# IO流

# 1 IO流相关概念

## 什么是IO流？

* IO流用来处理设备之间的数据传输
* Java对数据的操作是通过流的方式
* Java用于操作流的类都在**IO包**中

## IO流分类

### 按流向来分

* 输入流InpuStream（读取数据）
* 输出流OutStream（写数据）

### 按操作类型分

* 字节流 : 字节流可以操作任何数据,因为在计算机中任何数据都是以字节的形式存储的
* 字符流 : 字符流只能操作纯字符数据

## Java中常用IO流的类

### 字节流的抽象父类：

* InputStream 输入流
* OutputStream 输出流

### 字符流的抽象父类：

* Reader 字符读取流
* Writer 字符写入流

# 2 InputStream

* InputStream是**抽象类**，表示字节输入流,用于读数据。
* 直接已知子类：
* AudioInputStream
* ByteArrayInputStream
* **FileInputStream(学习文件输入流)**
* FilterInputStream
* ObjectInputStream
* PipedInputStream
* SequenceInputStream
* StringBufferInputStream

# 3 FileInputStream

## 3.1概述：

FileInputStream 从文件系统中的某个文件中获得输入字节。FileInputStream 用于读取诸如图像数据之类的原始字节流。要读取字符流，请考虑使用 FileReader

## 3.2 构造方法

* FileInputStream(File file)
* FileInputStream(String name)

## 3.3 方法

* Int read()方法，读取一个字节

## 3.4 案例1：读取txt文件内容

|  |
| --- |
|  |

## 3.5 案例2：读取txt文件内容另外一种写法

|  |
| --- |
|  |
| 读到的数据是码表的对应值，读取a,就是97 |

## 3.6 关于read方法的面试题

* read()方法读取的是一个字节,为什么返回是int,而不是byte ？

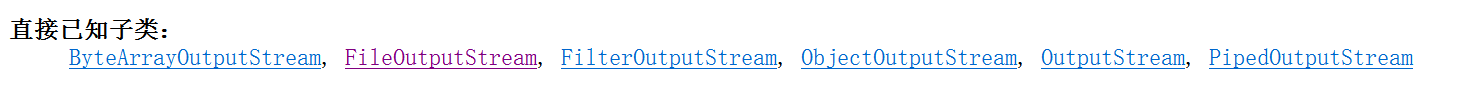
|  |
| --- |
| **public** **static** **void** main(String[] args) {  //面试题:read()方法读取的是一个字节,为什么返回是int,而不是byte ？  //如果使用FileInputStream读取图片数据  /\*\*  \* 00010111 00010111 00010111 00010111  \* 00010111 11111111 00010111 00010111  \*  \* byte:11111111 表示的-1  \* int:00000000 00000000 00000000 11111111  \*  \* 为什么11111111是表示-1？  \* 因为计算机是以补码的方式计算  \* 补码： 11111111 （-1）  \* 反码： 11111110  \* 原码: 10000001  \*  \* 假如使用FileInputStream读取图片的时候，  \* 图片中间有一段数据刚好是 11111111，这八个一表示的-1，用byte接收  \* java程序如果读取的是-1，后面的数据就不读取  \* 如果把8个1转成int,那就是00000000 00000000 00000000 11111111,这个表示255，就不是-1，归避后面数据没有读取问题  \*  \*/  } |

# 4 OutputStream

## 概述

* outputStream译为输出流
* OutputStream是一个抽象类，它用于往文件里写入内容

## 子类



# 5 FileOutputStream

## 概述

* 表示文件输出流，用于往文件写内容
* 构造方法
  + FileOutputStream(String name)
  + FileOutputStream(File file)
* 方法
  + write(int b);//每次写一个字节
  + write(byte[] b) ;//一次写多个字节

## 案例1：往b.txt 写个内容

|  |
| --- |
|  |

## 案例2: 拷贝图片和音频

* 原理，使用输入流读取数据，使用输出流写数据
* 画图讲原理和性能

**每次都读一次字节，太慢，效率太低**

|  |
| --- |
|  |
|  |

## 案例3: 拷贝图片【优化一】

* 一次性读取文件所有数据，一次性将所有数据写入文件
* **这种方式在大文件下不建议使用**
* FileInputStream有个available()方法，返回文件大小

|  |
| --- |
|  |

## 案例4: 拷贝图片【优化二】

* 这种方式开发中**建议经常使用**
* 拷由文件时应该使用**缓冲区思想、或者杯子思想**
* 缓冲区思想、杯子思想是指把文件数据先读取到**一个字节数组中【byte buf[]】,**然后再把字节数组中的数据写入文件中
* 字节流一次读写一个数组的速度明显比一次读写一个字节的速度快很多

