Java进阶1 第11天

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【应用】IO概述**
2. 【理解】能够阐述IO的作用
3. 【理解】能够阐述数据在计算机中的表现形式
4. **【应用】字节输入流和字节输出流**
5. 【应用】能够利用字节输出流向文件中写单个字节
6. 【应用】能够利用字节输出流向文件中写一个字节数组
7. 【应用】能够利用字节输入流从文件中读取单个字节
8. 【应用】能够利用字节输入流将文件中的内容读取到字节数组中
9. **【应用】文件复制与高效流**
10. 【应用】 能够利用字节数组和字节流完成文件复制案例
11. 【应用】 能够利用高效字节输出流向文件中写数据

# IO概述

## 字符流回顾

### 案例代码一:

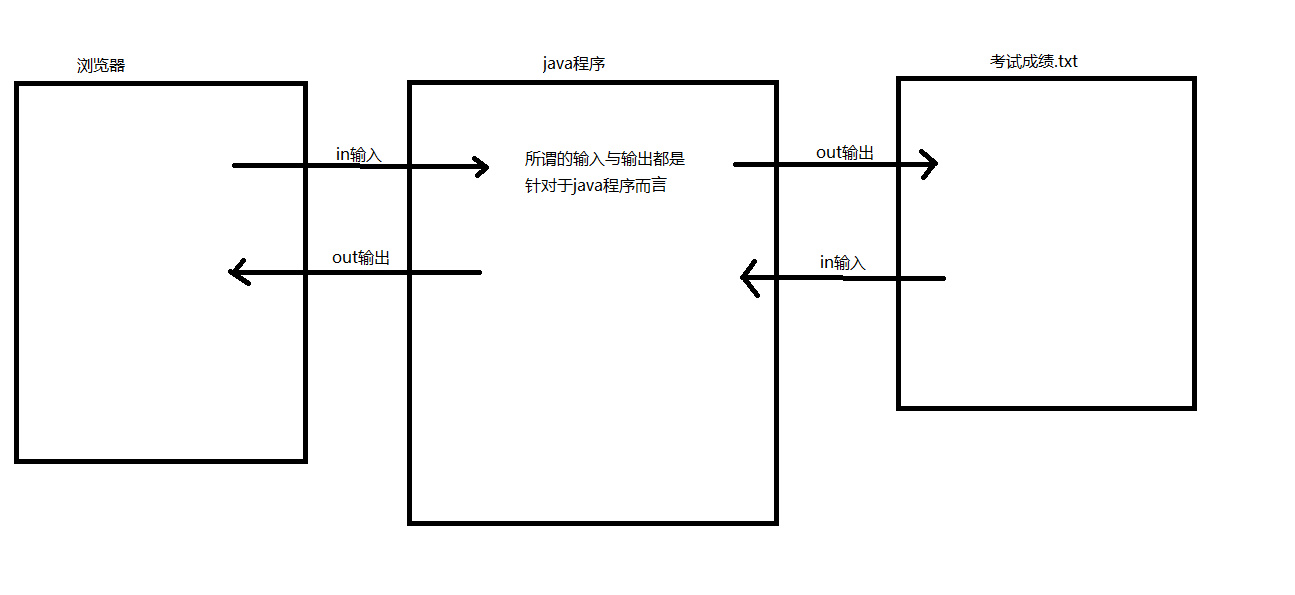
|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileReader;  **import** java.io.FileWriter;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: CharStreamDemo  \* **@Description**: 字符流回顾  \* **@date** 2017年11月22日 上午9:11:26  \* Company www.igeekhome.com  \*  \* 复习应该已经具备的IO知识:  \* 向一个文件中写出字符"hello java",从一个文件中读取字符  \*/  **public** **class** CharStreamDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //写出字符  FileWriter fw = **new** FileWriter("test.txt");  fw.write("hello java");  fw.close();    //读取字符  FileReader fr = **new** FileReader("test.txt");    **int** read = fr.read();  System.***out***.println((**char**)read);    **int** read2 = fr.read();  System.***out***.println((**char**)read2);    fr.close();  }  } |

