JAVA_IO流

1、java IO流分类

有的流是按照字节的方式读取数据，一次读取1个字节byte等同于一次读取8个二进制位。这种流是万能的，什么类型的文件都可以读取。包括文本文件、图片、声音文件、视频文件等等…

有的流是按照字符的方式读取数据，一次读取一个字符，这种流是为了方便读取普通文本文件存在的。**这种流不能读取：图片、声音、视频等文件，只能读取纯文本文件，连word文件都无法读取。**

```
java.io.InputStream 字节输入流
java.io.OutputStream 字节输出流
java.io.Reader 字符输入流
java.io.Writer 字符输出流

文件专属:
java.io.FileInputStream
java.io.FileOutputStream
java.io.FileReader
java.io.FileWriter

转换流:(将字节流转换成字符流)
java.io.InputStreamReader
java.io.OutputStreamWriter

缓冲流专属:
java.io.BufferedReader
java.io.BufferedWriter
java.io.BufferedInputStream
java.io.BufferedOutputStream

数据流专属:
java.io.DateInputStream
```

```
java.io.DateOutputStream

标准输出流:
java.io.printWriter
java.io.printStream

对象专属流:
java.io.ObjectInputStream
java.io.ObjectOutputStream
```

注意：在java中只要"类名"以Stream结尾的都是字节流。
以"Reader/Writer"结尾的都是字符流。

- `FileInputStream`不涉及字符编码，它直接读取原始字节。
- **`FileReader`依赖于平台的默认字符编码。**这意味着，如果文本文件不是使用平台默认的编码保存的，使用`FileReader`可能会导致乱码。在这种情况下，使用`InputStreamReader`和`FileInputStream`的组合，显式指定文件的字符编码，会是一个更好的选择。
- 当处理非文本文件（如二进制文件）时，使用`FileReader`不适用，因为它会尝试将读取的字节解释为字符，这在处理例如图片或音频文件时没有意义，可能会导致数据损坏。
- 当需要读取的文本文件的编码与平台的默认编码不匹配时，直接使用`FileReader`可能不适用，因为它可能无法正确解码字符。

2、FileInputStream 字节

> **FileInputStream的1个字节读入法**

try-with-resources自动关闭资源

fis.read()

```
public static void testSingleByteRead() {
    try (FileInputStream fis = new
         FileInputStream("E:\\个人学习资料\\Java笔记
\\IO_1.txt")) {
        int readData = fis.read(); // 读取单个字节
        System.out.println(readData); // 输出读取到的
数据
    } catch (FileNotFoundException e) {
        System.err.println("文件未找到: " +
e.getMessage());
        // 在实际应用中，这里可以记录日志或者提供更多错误处
理逻辑
    } catch (IOException e) {
        System.err.println("读取文件时发生错误: " +
e.getMessage());
        // 同样，这里可以进行更复杂的异常处理
    }
}
```

FileInputStream的byte[]读入法

fis.read(bytes)

```
public static void testMultipleByteRead() {
    try (FileInputStream fis =
         new FileInputStream("E:\\个人学习资料\\Java笔
记\\IO_1.txt")) {
        //准备一个4个长度的byte数组，一次最多读取4个字节。
        byte[] bytes = new byte[4];
        int readCount = 0;
        //这个方法的返回值是，读取到的字节数量。（不是字节本
身）
        while ((readCount = fis.read(bytes)) != -1)
{
            // 读取所有的内容  abcdsdsadad
```

```
            System.out.print(new String(bytes, 0,
readCount));
        }
    } catch (FileNotFoundException e) {
        System.err.println("文件未找到: " +
e.getMessage());
        // 在实际应用中，这里可以记录日志或者提供更多错误处
理逻辑
    } catch (IOException e) {
        System.err.println("读取文件时发生错误: " +
e.getMessage());
        // 同样，这里可以进行更复杂的异常处理
    }
}
```

