回顾流的使用步骤:

1. 创建流对象 2.使用对象的方法操作文件 3.关闭流对象

课程目标  
1. 会使用字符流:Writer和Reader读写文件

2. 掌握使用字符流BufferedWriter和BufferedReader进行文件读写

3. 扩展:

了解Java的序列化(Serializable)机制

1. 字符流

注意:Writer和Reader是字符流的基础

1. 使用FileWriter和FileReader进行文件写和读操作
2. 示例一:使用FileWriter进行写文件

|  |
| --- |
| *//使用FileWriter写文件* **try** {  *//1.创建FileWriter流对象* FileWriter fw=**new** FileWriter(**"d:\\a.txt"**,**true**); *//替换  //2.写内容  //2.1写单个字符  //fw.write('张');  //2.2写入字符串  fw.write("张三");*  *//3.关闭流*  fw.flush(); *//清空流* fw.close();  System.***out***.println(**"写入完毕"**); } **catch** (IOException e) {  e.printStackTrace(); } |

1. 使用FileReader读文本文件

|  |
| --- |
| *//1.创建FileReader对象* **try** {  FileReader reader=**new** FileReader(**"d:\\a.txt"**);  *//2.读文件  //2.1一次一个字符  // fw.flush(); //清空流  //int temp=reader.read();  ///System.out.println((char)temp);  //2.2循环读取所有字符  /\*int temp;  while((temp=reader.read())!=-1){  System.out.println((char)temp);  }\*/  //2.3采用字符数组读取内容 了解* **char** [] cs=**new char**[5];  **int** temp;  String allStr=**""**;  **while**((temp=reader.read(cs))!=-1){  allStr =allStr+**new** String(cs,0,temp);  }  System.***out***.println(allStr);  *//3.关闭流* reader.close(); } **catch** (FileNotFoundException e) {  e.printStackTrace(); } **catch** (IOException e) {  e.printStackTrace(); } |

1. 字符流
2. 使用带缓冲区的字符流读写文件.(高级流)

支持换行和按行读取的方法

1. 使用BufferedWriter和BufferedReader进行文件读写
2. 使用BufferedWriter进行写文件

|  |
| --- |
| **try** {  *//1.创建BufferedWriter对象* FileWriter fw = **new** FileWriter(**"d:\\a.txt"**,**true**);  BufferedWriter bw=**new** BufferedWriter(fw);  *//2.写操作* bw.write(**"103,王刚哥,女"**); *//一行写一个对象相关信息,相关信息使用分隔符隔开* bw.newLine(); *//换行  //3.关闭流* bw.flush();  bw.close();  fw.close();  System.***out***.println(**"写入完毕"**); } **catch** (IOException e) {  e.printStackTrace(); } |

1. 使用BufferedReader读取文件信息(按行读取)

|  |
| --- |
| *//使用BufferedReader读文件* **try** {  *//1.创建流对象* FileReader fr=**new** FileReader(**"d:\\a.txt"**);  BufferedReader br=**new** BufferedReader(fr);  *//2.读操作  //2.1按行读取  //String line=br.readLine();  //2.2读取所有行* System.***out***.println(**"学号\t姓名\t性别"**);  String line;  **while**((line=br.readLine())!=**null**){  *//System.out.println(line);  //拆分行* String infos[]=line.split(**","**);  System.***out***.println(infos[0]+**"\t"**+infos[1]+**"\t"**+infos[2]);  }  *//3. 关闭流对象* br.close();  fr.close(); } **catch** (FileNotFoundException e) {  e.printStackTrace(); } **catch** (IOException e) {  e.printStackTrace(); } |

1. 字节流和字符流的应用区别

字节流一般用于操作二进制文件.(如:视频、音频、图片)

字符流一般用于操作文本文件.(如:记事本)

1. 扩展内容:

Java的序列化(Serializable)机制

1. 什么是序例化

把对象转换为字节序列(文件)的过程称为对象的序列化。

1. 什么是反序例化:

反序列化：把字节序列恢复为对象的过程称为对象的反序列化。

1. (了解)序列化的应用场景

a.永久性保存对象，保存对象的字节序列到本地文件或者数据库中，实现了数据的持久化，通过序列化可以把数据永久的保存到硬盘上，

b.利用序列化实现远程通信，可以在网络上传送对象的字节序列。

c.在进程间传递对象

1. 使用ObjectOutputStream实现序例化的步骤:

注意:只有实现了java.io. Serializable接口的类才支持序列化操作.

|  |
| --- |
| *\*/* **public class** Student **implements** Serializable {  **public int xh**;  **public** String **name**;  **public** String **sex**; …. } |

|  |
| --- |
| **public static void** main(String[] args) {  *//创建一个学生* Student stu=**new** Student(); *//此对象保存在内存中 暂时* stu.**xh**=101;  stu.**name**=**"张三"**;  stu.**sex**=**"男"**;  stu.show();   *//将java对象序例化到文件中  //1.创建ObjectoutputStream对象* **try** {  FileOutputStream fos=**new** FileOutputStream(**"d:\\obj.txt"**);  ObjectOutputStream objectOutputStream=**new** ObjectOutputStream(fos);  *//2.写对象* objectOutputStream.writeObject(stu);  *//3.关闭流* objectOutputStream.close();  fos.close();  System.***out***.println(**"写入完毕"**);  } **catch** (IOException e) {  e.printStackTrace();  } } |

1. 使用ObjectInputStream实现反序化的步骤:

|  |
| --- |
| **public static void** main(String[] args) {  *//实现反序例化  //1.创建ObjectInputStream流对象* **try** {  InputStream is=**new** FileInputStream(**"d:\\obj.txt"**);  ObjectInputStream ois=**new** ObjectInputStream(is);  *//2.读操作* Student stu=(Student) ois.readObject();  *//3.关闭流* ois.close();  is.close();  stu.show(); *//操作对象* } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  }  } |