## IO概念

当使用集合持有数据时，这些数据都存在于内存中，一旦程序运行结束，这些数据将会从内存中清除，下次再想使用这些数据，已经没有了。

如果希望将运算永久地保存下来，可以使用IO，将这些数据持久化存储起来。要把数据持久化存储就需要把内存中的数据存储到内存以外的其他持久化设备(硬盘、光盘、U盘等)上。此时需要数据的输入(in)输出(out)。

数据输入输出相关的类均在io包下。



## 数据本质

在数据传输过程中，一切数据(文本、图像、声音等)最终存储的均为一个个字节，即二进制数字。所以数据传输过程中，使用二进制数据可以完成任意数据的传递。

我们向一个文件中存储一定的数据(一些数字)，如果使用文本方式打开，则会以文本的方式解释数据。如果以视频的方式打开，则会以视频的方式解释数据。音频、可行执行文件等亦是如此。所以，在文件传输过程中，我们要时刻明确，传输的始终为二进制数据。

# 字节输入流与字节输出流

## 字节输出流

输出流：

OutputStream(抽象类)

FileOutputStream(基本输出流)

构造方法：

public FileOutputStream([String](mk:@MSITStore:C:\Users\Administrator\Desktop\第三阶段\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) name)：创建输出流操作的目标

public FileOutputStream([String](mk:@MSITStore:C:\Users\Administrator\Desktop\第三阶段\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) name,boolean append)：append为true代表每次写入都向文件末尾追加,为false则每次都是覆盖写入

操作方法：

write(int n)：输出一个字节；

write(byte[] b)：输出一个字节数组；

write(byte[] b, int off , int len)：输出字节数组的一部分；

close()： 释放IO占用的windows底层资源；

### 案例代码二:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: OutputStreamDemo  \* **@Description**: 字节输出流的使用  \* **@date** 2017年11月22日 上午9:30:13  \* Company www.igeekhome.com  \*  \* OutputStream:字节输出流的根类,定义了所有字节输出流应该具备的方法  \* FileOutputStream:文件输出流是用于将数据写入文件的输出流  \*  \* public abstract void write(int b) 写出一个字节  \* public void write(byte[] b) 写出一个字节数组的字节  \* public void close() 关闭IO资源  \*/  **public** **class** OutputStreamDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建IO流对象,绑定IO资源  FileOutputStream fos = **new** FileOutputStream("test.txt");    //写出数据  //fos.write(0b00000000000000000000000001100001);  //fos.write(0b01100001);  fos.write(97);    //关闭资源  fos.close();  }  } |

### 案例代码三:

写一个字节数组的字节数据

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: OutputStreamDemo2  \* **@Description**: 字节输出流的使用  \* **@date** 2017年11月22日 上午9:33:13  \* Company www.igeekhome.com  \*  \* public void write(byte[] b) 写出一个字节数组的字节  \*  \* 输出流的源代码实现:如果输出时没有该文件,则会先创建该文件,再输出数据  \* 默认的写出是覆盖掉了文件以前的内容,如果想每次运行程序写出数据不覆盖之前的内容,使用特定的构造方法  \* public FileOutputStream(String name,boolean append) append参数:是否追加数据  \*  \* 在FileOutputStream的构造方法当中,指定的io资源可以是文件对象,也可以是文件路径,其本质是相同的.  \* 一般情况下直接使用给予文件路径的方式创建IO流对象,因为源代码中会自动帮助我们创建File对象.  \*/  **public** **class** OutputStreamDemo2 {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建IO流对象  //FileOutputStream fos = new FileOutputStream("test.txt");  FileOutputStream fos = **new** FileOutputStream("test.txt",**true**);    //输出数据  **byte**[] bytes = {97,98,99};  fos.write(bytes);    //关闭资源  fos.close();  }  } |

## 字节输入流

输入流：

InputStream(抽象类)：

FileInputStream(基本输入流)

构造方法：

public FileInputStream([String](mk:@MSITStore:C:\Users\Administrator\Desktop\第三阶段\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) name)

