

Java2 Lab 12

(Java I/o)

[Experimental Objective]

1. 学习理解Java的Io操作
2. 通过编写和调试程序，掌握JAVA序列化和反序列化方法

[Java中的I/O]

Java的输入/输出流中，根据它们的数据类型，主要可分为两类：字符流 (Character Stream) 和字节流 (Byte Stream)。

(1) 字符流

字符流的输入/输出数据是字符码，即Unicode字符。字符流有两个基本类：Reader类和Writer类。

Reader：抽象类，无法直接使用，通过其子类的子类FileReader从文件中获取字符。

Writer：抽象类，无法直接使用，通过其子类的子类FileWriter向文件写入字符。

(2) 字节流

字节流是按字节读/写二进制数据。字节流有两个基本的类：InputStream类和OutputStream类。

InputStream：抽象类，无法直接使用，通过其子类FileInputStream，从文件中获取字节。

OutputStream：抽象类，无法直接使用，通过其子类FileOutputStream，向文件写入字节。

[案例1 字符流]

使用字符流，主要使用FileReader，读取文件中的信息，可以得到一个字符，而且值为ASCII码类型(int)，而且需要经过转换才能变成char型。例子如下：

-
- **public class** Read1 {
- **public static void** main(String[] args) {
- **try** {

- `FileReader fr = new FileReader("File.txt");`
- `System.out.print("文件内容: \n");`
- `int s = fr.read();` //读取到的字符的 ASCII 值, 返回值是 int
- `//ASCII 值转成字符`
- `char ss = (char)s;`
- `System.out.print(ss);`
- `fr.close();`
- `} catch (IOException e) {`
- `System.out.println(e);`
- `}`
- `}`
- `}`

```

Console Problems Progress Debug St
<terminated> Read1 [Java Application] C:\Program Fi
文件内容:
q

```

[案例2 字节流]

字节流是按字节读/写二进制数据的, 下面的例子是读取文件第一行 (即前6个字符), 你会发现`is.read(buffer, 0, 6)`中把6改成7和8都显示的是第一行数据, 这是因为第7个字符是'\n', 第8个是'\r'.

```

1. public class Read2 {
2.
3.     public static void main(String[] args) {
4.         try {
5.             FileInputStream fis = new FileInputStream("File.txt");
6.             System.out.println("文档内容: ");
7.             byte[] buffer = new byte[1024];
8.             fis.read(buffer, 0, 6);
9.             System.out.println(new String(buffer));
10.            fis.close();
11.            System.out.println("reading finished");
12.        } catch (IOException e) {
13.            System.out.println(e);
14.        }
15.    }
16. }

```

```

Package Explorer Type Hierarchy Console
<terminated> Read2 [Java Application] C:\Program F
文档内容:
qwerty
reading finished

```

[案例3 过滤流]

Java过滤流本身不具IO功能，只是在别的流上加上过滤提高效率，像是为别的流装上一一种包装。过滤流分为字节和字符过滤流

字节过滤流为：

BufferedInputStream—字节输入过滤流

BufferedOutputStream—字节输出过滤流

字符过滤流为：

BufferedReader—字符输入过滤流

BufferedWriter—字符输出过滤流

如复制一个文件，不使用过滤流如下：

```
1. public class FileCopy {
2.     public static void main(String[] args) {
3.         try (FileInputStream in = new FileInputStream("./TestDir/src.zip");
4.             FileOutputStream out = new FileOutputStream("./TestDir/subDir/s
rc.zip")) {
5.             //开始时间，当前系统纳秒时间
6.             long startTime = System.nanoTime();
7.             // 准备一个缓冲区
8.             byte[] buffer = new byte[1024];
9.             // 首先读取一次
10.            int len = in.read(buffer);
11.            while (len != -1) {
12.                // 开始写入数据
13.                out.write(buffer, 0, len);
14.                // 再读取一次
15.                len = in.read(buffer);
16.            }
17.            //结束时间，当前系统纳秒时间
18.            long elapsedTime = System.nanoTime() - startTime;
19.            System.out.println("耗时: " + (elapsedTime / 1000000.0) + " 毫秒");
20.        } catch (FileNotFoundException e) {
21.            e.printStackTrace();
22.        } catch (IOException e) {
23.            e.printStackTrace();
24.        }
25.    }
26. }
```

Package Explorer Type Hierar

<terminated> FileCopy [Java Applic

耗时: 3714.898417 毫秒

使用过滤流如下：

```
1. public class FileCopyWithBuffer {
2.     public static void main(String[] args) {
3.         try (FileInputStream fis = new FileInputStream("./TestDir/src.zip");
4.             BufferedInputStream bis = new BufferedInputStream(fis);
5.             FileOutputStream fos = new FileOutputStream("./TestDir/subDir/s
rc.zip");
6.             BufferedOutputStream bos = new BufferedOutputStream(fos)) {
7.             //开始时间
8.             long startTime = System.nanoTime();
9.             // 准备一个缓冲区
10.            byte[] buffer = new byte[1024];
11.            // 首先读取一次
12.            int len = bis.read(buffer);
13.            while (len != -1) {
14.                // 开始写入数据
15.                bos.write(buffer, 0, len);
```

```
16.          // 再读取一次
17.          len = bis.read(buffer);
18.      }
19.      //结束时间
20.      long elapsedTime = System.nanoTime() - startTime;
21.      System.out.println("耗时: " + (elapsedTime / 1000000.0) + " 毫秒");
22.  } catch (FileNotFoundException e) {
23.      e.printStackTrace();
24.  } catch (IOException e) {
25.      e.printStackTrace();
26.  }
27.  }
28. }
```

