流-入门

使用步骤:

1. 创建字节输出流对象 ---告诉虚拟机,我要在那个文件中写数据

FileOutputStream fos = new FileOutputStream("a.txt");//相对路径,在当前包下创建

1. 写数据 ---写数据

fos.write(97);

1. 释放资源 ---关闭流

fos.close();

### 字节输出流-注意事项

1. 创建字节输出流对象的时候,如果指向文件不存在,会帮我们自动创建出来,如果文件已存在,则他会把文件里面的内容清空.
2. 写出的整数,实际上写出的是整数在码表上对应的字母
3. 每次使用完都必须要释放资源

### 字节输出流-一次写多个数据

void write(int b) : 一次写一个字节数据;

void write(byte[] b) : 一次写一个字节数组;

void write(byte[] b, int off, int len) : 一次写一个字节数组的部分数据.

### 字节输出流-两个问题

一 字节流写数据如何实现换行

在需要换行的位置加入换行符

Windows: \r\n

Linux: \n

Mac: \r

注意:假如换行符的方式也是使用字节流对象调用write方法写入换行符时,换行符时字符串,需要将其转换成字节数组才能被write方法写入,String对象调用getBytes​()方法转化为一个字节数组才能正真写入.

二 字节流写数据如何实现追加写入

FileOutputStream​(File file, boolean append) 通俗来说,这也是FileOutputStream类的构造方法,其在原来的基础上假如了一个开关,在不写的默认情况下为false

创建文件输出流以指定的名称写入文件,如果开关参数为true则不会清空原有内容,实现追加写入,反之则会清空,构造方法完成清空.

### 字节输出流-try..catch捕获异常

**FileOutputStream fos = null;//对象定义在外面,这样才能访问到**

**try {**

**fos = new FileOutputStream("a.txt");**

**fos.write('a');**

**} catch (IOException e) {**

**e.printStackTrace();**

**} finally {//释放资源是一定要执行的,所以引入了try..catch的最终版假如finally**

**if (fos!=null) {//因为对象初始赋值为null,所以这里要进行非null判断,否则会空指针**

**try {//释放资源一样会异常,继续抓取**

**fos.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

### 字节输出流-小结

步骤:

1.创建字节输出流对象

文件不存在,就创建

文件存在就清空,如果不想清空就加true

2.写数据

可以写一个子节,一个字节数组的一部分或者全部

换行符:\r\n

3.释放资源

### 子节输入流-入门

使用步骤:

1.创建字节输入流对象

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

<如果文件不存在会报错>

2.读数据

int read = fis.read();

System.out.println(read);

一次读取一个字节,返回值就是这个字节在码表中对应的数字,如果想要得到字节需要强转

3.释放资源

fis.close();

### 子节输入流-读多个字节

FileInputStream fis = new FileInputStream("mjyJavaseadvance\\src\\A\_mytest\\a.txt");

int read;

while ((read = fis.read()) != -1) {

System.out.print((char) read);

}

fis.close();

### 子节输入流-复制文件

FileInputStream fis = new FileInputStream("E:\\java学习视频\\Java基础视频资料11.18版\\第二阶段视频\\day9\\02\_字节流复制视频.avi");

FileOutputStream fos = new FileOutputStream("mjyJavaseadvance\\02\_字节流复制视频.avi");

int read;

while ((read = fis.read()) != -1) {

fos.write(read);

}

fis.close();

fos.close();

### 子节输入流-定义小数组拷贝

FileInputStream fis = new FileInputStream("E:\\java学习视频\\Java基础视频资料11.18版\\第二阶段视频\\day9\\02\_字节流复制视频.avi");

FileOutputStream fos = new FileOutputStream("mjyJavaseadvance\\02\_字节流复制视频.avi");

byte[] bytes = new byte[1024];

int read;

while ((read = fis.read(bytes)) != -1) {

fos.write(bytes,0,read);

}

fis.close();

fos.close();

注意点:一次读取和写入一个字符数组,能加快copy速度

### 子节输入流-小数组拷贝原理

原理分析:

byte[] bytes = new byte[1024];

int read;

while ((read = fis.read(bytes)) != -1) {

fos.write(bytes,0,read);

}

1.定义字节数组就是一次复制多少个字节

2. read​(byte[] b)返回的是读入缓冲区的总字节数的一个int值.

