

IO流

一、字节流

1. 字节输入流 InputStream

a. FileInputStream : 文件字节输入流

作用：以内存为基准，可以把磁盘文件中的数据以字节的形式读入到内存中

构造器：

- `public FileInputStream(File file)`
- `public FileInputStream(String pathname)`

方法：

- `int read()`

每次读取一个字节返回，若发现没有数据可读，则返回-1

- `int read(byte[] buffer)`

每次使用一个字节数组去读取数据，返回值是字节数组读取的字节数，若发现没有数据可读，则返回-1

- `byte[] readAllBytes() throws IOException`

直接将当前字节输入流对应的文件对象的字节数据装到一个字节数组并返回，可以一次性读完全部字节

b. BufferedInputStream : 缓冲字节输入流

作用：提高字节输入流读数据的性能

构造器：

```
public BufferedInputStream(InputStream is)
```

把**低级**的文件字节输入流包装成一个**高级**的缓冲字节输入流，从而提高**读**数据的性能

2. 字节输出流 OutputStream

a. FileOutputStream : 文件字节输出流

作用：以内存为基准，把内存中的数据以字节的形式写出到文件中去

构造器：

- `public FileOutputStream(File file)`
- `public FileOutputStream(String filepath)`

- `public FileOutputStream(File file,boolean append)`
append这个布尔变量用于决定可否追加数据
- `public FileOutputStream(String filepath,boolean append)`

方法:

- `void write(int b)`
写一个字节到指定文件中去
- `void write(byte[] buffer)`
写一个字节数组到指定文件中去
- `void write(byte[] buffer,int pos,int len)`
写一个字节数组的一部分到指定文件中去, `pos`是起始下标, `len`是截取长度
- `void close() throws IOException`
关闭流

b. `BufferedOutputStream`:缓冲字节输出流

作用: 提高字节输出流写数据的性能

构造器:

```
public BufferedOutputStream(OutputStream os)
```

把低级的文件字节输出流包装成一个高级的缓冲字节输出流, 从而提高写数据的性能

二、字符流

1.字符输入流 `Reader`

a. `FileReader`:文件字符输入流

作用: 以内存为基准, 可以把磁盘文件中的数据以字符的形式读入到内存中

构造器:

- `public FileReader(File file)`
- `public FileReader(String pathname)`

方法:

- `int read()`
每次读取一个字符返回, 若发现没有数据可读, 则返回-1
- `int read(char[] buffer)`

每次使用一个字符数组去读取数据，返回值是字符数组读取的字符数，若发现没有数据可读，则返回-1

b. BufferedReader :缓冲字符输入流

作用：自带8KB的字符缓冲流，可以提高字符输入流读数据的性能

构造器：

```
public BufferedReader(Reader r)
```

把**低级**的文件字符输入流包装成**高级**的缓冲字符输入流，从而提高**读**数据的性能

方法：

```
String readLine()
```

读取一行数据并返回，若无数据可读，则返回 null

2.字符输出流 writer

a. FileWriter :文件字符输出流

作用：以内存为基准，把内存中的数据以字符的形式写出到文件中去

构造器：

- `public FileWriter(File file)`
- `public FileWriter(String filepath)`
- `public FileWriter(File file,boolean append)`

append这个布尔变量用于决定可否追加数据

- `public FileWriter(String filepath,boolean append)`

方法：

- `void write(int c)`

写一个字符到指定文件中去

- `void write(String str)`

写一个字符串到指定文件中去

- `void write(String str,int off,int len)`

写一个字符串的一部分到指定文件中去

- `void write(char[] cbuf)`

写入一个字符数组

- `void write(char[] cbuf,int off,int len)`

写入一个字符数组的一部分

- `void flush() throws IOException`

刷新流，就是将内存中缓存的数据立即写入到磁盘文件中去

- `void close() throws IOException`

关闭流，包含了刷新流的操作

注意事项：

字符输出流写出数据后，必须刷新流或者关闭流，只有这样写入的数据才能生效！

b. `BufferedWriter`:缓冲字符输出流

作用：自带8KB的字符缓冲流，可以提高字符输出流写数据的性能

构造器：

```
public BufferedWriter(Writer w)
```

把**低级**的文件字符输出流包装成**高级**的缓冲字符输出流，从而提高**写**数据的性能

方法：

```
void newLine() 换行
```

三、释放资源的方式

1. `try-catch-finally`

格式：

```
1 try{  
2     ...  
3 }catch(Exception e){  
4     e.printStackTrace();  
5 }finally{  
6     ...  
7 }
```

