Java进阶1 第12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】字符流概述**
2. 【理解】能够阐述字符流的作用
3. 【应用】能够独立使用字符输出流的write(int n)和write(char[] c)方法
4. 【应用】能够独立使用字符输入流的read()和read(char[] c)方法
5. **【应用】高效字符流**
6. 【应用】能够独立使用BufferedWriter的write(String s)
7. 【应用】能够独立使用BufferedReader的read(intch)和read(char[] ch,intoff,intlen)方法
8. 【应用】能够独立使用BufferedWriter的newLine()方法
9. 【应用】能够独立使用BufferedReader的readLine()方法
10. **【应用】转换流**
11. 【应用】 能够利用OutputStreamWrtier将字节输出流转换为字符输出流
12. 【应用】 能够利用InputStreamReader将字节输入流转换为字符输入流
13. 【应用】 能够利用转换流指定编码表解决乱码问题
14. **【应用】其它流**
15. 【应用】 能够独立使用序列化流和反序列化流存储和读取对象
16. 【应用】 能够独立导入commonsIO相关jar包,并使用里面的类和方法
17. 【应用】 能够将自己写的类导出为jar包,并且导入其它工程使用

# 字符流概述

## 字符流概述

在IO开发过程中，我们传输最频繁的数据为字符，而以字节方式传输字符需要每次将字符串转换成字节再处理，而且也丧失了程序员对数据内容的判断(因为程序员只认识字符，不认识字节)。

所以，为了让程序员方便对字符进行操作，Java提供了专门以字符作为操作单位的类——字符流，其底层仍然为字节流。

显然，字符流只能操作字符，无法操作其他数据，如声音、视频等。

## 字符输出流

输出流：

Writer(抽象类)

FileWriter(基本字符输出流)

构造方法：

public FileWriter([String](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\第三阶段\\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html" \o "java.lang 中的类) fileName) 创建覆盖写出对象

public FileWriter([File](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\第三阶段\\JDK_API_1_6_zh_CN.CHM::/java/io/File.html" \o "java.io 中的类) file,boolean append) 创建指定是否追加写出对象

其他方法：

write(int n)：输出一个字符(使用int替代了char)

write(char[] c)：输出一个字符数组

write(char[] c , int off , int len)：输出字符数组的一部分

write(String s)：输出一个字符串

write(String s , int off , int len)：输出字符串的一部分

flush()：刷新缓冲区，将内容写出到指定IO资源

close()：由于每个IO流都需要绑定一个IO资源，在使用时，需要回收资源。而字符流输出流在关闭时，会先调用flush方法。

### 案例代码一:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileWriter;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: WriterDemo  \* **@Description**: 字符输出流  \* **@date** 2017年11月23日 上午9:21:48  \* Company www.igeekhome.com  \*  \* 字符输出流:  \* Writer字符输出流的根类,具备所有字符输出流所应该具备的方法  \* FileWriter文件字符输出流  \*  \* 构造方法：需要绑定IO资源  \* public FileWriter(String fileName) 创建覆盖写出对象  \* public FileWriter(File file,boolean append) 创建指定是否追加写出对象  \*  \* 普通方法:  \* write(int n)：输出一个字符(使用int替代了char)  \* write(char[] c)：输出一个字符数组  \* write(char[] c , int off , int len)：输出字符数组的一部分  \* write(String s)：输出一个字符串  \* write(String s , int off , int len)：输出字符串的一部分  \* 字符串>>>二进制1010101 >>写出二进制101010 >>  \* 文本文件打开时,将二进制101010翻译成对应的字符显示  \*  \* flush()：刷新缓冲区,将内容写出到指定IO资源,字符流都带缓冲区,都必须刷新.  \* 将"我爱JAVA"字符打碎成二进制字节10xxx1010x 写到缓冲区中,通过刷新一次性写出  \* close()：由于每个IO流都需要绑定一个IO资源,在使用时,需要回收资源.  \* 而字符流输出流在关闭时,会先调用flush方法.  \*/  **public** **class** WriterDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建流对象  FileWriter fw = **new** FileWriter("test.txt");    //写出数据  fw.write('a');  **char**[] arr = {'我','爱','j','a','v','a'};  fw.write(arr);  fw.write(arr,2,4);  fw.write("我爱java!!");    //省略刷新,直接用关闭资源刷新  fw.close();  }  } |

## 字符输入流

输入流：

Reader(抽象类)

FileReader(基本字符输入流)

