**JAVA基础**

**IO流**

**InputStream OutputStream Reader Writer**

**字节流（按字节单位读写）**

**InputStream - FileInputStream BufferedInputStream**

**OutputStream - FileOutputStream BufferedOutputStream**

**字符流（按字符单位读写）**

**Reader - FileReader BufferedReader**

**Writer - FileWriter BufferedWriter**

**转换流**

**InputStreamReader extends Reader输入流-将字节流转换为字符流**

**OutputStreamWriter extends Writer 输出流-…..**

**示例：**

**// 参数是字节输入流对象，将字节输入流装换为字符输入流**

**InputStreamReader isr = new InputStreamReader(new FileInputStream(“path”/File,false))**

**// 将字节输出流转换为字符输出流**

**OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream(“path”,true))**

**对象流**

**ObjectInputStream readObject()**

**ObjectOutputStream writeObject()**

**注意：对象类实现Serilizable**

输入流

从文件将内容输入到内存

OutputStream

Reader

CPU/内存

B文件

（目的）

A文件（源）

输出流

从内存将内容

写入到文件

OutputStream

Writer

**源：网络数据、本地文件**

**目的：本地**

# File

* 访问文件和目录

File类是java.io包下代表与平台无关的文件和目录

File类能新建、删除、重命名文件和目录

File类不能访问文件内容本身，访问文件内容本身需要使用输入/输出流

|  |
| --- |
| **import** java.io.File;  **import** java.io.IOException;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **public** **class** FileDemo {  **public** **static** **void** main(String[] args) **throws** IOException {    // 创建一个文件对象  File file = **new** File("D:/GZT/test/test/text.txt");  // 获取文件所在路径  System.***out***.println(file.getPath());  // 获取文件绝对路径，从根路径开始  System.***out***.println(file.getAbsolutePath());  // D:/GZT/test/test  System.***out***.println(file.getParentFile());  // 创建文件目录  **if** (!file.getParentFile().exists()) {  file.getParentFile().mkdirs();  }  // 创建文件  **if** (!file.exists()) {  // 创建文件  file.createNewFile();  }  // 文件分隔符  System.***out***.println(File.***separator***);    // text.txt  System.***out***.println("文件名：" + file.getName());  // D:\GZT\test\test  System.***out***.println("父目录名：" + file.getParent());    // File file1 = new File("D:/GZT/test/test/text1.txt");  // 从一个文件对象重名为另一个文件对象  // System.out.println("重命名：" + file.renameTo(file1));    System.***out***.println("是否可写：" + file.canWrite());    System.***out***.println("是否是文件：" + file.isFile());  System.***out***.println("是否是目录：" + file.isDirectory());    // 单位毫秒  System.***out***.println("最后修改时间：" + file.lastModified());  // 毫秒转日期  Date date = **new** Date();  date.setTime(file.lastModified());  // 日期格式转换  SimpleDateFormat sf = **new** SimpleDateFormat("yyyy-MM-dd hh:mm:ss");  String str = sf.format(date);  System.***out***.println(str);  // 获取文件的长度，文件内容大小  System.***out***.println(file.length());    // 我们已经知道这个文件已经存在  File file1 = **new** File("D:/GZT/test/test/text1.txt");  System.***out***.println(file1.delete());    String[] strPaths = file.getParentFile().list();  **if** (**null** != strPaths) {  **for**(String strP:strPaths) {  System.***out***.println(strP);  }  }    File[] fileArr = file.getParentFile().listFiles();  **if** (**null** != fileArr) {  **for** (**int** i = 0; i < fileArr.length; i ++) {  System.***out***.println(fileArr[i].getAbsolutePath());  }  }    }  } |