3.前面都是可以装满

4.最后一次读写,可能装不满一个一个字节数组

5.那就装多少,读多少fos.write(bytes,0,read);

### 字节缓冲流-一次读写一个字节

字节缓冲流:

BufferOutputStream : 字节缓冲输出流

BufferedInputStream ; 字节缓冲输入流

构造方法:

字节缓冲输出流 : BufferOutoutStream(OutputStream out)

字节缓冲输入流 : BufferedInputStream(InputStream in)

注意点:为什么在构造中传入的是字节流,而不是具体文件或者路径

字节缓冲区仅仅提供缓冲区,二真正读写数据的还是靠基本的字节流的字节流对象进行操作,缓冲流其实也类似于小组拷贝.

**读写操作核心:**

**int read;**

**while ((read = bif.read()) != -1) {**

**bof.write(read);**

**}**

**使用字节缓冲流读写文件,原理上类似小组拷贝,提高了读写效率**

### 字节缓冲流-一次读写一个字节原理

在底层创建了一个长度为8192的字节数组

### 字节缓冲流-一次读写一个字节数组

byte[] bytes = new byte[1024];

int read;

while ((read = fis.read(bytes)) != -1) {

fos.write(bytes,0,read);

}

字节缓冲流的一次读写一个数组,

跟字节流的区别在于.传入缓冲区后,每次不是读写一个字符而是一个字符数组,再次加快了读写

### 小结

字节流:

可以操作(拷贝)所有类型的文件

字节缓冲流:

可以提高效率

不能直接操作文件,需要传递字节流

拷贝文件的四种方式

1.字节流一次读写一个字节

2.字节流一次读写一个字节数组

3.字节缓冲流一次操作一个字节

4.字节缓冲流一次操作一个字节数组

## 字符流

### 码表

最初的码表是ASCII码表,后面出现了万国码Unicode,现在多以万国码

Windows默认使用的码表为:GBK(由汉字的码表),一个字符两个字节;

Idea和以后工作默认使用Unicode码表的UTF-8编解码格式,一个中文三个字节

### 字符流-编码和解码的方法

编码:

byte[] getBytes() : 使用平台的默认字符集(idea默认码表的解码格式是Unicode码表的UTF-8编解码格式)将该S听编码为一系列字节,将结果存储到一个字节数组中并返回

byte[] getBytes(String charsetName) ; 使用指定的字符集将该String编码为一系列字节,将结果存储到新的字节数组中.并返回

**示例:**

String s = "黑马程序员";

byte[] bytes1 = s.getBytes();

byte[] bytes2 = s.getBytes("GBK");

System.out.println(Arrays.toString(bytes1));

System.out.println(Arrays.toString(bytes2));

解码:

String(byte[] bytes) ; 通过使用平台的默认字符集解码指定的字符数组来构造新的String

String(byte[] bytes, String charsetName) : 通过指定的字符集解码指定的字节数组构造的新的String

**示例:**

String s1 = new String(bytes1);

System.out.println(s1);

String s2 = new String(bytes2,"gbk");//这里不加GBK会出现乱码

System.out.println(s2);

### 字节流读取中文出现乱码的原因

因为字节流一次读取一个字节,而不管是GBK还是UTF-8一个中文都是多个字节,用字节流每次只能读取一部分,所以就会出现乱码.

### 字符流-读取中文的过程

字符流本质上是字节流和编码表的组合;

不管在哪张码表中,中文的第一个字节一定是负数;

使用字符流读取时,他也是通过字节流一个字覅一个字符读取的,但是当他遇到负数时,会根据掺入的码表自动的转换成2-3和字节一组一起读取.

**结论:**

1.想要进行拷贝,一律使用字节流或者字节缓冲流

2.想要把文本文件中的数据读到内存中,请使用字符输入流

想要把内存中的数据写到文本文件中,请使用字符输出流

3.GBK码表的一个中文两个字节,Unicode的UTF-8编码格式一个中文三个字节.