|  |
| --- |
| **public** **class** Demo01 {  **public** **static** **void** main(String[] args) **throws** IOException {  //案例:拷贝图片【优化二、这种方式开发中建议经常使用】  /\*\*  \* 1.每次只读一个字节，又觉得太慢  \* 2.一次性读一个文件的所有数据，又怕内存装不下  \* 内存只有2G,视频3G,这样就会内存溢出  \*  \* 3.最终解决方法：折中，定义每次读8KB  \*  \*/    //1.输入流  FileInputStream fis = **new** FileInputStream("C:/Users/10301/Desktop/a/cm.jpg");  //2.输出流  FileOutputStream fos = **new** FileOutputStream("C:/Users/10301/Desktop/a/cm-副本2.jpg");    //3.定义个8kb字节数组，也叫缓冲区流  **byte**[] bytes = **new** **byte**[1024 \* 8];    **int** len = 0;  **int** i = 0;  **while**( (len = fis.read(bytes)) != -1){  i++;  //4.写入文件  fos.write(bytes,0,len);  }    System.***out***.println("读取的次数:" + i);    //5.关闭流  fis.close();  fos.close();  }  } |

# 6 带缓冲的字节输入输出流

## 6.1 BufferedInputStream

* BufferedInputStream内置了一个缓冲区(数组)
* 当使用BufferedInputStream读取一个字节时，BufferedInputStream会一次性从文件中读取8192个(8Kb), 存在缓冲区中,直到缓冲区装满了, 才重新从文件中读取下一个8192个字节数组

|  |
| --- |
|  |

## 6.2 BufferedOutputStream

* BufferedOutputStream也内置了一个**缓冲区**(数组)
* 向流中写入字节时, 不会直接写到文件, 先写到缓冲区中直到缓冲区写满, BufferedOutputStream才会把缓冲区中的数据一次性写到文件里

|  |
| --- |
|  |

## 6.3 案例：使用缓冲字节流读取数据

|  |
| --- |
| /\* BufferedInputStream和BufferedOutputStream  >这两个流是内置了缓冲区流,也就是说内部有一个字节数组  >这个两个类没有前面我们写的好用，因为它内部每次读8kb字节，如果想读80kb,没办法  \*/  //案例：拷贝音频文件  //1.创建缓冲输入流  FileInputStream fis = **new** FileInputStream("C:/Users/10301/Desktop/a/ghsy.mp3");  BufferedInputStream bis = **new** BufferedInputStream(fis);    //2.创建缓冲输出流  FileOutputStream fos = **new** FileOutputStream("C:/Users/10301/Desktop/a/ghsy-副本.mp3");  BufferedOutputStream bos =**new** BufferedOutputStream(fos);    //3.输入流读数据、输出流写数据    **int** b = 0;  // bis.read()方法，内部会读8kb数据  **while**( (b = bis.read()) != -1){  bos.write(b);//内部会写8kb数据  }    //4.关流,只需要关缓冲流，文件流不用关  bis.close();//【内部会关文件流输入流】  bos.close();//【内部会关文件流输出流】    System.***out***.println("音频拷贝完成..."); |
|  |

## 6.4 案例：一个字节的拷贝音频文件

|  |
| --- |
|  |

## 6.5 带Buffered的流和自己写的字节数组缓冲对比

* 自己写数组会略胜一筹,因为读和写操作的是同一个数组，而Buffered操作的是两个数组

|  |
| --- |
|  |
|  |

## 6.6 BufferedOutputStream的flush和close方法的区别

### flush()方法

1.用来刷新缓冲区的，把内部buf[] 变量的数据写入文件,刷新后可以再次写入数所

### close()方法

2.用来关闭流释放资源的

3.如果是带缓冲区的流对象的close()方法,不但会关闭流,还会再关闭流之前刷新缓冲区,关闭后不能再写出

|  |
| --- |
|  |

# 7字节流读写中文乱码问题

## 7.1字节流读取中文

* 字节流在读中文的时候有可能会读到半个中文,造成乱码
* 解决方案：Java提供一个类Reader（字符流）

|  |
| --- |
|  |

## 7.2 字节流写入中文

* 使用字节流写入中文必须将字符串转换成字节数组
* 如果想添加换行添加write("\r\n".getBytes())代码
* 将字符串写入文件中，也可以指定写入字符的码表 如getBytes(“UTF-8”)

