# 20 IO字节流

## 20.01\_IO流(IO流概述及其分类)

**1.概念**

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

流按流向分为两种：输入流，输出流。

流按操作类型分为两种：

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，比较方便。

**2.IO流常用父类**

**字节流的抽象父类：**

**InputStream**

**OutputStream**

**字符流的抽象父类：**

**Reader**

**Writer**

**3.IO程序书写**

使用前，导入IO包中的类

使用时，进行IO异常处理（硬盘和内存之间的操作，硬盘可能不存在啊！）

**使用后，释放资源.（关闭硬盘和内存之间的那个桥梁管道）**

## 20.02\_IO流(FileInputStream)

read()一次读取一个字节

---------------------------------------------------------------------------------------------------------------------------

FileInputStream fis = new FileInputStream("aaa.txt"); //创建一个文件输入流对象,并关联aaa.txt

int b; //定义变量,记录每次读到的字节

while((b = fis.read()) != -1) { //将每次读到的字节赋值给b并判断是否是-1

System.out.println(b); //打印每一个字节，读的是硬盘上存储的二进制

}

fis.close(); //关闭流释放资源

---------------------------------------------------------------------------------------------------------------------------

### 20.03\_IO流(read()方法返回值为什么是int)

read()方法读取的是一个字节,为什么返回是int,而不是byte

因为字节输入流可以操作任意类型的文件,比如图片音频等,这些文件底层都是以二进制形式的存储的,如果每次读取都返回byte,有可能在读到中间的时候遇到111111111，那么这11111111是byte类型的-1,我们的程序是遇到-1就会停止不读了,后面的数据就读不到了,所以在读取的时候用int类型接收,如果11111111会在其前面补上

24个0凑足4个字节（不只是在这个地方是所有地方都是这样的，**读的时候补上写的时候砍掉**）,那么byte类型的-1就变成int类型的255了这样可以保证整个数据读完,而结束标记的-1就是int类型

## 20.04\_IO流(FileOutputStream★★★★★)

write()一次写出一个字节

---------------------------------------------------------------------------------------------------------------------------

FileOutputStream fos = new FileOutputStream("bbb.txt"); //**如果没有bbb.txt,会创建出一个，但是路径你的正确，读的时候必须有的！**

//fos.write(97); //虽然写出的是一个int数,但是在写出的时候会将前面的24个0去掉,所以写出的一个byte

fos.write(98);

fos.write(99);

fos.close();

---------------------------------------------------------------------------------------------------------------------------

### 20.05\_IO流(FileOutputStream追加)

A:案例演示

FileOutputStream的构造方法写出数据如何实现数据的追加写入**(内存中)**

---------------------------------------------------------------------------------------------------------------------------

FileOutputStream fos = **new FileOutputStream("bbb.txt",true)**; //如果没有bbb.txt,会创建出一个，如果有这个文件就会先将文件清空，每一次都是重新写入。有个true就是追加。

//fos.write(97); //虽然写出的是一个int数,但是在写出的时候会将前面的24个0去掉,所以写出的一个byte

fos.write(98);

fos.write(99);

fos.close();

---------------------------------------------------------------------------------------------------------------------------

## IO流拷贝文件的三种方式(直接,aviable,数组★★★★)

### 20.06\_IO流(拷贝图片)第一种拷贝方式

FileInputStream读取

FileOutputStream写出

---------------------------------------------------------------------------------------------------------------------------

**FileInputStream fis = new FileInputStream("致青春.mp3");** //创建输入流对象,关联致青春.mp3

**FileOutputStream fos = new FileOutputStream("copy.mp3");** //创建输出流对象,关联copy.mp3

**int b;**

**while((b = fis.read()) != -1) {**

**fos.write(b);**

**}**

**fis.close();**

**fos.close(); //掌握核心的代码！**

---------------------------------------------------------------------------------------------------------------------------