构造方法：需要绑定IO资源

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

其他方法：

读取时，返回数据时，使用int替代了char

int read()：读取一个字符；

int read(char[] c)：读取一个字符数组；

close()：由于每个IO流都需要绑定一个IO资源，在使用时，需要回收资源。

### 案例代码二:

|  |
| --- |
| **package** com.igeek\_03;  **import** java.io.FileReader;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: ReaderDemo  \* **@Description**: 字符输入流  \* **@date** 2017年11月23日 上午9:29:32  \* Company www.igeekhome.com  \*  \* 输入流：  \* Reader(抽象类)  \* FileReader(基本字符输入流)  \* 构造方法：需要绑定IO资源  \* public FileReader(String name)  \*  \* 其他方法：读取时，返回数据时，使用int替代了char  \* int read()：读取一个字符；  \* int read(char[] c)：读取一个字符数组；  \* close()：由于每个IO流都需要绑定一个IO资源，在使用时，需要回收资源。  \*/  **public** **class** ReaderDemo {  **public** **static** **void** main(String[] args) **throws** IOException {    //一次读取一个字符  FileReader fr = **new** FileReader("test.txt");  //定义变量,记录每次读取到的字符  **int** c=-1;  **while**((c=fr.read())!=-1) {  System.***out***.println((**char**)c);  }  fr.close();  System.***out***.println("================");    //一次读取一个字符数组  FileReader fr2 = **new** FileReader("test.txt");  //定义字符数组,用于存储每次读取到的字符  **char**[] arr = **new** **char**[1024];  //定义变量,记录每次读取到的字符个数  **int** len=-1;  **while**((len=fr2.read(arr))!=-1) {  //字符数组比较特殊,直接打印引用变量,不是打印的地址,而是内容.  //System.out.println(arr);  //可以将字符数组包装成字符串,再打印  String s = **new** String(arr,0,len);  System.***out***.println(s);  }  fr2.close();  }  } |

# 高效字符流

## 高效字符流基本使用

Java在常规IO流的基础上，提供了更为高效的缓冲流，如下：

高效字符流：BufferedReader/BufferedWriter

### 案例代码三:

A：需求：

利用高效字符流完成以下操作

1.向当前工程下的test.txt中写入"i miss u,i want u,i with u"

2.读取当前工程下的test.txt文件中的内容

3.将读取到的内容打印到控制台上

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.BufferedReader;  **import** java.io.BufferedWriter;  **import** java.io.FileReader;  **import** java.io.FileWriter;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: BufferedWriterReaderDemo  \* **@Description**: 高效字符流基本使用  \* **@date** 2017年11月23日 上午9:38:07  \* 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  \*  \* 高效缓冲流:  \* BufferedReader/BufferedWriter  \*  \* 高效流字符流的方法比普通流的方法更多  \* 高效流的构造方法接收普通流对象,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()方法会调用flush方法  \*/  **public** **class** BufferedWriterReaderDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建普通流对象  FileWriter fw = **new** FileWriter("test.txt");  //创建高效流对象  BufferedWriter bw = **new** BufferedWriter(fw);    //写  bw.write("i miss u,i want u,i with u");  //刷新数据  bw.flush();  //关闭高效流对象  bw.close();    //创建普通流对象  FileReader fr = **new** FileReader("test.txt");  //创建高效流对象  BufferedReader br = **new** BufferedReader(fr);    //读  **char**[] chars = **new** **char**[1024];  //获取读取到的字符个数  **int** len = br.read(chars);  //将读取到的数据转成字符串  String s = **new** String(chars,0,len);  System.***out***.println(s);    //关闭高效流对象  br.close();  }  } |

## 高效字符输出流的特殊方法

void newLine(): 根据当前的系统，写入一个换行符,

在windows系统下调用该方法,写入的是\r\n

在mac系统下调用该方法写入的是\r

在linux系统下调用该方法写入的是\n

### 案例代码四:

A：需求:

向test.txt中写入三行"i miss u,i want u,i with u"

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.BufferedReader;  **import** java.io.BufferedWriter;  **import** java.io.FileReader;  **import** java.io.FileWriter;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: BufferedWriterReaderDemo  \* **@Description**: 高效字符输出流的特殊方法  \* **@date** 2017年11月23日 上午10:03:28  \* 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  \*  \* 高效缓冲流:  \* BufferedReader/BufferedWriter  \*  \* 换行时,每种环境对换行的识别不同.windows识别\r\n,有的识别\r有的识别\n  \* BufferedWriter特殊方法:  \* public void newLine() 根据不同的环境生成不同的换行标记  \*/  **public** **class** BufferedWriterReaderDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建普通流对象  FileWriter fw = **new** FileWriter("test.txt");  //创建高效流对象  BufferedWriter bw = **new** BufferedWriter(fw);    //写  bw.write("i miss u,i want u,i with u");    //换行时,每种环境对换行的识别不同.windows识别\r\n,有的识别\r有的识别\n  //bw.write("\n");  //调用高效流的换行方法  bw.newLine();    bw.write("i miss u,i want u,i with u");  bw.newLine();    bw.write("i miss u,i want u,i with u");  bw.newLine();  //刷新数据  bw.flush();  //关闭高效流对象  bw.close();    //创建普通流对象  FileReader fr = **new** FileReader("test.txt");  //创建高效流对象  BufferedReader br = **new** BufferedReader(fr);    //读  **char**[] chars = **new** **char**[1024];  //获取读取到的字符个数  **int** len = br.read(chars);  //将读取到的数据转成字符串  String s = **new** String(chars,0,len);  System.***out***.println(s);    //关闭高效流对象  br.close();  }  } |