|  |
| --- |
|  |

# 8 流的标准处理异常方式

## 8.1 jdk1.6以前的处理方式

* 读写的操作放在try里面
* 关闭流的操作放finally

|  |
| --- |
|  |

## 8.2 jdk1.7以后的处理方式

**实现方式：**

1.把流对象的声明放在try() **括号**里面

2.操作流【读写数据】的代码放在**花括号**里面

3.这样写不用关闭流，java自动关闭流

4.在try的括号里的对象，必须是实现**AutoCloseable**的接口

|  |
| --- |
|  |
|  |

# 9 字节流练习题

## 9.1 练习一 图片加密

|  |
| --- |
|  |

## 9.2 练习二 根据路径拷贝文件

* 根据用户输入的文件[图片]拷贝文件到当前项目目录

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** IOException {  //练习:根据用户输入的文件[图片]拷贝文件到当前项目目录  //C:/Users/10301/Desktop/a/cm2.jpg    //1.获取用户输入的文件路径  Scanner scanner = **new** Scanner(System.***in***);  System.***out***.println("请输入一个文件的路径[请不要输入文件夹路径]:");  String path = scanner.nextLine();    //2.判断这个路径是否为文件  File file = **new** File(path);  **if**(file.isDirectory()){  System.***out***.println("您输入的路径不是一个文件");  **return**;  }    //3.如果是文件就拷贝  //3.1 输入流  FileInputStream fis = **new** FileInputStream(path);    //3.2输出流  FileOutputStream fos = **new** FileOutputStream(file.getName());    //3.3读写【性能】  **byte**[] buf = **new** **byte**[8 \* 1024];    **int** len = 0;  **while**((len = fis.read(buf)) != -1){  fos.write(buf, 0, len);  }    //3.4关流  fis.close();  fos.close();    System.***out***.println("文件拷贝完成...");    } |

## 9.3 练习三 录入数据拷贝到文件

* 根据用户在键盘中输入的内容，然后写到一个文件中去

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** IOException {  /\*\*  \* 根据用户在键盘中输入的内容，然后写到一个文件中去  \*/  //1.创建一个文件输出流  FileOutputStream fos = **new** FileOutputStream("new.txt");    //2.创建一个Scanner对象  Scanner scanner = **new** Scanner(System.***in***);  System.***out***.println("请输入需要录入内容...输入quit代表退出");    //3.通过死循环来获取用户输入【当输入quit，代表不录入】  **while**(**true**){  //获取用户的输入  String line = scanner.nextLine();    //退出循环  **if**(line.equals("quit")){  **break**;  }    //写入文件  fos.write(line.getBytes());  //换行  fos.write("\r\n".getBytes());  }    //4.关流  fos.close();  } |

# 10 字符流

## 10.1 什么是字符流

* 字符流是可以直接读写**字符**的IO流
* 使用字符流从文件中读取字符时, 需要先读取到**字节**数据, 然后转为**字符**.
* 使用字符流往文件写入字符时, 需要把字符转为字节再写入文件.

## 10.2 字符流有两大派

* Reader和Writer，这两个类是抽象类，一般使用它们的子类
* FileReader和FileWriter

## 10.3 FileReader的使用

* FileReader 用于读取字符,调用read方法的时候，是读取一个**字符**
* read方法的返回值是int,它是把char 转成int
* 以后读纯文本文件，使用FileReader这个类，为什么？因为我们不用考虑乱码问题

|  |
| --- |
| **public** **static** **void** test1() **throws** FileNotFoundException, IOException {  //FileReader  /\*\*  \* FileReader 用于读取字符  \*/    //1.创建对象  FileReader reader = **new** FileReader("a.txt");    //2.读取数据  /\*\*  \* reader.read() : 先读取到字节数据, 然后转为字符  \*/  /\* System.out.println((char)reader.read());  System.out.println((char)reader.read());  System.out.println((char)reader.read());  System.out.println((char)reader.read());  System.out.println((char)reader.read());  System.out.println(reader.read());\*/    **int** c;  **while**((c = reader.read()) != -1){  System.***out***.println((**char**)c);  }    //3.关闭流  reader.close();  } |

## 10.5 FileWriter的使用

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  //FileWiter 文件写入流、文件写入字符流  //案例：使用FileWiter往一个文件a.txt写内容    //1.创建writer对象  FileWriter writer = **new** FileWriter("a.txt");    //2.写内容  writer.write("你好!");  writer.write("\r\n");  writer.write("你好!");  writer.write("\r\n");  writer.write("你好!");  writer.write("\r\n");  writer.write("你好!");    //3.关闭  writer.close();  } |

## 10.6 案例：字符流的拷贝

* 需求：将a.txt文件内容拷。贝到b.txt文件中，必须使用**字符流**
* 思路：用FileReader读字符,用FileWriter写字符

|  |
| --- |
| 方式一：**每次读写一个字符** |
| //1.创建 “读取流” 对象  FileReader reader = **new** FileReader("a.txt");    //2.创建 "写入流" 对象  FileWriter writer = **new** FileWriter("b.txt");    //3.读取和写入  **int** c = 0;  **while**((c = reader.read()) != -1){  writer.write(c);  }      //4.关闭流  reader.close();  writer.close(); |
| 方式一：**每次读写多个字符** |
| //1.创建 “读取流” 对象  FileReader reader = **new** FileReader("a.txt");    //2.创建 "写入流" 对象  FileWriter writer = **new** FileWriter("b.txt");    //3.读取和写入【缓冲思想】  **char**[] buf = **new** **char**[8 \* 1024];  **int** len = 0;  **while**( (len = reader.read(buf)) != -1){  writer.write(buf, 0, len);  }  //4.关闭流  reader.close();  writer.close(); |