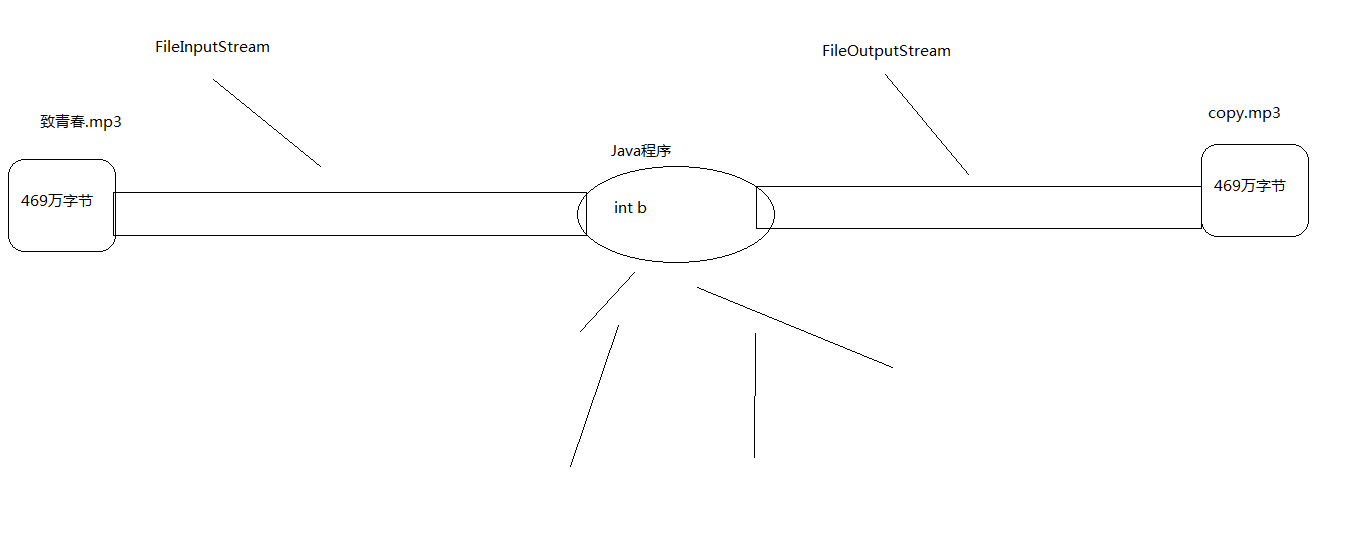
### 20.07\_IO流(拷贝音频文件画原理图)

A:案例演示

字节流一次读写一个字节复制音频

弊端:效率太低

---------------------------------------------------------------------------------------------------------------------------



---------------------------------------------------------------------------------------------------------------------------

### 20.08\_IO流(字节数组拷贝之available()方法)第二种拷贝方式

A:案例演示

int read(byte[] b): 一次读取一个字节数组

Int write(byte[] b): 一次写出一个字节数组

**available()获取读的文件所有的字节个数**

**弊端:有可能会内存溢出，就是要在内存中创建一个数组大小的空间。其实系统给内存分配的空间大小只有170M左右的（jdk1.7版本）。**

---------------------------------------------------------------------------------------------------------------------------

FileInputStream fis = new FileInputStream("致青春.mp3");

FileOutputStream fos = new FileOutputStream("copy.mp3");

**byte[] arr = new byte[fis.available()];** //根据文件大小做一个字节数组

fis.read(arr); //将文件上的所有字节读取到数组中

fos.write(arr); //将数组中的所有字节一次写到了文件上

fis.close();

fos.close();

---------------------------------------------------------------------------------------------------------------------------

### 20.09\_IO流(定义小数组)第三种拷贝方式

write(byte[] b)

write(byte[] b, int off, int len)写出有效的字节个数

---------------------------------------------------------------------------------------------------------------------------

public static void demo2() throws FileNotFoundException, IOException {

FileInputStream fis = new FileInputStream("xxx.txt");

FileOutputStream fos = new FileOutputStream("yyy.txt");

byte[] arr = new byte[2];

int len; //读取的有效的字节数组大小。

while((len = fis.read(arr)) != -1) {

fos.write(arr,0,len);

}

fis.close();

fos.close();

}

public static void demo1() throws FileNotFoundException, IOException {

FileInputStream fis = new FileInputStream("xxx.txt");

byte[] arr = new byte[2]; //假定TXT中有三个字符abc

int a = fis.read(arr); //将文件上的字节读取到字节数组中

System.out.println(a); //读到的有效字节个数两个

for (byte b : arr) { //第一次获取到文件上的a和b

System.out.println(b);

}

System.out.println("-----------------------");

int c = fis.read(arr);

System.out.println(c); //有效字节数组一个

for (byte b : arr) { //第一次进来的c和残留下来的b

System.out.println(b);

}

fis.close();