## 高效字符输入流的特殊方法

public [String](mk:@MSITStore:D:\\黑马基础班\\黑马04期\\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html" \o "java.lang 中的类) readLine()

读取一个文本行，包含该行内容的字符串，不包含任何行终止符，

**如果已到达流末尾，则返回 null**

### 案例代码五:

|  |
| --- |
| **package** com.igeek\_03;  **import** java.io.BufferedReader;  **import** java.io.FileReader;  **import** java.io.IOException;  /\*\*  \* **@ClassName**: BufferedWriterReaderDemo  \* **@Description**: 高效字符输入流的特殊方法  \* **@date** 2017年11月23日 上午10:08:42  \* 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  \*  \* 高效缓冲流:  \* BufferedReader/BufferedWriter  \*  \* BufferedReader特殊方法:  \* public String readLine() 读取数据,一次读取一行,读到末尾返回null  \*/  **public** **class** BufferedWriterReaderDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //一次读取一个字符,回车换行也会读取进来  //创建普通流对象  FileReader fr = **new** FileReader("test.txt");  //创建高效流对象  BufferedReader br = **new** BufferedReader(fr);    //获取读取到的字符个数  **int** c=-1;  **while**((c=br.read())!=-1) {  System.***out***.println((**char**)c);  }    //关闭资源  br.close();    System.***out***.println("=======================");  //使用高效流特殊方法,一次读取一行  //创建普通流对象  FileReader fr2 = **new** FileReader("test.txt");  //创建高效流对象  BufferedReader br2 = **new** BufferedReader(fr2);    //System.out.println(br2.readLine());  //System.out.println(br2.readLine());  //System.out.println(br2.readLine());  //System.out.println(br2.readLine());  //System.out.println(br2.readLine());  //System.out.println(br2.readLine());  //System.out.println(br2.readLine());    //定义变量,用于记录每次读取到的字符串  String s2=**null**;  **while**((s2 = br2.readLine())!=**null**) {  System.***out***.println(s2);  }    //关闭资源  br2.close();  }  } |

## 高效流练习

需求:将文件test.txt中所有名称读取,并倒序再输入进test.txt中

### 案例代码六:

|  |
| --- |
| **package** com.igeek\_04;  **import** java.io.BufferedReader;  **import** java.io.BufferedWriter;  **import** java.io.FileReader;  **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.util.ArrayList;  **import** java.util.Collections;  /\*\*  \* **@ClassName**: BufferedWriterReaderTest  \* **@Description**: 将文件test.txt中所有名称读取,并倒序再输入进test.txt中  \* **@date** 2017年11月23日 上午10:20:44  \* Company www.igeekhome.com  \*  \* 将文件g.txt中所有名称读取,并倒序再输入进g.txt中  \*  \* 1.定义集合ArrayList<String>存储姓名  \* 2.使用高效流对象,一次读取一行(每次读取一个姓名)  \* 3.将读取到的姓名放到集合中  \* 4.Collections的reverse方法反转集合内容  \* 5.使用高效流,一次写出一个姓名  \*/  **public** **class** BufferedWriterReaderTest {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建普通流对象  FileReader fr = **new** FileReader("test.txt");  //创建高效流对象  BufferedReader br = **new** BufferedReader(fr);    //定义集合ArrayList<String>存储姓名  ArrayList<String> names = **new** ArrayList<String>();  //读取内容,一次读取一行  String thisName=**null**;  **while**((thisName = br.readLine())!=**null**) {  //将读取到的姓名放到集合中  names.add(thisName);  }  //关闭流  br.close();    //使用Collections的工具类方法,反转集合中内容  Collections.*reverse*(names);    //创建普通流对象  FileWriter fw = **new** FileWriter("test.txt");  //创建高效流对象  BufferedWriter bw = **new** BufferedWriter(fw);  //迭代集合内容  **for**(String name : names) {  //将每个姓名写出到g.txt中  bw.write(name);  bw.newLine();  }  //关闭流  bw.close();  }  } |

# 转换流

## 转换流

操作纯文本文件时，使用字节流对于程序来说过于繁琐，程序员更偏爱操作字符。所以，当面对一些字节流的操作时，我们可以将其转换为字符流再进行操作，这样便非常方便了。

Reader:

InputStreamReader可以完成字节输入流转换为字符输入流

Writer:

OutputStreamWriter可以完成字节输出流转换为字符输出流

由上边可以知道，转换流是字符流的一种，创建对象时传入对应字节流对象即可完成转换动作。

转换流同样使用了包装的思想，其构造方法接收的同样为IO流对象，并非某个文件资源。关闭转换流的同时即关闭了对应的字节流。

### OutputStreamWriter

#### 案例代码七:

需求:利用转换流向test.txt中写入"hello:中国"