## 10.7 使用字符流的场景

|  |
| --- |
| /\*\*  \* 1.使用FileReader读取字符  \* 原理：先读字节，把字节转成字符（byte -> char）  \* 2.使用FileWriter写入字符  \* 原理：writer(String str),str 转成 字符数组 再转成 字节写入文件  \* (string -> char[] -> char -> byte)  \* 如果是读取文本文件，使用FileReader比较好,不考虑乱码问题  \* 如果是写入文本文件，不建议使用FileWriter,直接使用FileOutputStream好一点  因为是字节直接写入文件，不需要过多数据类型转换  \*/ |

# 11 带缓冲的字符流

## 11.1 概述

* BufferedReader/BufferedWriter

1.这两个类，内部都一个缓冲区，字符数组

2.br.read()方法，内部会读8192（8\*1024）个字符

3.bw.write(),内部会写8192（8\*1024）个字符

|  |
| --- |
|  |

回顾：带缓冲的字节流,BufferedInputStream/BufferedOutputStream

1. 这两个类，内部都一个缓冲区，字节数组

## 11.2带缓冲字符流的使用

* 使用BufferedReader、BufferedWriter读写文件

|  |
| --- |
| //1.创建Reader  FileReader fr = **new** FileReader("a.txt");  BufferedReader br = **new** BufferedReader(fr);    //2.创建Writer  FileWriter fw =**new** FileWriter("b.txt");  BufferedWriter bw = **new** BufferedWriter(fw);    //3.读写  **int** c = 0;  **while**( (c = br.read()) != -1){  bw.write(c);  }    //4.关流  br.close();//内部关闭FileReader  bw.close();//内部关闭FileWriter |

## 11.3 BufferedReader的readLine()

* 这个方法用于每次读取一行数据
* 案例：使用readLine方法来读取文件

|  |
| --- |
| //1.创建Reader  FileReader fr = **new** FileReader("a.txt");  BufferedReader br = **new** BufferedReader(fr);    //2.读一行数据  /\* System.out.println(br.readLine());  System.out.println(br.readLine());  System.out.println(br.readLine());  System.out.println(br.readLine());  System.out.println(br.readLine());  System.out.println(br.readLine());\*/  String line = **null**;  **while**((line = br.readLine()) != **null**){  System.***out***.println(line);  }    //3.关流  br.close(); |

## 11.4 BufferedWriter的newLine()

* newLine方法是用于写文件时，添加换行，相当于”\r\n”

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** IOException {  //需求：往b.txt写a.txt文件的内容【一行一行拷贝,读一行写一行】  //1.创建Reader  FileReader fr = **new** FileReader("a.txt");  BufferedReader br = **new** BufferedReader(fr);    //2.创建writer  FileWriter fw = **new** FileWriter("b.txt");  BufferedWriter bw = **new** BufferedWriter(fw);    //3.读一行写一行  String line = **null**;  **while**((line = br.readLine()) != **null**){  bw.write(line);  /\*\*  \* 当使用readline读取一行数据时，不会读回车换行"\r\n"字符  \*/  //bw.write("\r\n");  bw.newLine();//换行  }    //4.关流  br.close();  bw.close();  } |

## 11.5 案例:将文本反转写入另一个文件

|  |
| --- |
| /\*\*  \* 掌握思想：  \* 1.for循环的倒序遍历  \*/  **public** **class** Demo01 {  **public** **static** **void** main(String[] args) **throws** IOException {  //案例:将文本反转写入另一个文件    //1.把文本每一行内容读取出来存在List<String> 集合  //1.1 集合对象  List<String> list = **new** ArrayList<String>();    //1.2一行行读取文件  BufferedReader br = **new** BufferedReader(**new** FileReader("a.txt"));  String line = **null**;  **while**((line = br.readLine()) != **null**){  list.add(line);  }    System.***out***.println(list);    //2.当我们要写入文件时，倒序保存List集合中的内容到文件中  BufferedWriter bw = **new** BufferedWriter(**new** FileWriter("b.txt"));  **for**(**int** i = list.size() - 1 ; i>=0; i--){  bw.write(list.get(i));  bw.newLine();//换行  }    //3.关闭流  br.close();  bw.close();  System.***out***.println("finish....");  }  } |