}

---------------------------------------------------------------------------------------------------------------------------

### 20.10\_IO流(定义小数组的标准格式)第三种拷贝的标准格式

A:案例演示

字节流一次读写一个字节数组复制图片和视频

---------------------------------------------------------------------------------------------------------------------------

FileInputStream fis = new FileInputStream("致青春.mp3");

FileOutputStream fos = new FileOutputStream("copy.mp3");

int len;

byte[] arr = new byte[1024 8]; //自定义字节数组

while((len = fis.read(arr)) != -1) { //如果忘记加arr,返回的就不是读取的字节个数,而是字节的码表值

//fos.write(arr);

fos.write(arr, 0, len); //写出字节数组写出有效个字节个数

}

fis.close();

fos.close();

---------------------------------------------------------------------------------------------------------------------------

## 20.11\_IO流(BufferedInputStream和BufferOutputStream拷贝★★★★)

A:缓冲思想

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，这是加入了数组这样的缓冲区效果，java本身在设计的时候，也考虑到了这样的设计思想(装饰设计模式后面讲解)，所以提供了字节缓冲区流

B.BufferedInputStream

BufferedInputStream内置了一个缓冲区(数组)，从BufferedInputStream中读取一个字节时，BufferedInputStream会一次性从文件中读取8192个, 存在缓冲区中, 返回给程序一个，程序再次读取时, 就不用找文件了, 直接从缓冲区中获取直到缓冲区中所有的都被使用过, 才重新从文件中读取8192个

C.BufferedOutputStream

BufferedOutputStream也内置了一个缓冲区(数组)，程序向流中写出字节时, 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

D.拷贝的代码

---------------------------------------------------------------------------------------------------------------------------

*FileInputStream fis = new FileInputStream("致青春.mp3"); //创建文件输入流对象,关联致青春.mp3*

*BufferedInputStream bis = new BufferedInputStream(fis); //创建缓冲区对fis装饰*

*FileOutputStream fos = new FileOutputStream("copy.mp3"); //创建输出流对象,关联copy.mp3*

*BufferedOutputStream bos = new BufferedOutputStream(fos); //创建缓冲区对fos装饰*

*int b;*

*while((b = bis.read()) != -1) {*

*bos.write(b);*

*}*

*bis.close(); //只关装饰后的对象即可*

*bos.close();*

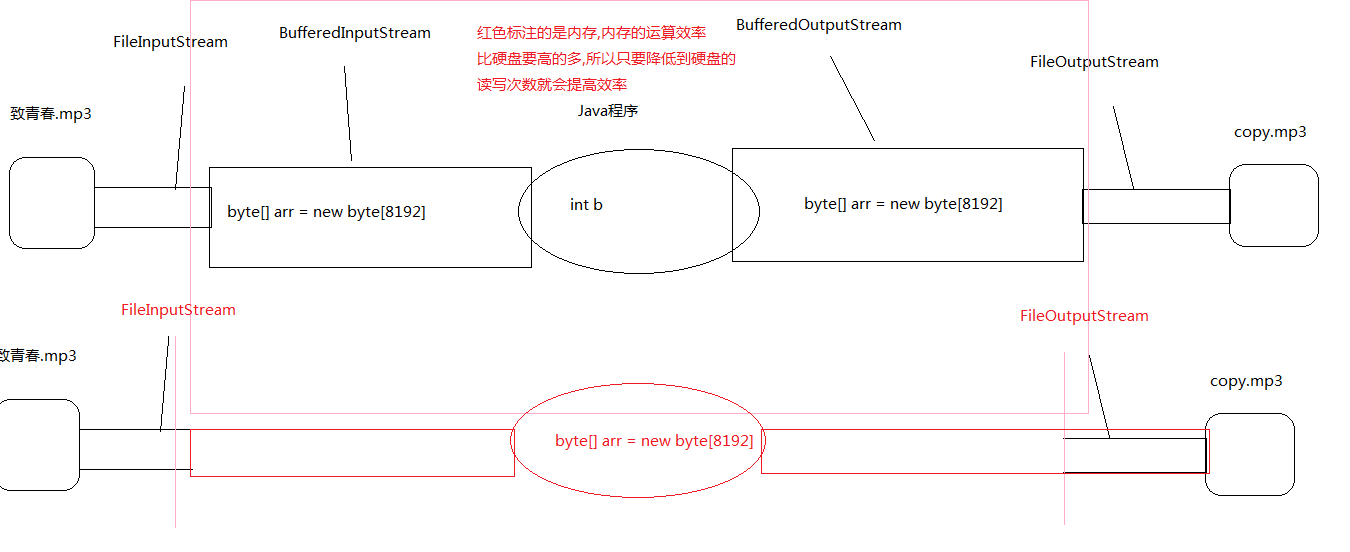
---------------------------------------------------------------------------------------------------------------------------