操作方法：

int read()：读取一个字节, 返回的是字节内容本身，读取到末尾返回-1

int read(byte[] b)：读取一个字节数组

close()：释放IO占用的系统底层资源

### 案例代码四:

读一个字节,写入一个字节

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileInputStream;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: InputStreamDemo  \* **@Description**: 字节输入流  \* **@date** 2017年11月23日 上午8:46:12  \* Company www.igeekhome.com  \*  \* InputStream:字节输入流的根类,定义了所有字节输入流应该具备的方法  \* FileInputStream:文件输入流是用于从文件当中读取数据的输入流  \*  \* public int read() 一次读取一个字节,返回的就是这个字节本身,读取到文件末尾返回-1  \*/  **public** **class** InputStreamDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建IO流对象,绑定IO资源  FileInputStream fis = **new** FileInputStream("test.txt");    //读取数据  //int c = fis.read();  //System.out.println(c);  //System.out.println(Integer.toBinaryString(c));  //System.out.println(fis.read());  //System.out.println(fis.read());  //System.out.println(fis.read());    //使用循环反复读取文件中的内容,直到读取结束为止  **int** c=-1;    **while**((c=fis.read())!=-1) {  System.***out***.println(c);  }  //关闭IO流对象  fis.close();  }  } |

### 案例代码五:

读一个字节数组的数据

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileInputStream;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: InputStreamDemo2  \* **@Description**: 字节输入流  \* **@date** 2017年11月23日 上午8:51:03  \* Company www.igeekhome.com  \*  \* 一次读取一个字节数组  \* int read(byte[] b)：读取一个字节数组  \* 返回值是本次读取到的字节的个数  \* 参数用于每次存储字节数据的数组  \* 读取到文件末尾返回-1  \*/  **public** **class** InputStreamDemo2 {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建IO流对象,绑定IO资源  FileInputStream fis = **new** FileInputStream("test.txt");    //读入数据  //byte[] bytes = new byte[3];  //int len = fis.read(bytes);  //System.out.println(len);  //System.out.println(Arrays.toString(bytes));  //System.out.println(fis.read(bytes));  //System.out.println(Arrays.toString(bytes));  //System.out.println(fis.read(bytes));  //System.out.println(Arrays.toString(bytes));  //System.out.println(fis.read(bytes));  //System.out.println(Arrays.toString(bytes));    //定义数组,用于接受每次读取到的字节数据,一般数组长度为1024或者其整数倍  **byte**[] bytes = **new** **byte**[3];  //定义变量,用于记录每次读取到的字节长度  **int** len=-1;  //循环读取,直到文件末尾  **while**( (len=fis.read(bytes)) != -1) {  //将字节数组变成字符串,方便观察(通过字符串的构造方法)  //本次读取到几个字节,就转换成几个字节为字符串  String s = **new** String(bytes,0,len);  //打印字符串  System.***out***.println(s);  }  //关闭流  fis.close();  }  } |

# 文件复制与高效流

## 文件复制不使用字节数组

利用字节流将当前工程下的ASCII码表完整版.doc复制到当前工程下的ASCII码表完整版(副本).doc

### 案例代码六:

A：分析:

最终两个文件中的数据一致

1. 利用字节输入流关联ASCII码表完整版.doc文件
2. 利用字节输出流关联ASCII码表完整版(副本).doc文件
3. 循环从ASCII码表完整版.doc读一个字节,然后将这个字节写入到ASCII码表完整版(副本).doc,直到读到ASCII码表完整版.doc的末尾

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: CopyFileDemo  \* **@Description**: 文件复制不使用字节数组  \* **@date** 2017年11月23日 上午8:58:46  \* Company www.igeekhome.com  \*  \* 分析：  \* 1. 利用字节输入流关联ASCII码表完整版.doc文件  \* 2. 利用字节输出流关联ASCII码表完整版(副本).doc文件  \* 3. 循环从ASCII码表完整版.doc读一个字节,  \* 然后将这个字节写入到ASCII码表完整版(副本).doc,  \* 直到读到ASCII码表完整版.doc的末尾  \* 即先读后写  \*/  **public** **class** CopyFileDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //使用一次一个字节的方式完成复制  //创建IO流对象绑定IO资源  FileInputStream fis = **new** FileInputStream("ASCII码表完整版.doc");  FileOutputStream fos = **new** FileOutputStream("ASCII码表完整版(副本).doc");    //读取数据,读一个字节,写一个字节  //先读  //定义变量c,用于存储每次读取到的字节  **int** c=-1;  **while**((c=fis.read())!=-1) {  //后写  fos.write(c);  }    //关闭流  fos.close();  fis.close();  }  } |