## 11.6 案例:获取文本上字符出现的次数

|  |
| --- |
| /\*\* 掌握:当键盘敲 enter键，两个字符\r\n : 回车换行\*/  **public** **class** Demo01 {  **public** **static** **void** main(String[] args) **throws** IOException {  //练习:获取文本上字符出现的次数,把数据写入文件  /\* 思路：  1.遍历文本每一个字符  2.字符出现的次数存在Map中  \*//\*\*  \* Map<Character,Integer> map = new HashMap<Character,Integer>();  \* map.put('a',18);  \* map.put('你',2);  \*//\*  3.把map中的数据写入文件\*/  //==================================  //1.创建Map集合  Map<Character,Integer> map = **new** HashMap<Character,Integer>();  //System.out.println(map.get('a'));    //2.遍历每一个字符,每一个字符出现的次数放到map中  FileReader reader = **new** FileReader("a.txt");  **int** c = 0;  **while**((c = reader.read()) != -1){  //int 还原 char  **char** ch = (**char**)c;  // 判断char是否在map中第一次出现  **if**(map.get(ch) == **null**){  map.put(ch, 1);  }**else**{  map.put(ch, map.get(ch) + 1);  }  }    //3.把map中数据存在文件count.txt  //3.1 创建Writer  BufferedWriter bw = **new** BufferedWriter(**new** FileWriter("count.txt"));    //3.2 遍历map,再写入数据  **for**(Entry<Character, Integer> entry :map.entrySet()){  **switch** (entry.getKey()) {  **case** ' ':  bw.write("空格=" + entry.getValue());  **break**;  **case** '\t'://\t表示tab 键字符  bw.write("tab键=" + entry.getValue());  **break**;  **case** '\r'://  bw.write("回车=" + entry.getValue());  **break**;  **case** '\n'://  bw.write("换行=" + entry.getValue());  **break**;  **default**:  bw.write(entry.getKey() + "=" + entry.getValue());  **break**;  }  bw.newLine();  }    //4.关流  reader.close();  bw.close();  }  } |

# 12 装饰(Decorator)设计模式

|  |
| --- |
| /\*\*  \* 一、设计模式【一种固定代码风格】  \* 面试题：Java中有哪些设计模式  \* 常用设计模式：装饰模式、单例模式、模版模式、适配器模式、代理模式、工厂模式...  \*  \* 二、装饰(Decorator)设计模式  \* 1.装饰模式的概念:  装饰模式是动态的给一个对象添加一些额外的功能，就增加功能来说，装饰模式比生成子类更为灵活。  装饰模式是在不必改变原类文件和使用继承的情况下，动态的扩展一个对象的功能。  装饰模式是创建一个包装对象，也就是使用装饰来包裹真实的对象。    2.装饰模式的实现方式  1. 装饰对象和真实对象有相同的接口/抽象类。  2. 装饰对象包含一个真实对象的引用（reference）  3. 装饰对象的所有方法，内部实现都是通过真实对象的引用来调用，然后实现自己的功能  3.适用性  1. 需要扩展一个类的功能，或给一个类添加附加职责。  2. 当不能采用生成子类来实现，比如final类  \*/ |
| **public** **class** Demo01 {  **public** **static** **void** main(String[] args) {  /\* BufferedInputStream bis = new BufferedInputStream(new FileInputStream("a.txt"));  BufferedOutputStream bos;  BufferedReader br;  BufferedWriter bw;\*/    /\*GYFStudent stu = new GYFStudent();  stu.code();\*/    GYFStudent stu = **new** GYFStudent(**new** UniversityStudent());  stu.code();  }  }  //1.接口  **interface** Person{  **public** **void** code();  }  //2.接口实现类  **final** **class** UniversityStudent **implements** Person{  @Override  **public** **void** code() {  // **TODO** Auto-generated method stub  System.***out***.println("在大学期间学了一点点C语言...");  }  }  /\*class GYFStudent extends UniversityStudent{  @Override  public void code() {  // **TODO** Auto-generated method stub  super.code();  System.out.println("在GYF IT EDU 学习Java语言...");  }  }\*/  **class** GYFStudent **implements** Person{  **private** UniversityStudent us;  **public** GYFStudent(UniversityStudent us){  **this**.us = us;  }  //装饰对象的所有方法，内部实现都是通过真实对象的引用来调用，然后实现自己的功能  @Override  **public** **void** code() {  us.code();  System.***out***.println("在GYF IT EDU 学习Java语言...");  }  } |