*1.输入流和输出流(按照流的流向来分,注意选择参照对象)*

*输入流:只能从中读取数据，而不能向其写入数据*

*输出流:只能向其写入数据，而不能从中读取数据*

*2.字节流与字符流(按操作数据单元不同)*

*字节流与字符流用法一样，只是操作数据单元不同:一个是字节(8位)，一个是字符(16位)*

*字节流以InputStream与OutputStream为基类*

*字符流以Reader与Writer为基类*

* 流的概念模型

Java的IO流共有40多个类，但这40多个类都是从如下4个抽象基类派生的

*1.InputStream/Reader:所有输入流的基类，前一个是字节输入流，后一个是字符输入流*

*2.OutputStream/Writer:所有输出流的基类，前一个是字节输出流，后一个是字符输出流*

FileInputStream

|  |
| --- |
| **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.IOException;  **import** java.io.InputStream;  **public** **class** FileIsDemo {  **public** **static** **void** main(String[] args) {  FileIsDemo fid = **new** FileIsDemo();  fid.readInfo();  }  **public** **void** readInfo() {  // 创建一个文件对象  File file = **new** File("D:/GZT/test/test/text.txt");  InputStream is = **null**;  **try** {  is = **new** FileInputStream(file);  // is = new FileInputStream("D:/GZT/test/test/text.txt");  // 读取下一个字节，如果读到流（is）的结尾返回-1  **int** data = 0;  // 有小括号先算小括号里面的  // while((data=is.read()) != -1) {  // // 中文会被读成两截，产生乱码  // System.out.println((char)data);  // }  **byte**[] bs = **new** **byte**[100];  // 将读取到的内容放到bs中，从bs0位置开始存放，存放长度是100  // 这里会因为文件本身内容长度不足以填满bs时，其后会议空格填充  // data读取到的文本字节数据个数  // bs内存中，在读文件内容的时候，先读一个字节数据到bs中，  // bs没有满，文件也没有读完，接着读第二个放到bs中，bs没有满  // 文件也没有读完，一次类推，要么bs读满，要么读到文件结束  // 可以操作bs获取数据  **while** ((data = is.read(bs, 0, bs.length)) != -1) {  System.***out***.println("======");  // 中文也会被读成两截，产生乱码  System.***out***.println(**new** String(bs, 0, data));  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  // 关闭流  **try** {  **if** (**null** != is) {  is.close();  }  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  } |

FileOutputStream

|  |
| --- |
| **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.OutputStream;  **public** **class** FileOsDemo {  **public** **static** **void** main(String[] args) {  FileOsDemo fod = **new** FileOsDemo();  fod.outputInfo();    }    **public** **void** outputInfo() {  OutputStream os = **null**;  **try** {    // 输出流的初始化  // 默认是覆盖的写入，当后面传入的参数是true时，便是追加的写入  os = **new** FileOutputStream("D:/GZT/test/test/text.txt",**false**);    String info = "韩国部署萨德是不道德的 韩国部署萨德是不道德的";    //os.write(97);// a  **byte**[] bs = info.getBytes();  // 一个中文3个字节  os.write(bs);// os.write(bs,0,bs.length);  os.write(bs,0,6);// 写入整个字节数组    // 从第二个位置开始，写入长度为3  // os.write("acbdef".getBytes(), 2, 3);    // os.flush();    } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **if** (**null** != os) {  **try** {  os.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  }  } |

文件复制

|  |
| --- |
| **import** java.io.Closeable;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.io.OutputStream;  **public** **class** FileCopyDemo {  // 读文件的路径  **private** String iPath = "D:/GZT/test/test/text.txt";  // 写文件的路径  **private** String oPath = "D:/GZT/test/test/text1.txt";  **public** **static** **void** main(String[] args) {  FileCopyDemo fcd = **new** FileCopyDemo();  fcd.copyFile();  }  /\*\*  \*  \* **@param** path  \* **@return** true表示文件存在，false文件不存在  \*/  **public** **boolean** existFile(String path) {  File file = **new** File(path);  // 先判读目录存在不存在，如果不存在，则创建，否则不管  **if** (!file.getParentFile().exists()) {  file.getParentFile().mkdirs();  }  // 判断文件存在不存在，如果不存在，则创建  **if** (!file.exists()) {  **try** {  // 如果不存在，并创建成功返回true，否则返回false  **return** file.createNewFile();  } **catch** (IOException e) {  **return** **false**;  }  }  **return** **true**;  }  **public** **boolean** fileError() {  // 判断读写文件是否都存在，如果有一个不存在，则执行return结束方法  **if** (!existFile(iPath)) {  System.***out***.println("读的文件不存在，请检查后在执行");  **return** **false**;  }    **if** (!existFile(oPath)) {  System.***out***.println("写的文件不存在，请检查后在执行");  **return** **false**;  }  **return** **true**;  }  **public** **void** copyFile() {  // 输入流 读 输出流 写  InputStream is = **null**;  OutputStream os = **null**;  **if** (!fileError()) {  **return**;  }  **try** {  is = **new** FileInputStream(iPath);  os = **new** FileOutputStream(oPath);  **byte**[] b = **new** **byte**[100];  **int** data = 0;  **while** ((data = is.read(b)) != -1) {  os.write(b, 0, data);  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **finally** {  // FileOutputStream extends OutputStream implements Closeable  // Closeable中有close方法，  // Closeable cl = os;  // cl.close();  closeStream(os);  closeStream(is);  }  }  /\*\*  \* 关闭流  \*  \* **@param** stream  \*/  **public** **void** closeStream(Closeable stream) {  **try** {  stream.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