### 字符流-写出数据

步骤:

<1>. 创建字符输出流对象

FileWriter(File file) : 给一个File对象构造一个FileWriter对象;

**FileWriter(String fileName) : 构造一个给定文件名的FileWriter对象;**

<2>. 写数据

void write(int c) 写一个字符

void write(char[] cbuf) 写出一个字符数组

void write(char[] cbuf, int off, int len) 写出字符数组的一部分

**void write(String str) 写一个字符串**

void write(String str, int off, int len) 写一个字符串的一部分

<3>. 释放资源

close()

### 字符流-写出数据的注意事项

1.创建字符输出流对象时,如果文件不存在,就创建,但是前提是保证父级路径正确并存在.如果文件存在就清空

2.写数据时,写出int类型的整数,实际写出的时整数在码表上对应的字母.写出的时字符串数据,是把字符串本身原样写出.

3.每次使用完必须释放资源

### 字符流-flush和close方法

flush() : 刷新流,刷新完毕后还可以继续写数据

close() : 关闭流,释放资源,一旦调用,就不能写数据了

### 字符流-读取数据

1.创建字符输入流对象

FileReader fr = new FileReader

("E:\\Develop\\workspace\\jyworkspace\\mjyJavaseadvance\\src\\A\_mytest\\a.txt");

2.定义字符数组一次读取一个字符数组

read方法一次读取多个字符,把读到的字符存入字符数组,返回值表示读到的字符个数

char[] chars = new char[1024];

int read;

while ((read = fr.read(chars)) != -1) {

System.out.println(new String(chars, 0, read));

}

3.释放资源

fr.close();

### 字符流-练习

需求:键盘录入用户名密码,实现永久存储

FileWriter fw = new FileWrite ("E:\\Develop\\workspace\\jyworkspace\\mjyJavaseadvance\\src\\A\_mytest\\a.txt");

Scanner scanner = new Scanner(System.in);

System.out.println("请输入用户名:");

String userName = scanner.nextLine();

System.out.println("请输入密码:");

String passWord = scanner.nextLine();

fw.write(userName);

fw.write("\r\n");

fw.write(passWord);

fw.close();

### 字符缓冲输入流-读取数据

BufferedReader : 可以将数据高效的读取到内存中

**继承体系:**

public class BufferedReader extends Reader{}

**概述:**

从字符输入流读取文本，缓冲字符，以提供字符，数组和行的高效读取。

构造一个字符缓冲器,一次从内存中读取一个长度为8192的字符数组.

**使用步骤:**

1.创建字符输入流对象

BufferedReader​(Reader in) 创建使用默认大小的输入缓冲区的缓冲字符输入流。

BufferedReader​(Reader in, int sz) 创建使用指定大小的输入缓冲区的缓冲字符输入流。

BufferedReader br = new BufferedReader(newFileReader

("E:\\Develop\\workspace\\jyworkspace\\mjyJavaseadvance\\src\\A\_mytest\\a.txt"));

2.定义字符数组一次读取一个字符数组

char[] chars = new char[1024];

int len;

while ((len = br.read(chars)) != -1) {

System.out.println(new String(chars, 0, len));

}

3.释放资源

br.close();

### 字符缓冲输出流-输出数据

BufferedWriter : 可以将数据高效的写出

**继承体系:**

public class BufferedWriter extends Writer

**概述:**

将文本写入字符输出流，缓冲字符，以提供单个字符，数组和字符串的高效写入.

构造一个字符缓冲器,一次从读取长度为8192的字符数组.

**使用步骤:**

**1.创建字符输入流对象**

BufferedWriter bw = new BufferedWriter(new FileWriter

("E:\\Develop\\workspace\\jyworkspace\\mjyJavaseadvance\\src\\A\_mytest\\a.txt", true));

true 追加写入

**2.写数据**

bw.write("\r\n");

bw.write("wangwu");

bw.write("\r\n");

bw.write("123456");

**3.释放资源**

bw.flush();

bw.close();

### 缓冲流-特有方法

BufferedWriter :

**void newLine() :** 表示行分隔符,具体分隔符由所用系统决定,windows是”\r\n”

BufferedReader :