# 13 编码字符流

## 13.1 InputStreamReader

* 这个类用于使用指定的码表读写字符

|  |
| --- |
| //案例1：使用指定 “码表UTF-8&GBK” 读取字符  /\*\* charsetName：字符编码的名称 \*/  FileInputStream fis = **new** FileInputStream("a.txt");  InputStreamReader isr = **new** InputStreamReader(fis, "UTF-8");  //读一个字符  /\* System.out.println((char)isr.read());  System.out.println((char)isr.read());\*/    //读一行  BufferedReader br = **new** BufferedReader(isr);  System.***out***.println(br.readLine());  br.close(); |

## 13.2 OutputStreamWriter

* 这个类使用指定的编码 写入文件

|  |
| --- |
| //1.创建Writer  FileOutputStream fos = **new** FileOutputStream("c.txt");  OutputStreamWriter osw = **new** OutputStreamWriter(fos, "GBK");  BufferedWriter bw = **new** BufferedWriter(osw);    //2.写  bw.write("你好，IO流...搞的我有点晕...");  bw.newLine();  bw.write("你好，IO流...搞的我有点晕...");  bw.newLine();    //3.关流  bw.close(); |

# 14 递归

## 概述

* 递归是一种算法,在程序设计语言中广泛应用。
* 递归：**就方法内部调用自己**
* 递归的弊端:不能调用次数过多,容易导致栈内存溢出
* 递归的好处:不用知道循环次数
* 构造方法不能递归
* 递归的方法可以有返回值，也可以没返回值

## 案例：递归5的阶乘

|  |
| --- |
| 方法1 |
|  |
| 方法2 |
|  |

## 案例：递归文件目录

|  |
| --- |
| **public** **class** Demo01 {  **public** **static** **void** main(String[] args) {  //递归:文件目录  /\*\*打印出a目录所有文件名称，包括子文件夹的文件\*/    //1.创建目录对象  File dir = **new** File("C:/Users/10301/Desktop/a");    //2.打印目录的子文件  *printSubFile*(dir);  }    **public** **static** **void** printSubFile(File dir){  //打印目录的子文件  File[] subfiles = dir.listFiles();    **for**(File f : subfiles){  **if**(f.isDirectory()){//文件夹  *printSubFile*(f);  }**else**{//文件  System.***out***.println(f.getAbsolutePath());  }    }  }  } |

## 案例：从键盘输入文件夹路径,打印该文件夹下所有的.jpg文件名

|  |
| --- |
| **public** **class** Demo02 {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  //练习:从键盘输入接收一个文件夹路径,打印出该文件夹下所有的.jpg文件名  //1.获取文件路径  Scanner scanner = **new** Scanner(System.***in***);  System.***out***.println("请输入一个文件夹路径...");  String path = scanner.nextLine();    //2.把路径封装成File对象  File file = **new** File(path);  **if**(!file.exists()){  System.***out***.println("此路径对应的文件不存在");  **return**;  }    **if**(!file.isDirectory()){  System.***out***.println("请输入文件夹路径，而不文件路径");  **return**;  }    //3.遍历文件夹下的子文件  *printSubFile*(file);    }    **public** **static** **void** printSubFile(File dir){  //打印目录的子文件  File[] subfiles = dir.listFiles();    **for**(File f : subfiles){  **if**(f.isDirectory()){//文件夹  *printSubFile*(f);  }**else**{//文件  **if**(f.getName().endsWith(".jpg")){  System.***out***.println(f.getAbsolutePath());  }  }  }  }  } |

# 15 SequenceInputStream

## 序列流

* 序列流可以把**多个字节输入流**整合成一个, 从序列流中读取数据时,
* 将从被整合的第一个流开始读, 读完一个之后继续读第二个, 以此类推.

## 案例1：把两个文件内容写到一个文件中

|  |
| --- |
| 方法一 |
|  |
| 方法二 |
|  |

## 案例2：把三个文件内容写到一个文件中

|  |
| --- |
| **public** **class** Demo01 {  **public** **static** **void** main(String[] args) **throws** Exception {  //SequenceInputStream序列流使用二  //需求，把a.txt b.txt c.txt 的内容拷贝到d.txt  //1.Vector是集合  Vector<InputStream> vector = **new** Vector<InputStream>();  vector.add(**new** FileInputStream("a.txt"));  vector.add(**new** FileInputStream("b.txt"));  vector.add(**new** FileInputStream("c.txt"));  //2.Enumeration枚举  Enumeration<InputStream> e = vector.elements();  //3.序列流  SequenceInputStream sis = **new** SequenceInputStream(e);  //4.创建输出流  FileOutputStream fos = **new** FileOutputStream("d.txt");  //读写  **int** b = 0;  **while**((b = sis.read()) != -1){  fos.write(b);  }    sis.close();  fos.close();  }  } |