> **int available(): 返回流当中剩余的没有读取到的字节的数量**

fis.available()

```
public static void testAvailable() {
    try (FileInputStream fis =
         new FileInputStream("E:\\个人学习资料\\Java笔
记\\IO_1.txt")) {
        System.out.println("总字节数量:" +
fis.available());// 11
        //读一个字节
        int readDate = fis.read();
        System.out.println("还剩下的可读字节的数量:" +
fis.available());// 10
        //读取剩余的内容
        byte[] bytes = new byte[fis.available()];
        //不需要循环了！直接读一次就行了。
        int readCount = fis.read(bytes);
        System.out.println(readCount); // 10
        System.out.println(new String(bytes)); //
bcdsdsadad
    } catch (FileNotFoundException e) {
```

```
            System.err.println("文件未找到: " +
e.getMessage());
            // 在实际应用中，这里可以记录日志或者提供更多错误处
理逻辑
        } catch (IOException e) {
            System.err.println("读取文件时发生错误: " +
e.getMessage());
        }
}
```

3、FileOutputStream 字节

当在FileOutputStream的构造函数中将第二个参数设为true时，就指定了以追加模式打开文件。这意味着当你向文件写入数据时，新的数据会被追加到文件的末尾，而不是覆盖文件原有的内容。

fos.write(bytes); // 将byte数组全部写出
fos.write(bytes, 0, 2); // 将byte数组的一部分写出

```
public static void testFileWrite() {
    String filePath = "E:\\个人学习资料\\Java笔记
\\IO_2.txt";
    File file = new File(filePath);
    // 检查文件是否存在，如果不存在则尝试创建
    if (!file.exists()) {
        try {
            boolean created = file.createNewFile();
            if (created) {
                System.out.println("文件创建成功");
            } else {
                System.out.println("文件创建失败");
                return; // 如果文件创建失败，则不继续往下
执行
            }
        } catch (IOException e) {
            e.printStackTrace(); // 打印异常信息
```

```
                return; // 如果发生异常，则不继续往下执行
            }
        }
        // 使用try-with-resources确保FileOutputStream正确
关闭
        try (FileOutputStream fos = new
FileOutputStream("E:\\个人学习资料\\Java笔记
\\IO_2.txt", true)) {
            // 开始写入字节数据
            byte[] bytes = {97, 98, 99, 100}; // 对应abcd
            fos.write(bytes); // 将byte数组全部写出
            fos.write(bytes, 0, 2); // 将byte数组的一部分写
出

            // 写入字符串
            String s = "我是中国人! ";
            byte[] bs = s.getBytes(); // 将字符串转换成
byte数组
            fos.write(bs); // 写入字符串对应的byte数组

            // 刷新输出流，确保数据全部写入文件
            fos.flush();
        } catch (IOException e) {
            e.printStackTrace(); // 打印异常信息
        }
    }
```

应用：文件复制

```
public static void fileCopy() {
    String filePath = "E:\\个人学习资料\\Java笔记
\\IO_3.txt";
    File file = new File(filePath);
    // 检查文件是否存在，如果不存在则尝试创建
    if (!file.exists()) {
        try {
```

```
            boolean created = file.createNewFile();
            if (created) {
                System.out.println("文件创建成功");
            } else {
                System.out.println("文件创建失败");
                return; // 如果文件创建失败，则不继续往下
执行
            }
        } catch (IOException e) {
            e.printStackTrace(); // 打印异常信息
            return; // 如果发生异常，则不继续往下执行
        }
    }
    // 使用try-with-resources语句自动管理资源
    try (FileInputStream fis = new
FileInputStream("E:\\个人学习资料\\Java笔记
\\IO_1.txt");
         FileOutputStream fos = new
FileOutputStream("E:\\个人学习资料\\Java笔记
\\IO_3.txt")) {

        byte[] bytes = new byte[1024 * 1024]; // 1MB
（一次最多拷贝1MB）
        int readCount;
        while ((readCount = fis.read(bytes)) != -1)
{
            fos.write(bytes, 0, readCount);
        }

        // try-with-resources会自动关闭资源，所以不需要显
式调用flush
        // fos.flush(); // 输出流在关闭时会自动刷新
    } catch (IOException e) {
        e.printStackTrace(); // 打印异常信息
    }
}
```

4、FileReader字符

reader.read(chars)

```
public static void testFileReader() {
    //创建文件字符输入流
    try (FileReader reader = new FileReader("E:\\个人学习资料\\Java笔记\\IO_3.txt")){
        //开始读
        char[] chars = new char[4]; //一次读取4个字符(1个字符2个字节)
        int readCount = 0;
        while ((readCount = reader.read(chars)) != -1) {
            System.out.print(new String(chars, 0, readCount));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

5、FileWriter字符

> 应用：复制普通文本文件

in.read(chars))

out.write(new String(chars, 0, readCount))