**public String readLine() :**读一行文字,结果包含的内容的字符串,不包括任何终止字符,如果流的结尾已经到达,则为null

**示例:**

BufferedReader br = new BufferedReader(new FileReader

("E:\\Develop\\workspace\\jyworkspace\\mjyJavaseadvance\\src\\A\_mytest\\a.txt"));

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

### IO流-小结

**练习:**从文件中读取数据,排序,再按照原有格式写回文件

BufferedReader br = new BufferedReader(new FileReader

("E:\\Develop\\workspace\\jyworkspace\\mjyJavaseadvance\\src\\A\_mytest\\a.txt"));

String line = br.readLine();

String[] strings = line.split(" ");

int[] strArr = new int[strings.length];

for (int i = 0; i < strArr.length; i++) {

strArr[i] = Integer.parseInt(strings[i]);

}

Arrays.sort(strArr);

System.out.println(Arrays.toString(strArr));

BufferedWriter bw = new BufferedWriter(new FileWriter

("E:\\Develop\\workspace\\jyworkspace\\mjyJavaseadvance\\src\\A\_mytest\\a.txt")

);

for (int i = 0; i < strArr.length; i++) {

bw.write(strArr[i] + " ");

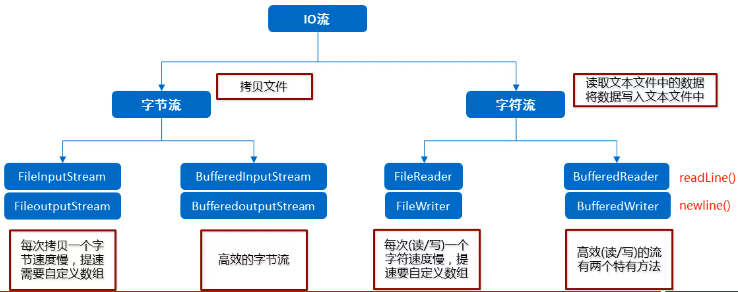
bw.flush();

}

br.close();

bw.close();

小结:



### 转换流-概念

转换流的作用是将字符流和字节流通过指定或者默认的编码表进行转换

流程:

**理解一下:**

字符流 = 字节流 + 编码表 ;

而转换流顾名思义就是在字符流和字节流转换的时候起作用的,它的依据就是指定的或者默认的编码表,归根结底还是字节流进行数据的写入和读取的

### 转换流-指定编码读写

Idea默认编码格式是UTF-8,在读写文件的时候不一定会是这个编码格式,在使用IO流读写时如遇乱码,很有可能就是i编码格式不符导致,在**JDK11以前**是使用转换流<InputStreamReader>和<OutputStreamWriter>创建对象,在构造中传入指定的编码格式就不会出现乱码.**从JDK11开始**字符流构造中可以传入字符<char>的**Charset.forName()**方法传入编码格式,也可以解决这个问题

JDK11开始:

FileReader fr = new FileReader("mjyJavaseadvance\\src\\A\_mytest\\a.txt",

Charset.forName("GBK"));

### 对象操作流-基本特点

将数据以字节的形式存储到本地文件中,在需要使用的时候再使用对象操作流读取到内存(控制台),以提高数据的安全性

### 对象操作流-序列化

对象操作输入流 : ObjectInputStream

<对象反序列化流> : 把写到本地文件中的对象读到内存中,或者接收网络中传输的对现象

对象操作输出流 : ObjectOutputStream

<对象序列化流> : 就是将对象写到本地文件中,或者在网络中传输对象

**创建序列化流对象:**

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("a.txt"));

**Serializable接口的意义:**

1.如果想要这个类的对象能被序列化,那么这个类必须要实现一个接口,Serializable;

2. Serializable接口称之为是一个标记性接口,里面没有任何的抽象方法;

3.只要一个类实现了Serializable接口,那么就表示这个类的对象可以被序列化.