`finally` 代码块的特点：

无论 `try` 代码块中的程序是否正常执行，最后都一定会执行 `finally` 代码块，除非 JVM 虚拟机终止

作用：一般用于在程序执行完后就，进行资源的释放操作（专业级做法）

代码：

```
1 InputStream is=null;  
2 OutputStream os=null;  
3 try{  
4     ...  
5 }
```

```
5 }catch (Exception e){  
6     e.printStackTrace();  
7 }finally {  
8     try{  
9         if(os!=null){os.close();}  
10    }catch (Exception e){  
11        e.printStackTrace();  
12    }  
13  
14    try{  
15        if(is!=null){is.close();}  
16    }catch (Exception e){  
17        e.printStackTrace();  
18    }  
19 }
```

2. try-with-resource (After JDK1.7)(推荐使用)

格式:

```
1 try(定义资源1;定义资源2;...){  
2     可能出现异常的代码;  
3 }catch(Exception e){  
4     处理异常的代码;  
5 }  
6 //资源: 一般指的是最终实现了AutoCloseable接口的类  
7 //try()中只能放置资源, 否则会报错  
8 //资源使用完毕后, 会自动调用其close()方法, 完成对资源的释放
```

代码:

```
1 try(  
2     InputStream is=new FileInputStream("");  
3     OutputStream os=new FileOutputStream("")  
4 ){  
5     ...  
6 }catch (Exception e){  
7     e.printStackTrace();  
8 }
```

四、转换流

1. 引入背景:

代码编码和被读取的文本文件编码不一致时, 使用字符流读取文本文件就会产生乱码

为了解决这一问题, 我们引入了转换流

2. 分类

a. InputStreamReader : 字符输入转换流

作用：先获取文件的原始**字节输入流**，再将其按照真实的字符集编码转换成**字符输入流**，这样字符输入流中的字符就不会乱码

构造器：

- `public InputStreamReader(InputStream is)`

把原始的 `InputStream` 按照代码的默认编码转换成 `Reader`，与直接使用 `FileReader` 的效果一样

- `public InputStreamReader(InputStream is, String charset)`

把原始的 `InputStream` 按照指定的字符集编码转换成 `Reader`

b. OutputStreamWriter : 字符输出转换流

作用：先获取**字节输出流**，再将其按照指定的字符集编码转换成**字符输出流**，这样写出去的字符就会用我们指定的字符集进行编码

构造器：

- `public OutputStreamWriter(OutputStream os)`

把原始的 `OutputStream` 按照代码的默认编码转换成 `Writer`，与直接使用 `FileWriter` 的效果一样

- `public OutputStreamWriter(OutputStream os, String charset)`

把原始的 `OutputStream` 按照指定的字符集编码转换成 `Writer`

五、打印流

1. PrintStream

作用：打印流可以实现更方便、更高效地打印数据出去

构造器：

- `public PrintStream(OutputStream os/File file/String filepath)`

打印流直接通向字节输出流/文件/文件路径

- `public PrintStream(String fileName, Charset charset)`

可以指定写出去的字符编码

- `public PrintStream(OutputStream os, boolean autoFlush)`

可以指定是否自动刷新

- `public PrintStream(OutputStream os, boolean autoFlush, String encoding)`

既可以指定是否自动刷新，又可以指定写出去的字符编码

方法：

- `void println(xxx xx)`
打印任意类型的数据出去
- `void write(int/byte[]/byte[]的一部分)`
可以支持写字节数据出去

2. PrintWriter

作用: 同 PrintStream

构造器:

- `public PrintWriter(OutputStream os/writer w/File file/string filepath)`
- `public PrintWriter(String fileName,Charset charset)`
- `public PrintWriter(OutputStream os/writer w,boolean autoFlush)`
- `public PrintWriter(OutputStream os,boolean autoFlush,string encoding)`

方法:

- `void println(xxx xx)`
打印任意类型的数据出去
- `void write(int/String/char[]/...)`
可以支持写字符数据出去

3. 应用：输出语句的重定向

介绍: 我们的输出语句默认是打印在控制台上，而利用打印流，我们可以把输出语句打印到某个文件中去

代码:

```
1 try( PrintStream ps=new PrintStream("D:\\ZSH-Computer  
2 Science\\Java\\JavaFile\\a.txt"); ){  
3     //把系统默认的打印流对象改成我们自己设置的打印流  
4     System.setOut(ps);  
5  
6     //打印输出语句到a.txt这个文件中去  
7     System.out.println("...");  
8 }catch(Exception e){  
9     e.printStackTrace();  
}
```