```
public static void copyFile() {
    try(FileReader in = new FileReader("E:\\个人学习资料\\Java笔记\\IO_1.txt");
        FileWriter out = new FileWriter("E:\\个人学习资料\\Java笔记\\IO_4.txt")) {
```

```
        //一边读一边写
        char[] chars = new char[1024 * 512];//1MB
        int readCount = 0;
        while((readCount = in.read(chars)) != -1) {
            out.write(new String(chars, 0,
readCount));
        }
        //刷新
        out.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

6、BufferedInputStream 字节缓冲区

BufferedInputStream是Java I/O库中的一个包装器类，**用于添加缓冲功能到输入流上，使之更加高效**。它是InputStream的一个子类，提供了缓冲读取功能，可以包装其他类型的输入流（如FileInputStream），通过预先从实际输入源（如文件）中读取较大的数据块到内部缓冲区，**减少实际的读取操作次数**，从而提高读取效率。

bis.read(buffer)

```
public static void testBufferInputOutputStream() {
    try (FileInputStream fis = new
FileInputStream("E:\\个人学习资料\\Java笔记
\\IO_1.txt");
         BufferedInputStream bis = new
BufferedInputStream(fis)) {
        byte[] buffer = new byte[1024];
        int bytesRead = 0;
        //从文件中按字节读取内容，到文件尾部时read方法将返
回-1
```

```
        while ((bytesRead = bis.read(buffer)) != -1)
{
            //将读取的字节转为字符串对象
            String chunk = new String(buffer, 0,
bytesRead);
            System.out.print(chunk);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

7、BufferedOutputStream 字节缓冲区

bos.write(97)

```
public static void testBufferOutputStream() {
    try (FileOutputStream fos = new
FileOutputStream("E:\\个人学习资料\\Java笔记
\\IO_5.txt");
         BufferedOutputStream bos = new
BufferedOutputStream(fos)) {
        //开始写
        bos.write(97);
        bos.write(99);
        bos.write(98);
        //刷新（输出流记得刷新）
        bos.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

> 应用：复制普通文本文件

```
public static void fileCopy() {
    String filePath = "E:\\个人学习资料\\Java笔记
\\IO_7.txt";
    File file = new File(filePath);
    // 检查文件是否存在，如果不存在则尝试创建
    if (!file.exists()) {
        try {
            boolean created = file.createNewFile();
            if (created) {
                System.out.println("文件创建成功");
            } else {
                System.out.println("文件创建失败");
                return; // 如果文件创建失败，则不继续往下
执行
            }
        } catch (IOException e) {
            e.printStackTrace(); // 打印异常信息
            return; // 如果发生异常，则不继续往下执行
        }
    }
    // 使用try-with-resources语句自动关闭所有资源
    try (FileInputStream fis = new
FileInputStream("E:\\个人学习资料\\Java笔记
\\IO_1.txt");
         BufferedInputStream bis = new
BufferedInputStream(fis);
         FileOutputStream fos = new
FileOutputStream(filePath);
         BufferedOutputStream bos = new
BufferedOutputStream(fos)) {
        byte[] bytes = new byte[1024 * 1024]; // 1MB
（一次最多拷贝1MB）
        int bytesRead;
        while ((bytesRead = bis.read(bytes)) != -1)
{
            bos.write(bytes, 0, bytesRead);
```

```
        }
        // 在try-with-resources中，不需要显式调用flush，
因为关闭流时会自动调用
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

8、BufferedReader字符缓冲区

br.readLine()

```
public static void testBufferInputOutputStream() {
    try (FileReader fr = new FileReader("E:\\个人学习
资料\\Java笔记\\IO_1.txt");
         BufferedReader br = new BufferedReader(fr))
{
        //br.readLine()方法读取一个文本行，但不带换行符。
        String line = null;
        while((line = br.readLine()) != null) {
            System.out.println(line);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

> 字节流转换成字符流

```
try (FileInputStream fis = new
FileInputStream(filePath);
     InputStreamReader isr = new
InputStreamReader(fis, "UTF-8");
     BufferedReader br = new BufferedReader(isr)) {

    String line;
    while ((line = br.readLine()) ≠ null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

> 应用：复制普通文本文件