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.OutputStreamWriter;  /\*\*  \* **@ClassName**: OutputStreamWriterDemo  \* **@Description**: OutputStreamWriter的使用  \* **@date** 2017年11月23日 上午10:47:22  \* Company www.igeekhome.com  \*  \* 需求:  \* 已经具备字节流,绑定了IO资源  \* 需要将字节流转成字符流  \*  \* 此时,可以使用转换流,接收一个字节流对象,进行字符流的操作.  \* 转换流本身就是字符流  \*  \* 转换流:  \* Reader:  \* InputStreamReader可以完成字节输入流转换为字符输入流  \* Writer:  \* OutputStreamWriter可以完成字节输出流转换为字符输出流。  \*  \* 则转换流的使用方法:  \* 构造方法传入字节流对象  \* 自身调用字符流的方法  \*/  **public** **class** OutputStreamWriterDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建字节输出流  FileOutputStream fos = **new** FileOutputStream("test.txt");  //将字节流转为字符流,即通过字节流对象创建转换流对象  OutputStreamWriter osw = **new** OutputStreamWriter(fos);  //使用字符流的方法  osw.write("hello:中国");  //与高效流类似,转换流包含字节流,所以直接关闭转换流,字节流也关闭了  osw.close();  }  } |

### InputStreamReader

#### 案例代码八:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  /\*\*  \* **@ClassName**: InputStreamReaderDemo  \* **@Description**: InputStreamReaderDemo的使用  \* **@date** 2017年11月23日 上午10:55:03  \* Company www.igeekhome.com  \*  \* 需求:  \* 已经具备字节流,绑定了IO资源  \* 需要将字节流转成字符流  \*  \* 此时,可以使用转换流,接收一个字节流对象,进行字符流的操作.  \* 转换流本身就是字符流  \*  \* 转换流:  \* Reader:  \* InputStreamReader可以完成字节输入流转换为字符输入流  \* Writer:  \* OutputStreamWriter可以完成字节输出流转换为字符输出流。  \*  \* 则转换流的使用方法:  \* 构造方法传入字节流对象  \* 自身调用字符流的方法  \*/  **public** **class** InputStreamReaderDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //对比字节流操作字符与字符流操作字符的区别  //普通字节流  FileInputStream fis = **new** FileInputStream("test.txt");    **int** b=-1;  **while**((b=fis.read())!=-1) {  System.***out***.println((**char**)b);  }    fis.close();  System.***out***.println("=====================");    //字符流(这里使用转换流转成一个字符流)  FileInputStream fis2 = **new** FileInputStream("test.txt");  InputStreamReader isr = **new** InputStreamReader(fis2);    **int** c=-1;  **while**((c=isr.read())!=-1) {  System.***out***.println((**char**)c);  }    isr.close();  }  } |

## 编码表

### 编码表概述

我们知道计算机底层数据存储的都是二进制数据，而我们生活中的各种各样的数据，如何才能和计算机中存储的二进制数据对应起来呢？

这时老美他们就把每一个字符和一个整数对应起来，就形成了一张编码表，老美他们的编码表就是ASCII表。其中就是各种英文字符对应的编码。

编码表：其实就是生活中字符和计算机二进制的对应关系表。

1、ASCII：一个字节中的7位表示。对应的字节都是正数：0-xxxxxxx。

2、ISO-8859-1：拉丁latin码表，用一个字节8位表示。1-xxxxxxx：负数。

3、GB2312：简体中文码表。包含6000-7000中文和符号。用两个字节表示。两个字节第一个字节是负数，第二个字节可能是正数。

GBK：目前最常用的中文码表，2万的中文和符号。用两个字节表示，其中的一部分文字，第一个字节开头是1，第二字节开头是0。

GB18030：最新的中文码表，目前还没有正式使用。

4、Unicode：国际标准码表：无论是什么文字，都用两个字节存储。

Java中的char类型用的就是这个码表：char c = 'a';占两个字节。

5、UTF-8：基于Unicode，一个字节就可以存储数据，不要用两个字节存储，而且这个码表更加的标准化，中文一般使用3个字节表示。

能识别中文的码表：GB2312、GBK、Unicode、UTF-8

### 乱码问题

A:乱码：

当字符与字节转换过程中使用了不同的码表，会造成乱码的情况。

B:在字符串中:

当我们将字符串转为对应的数字字节时，需要指定码表，则存储为该 字符该码表对应的数字字节，如果使用了其他码表重写翻译回字符串，则 拼写的新字符串会乱码。

C:在IO中:

与字符串编码表使用类似，当以某个码表写出字节数据时，又使用另外 码表展示，会出现乱码。

#### 案例代码九:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **import** java.io.OutputStreamWriter;  /\*\*  \* **@ClassName**: CharsetDemo  \* **@Description**: 字符编码问题  \* **@date** 2017年11月23日 上午10:59:47  \* Company www.igeekhome.com  \*  \* 编码表:  \* 编码表即字符与存储数据的对应关系表，每一个字符都对应一个数字。  \* 字符 = 字节 + 编码表  \*  \* 转换的构造方法当中:  \* public Output StreamWriter(OutputStream out, String charsetName)  \* public Input StreamReader(InputStream in, String charsetName)  \* 默认都会使用GBK码表,也可以自己指定编码表  \*  \* 乱码:  \* 用一个A编码表将字符转成字节,又用B编码表与将字节转回字符,两个码表对应的关系不同,查到了不相同的结果  \*  \* 保证不乱码的方式:编码与解码保持相同.  \* 中后期,所有的码表均为UTF-8  \*/  **public** **class** CharsetDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //使用UTF-8的方式将字符输出到文件中,文件使用GBK观看会乱码,使用UTF-8观看时正常的  FileOutputStream fos = **new** FileOutputStream("test.txt");  OutputStreamWriter osw = **new** OutputStreamWriter(fos,"UTF-8");  osw.write("你好");  osw.close();    //如果用UTF-8的方式写出两个汉字(6个字节),在那个写出的文件中观看时乱码了.这时再使用UTF-8的方式从那个文件中读回,还会乱码么?  FileInputStream fis = **new** FileInputStream("test.txt");  InputStreamReader isr = **new** InputStreamReader(fis,"UTF-8");  System.***out***.println((**char**)isr.read());  System.***out***.println((**char**)isr.read());  isr.close();  }  } |

# 其它流

## 序列化与反序列化

### 概述

用于向流中写入对象的操作流 ObjectOutputStream称为序列化流

用于从流中读取对象的操作流 ObjectInputStream称为反序列化流

特点：用于操作对象。可以将对象写入到文件中，也可以从文件中读取对象。

ObjectOutputStream序列化流

public final void writeObject(Object obj)

ObjectInputStream反序列化流

public final Object readObject()

每个被序列化操作的类型必须实现Serializable接口(具备序列化功能的标记接口),让其具备序列化能力。

### 对象序列化流ObjectOutputStream

ObjectOutputStream将Java对象写入OutputStream，利用输出流将对象写入到指定的文件中。

#### 案例代码十:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.Serializable;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月23日 下午1:43:01  \* Company www.igeekhome.com  \*  \* 普通类需要实现序列化接口,开启其序列化功能  \*/  **public** **class** Person **implements** Serializable{  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;    /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;    /\*\*  \* **@Title**: Person  \*/  **public** Person() {  **super**();  }    /\*\*  \* **@Title**: Person  \* **@param** name  \* **@param** age  \*/  **public** Person(String name, **int** age) {  **super**();  **this**.name = name;  **this**.age = age;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  /\*\*  \* **@return** the age  \*/  **public** **int** getAge() {  **return** age;  }  /\*\*  \* **@param** age the age to set  \*/  **public** **void** setAge(**int** age) {  **this**.age = age;  }    } |

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.ObjectOutputStream;  /\*\*  \* **@ClassName**: SerializableDemo  \* **@Description**: 对象的序列化操作  \* **@date** 2017年11月23日 下午1:44:51  \* Company www.igeekhome.com  \*  \* 序列化流可以用来将一个对象整体转为二进制数并存储到文件中。  \* 将对象永久性存入一个文件中，叫做序列化，读取叫做反序列化。  \*  \* 每个被序列化操作的类型必须实现Serializable接口(具备序列化功能的标记接口),让其具备序列化能力  \*  \* ObjectOutputStream序列化流  \* public final void writeObject(Object obj)  \* ObjectInputStream反序列化流  \* public final Object readObject()  \*/  **public** **class** SerializableDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //准备被序列化的对象  Person p = **new** Person("Jack", 18);    //创建序列化流对象  FileOutputStream fos = **new** FileOutputStream("person.txt");  ObjectOutputStream oos = **new** ObjectOutputStream(fos);    //序列化对象  oos.writeObject(p);  //关闭流  oos.close();  }  } |

### 对象反序列化流ObjectInputStream

ObjectInputStream对以前使用ObjectOutputStream写入的对象进行反序列化。支持 java.io.Serializable接口的对象才能从流读取。

#### 案例代码十一:

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** java.io.ObjectInputStream;  /\*\*  \* **@ClassName**: SerializableDemo2  \* **@Description**: 对象的反序列化操作  \* **@date** 2017年11月23日 下午1:50:50  \* Company www.igeekhome.com  \*  \* 序列化流可以用来将一个对象整体转为二进制数并存储到文件中。  \* 将对象永久性存入一个文件中，叫做序列化，读取叫做反序列化。  \*  \* 每个被序列化操作的类型必须实现Serializable接口(具备序列化功能的标记接口),让其具备序列化能力  \*  \* ObjectOutputStream序列化流  \* public final void writeObject(Object obj)  \* ObjectInputStream反序列化流  \* public final Object readObject()  \*/  **public** **class** SerializableDemo2 {  **public** **static** **void** main(String[] args) **throws** ClassNotFoundException, IOException {  //创建反序列化流对象  FileInputStream fis = **new** FileInputStream("person.txt");  ObjectInputStream ois = **new** ObjectInputStream(fis);    //反序列化  Person p = (Person) ois.readObject();  System.***out***.println(p.getName());  System.***out***.println(p.getAge());  System.***out***.println(p);    //关闭流  ois.close();  }  } |