BufferedInputStream

|  |
| --- |
| **import** java.io.BufferedInputStream;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.IOException;  **import** java.io.InputStream;  **public** **class** BisDemo {  **private** **static** String *path* = "D:/GZT/test/test/text.txt";  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  *bufferIs*();  }    **private** **static** **void** bufferIs() {  FileInputStream fis = **null**;  InputStream is = **null**;  **try** {  fis = **new** FileInputStream(*path*);    is = **new** BufferedInputStream(fis);  **byte**[] b = **new** **byte**[100];  **int** data = 0;    // 关于flush方法  // 刷新缓冲区 缓冲区在没有满的情况下他还在内存中，我们是无法操作其中的数据  // 当缓冲区满了之后，我们便可以通过变量访问到他        **while** ((data = is.read(b)) != -1) {  System.***out***.println(**new** String(b,0,data));  }    } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **finally** {  **if** (**null** != is) {  **try** {  is.close();  } **catch** (IOException e) {    e.printStackTrace();  }  }  }    }  } |

BufferedOutputStream

|  |
| --- |
| **import** java.io.BufferedOutputStream;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **public** **class** BufOsDemo {  **private** **static** String *path* = "D:/GZT/test/test/text.txt";  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  *bufOs*();  }    **private** **static** **void** bufOs() {  BufferedOutputStream bos = **null**;      **try** {  bos = **new** BufferedOutputStream(**new** FileOutputStream(*path*, **true**));  String info = "\*\*韩国滚蛋了";    bos.write(info.getBytes());    // 如果发现内容没有写全，那么就调用flush方法  // 刷新缓冲区  bos.flush();        } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **finally** {  **if** (**null** != bos) {  **try** {  bos.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  }  } |

FileReader

|  |
| --- |
| **import** java.io.FileNotFoundException;  **import** java.io.FileReader;  **import** java.io.IOException;  **import** java.io.Reader;  /\*\*  \*  \* InputStream OutputStream  \* FileInputStream FileOutputStream  \* BufferedInputStream BufferedOutputStream  \*  \* Reader Writer  \* FileReader FileWriter  \* BufferedReader BufferedWriter  \*  \*/  **public** **class** FileReaderDemo {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  *fileReader*();  }      **private** **static** **void** fileReader() {  // IputStream is = new FileInputStream("");  Reader reader = **null**;  **try** {  reader = **new** FileReader("D:/GZT/test/test/text.txt");    **int** data = 0;  **char**[] ch = **new** **char**[20];  **while** ((data = reader.read(ch)) != -1) {  System.***out***.println(**new** String(ch, 0, data));  }    // 少用这种方式  // while ((data=reader.read()) !=-1) {  // System.out.print((char)data);  // }      } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally**{  **if** (**null** != reader) {  **try** {  reader.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  }  } |

FileWriter

|  |
| --- |
| **import** java.io.FileWriter;  **import** java.io.IOException;  **import** java.io.Writer;  **public** **class** FileWriterDemo {  **public** **static** **void** main(String[] args) {  *write*();  }    **private** **static** **void** write() {  Writer fw = **null**;    **try** {  fw = **new** FileWriter("D:/GZT/test/test/text.txt", **true**);    String info = "上海";  fw.write(info.toCharArray());    } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **if** (**null** != fw) {  **try** {  fw.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }      }  } |

BufferedReader readLine方法

BufferedWriter write(String)

自行练习

转换流

|  |
| --- |
| **import** java.io.BufferedReader;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **import** java.io.Reader;  **public** **class** TransferStreamDemo {  **public** **static** **void** main(String[] args) {  *isToReader*();  }    /\*\*  \* 字节输入流转换为字符输入流  \*/  **private** **static** **void** isToReader() {    // InputStreamReader extends Reader  // 参数是字节输入流  FileInputStream fis;  InputStreamReader isr;  BufferedReader br = **null**;  **try** {  fis = **new** FileInputStream("D:/GZT/test/test/text.txt");  // 实现字节流转换为字符流  isr = **new** InputStreamReader(fis);  // 普通字符流转换为带缓冲的字符流  br = **new** BufferedReader(isr);  String str = **null**;  **while** ((str=br.readLine()) != **null**) {  System.***out***.println(str);  }    } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **finally** {  **if** (**null** != br) {  **try** {  br.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }    }    // 自己练习    } |