```
public static void
bufferedInputOutputStreamTest(String inputFilePath,
String outputFilePath) {
    // 使用try-with-resources语句自动关闭所有资源
    try (FileInputStream fis = new
FileInputStream(inputFilePath);
         BufferedInputStream bis = new
BufferedInputStream(fis);
         FileOutputStream fos = new
FileOutputStream(outputFilePath);
         BufferedOutputStream bos = new
BufferedOutputStream(fos)) {

        byte[] bytes = new byte[1024 * 1024]; // 1MB
(一次最多拷贝1MB)
        int bytesRead;
        while ((bytesRead = bis.read(bytes)) ≠ -1)
{
            bos.write(bytes, 0, bytesRead);
        }
```

```
        // 在try-with-resources中，不需要显式调用flush，因为关闭流时会自动调用
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

9、BufferedWriter字符缓冲区

bw.write

```
public static void testBufferWriter() {
    try (FileWriter fw = new FileWriter("E:\\个人学习资料\\Java笔记\\IO_6.txt");
         BufferedWriter bw = new BufferedWriter(fw))
{
        //开始写
        bw.write("123");
        bw.write("\n");
        bw.write("456");
        //刷新（输出流记得刷新）
        bw.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

10、DataInputStream Java数据类型

DataInputStream的应用场景相对来说是更加局限。

DataInputStream是Java I/O库中的一个类，**它允许应用程序以可移植的方式从底层输入流中读取基本Java数据类型（如int、long、float、double、String等）而不是仅仅是字节。**这种能力使得DataInputStream非常适合读取数据流或者文件的内容，这些内容是按照Java的基本数据类型编码的。

读取特定格式的数据文件：当你有一个数据文件，其中包含了按照一定顺序排列的Java基本类型数据（如二进制数据文件），`DataInputStream`可以直接读取并转换为相应的Java数据类型。

```
public static void testDataInputStream() {
    String filePath = "E:\\个人学习资料\\Java笔记
\\IO_8.txt";
    // 使用try-with-resources自动关闭资源
    try (FileInputStream fis = new
FileInputStream(filePath);
         DataInputStream dis = new
DataInputStream(fis);
        ) {
        // 开始读取各种基本类型的数据
        byte b = dis.readByte();
        short s = dis.readShort();
        int i = dis.readInt();
        long l = dis.readLong();
        float f = dis.readFloat();
        double d = dis.readDouble();
        boolean sex = dis.readBoolean();
        char c = dis.readChar();

        // 打印读取的数据
        System.out.println(b);
        System.out.println(s);
        System.out.println(i);
        System.out.println(l);
        System.out.println(f);
        System.out.println(d);
        System.out.println(sex);
        System.out.println(c);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