Package Explorer Type Hierarchy
<terminated> FileCopyWithBuffer [Java App
耗时: 45.812339 毫秒

可以看出，使用过滤流后，显著提高了复制的速度。

【java中的序列化和反序列化】

序列化：把Java对象转换为字节序列的过程称为对象的序列化。

反序列化：把字节序列恢复为Java对象的过程称为对象的反序列化。

对象的序列化主要有两种用途：

- 1) 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中；
- 2) 在网络上传送对象的字节序列。

对象序列化包括如下步骤：

- 1) 创建一个对象输出流，它可以包装一个其他类型的目标输出流，如文件输出流；
- 2) 通过对象输出流的writeObject()方法写对象。

[案例4 序列化]

首先创建一个**student**类如下，切记要把它定义成可以被序列化的类。其成员包括名字，年龄，学号，和考试分数。

```
1.  class Student implements Serializable {
2.      String name;
3.      int age;
4.      String num;
5.      double score;
6.      public Student() {
7.      }
8.      public Student(String name, int age, String num, double score) {
9.          this.name = name;
10.         this.age = age;
11.         this.num = num;
12.         this.score = score;
13.     }
14.     public String toString() {
15.         return name + "\t" + age + "\t" + num + "\t" + score;
16.     }
17. }
```

然后用**student**类创建两个实例，并写入到txt文件：

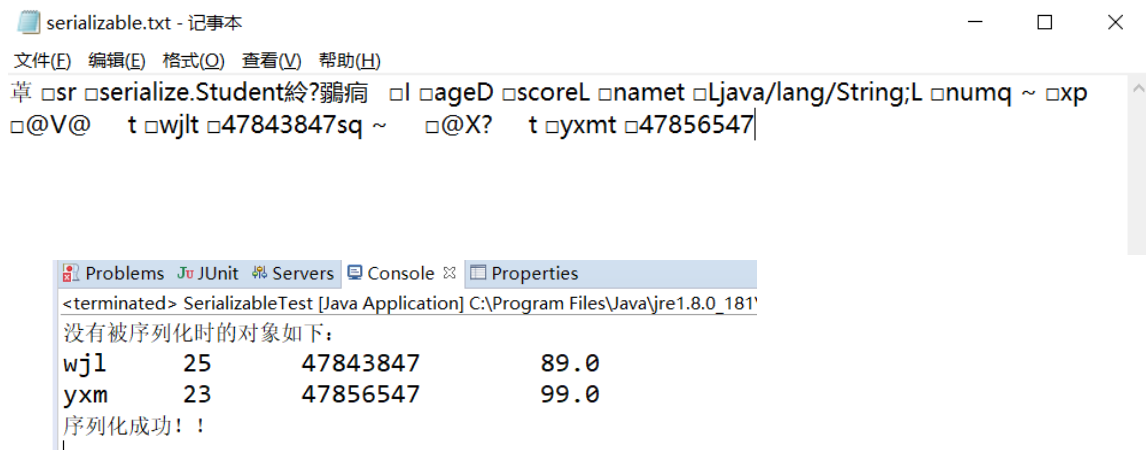
```
1.  import java.io.File;
2.  import java.io.FileOutputStream;
3.  import java.io.IOException;
4.  import java.io.ObjectOutputStream;
5.  public class SerializableTest {
6.      public static void main(String[] args) {
7.          Student stu_1 = new Student("wj", 25, "47843847", 89);
8.          Student stu_2 = new Student("yxm", 23, "47856547", 99);
```

```

9.         File f = new File("serializable.txt");
10.        try {
11.            FileOutputStream fos = new FileOutputStream(f);
12.            ObjectOutputStream oos = new ObjectOutputStream(fos);
13.            System.out.println("没有被序列化时的对象如下: ");
14.            System.out.println(stu_1);
15.            System.out.println(stu_2);
16.            oos.writeObject(stu_1);
17.            oos.writeObject(stu_2);
18.            System.out.println("序列化成功! ! ");
19.            oos.flush();
20.            fos.close();
21.            oos.close();
22.        } catch (IOException e) {
23.            e.printStackTrace();
24.        }
25.    }
26. }

```

可以看到序列化后输出的内容如下:



The screenshot shows a Java IDE with two windows. The top window is a text editor titled 'serializable.txt - 记事本' (serializable.txt - Notepad). It contains the following text:

```

草 □sr □serialize.Student 蛉? 鸚痼 □l □ageD □scoreL □namet □Ljava/lang/String;L □numq ~ □xp
□@V@    t □wjlt □47843847sq ~    □@X?    t □yxmt □47856547|

```

The bottom window is the 'Console' tab, showing the output of the program:

```

<terminated> SerializableTest [Java Application] C:\Program Files\Java\jre1.8.0_181'
没有被序列化时的对象如下:
wj1      25      47843847      89.0
yxm      23      47856547      99.0
序列化成功! !
|

```