## 文件复制使用字节数组

利用字节流和字节数组将当前工程下的ASCII码表完整版.doc复制到当前工程下的ASCII码表完整版(副本).doc

### 案例代码七:

A：分析:

最终两个文件中的数据一致

1. 定义一个长度为1024整数倍的字节数组
2. 利用字节输入流关联ASCII码表完整版.doc文件
3. 利用字节输出流关联ASCII码表完整版(副本).doc文件
4. 循环从ASCII码表完整版.doc读一个字节数组,然后将这个字节数组中有效的内容写入到ASCII码表完整版(副本).doc,直到读到ASCII码表完整版.doc的末尾

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: CopyFileDemo  \* **@Description**: 文件复制使用字节数组  \* **@date** 2017年11月23日 上午9:05:23  \* Company www.igeekhome.com  \*  \* 分析：  \* 1. 定义一个长度为1024整数倍的字节数组  \* 2. 利用字节输入流关联ASCII码表完整版.doc文件  \* 3. 利用字节输出流关联ASCII码表完整版(副本).doc文件  \* 4. 循环从ASCII码表完整版.doc读一个字节数组,  \* 然后将这个字节数组中有效的内容写入到ASCII码表完整版(副本).doc,  \* 直到读到ASCII码表完整版.doc的末尾  \* 即先读后写  \*/  **public** **class** CopyFileDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //使用一次一个字节数组的方式完成复制  //创建IO流对象绑定IO资源  FileInputStream fis = **new** FileInputStream("ASCII码表完整版.doc");  FileOutputStream fos = **new** FileOutputStream("ASCII码表完整版(副本).doc");  //读取数据,读一个字节数组,写一个字节数组    //定义字节数组,用于存储每次读取到的数据  **byte**[] bytes = **new** **byte**[1024];  //定义变量,用于记录每次读取到数据的长度  **int** len=-1;  //先读  **while**((len = fis.read(bytes))!=-1) {  //后写  fos.write(bytes,0,len);  }    //关闭流  fos.close();  fis.close();  }  } |

## 高效流

Java在常规IO流的基础上，提供了更为高效的缓冲流，如下：

高效字节缓冲流：BufferedInputStream/BufferedOutputStream

高效流使用普通流对象作为构造方法参数。将普通流包装，提供高效的装饰。

高效流write写出数据时，写出位置为缓冲区，并非目标资源。需要通过flush刷新方法将缓冲区的内容写出到目标文件中。

高效输出流的关闭方法会自动调用flush方法。

### 案例代码八:

|  |
| --- |
| **package** com.igeek\_03;  **import** java.io.BufferedOutputStream;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: BufferedStreamDemo  \* **@Description**: 高效字节流  \* **@date** 2017年12月1日 上午9:47:06  \* Company www.igeekhome.com  \*  \* 高效原理：缓冲区临时存储多个数据，一次调用底层资源，减少调用底层资源的次数，提高了程序运行的速度  \*  \* 高效字节缓冲流：  \* BufferedInputStream/BufferedOutputStream  \*  \* 高效流的方法与普通流完全一致  \* 高效流使用普通流对象作为构造方法参数。在普通流的基础上增加了缓冲区。  \*  \* 高效流write写出数据时，写出位置为缓冲区，并非目标资源。  \* 需要通过flush刷新方法将缓冲区的内容写出到目标文件中。  \*  \* public void flush() 刷新缓冲区，将内容写到目标文件中  \* public void close() 会调用flush方法  \*/  **public** **class** BufferedStreamDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //以高效字节输出流为例  //创建普通流对象  FileOutputStream fos = **new** FileOutputStream("test.txt");  //通过普通流对象创建高效流对象，这个过程加入了缓冲区  BufferedOutputStream bos = **new** BufferedOutputStream(fos);    //高效流的方法与普通流的方法一致，写出数据  bos.write(97);  bos.write(**new** **byte**[]{98,99,100});    //刷新数据  bos.flush();    //关闭流,关闭高效流即可，高效流会关闭普通流  bos.close();  }  } |

重点和总结

1、IO流概述，字符流的回顾

2、字节输入流和字节输出流

3、使用字节流完成文件的复制

4、高效流的使用