11、DataOutputStream Java数据类型

DataOutputStream:数据专属的流 这个流可以将数据连通数据的类型一并写入文件。

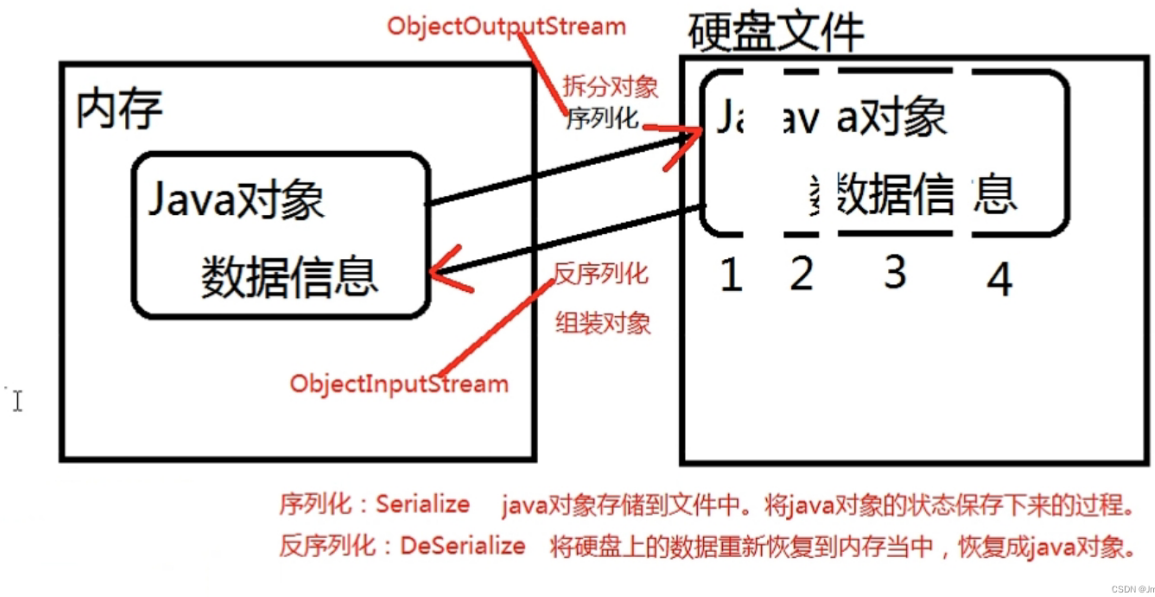
```
public static void testDataOutputStream() {
    String filePath = "E:\\个人学习资料\\Java笔记
\\IO_8.txt";
    File file = new File(filePath);
    // 检查文件是否存在，如果不存在则尝试创建
    if (!file.exists()) {
        try {
            boolean created = file.createNewFile();
            if (created) {
                System.out.println("文件创建成功");
            } else {
                System.out.println("文件创建失败");
                return; // 如果文件创建失败，则不继续往下
执行
            }
        } catch (IOException e) {
            e.printStackTrace(); // 打印异常信息
            return; // 如果发生异常，则不继续往下执行
        }
    }
    // 使用try-with-resources自动关闭资源
    try (FileOutputStream fos = new
FileOutputStream(filePath);
         DataOutputStream dos = new
DataOutputStream(fos)
        ) {
        dos.writeByte(100); // 写入一个byte
        dos.writeShort(30000); // 写入一个short
        dos.writeInt(123456789); // 写入一个int
        dos.writeLong(12345678901L); // 写入一个long
```

```
            dos.writeFloat(123.45f); // 写入一个float
            dos.writeDouble(1234567.89); // 写入一个
double
            dos.writeBoolean(true); // 写入一个boolean
            dos.writeChar('A'); // 写入一个char
            dos.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

12、序列化和反序列化

ObjectInputStream和ObjectOutputStream是Java标准库中用于序列化和反序列化对象的流类。它们使得可以将一个对象转换为字节序列，这样就可以轻松地将其**保存到磁盘上或通过网络发送给其他程序**，然后再还原回原来的对象。

序列化： Serialize **java对象存储到文件中。**
反序列化:DeSerialize **将硬盘上的数据重新恢复到内存当中，恢复成java对象。**



> ObjectInputStream Java对象

```
ObjectInputStream ois = null;

FileInputStream fis = new
FileInputStream("D:\\JAVAEE\\Students.txt");
ois = new ObjectInputStream(fis);
//开始反序列化，读
Object obj = ois.readObject();
//反序列化回来是一个学生对象，所以会调用学生对象的toString方
法。
System.out.println(obj);
ois.close();
```

> ObjectOutputStream Java对象

```
 //创建java对象
Student s = new Student(1111,"zhansan");
ObjectOutputStream oos = null;

FileOutputStream fos = new
FileOutputStream("D:\\JAVAEE\\Students.txt");
//序列化
oos = new ObjectOutputStream(fos);
//序列化对象
oos.writeObject(s);
//刷新
oos.flush();
```

```
List<Student> list = new ArrayList<>();
list.add(new Student(1111, "zhangsan"));
list.add(new Student(2222, "lisi"));
list.add(new Student(3333, "wangwu"));
list.add(new Student(4444, "zhaoliu"));
ObjectOutputStream oos = null;

FileOutputStream fos = new
FileOutputStream("D:\\JAVAEE\\Students.txt");
```

```
oos = new ObjectOutputStream(fos);
//序列化一个集合，这个集合对象中放了很多其他对象。
oos.writeObject(list);
//刷新
oos.flush();


ObjectInputStream ois = null;
FileInputStream fis = new
FileInputStream("D:\\JAVAEE\\Students.txt");
//反序列化
ois = new ObjectInputStream(fis);
List<Student> list = (List<Student>)
ois.readObject();
for(Student student: list) {
    System.out.println(student);
}
```