|  |
| --- |
| **package** com.igeek\_01;  **import** java.io.Serializable;  /\*\*  \* **@ClassName**: Person  \* **@Description**: Person类  \* **@date** 2017年11月23日 下午1:55:50  \* Company www.igeekhome.com  \*  \* 普通类需要实现序列化接口，开启其序列化功能  \*  \* 序列化ID(serialVersionUID,序列化版本号)，用于记录该对象的类型。  \* 每个被序列化操作的对象其对象ID与接收类型的ID必须相同。  \*/  **public** **class** Person **implements** Serializable{  /\*\*  \* **@Fields** serialVersionUID : 序列化ID  \*/  **private** **static** **final** **long** ***serialVersionUID*** = -8438404800188984801L;  /\*\*  \* **@Fields** name : 姓名  \*/  **private** String name;  /\*\*  \* **@Fields** age : 年龄  \*/  **private** **int** age;  /\*\*  \* **@Title**: Person  \*/  **public** Person() {  **super**();  }  /\*\*  \* **@Title**: Person  \* **@param** name  \* **@param** age  \*/  **public** Person(String name, **int** age) {  **super**();  **this**.name = name;  **this**.age = age;  }  /\*\*  \* **@return** the name  \*/  **public** String getName() {  **return** name;  }  /\*\*  \* **@param** name the name to set  \*/  **public** **void** setName(String name) {  **this**.name = name;  }  /\*\*  \* **@return** the age  \*/  **public** **int** getAge() {  **return** age;  }  /\*\*  \* **@param** age the age to set  \*/  **public** **void** setAge(**int** age) {  **this**.age = age;  }  /\*\*  \* **@Title**: toString  \* **@Description**: 重写toString方法  \* **@return**  \* **@see** java.lang.Object#toString()  \*/  @Override  **public** String toString() {  **return** "Person [name=" + name + ", age=" + age + "]";  }  } |

## 打印流

打印流添加输出数据的功能，使它们能够方便地打印各种数据值表示形式

打印流只有输出，没有输入

PrintStream 字节打印流，调用println方法时自动刷新

PrintWriter 字符打印流，指定自动刷新开关后，调用println方法时自动刷 新，无需手动调用flush()方法。

### 案例代码十二:

|  |
| --- |
| **package** com.igeek\_02;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.PrintStream;  **import** java.io.PrintWriter;  /\*\*  \* **@ClassName**: PrintStreamPrintWriterDemo  \* **@Description**: 打印流  \* **@date** 2017年11月23日 下午1:54:06  \* Company www.igeekhome.com  \*  \* 打印流只有输出,没有输入  \* PrintStream字节打印流,调用println方法时自动刷新  \* PrintWriter字符打印流,指定自动刷新开关后,调用println方法时自动刷新,无需手动调用flush()方法  \*/  **public** **class** PrintStreamPrintWriterDemo {  **public** **static** **void** main(String[] args) **throws** FileNotFoundException {  //创建流对象  PrintStream ps = **new** PrintStream("test1.txt");    //写  ps.println('a');    //关闭流  //ps.close();    //创建流对象  FileOutputStream fos = **new** FileOutputStream("test2.txt");  PrintWriter pw = **new** PrintWriter(fos, **true**);    //写  pw.write('c');  pw.println('b');    //关闭流  //pw.close();  }  } |

## 第三方工具CommonsIO

### 导入classpath

A：加入classpath的第三方jar包内的class文件才能在项目中使用

B：创建lib文件夹

将commons-io.jar拷贝到lib文件夹

右键点击commons-io.jar，Build Path→Add to Build Path

### FilenameUtils

这个工具类是用来处理文件名（译者注：包含文件路径）的，他可以轻松解决不同操作系统文件名称规范不同的问题。

常用方法：

getExtension(String path)：获取文件的扩展名；

getName()：获取文件名；

isExtension(String fileName,String ext)：判断fileName是否是ext 后缀名；

### FileUtils

提供文件操作（移动文件，读取文件，检查文件是否存在等等）的方法。

常用方法：

readFileToString(File file)：读取文件内容，并返回一个String；

writeStringToFile(File file，String content)：将内容content写入到 file中；

copyDirectoryToDirectory(File srcDir,File destDir)：文件夹复制；

copyFile(File srcFile,File destFile)：文件复制；

### 案例代码十三:

|  |
| --- |
| **package** com.igeek\_03;  **import** java.io.File;  **import** java.io.IOException;  **import** org.apache.commons.io.FileUtils;  /\*\*  \* **@ClassName**: FileUtilsDemo  \* **@Description**: FileUtils工具类的使用  \* **@date** 2017年11月23日 下午2:12:52  \* Company www.igeekhome.com  \*  \* 多级文件夹复制:  \* 可以使用第三方commons-io提供的FileUtils类的方法  \* copyDirectoryToDirectory(File srcDir,File destDir);文件夹复制  \*  \* 第三方Jar包的使用步骤:  \* 创建lib文件夹  \* 将commons-io.jar拷贝到lib文件夹  \* 右键点击commons-io.jar，Build Path→Add to Build Path  \*/  **public** **class** FileUtilsDemo {  **public** **static** **void** main(String[] args) **throws** IOException {  //创建文件夹的文件对象  File srcDir = **new** File("src");  File destDir = **new** File("dest");  FileUtils.*copyDirectoryToDirectory*(srcDir, destDir);  }  } |