# ObjectInputStream、ObjectOutputStream

目标:将对象保存到磁盘中，或允许在网络中直接传输对象

对象序列化机制允许把内存中的Java对象转换成平台无关的二进制流，从而允许把这种二进制流持久地保存在磁盘上，通过网络将这种二进制流传输到另一个网络结点。其他程序一旦获得了这种二进制作流都可以将这种二进制流恢复成原来的对象

* 序列化的含义与意义

序列化机制将实现序列化的Java对象转换成字节序列，这些字节序列可以保存到磁盘或通过网络进行传输，以备在需要时恢复成原来的对象。序列化机制使得对象可以脱离程序的运行而独立存在

对象序列化(Serialize)指将一个Java对象写入IO流中

对象反序列化(Deserialize)指从一个IO流恢复Java对象

需要让某个对象支持序列化，则必须将这个对象的类可序列化(Serializable)

类要想可序列化，则必须实现如下两个接口之一:

Serializable

Externalizable

Java的序列化机制是通过在运行时判断类的serialVersionUID来验证版本一致性的。在进行反序列化时，JVM会把传来的字节流中的serialVersionUID与本地相应实体（类）的serialVersionUID进行比较，如果相同就认为是一致的，可以进行反序列化，否则就会出现序列化版本不一致的异常。

当实现java.io.Serializable接口的实体（类）没有显式地定义一个名为serialVersionUID，类型为long的变量时，Java序列化机制会根据编译的class自动生成一个serialVersionUID作序列化版本比较用，这种情况下，只有同一次编译生成的class才会生成相同的serialVersionUID 。

如果我们不希望通过编译来强制划分软件版本，即实现序列化接口的实体能够兼容先前版本，未作更改的类，就需要显式地定义一个名为serialVersionUID，类型为long的变量，不修改这个变量值的序列化实体都可以相互进行串行化和反串行化

|  |
| --- |
| **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.ObjectInputStream;  **import** java.io.ObjectOutputStream;  **public** **class** ObjectStreamDemo {  **public** **static** **void** main(String[] args) {    // writeObject();    *readObject*();    }      **private** **static** **void** readObject() {  ObjectInputStream ois = **null**;    **try** {    ois = **new** ObjectInputStream(**new** FileInputStream("D:/GZT/test/test/text.txt"));  // 直接从文件中将对象读出来，称为反序列化  Object obj = ois.readObject();  // String str = "AAA";  System.***out***.println(obj);  } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (ClassNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **finally** {  **if** (**null** != ois) {  **try** {  ois.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  }    **private** **static** **void** writeObject() {  ObjectOutputStream oos = **null**;    **try** {  oos = **new** ObjectOutputStream(**new** FileOutputStream("D:/GZT/test/test/text.txt"));  Student stu = **new** Student("张三","上海");  // 直接向文件中写入对象，将序列化的数据写入文件  oos.writeObject(stu);  } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **finally** {  **if** (**null** != oos) {  **try** {  oos.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  }  }  } |
| **public** **class** Student **implements** Serializable {    /\*\*  \* 序列化的版本ID  \*/  **private** **static** **final** **long** ***serialVersionUID*** = 2046747353589527261L;    // transient关键字，修饰的实例变量，不会被序列化，在使用序列化存储时  // 这个实例变量不会被存储  // private transient String name;  // private static String name;  **private** String name;  **private** String address;  **public** Student() {  **super**();  // **TODO** Auto-generated constructor stub  }  **public** Student(String name, String address) {  **super**();  **this**.name = name;  **this**.address = address;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** String getAddress() {  **return** address;  }  **public** **void** setAddress(String address) {  **this**.address = address;  }  @Override  **public** String toString() {  **return** "Student [name=" + name + ", address=" + address + "]";  }  } |

**把对象转换为字节序列的过程称为对象的序列化**。  
　　**把字节序列恢复为对象的过程称为对象的反序列化**。  
　　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。

　　在很多应用中，需要对某些对象进行序列化，让它们离开内存空间，入住物理硬盘，以便长期保存。比如最常见的是Web服务器中的Session对 象，当有 10万用户并发访问，就有可能出现10万个Session对象，内存可能吃不消，于是Web容器就会把一些seesion先序列化到硬盘中，等要用了，再 把保存在硬盘中的对象还原到内存中。

　　当两个进程在进行远程通信时，彼此可以发送各种类型的数据。无论是何种类型的数据，都会以二进制序列的形式在网络上传送。发送方需要把这个Java对象转换为字节序列，才能在网络上传送；接收方则需要把字节序列再恢复为Java对象。