六、数据流

1. DataOutputStream : 数据输出流

作用：允许把**数据和其类型**一并写入磁盘文件中

构造器： `public DataOutputStream(OutputStream os)` 创建新的数据输出流来包装基础的字节输出流

方法：

- `final void writeByte(int v) throws IOException`
将 byte 类型的数据写入基础的字节输出流
- `final void writeInt(int v) throws IOException`
将 int 类型的数据写入基础的字节输出流
- `final void writeDouble(Double v) throws IOException`
将 double 类型的数据写入基础的字节输出流
- `final void writeUTF(String str) throws IOException`
将字符串数据以 UTF-8 编码成字节，然后写入基础的字节输出流
- `void write(int/byte[]/byte[] 的一部分)`
支持写字节数据出去

2. DataInputStream : 数据输入流

作用：读取数据输出流 DataOutputStream 写出去的数据

构造器： `public DataInputStream(InputStream is)` 创建新的数据输入流来包装基础的字节输入流

方法：

- `final byte readByte() throws IOException`
读取字节数据并返回
- `final int readInt() throws IOException`
读取 int 类型的数据并返回
- `final double readDouble() throws IOException`
读取 double 类型的数据并返回
- `final String readUTF() throws IOException`
读取 UTF-8 编码形式的字符串数据并返回
- `int readInt()`
`int read(byte[])`
支持读字节数据进来

七、序列化流

1.前置知识

对象序列化: 把 Java 对象写入到磁盘文件中

对象反序列化: 把磁盘文件中的 Java 对象读出到内存中来

2. ObjectOutputStream:对象字节输出流

作用: 把 Java 对象进行序列化

构造器: `public ObjectOutputStream(OutputStream os)` 创建新的对象字节输出流来包装基础的字节输出流

方法: `final void writeObject(Object o) throws IOException` 把对象写出去

注意: 对象如果要参与序列化，必须实现序列化接口 (`java.io.Serializable`)

3. ObjectInputStream:对象字节输入流

作用: 把 Java 对象进行反序列化

构造器: `public ObjectInputStream(InputStream os)` 创建新的对象字节输入流来包装基础的字节输入流

方法: `final Object readObject() throws IOException` 把对象读进来

4.代码演示:

(1) 先创建一个 Java 对象 `User` 类

```
1 package IO;
2
3 import java.io.Serializable;
4
5 public class User implements Serializable{//必须实现序列化接口
6     (`java.io.Serializable`))!!!
7     /*类变量*/
8     private String loginName;
9     private String userName;
10    private int age;
11    private String password;
12
13    /*无参构造器*/
14    public User(){
15
16    }
17
18    /*有参构造器*/
19    public User(String loginName, String userName, int age, String password)
20    {
```

```
19     this.loginName = loginName;
20     this.userName = userName;
21     this.age = age;
22     this.password = password;
23 }
24
25 /*get and set方法*/
26
27 public String getLoginName() {
28     return loginName;
29 }
30
31 public void setLoginName(String loginName) {
32     this.loginName = loginName;
33 }
34
35 public String getUserName() {
36     return userName;
37 }
38
39 public void setUserName(String userName) {
40     this.userName = userName;
41 }
42
43 public int getAge() {
44     return age;
45 }
46
47 public void setAge(int age) {
48     this.age = age;
49 }
50
51 public String getPassword() {
52     return password;
53 }
54
55 public void setPassword(String password) {
56     this.password = password;
57 }
58
59 /*重写toString方法*/
60 @Override
61 public String toString() {
62     return "User{" +
63             "loginName='" + loginName + '\'' +
64             ", userName='" + userName + '\'' +
65             ", age=" + age +
66             ", password='" + password + '\'' +
67             '}';
68 }
69 }
```

(2) 进行对象的序列化

```
1 package IO;
2
3 import java.io.FileOutputStream;
4 import java.io.ObjectOutputStream;
5 import java.io.Exception;
6
7
8 public class test2 {
9     public static void main(String[] args) {
10         try {
11             (
12                 //1.创建一个ObjectOutputStream, 用于包装原始的FileOutputStream
13                 ObjectOutputStream oos =new ObjectOutputStream(new
14                 FileOutputStream("文件地址"));
15             ) {
16                 //2.创建一个Java对象
17                 User user=new User("admin","zsh",20,"123456");
18
19                 //3.将该对象序列化到文件中去
20                 oos.writeObject(user);
21                 System.out.println("序列化对象成功! ");
22             }
23             catch (Exception e){
24                 e.printStackTrace();
25             }
26         }
27     }
28 }
```