## 使用Eclipse生成jar包

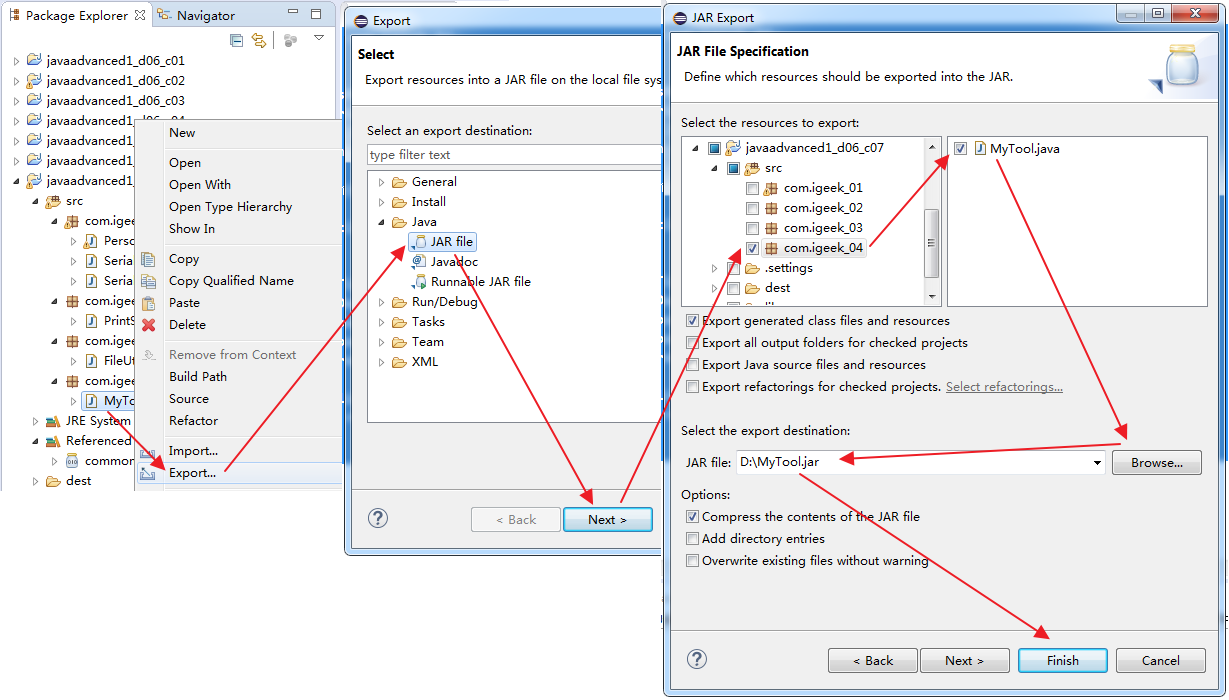
### 导出jar包

我们写一个自定义的工具类作为测试类

#### 案例代码十四:

|  |
| --- |
| **package** com.igeek\_04;  /\*\*  \* **@ClassName**: MyTool  \* **@Description**: 打包  \* **@date** 2017年11月23日 下午2:23:28  \* Company www.igeekhome.com  \*  \* 如何创建jar包  \*/  **public** **class** MyTool {    /\*\*  \* **@Title**: add2Number  \* **@Description**: 两个整数相加的方法  \* **@param** a  \* **@param** b  \* **@return**  \*/  **public** **int** add2Number(**int** a,**int** b) {  **return** a+b;  }  } |