# 16字节数组输出流

## 概述

* ByteArrayOutputStream 字节数组输出流
* ByteArrayOutputStream 不需要关联文件
* 此类实现了一个输出流，其中的数据被写入一个 byte 数组 缓冲区。
* 缓冲区会随着数据的不断写入而**自动增长**。
* 可使用 toByteArray() 和 toString() 获取数据。

|  |
| --- |
|  |

## ByteArrayOutputStream的使用

|  |
| --- |
| //1.创建字节数组输出流对象  ByteArrayOutputStream baos = **new** ByteArrayOutputStream();  //2.输入流  FileInputStream fis = **new** FileInputStream("a.txt");  //3.写入输出流  **int** b = 0;  **while**((b = fis.read()) != -1){  baos.write(b);  }  //自动把字节数组 转成 字符串  System.***out***.println(baos.toString());  System.***out***.println(baos.toString("UTF-8"));//指定编码格式的字符串    //获取文件数据  **byte**[] bytes = baos.toByteArray();  System.***out***.println(**new** String(bytes));  System.***out***.println(**new** String(bytes, "UTF-8"));//指定编码格式的字符串  //4.关流  fis.close(); |

## 面试题：找Bug

|  |
| --- |
| **public** **class** Demo01 {  **public** **static** **void** main(String[] args) **throws** IOException {  /\* 面试题：找bug  定义一个文件输入流,调用read(byte[] b)方法,  将a.txt文件中的内容打印出来(byte数组大小限制为5)\*/  //1.文件输入流  FileInputStream fis = **new** FileInputStream("a.txt");    //2.字节数组输出流  ByteArrayOutputStream baos = **new** ByteArrayOutputStream();    //3.字节数组【缓冲区思想，杯子思想】  **byte**[] arr = **new** **byte**[5];  **int** len;  **while**((len = fis.read(arr)) != -1) {  //通过字节缓冲数组写入  baos.write(arr, 0, len);    //把字节转成字符串  /\*\*中文乱码问题\*/  System.***out***.println(**new** String(arr,0,len));  }  //内部会调用toString方法,把字节数组转成字符串  System.***out***.println(baos);  fis.close();  }  } |

# 17对象操作流

## ObjectOutputStream

* 这个类是将一个对象写入文件
* 如果使用这个类写入对象，这个对象需要序列化
* 序列化就是让这个对象实现一个**Serializable接口**
* 如果没实现Serializable接口，会抛异常**NotSerializableException**
* **对象写入文件时，乱码没有关系，取出来正确就行了**

## ObjectInputStream

* 这个类是从文件中读取对象

## 案例：直接把对象存在文件中

|  |
| --- |
|  |
|  |

## 案例：从文件中直接取出对象

|  |
| --- |
|  |

## 序列化相关概念

* 归档（序列化） ：将对象存在一个文件的过程
* 解归档(反序列化)：把一个文件解析出对象

## 案例：对象操作流优化

* 思路:把多个对象存到List中，然后把List存入文件

|  |
| --- |
| 存List到文件 |
|  |
| 从文件中取出List |
|  |

## Serializable接口的ID讲解

1.要归档或者序列化的对象必须实现Serializable接口才能被序列化

2.Serializable 中有个id，但ID不是一定要加的

3.SerialVersionUid，简言之，其目的是以序列化对象进行版本控制，有关各版本反序列化时是否兼容。

4.如果在新版本中这个值修改了，新版本就不兼容旧版本，反序列化时会抛出InvalidClassException异常。

5.如果修改较小，比如仅仅是增加了一个属性，我们希望向下兼容，老版本的数据都能保留，那就不用修改；

6.如果我们**删除了一个属性，或者更改了类的继承关系**，必然不兼容旧数据，这时就应该手动更新版本号，即SerialVersionUid。

7.一般不会添加ID,就算添加了ID,版本号最好不要修改

|  |
| --- |
|  |

# 18 打印流

## PrintStream 打印字节流

* System.out就是一个PrintStream, 其默认向控**制台输出**信息
* 该流可以很方便的将对象的**toString()**结果输出, 并且自动加上换行

## PrintWriter 打印字符流

* 这个流是向文件打印信息，也就是将打印的内容写入文件

|  |
| --- |
| //1.打印的字节流【打印到控制台】  PrintStream ps = System.***out***;  ps.println("你好，元宵节快乐");//打印字符串  ps.println(19.6);    //2.打印的字符流  /\*\*  \* 1.PrintWriter调用打印方法，控制台是没有内容，它是把内容写到文件中  \* 2.如果打印了内容，没有调用flush或者close，内容在文件中也不存在  \*/  PrintWriter pw = **new** PrintWriter("test.txt");  pw.println("吃汤圆了...");  pw.print("超市一包汤圆10几块钱");  pw.print(13.01);  //pw.flush();  pw.close(); |

# 19 标准输入输出流

* System.in是InputStream, 标准输入流, 默认可以从键盘输入读取字节数据
* System.out是PrintStream, 标准输出流, 默认可以向Console中输出字符和字节数据