(3) 进行对象的反序列化

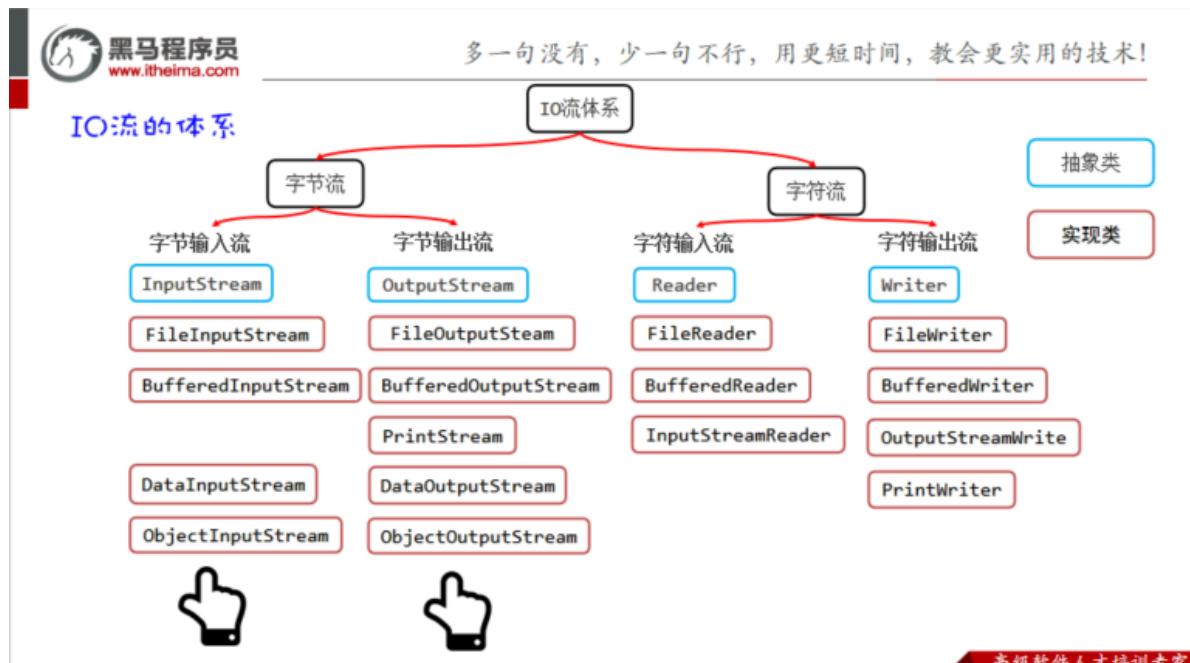
```
1 package IO;
2
3 import java.io.FileInputStream;
4 import java.io.ObjectInputStream;
5 import java.io.Exception;
6
7
8 public class test2 {
9     public static void main(String[] args) {
10         try {
11             (
12                 //1.创建一个ObjectInputStream, 用于包装原始的FileInputStream, 与源文件
13                 //接通
14                 ObjectInputStream ois=new ObjectInputStream(new
15                 FileInputStream("文件地址"));
16             ) {
17                 //2.进行对象的反序列化
18                 User user=(User) ois.readObject();
19                 System.out.println("反序列化对象成功! ");
20             }
21         }
22     }
23 }
```

5.一次性序列化多个对象的方法：

使用一个 `ArrayList` 集合存储多个对象，然后直接对该集合进行序列化即可

注意：`ArrayList` 集合已经实现了序列化接口！

八、总结



九、拓展：IO框架

1.框架的定义：

为了解决某一类问题而编写的一套类和接口，可以理解成一个半成品

2.使用框架的好处：

在框架的基础上进行软件开发，可以得到优秀的软件架构，并且大大提高开发效率

3.框架的形式：

通常是把所有类和接口等编译成 `class` 形式，再压缩成一个 `.jar` 结尾的文件包发行到市面上

4.IO框架:

封装了 Java 提供的对文件和数据进行操作的代码，对外提供了更简单的方式来对文件进行操作、对数据进行读写等

5. Commons-io:

- Commons-io 是 Apache 开源基金组织提供的一组有关IO操作的框架，用于提高IO流的开发效率
- FileUtils 类提供的部分方法：

- `public static void copyFile(File srcFile, File destFile)`
 复制文件 (`srcFile` 是源文件, `destFile` 是目标文件)
- `public static void copyDirectory(File srcDir, File destDir)`
 复制文件夹 (`srcDir` 是源文件夹, `destDir` 是目标文件夹)
- `public static void deleteDirectory(File directory)`
 删除文件夹
- `public static String readFileToString(File file, String encoding)`
 读数据
- `public static void writeStringToFile(File file, String data, String charsetName, boolean append)`
 写数据

- IOUtils 类提供的部分方法：

- `public static int copy(InputStream is, OutputStream os)`
 复制文件
- `public static int copy(Reader r, Writer w)`
 复制文件
- `public static void write(String data, OutputStream os, String charsetName)`
 写数据