### 对象操作流-反序列化

和前面流一模一样,不做赘述

### 对象操作流-两个注意点

1.用对象序列化流序列化了一个对象后,如果修改了对象所属的Javabean类,读取数据是会出问题的,会抛出InvalidClassException异常

解决方法:

给对象所属的类加一个serialVersionUID

private static final long serialVersionUID = 1L;

2.如果一个对象中的某一个成员变量的值不想被序列化时

给该成员变量加**transient**关键字修饰,该关键字标记的成员变量不能被序列化

### 对象操作流-练习

**需求:**创建多个Javabean类对象写到文件中,再次读取到内存中

**自定义学生类:**

/\*\*

\* @author Darwin

\* @Time 2020/08/14 13:05

\*/

public class Student implements Serializable {

private static final long serialVersionUID = 2L;

private String name;

private int age;

@Override

public String toString() {

return "Student{" +

"name='" + name + '\'' +

", age=" + age +

'}';

}

public static long getSerialVersionUID() {

return serialVersionUID;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public Student(String name, int age) {

this.name = name;

this.age = age;

}

public Student() {

}

}

**测试类中的main方法:**

Student stu1 = new Student("张三", 23);

Student stu2 = new Student("李四", 24);

Student stu3 = new Student("王五", 25);

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("a.txt"));

/\*ArrayList<Student> students = new ArrayList<>(List.of(stu1, stu2, stu3));

oos.writeObject(students);

oos.close();

//第一种方法,将数据存储到集合中,写入和读取集合对象

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("a.txt"));

ArrayList<Student> newStudents = (ArrayList<Student>) ois.readObject();

for (Student newStudent : newStudents) {

System.out.println(newStudent);

}\*/

/\*//第二种方法,出现异常try..catch异常

oos.writeObject(stu1);

oos.writeObject(stu2);

oos.writeObject(stu3);

oos.close();

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("a.txt"));

while (true){

try {

Student newStudents = (Student) ois.readObject();

System.out.println(newStudents);

} catch (EOFException e) {

break;

}

}

ois.close();\*/

### Properties-概述

public class Properties extends Hashtable<Object,Object>

Hashtable实现了Map接口

1.是一个Map体系的集合类

2. Properties中有跟IO有关的方法

3.没有泛型,可以存储任意类型,但是一般只存字符串

### Properties-作为map集合的基本使用

1.创建对象

Properties prop = new Properties();

2.增删改查

prop.put("杨过", "小龙女");

prop.put("郭靖", "黄蓉");

prop.put("东方不败", "东方失败");

prop.remove("东方不败");

prop.get("东方不败");

3.遍历

System.out.println("-----增强for循环keySet遍历-----");

Set<Object> keySets1 = prop.keySet();

for (Object keySet : keySets1) {

Object vlaue = prop.get(keySet);

System.out.println(keySet + " " + vlaue);

}

System.out.println("-----增强for循环entrySet遍历-----");

Set<Map.Entry<Object, Object>> entries = prop.entrySet();

for (Map.Entry<Object, Object> entry : entries) {

Object key = entry.getKey();

Object value = entry.getValue();

System.out.println(key + " " + value);

}

### Properties-特有方法

Object setProperty(String key, String value) 设置集合的键和值,都是String类型

对应Map集合put方法

String getProperty(String key) 通过键,获取值

对应Map集合get方法

Set<String>stringPropertyNames() 返回键的集合,该集合是字符串集合

对应Map集合keySer方法

### Properties-load

void load(InputStream inStream) 从输入字节流读取属性列表(键值对)

void load(Reader reader) 从输入字符流读取属性列表(键值对)

**注意点:**

1.保证本地文件中的数据是以键值对存储的;

2. Properties作为文件的后缀名,一般作为配置文件

**示例:**

Properties prop = new Properties();

FileReader fr = new FileReader("mjyJavaseadvance\\prop.properties");

prop.load(fr);

System.out.println(prop);

### Properties-store

void store(OutputStream out, String comments) 将此属性列表(键值对)写入Properties表中,以适合于使用void load(InputStream inStream)方法的格式写入输出字节流

void store(Writer writer, String comments) 将此属性列表(键值对)写入Properties表中,以适合于使用void load(Reader reader)方法的格式写入输出字节流

**示例:**

FileWriter fw = new FileWriter("mjyJavaseadvance\\prop.properties");

prop.put("tianzhen","2");

prop.put("keai","3");

prop.store(fw,null);//null位置表示在配置文件中写入一些注释,不写放null