**E.小数组的读写和带Buffered的读取哪个更快?**

定义小数组如果是8192个字节大小和Buffered比较的话

定义小数组会略胜一筹,因为读和写操作的是同一个数组

而Buffered操作的是两个数组



### 20.12\_IO流(flush和close方法的区别)

flush()方法

用来刷新缓冲区的,刷新后可以再次写出。主要是针对像QQ聊天类的运用的时候。

close()方法

用来关闭流释放资源的的,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

---------------------------------------------------------------------------------------------------------------------------

public static void main(String[] args) throws IOException {

//demo1();

//flush和close方法的区别

BufferedInputStream bis = new BufferedInputStream(new FileInputStream("致青春.mp3"));

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("copy.mp3"));

int b;

while((b = bis.read()) != -1) {

bos.write(b);

}

bis.close();

bos.close();

}

---------------------------------------------------------------------------------------------------------------------------

## 20.13\_IO流(字节流读写中文)读不成，但是能比较好的书写(★★★★)

字节流读取中文的问题

字节流在读中文的时候有可能会读到半个中文,造成乱码。因为一个中文是两个字节的，而字节流只能是每一次只读一个字节，当然你一可以用小数组进行制定规则进行读取但是你的规则只是针对汉字有效，那么里边的符号你是怎么办呢，符号只是一份字节而已！这是行不通的，只能是通过字符进行读。

字节流写出中文的问题

字节流直接操作的字节,所以写出中文必须将字符串转换成字节数组

写出回车换行 **write("\r\n".getBytes());**

---------------------------------------------------------------------------------------------------------------------------

public class Demo6\_Chinese {

public static void main(String[] args) throws IOException {

demo1();

// FileOutputStream fos = new FileOutputStream("zzz.txt");

// fos.write("我读书少,你不要骗我".getBytes()); //对象点方法成byte[ ];

// fos.write("\r\n".getBytes());

// fos.close();

}

public static void demo1() throws FileNotFoundException, IOException {

FileOutputStream fos = new FileOutputStream("zzz.txt");

FileInputStream fis = new FileInputStream("yyy.txt");

byte[] arr = new byte[4]; //虽然是整数倍但是有标点符号怎么办呢，还是不行的

int len;

while((len = fis.read(arr)) != -1) {

fos.write(arr,0,len);

// System.out.println(new String(arr,0,len));

}

fis.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

## 流的异常处理

### 20.14\_IO流(流的标准处理异常代码1.6版本及其以前)

try finally嵌套,可能有问题的写在了try中，将关流机制全部放在了finally中。

---------------------------------------------------------------------------------------------------------------------------

FileInputStream fis = null; //成员变量开始的时候应该是赋给初始化值的，后边的try中可能是读取不到的。

FileOutputStream fos = null;

try {

fis = new FileInputStream("aaa.txt");

fos = new FileOutputStream("bbb.txt");

int b;

while((b = fis.read()) != -1) {

fos.write(b);

}

} finally {

try {

if(fis != null) //null不能调用方法的，所以就应该判断的。

fis.close();

}finally { //能关一个是一个，读和写的流尽量关掉。

if(fos != null)

fos.close();

}

} //开发中不常见，只是在面试的时候比较常见，但是也是建议尽量在开发中也这么使用的，因为节省内存。

---------------------------------------------------------------------------------------------------------------------------

### 20.15\_IO流(流的标准处理异常代码1.7版本)

try close

---------------------------------------------------------------------------------------------------------------------------

try(

FileInputStream fis = new FileInputStream("aaa.txt");

FileOutputStream fos = new FileOutputStream("bbb.txt");

MyClose mc = new MyClose();

){

int b;

while((b = fis.read()) != -1) {

fos.write(b);

}

}

class MyClose implements **AutoCloseable { //若是没有继承就是无法写到try中的！**

public void close() {

System.out.println("我关了");

}

}

---------------------------------------------------------------------------------------------------------------------------