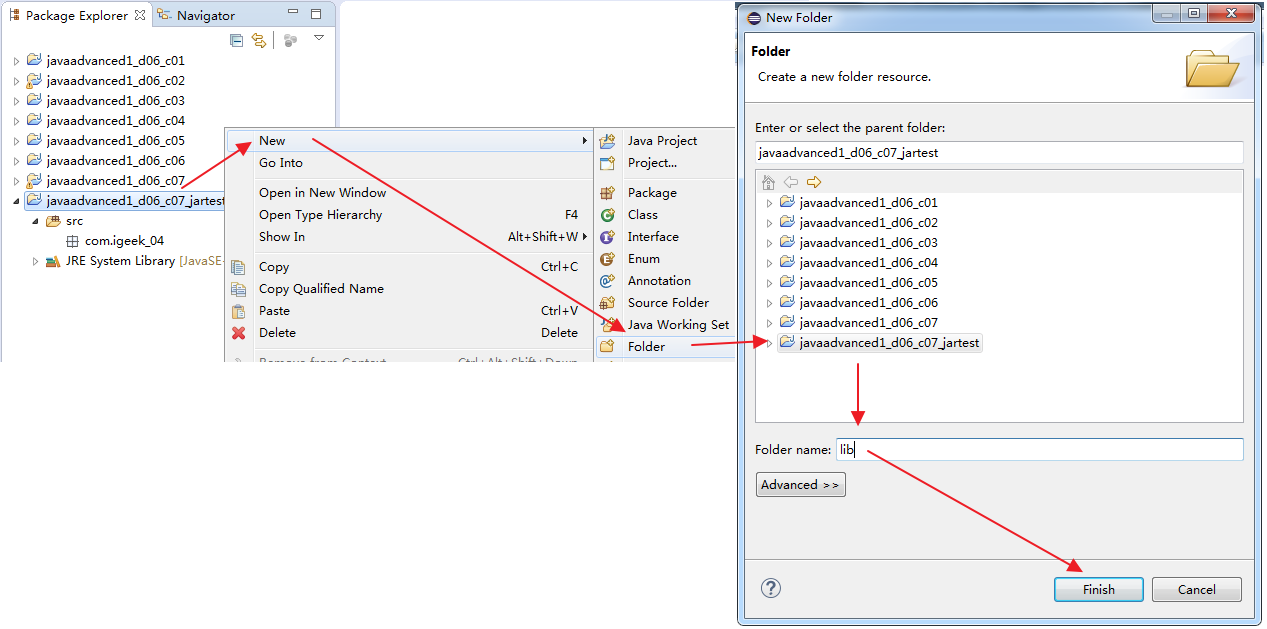
接下来我们选中该类所在的MyTool.java文件:



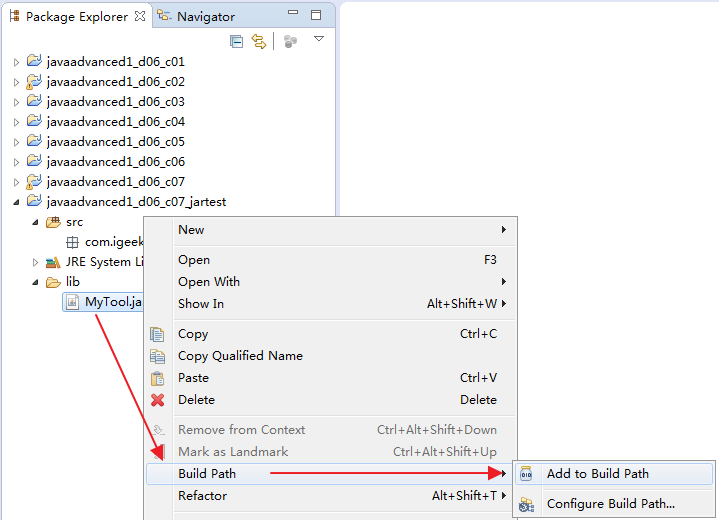
接下来填写导出的jar包的名字和位置，这里导出到d:盘下，名字为MyTool.jar。

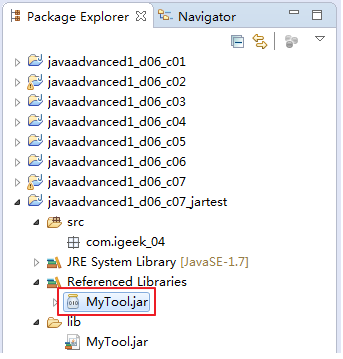
### 使用jar包:

在要使用该jar包的工程下新建一个lib的文件夹



将d盘下的MyTool.jar拷贝到lib文件夹下，在MyTool.jar上右键选择build path，直到MyTool.jar变成一个小奶瓶，代表Eclipse可以找到该jar包，可以使用：





#### 代码案例十五:

javaadvanced1\_d06\_c07\_jartest工程中的代码：

|  |
| --- |
| **package** com.igeek\_01;  **import** com.igeek\_04.MyTool;  /\*\*  \* **@ClassName**: JarTest  \* **@Description**: 导入jar包  \* **@date** 2017年11月23日 下午2:46:51  \* Company www.igeekhome.com  \*  \* 在要使用该jar包的工程下新建一个lib的文件夹  \*  \* 将d盘下的MyTool.jar拷贝到lib文件夹下,  \* 在MyTool.jar上右键选择build path,  \* 直到MyTool.jar变成一个小奶瓶,代表Eclipse可以找到该jar包,可以使用  \*/  **public** **class** JarTest {  **public** **static** **void** main(String[] args) {  //创建导入jar包中的工具对象  MyTool myTool = **new** MyTool();    **int** resutl=myTool.add2Number(1, 2);    System.***out***.println("result:"+resutl);  }  } |

重点和总结

1、字符输入流和字符输出流

2、高效字符流的使用

3、转换流及乱码问题的处理

4、Java对象的序列化和反序列号

5、打印流简单介绍

6、第三方工具：commons-io.jar的使用

7、如何使用Eclipse工具生成自己jar工具包