# 20 随机访问流

* RandomAccessFile类不属于流，是Object类的子类。
* 但它融合了InputStream和OutputStream的功能。
* 支持对随机访问文件的读取和写入。

|  |
| --- |
| /\*\*  \* 构造方法两个参数：  \* name:文件名称、路径  \* mode:模式 ，r=read 只读、 w=write  \*/  //1.创建一个随机访问流对象，以读写的方式打开文件  RandomAccessFile raf = **new** RandomAccessFile("a.txt", "rw");    //2.读字符  /\*\*  \* 使用RandomAccessFile的readChar/readLine方法读文件有乱码问题  \*/  /\* System.out.println(raf.readChar());  System.out.println(raf.readChar());  System.out.println(raf.readChar());  System.out.println(raf.readLine());\*/    //使用字节数组来读比较好  /\* byte[] buf = new byte[1024];  int len;  while((len = raf.read(buf)) != -1){  System.out.println(new String(buf,0,len));  }\*/    //3.写数据  //raf.writeChars("abc");  //raf.writeBytes("abc");  raf.seek(4);//指定位置  raf.write(97);  raf.write(98);  raf.write(99);  } |

# 21数据输入输出流

* DataInputStream, DataOutputStream可以按照基本数据类型大小读写数据
* 例如按Long大小写出一个数字, 写出时该数据占8字节.
* 读取的时候也可以按照Long类型读取, 一次读取8个字节.

|  |
| --- |
| **long** a = 997;  **long** b = 998;  **long** c = 999;    //使用FileOutputStream没法写入long类型数据  FileOutputStream fos = **new** FileOutputStream("a.txt");  //byte -128~127 0~255  /\* fos.write(997);只会写一个字节，不会写8个字节  fos.write(998);  fos.write(999);\*/    //fos.wr  DataOutputStream dos = **new** DataOutputStream(fos);  dos.writeLong(a);//写8个字节  dos.writeLong(b);  dos.writeLong(c);    dos.close(); |
|  |
| //读3个long数据  DataInputStream dis = **new** DataInputStream(**new** FileInputStream("a.txt"));  System.***out***.println(dis.readLong());//读8个字节  System.***out***.println(dis.readLong());//读8个字节  System.***out***.println(dis.readLong());//读8个字节 |

# 22 Properties类【掌握-经常用】

* Properties:属性，与Map的使用有点类似
* Properties 类表示了一个持久的属性集。
* Properties 可保存在流中或从流中加载,这个类可以读写文件。
* 属性列表中每个键及其对应值**都是一个字符串**。
* 存中文时，会转成Unicode编译存储

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# 23 捋一捋IO流

|  |
| --- |
| **public** **static** **void** main(String[] args) {  /\*\*  \* 一、IO流-输入输出流  \* I:input  \* O:out  \* IO流 操作文件，读取文件内容，往文件写内容  \*  \* 二、字节流  \*  \* >InputStream  \* -FileInputStream 文件输入流，读的单位是字节  \* -BufferedInputStream 缓冲输入流，内部有个byte[]字节数组  \* -SequenceInputStream 序列流，把多个字节流整合成一个流  \* -ObjectInputStream 对象输入流，直接从文件中读取一个对象,这个对象要实现serilazable接口  \* -Sytem.in 标准输入流-指键盘  \* -DataInputStream 数据输入流，按基本数据类型的大小(long) 读取文件  \*  \* >OutputStream  \* -FileOutputStream 文件输出流，写的单位是字节  \* -BufferedOutputStream 缓冲输出流，内部有个byte[]字节数组  \* -ByteArrayOutputStream 字节数组输出流，把数据读取到内存中，这个类不需要关联文件  \* -ObjectOutputStream 对象输出流，直接把一个对象存入文件，  \* -PrintStream 打印流，把内容打印到控制台  \* -System.out 标准输出流-指控制台  \* -DataOutputStream 数据输出流，按基本数据类型的大小(long) 写入文件  \*  \* 三、字符流  \* >Reader  \* -FileReader 文件读取流，读取的单位是字符  \* -BufferedReader 缓冲读取流，内部有个char[] 字符数组  \* -InputStreamReader 指定字符编码读取文件  \*  \* >Writer  \* -FileWriter 文件写入流,写入的单位是字符  \* -BufferedWriter,缓冲写入流，内部有个char[] 字符数组  \* -OutputStreamWriter 指定字符编码写入文件  \* -PrintWriter 打印流，把内容打印一个文件  \*  \*  \* RandomAccessFile 随机访问流，特点：读和写都在一个类中  \*  \* Properties 相当于Map一样使用，这个类把数据存在一个后缀名为.properties文件  \*/  } |