

Java-09

一、Task1.字节流

1. **读写**: 既可以单个字节读写, 也可以用数组读写, 数组效率更高

2. **其他流**:

- **字符流**

如果使用字节流处理中文, 一旦将一个字符对应的字节分裂开来, 就会出现乱码了, 为了更方便地处理中文这些字符, Java就推出了 **字符流**

字符流 自带缓存区, 效率更高

- **缓冲字节流(BufferedInputStream)**

缓冲流 在内存中设置一个缓存区, 缓冲区先存储足够的待操作数据后, 再与内存或磁盘进行交互

使用数组读写便是这个原理

3. **开关顺序**:

流的关闭顺序应该与打开顺序相反, 即最后打开的流应该最先关闭



对于 **包装流** (如缓冲流), 应该先关闭外层包装流, 再关闭内层基础流

但开关顺序对于此处代码应该没有影响(流已经使用完毕)

4. 具体代码如下:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class Main {
    public static void main(String[] args) throws Exception {
        FileInputStream fis = new
FileInputStream("F:\\Idea\\Code\\Test\\doro.jpg");
        FileOutputStream fos = new FileOutputStream("doro_copy.jpg");
        byte[] a = new byte[1024];
        int len;
        while ((len = fis.read(a)) != -1) {
            fos.write(a, 0, len);
        }
        fos.close();
        fis.close();
    }
}
```

 doro.jpg	2025/9/21 19:03	JPG 文件	102 KB
 doro_copy.jpg	2025/9/21 19:21	JPG 文件	102 KB

二、Task2.字符流

1. **读取**: 最开始我想要直接使用 **FileReader** 读取文件, 再写入
但这种方式依赖平台默认编码 (可能存在乱码风险) 所以最终采用了提示
2. **处理**: `.filter(s -> !s.isBlank())` 处理空行
`.map(String::trim)` 处理多余空格
`.sorted()` 自然排序
由于使用 `try-with-resources` 语法, 不需要关闭流
3. **写入**: 按要求使用UTF-8编码
`newLine`的优势: 兼容各平台(**`newLine`在各平台生成一致**)

代码如下:

```
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.stream.Collectors;

public class Readtxt {
    public static void main(String[] args) throws IOException{
        try (
            BufferedReader bufferedReader = new BufferedReader
                (new InputStreamReader
                    (new
FileInputStream("F:\\Idea\\Code\\Test\\Zhaoxing\\src\\cn\\org\\glimmer\\Java09\\
name.txt"),StandardCharsets.UTF_8));
            BufferedWriter writer = new BufferedWriter(
                new OutputStreamWriter
                    (new FileOutputStream("name_sorted.txt"),
StandardCharsets.UTF_8)
            )
        ) {
            ArrayList<String> list = new ArrayList<>();
            String line;
            while ((line = bufferedReader.readLine()) != null){
                list.add(line);
            }
            List<String> sortlist = list.stream()
                .filter(s -> !s.isBlank())
                .map(String::trim)
                .sorted()
                .collect(Collectors.toList());
            for (String sortedData : sortlist){
                writer.write(sortedData);
                writer.newLine();
            }
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

三、Task3.序列化与反序列化

1. 让Student类实现 `Serializable` 接口

(如果一个类没有实现 `Serializable` 接口, 尝试对它进行序列化将会抛出 `java.io.NotSerializableException`)

如:

```
Exception in thread "main" java.io.NotSerializableException
```

注意 `serialVersionUID` 应当手动设置

因为在反序列化时, JVM 会检查序列化数据的 `serialVersionUID` 和当前类的 `serialVersionUID` 是否一致。如果不一致, 会抛出 `InvalidClassException`, 防止不兼容的类版本被反序列化

而自动设置的 `serialVersionUID` 可能与反序列化时不一致

2. 使用 `ObjectOutputStream` 和 `ObjectInputStream` 实现读写对象

3. 其他:

- **transient** 关键字可以防止成员变量被序列化, 被修饰的变量会被设置成 **默认值** (如 `null, 0, false`)
- 如果一个父类实现了 `Serializable`, 其子类也是可序列化的
- **静态变量** 不会序列化(因为其属于类而非对象)

详细代码如下:

```
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;}
//其余一致故省略
```

```
import java.io.*;

public class Test09 {
    public static void main(String[] args) throws IOException {
        Student student = new Student(1, "doro", 2, "114514");
        Student doro;
        try (ObjectOutputStream oos = new ObjectOutputStream(
            new FileOutputStream(new File("student.dat"))))
        {
            oos.writeObject(student);
        }
        try (ObjectInputStream ois = new ObjectInputStream(
            new FileInputStream("student.dat")))
        {
            doro = (Student) ois.readObject();
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
        System.out.println(doro);
    }
}
```

输出结果

```
Student{id=1, name='doro', gender=2, phone='114514'}
```