原理

在try()中创建的流对象**必须实现了AutoCloseable这个接口,**如果实现了,在try后面的{}(读写代码)执行后就会自动调用,流对象的close方法将流关掉

## 流的一些关键操作

### 20.16\_IO流(图片加密)

给图片加密

---------------------------------------------------------------------------------------------------------------------------

BufferedInputStream bis = new BufferedInputStream(new FileInputStream("a.jpg"));

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("b.jpg"));

int b;

while((b = bis.read()) != -1) {

bos.write(b ^ 123);

}

bis.close();

bos.close();

---------------------------------------------------------------------------------------------------------------------------

### 20.17\_IO流(拷贝文件)

在控制台录入文件的路径,将文件拷贝到当前项目下

---------------------------------------------------------------------------------------------------------------------------

分析:

1,定义方法对键盘录入的路径进行判断,如果是文件就返回

2,在主方法中接收该文件

3,读和写该文件

public class Test2 {

public static void main(String[] args) throws IOException {

File file = getFile(); //获取文件

BufferedInputStream bis = new BufferedInputStream(new FileInputStream(file));

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(file.getName()));

int b;

while((b = bis.read()) != -1) {

bos.write(b);

}

bis.close();

bos.close();

}

public static File getFile() {

Scanner sc = new Scanner(System.in); //创建键盘录入对象

System.out.println("请输入一个文件的路径:");

while(true) {

String line = sc.nextLine(); //接收键盘录入的路径

File file = new File(line); //封装成File对象,并对其进行判断

if(!file.exists()) {

System.out.println("您录入的文件路径不存在,请重新录入:");

}else if(file.isDirectory()) {

System.out.println("您录入的是文件夹路径,请重新录入:");

}else {

return file;

}

}

}

}

---------------------------------------------------------------------------------------------------------------------------

### 20.18\_IO流(录入数据拷贝到文件)

将键盘录入的数据拷贝到当前项目下的text.txt文件中,键盘录入数据当遇到quit时就退出

---------------------------------------------------------------------------------------------------------------------------

分析:

1,创建键盘录入对象

2,创建输出流对象,关联text.txt文件

3,定义无限循环

4,遇到quit退出循环

5,如果不quit,就将内容写出

6,关闭流

Scanner sc = new Scanner(System.in);

FileOutputStream fos = new FileOutputStream("text.txt");

System.out.println("请输入:");

while(true) {

String line = sc.nextLine();

if("quit".equals(line))

break;

fos.write(line.getBytes());

fos.write("\r\n".getBytes());

}

fos.close();

---------------------------------------------------------------------------------------------------------------------------

# 21 IO字符流

## 字符流的概念

### 21.01\_IO流(字符流FileReader★★)

1.字符流是什么

字符流是可以直接读写字符的IO流

**字符流读取字符, 就要先读取到字节数据, 然后转为字符. 如果要写出字符, 需要把字符转为字节再写出.**

2.FileReader

FileReader类的read()方法可以按照字符大小读取

---------------------------------------------------------------------------------------------------------------------------

public class Demo1\_FileReader {

public static void main(String[] args) throws IOException {

//demo1(); 下边进行了代码的优化。

**FileReader fr = new FileReader("xxx.txt");**  //创建输入流对象,关联aaa.txt

**int c;**

//码表功劳,但是不研究码表.

**while((c = fr.read()) != -1) {** //通过项目默认的码表一次读取一个字符,也就是说当遇到汉字的时候是读两个字节，遇到汉字的第一个字节的时候是负数，直接就将后边的也读了。当遇到符号的时候，符号人家是正值所以就是只读了一个。这其实是最终的就是人家码表的功劳。将读到的字符赋值给ch。

**System.out.print((char)c);**  **//将读到的字符强转后打印，但是运用后边的字符写就不用强转的，还是流程原理的问题。**

}

**fr.close();**

}

public static void demo1() throws FileNotFoundException, IOException {

FileReader fr = new FileReader("xxx.txt");

int x = fr.read();

System.out.println(x); //读的是字节那么在这里打印的将是字节数字。

char c = (char)x; //打印那个根本就是没有意义的，那么就需要进行强转的。

System.out.println(c);

fr.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

### 21.02\_IO流(字符流FileWriter★★)

FileWriter类的write()方法可以自动把字符转为字节写出。将字符转换成字节写在了硬盘上，然后通过编码表再翻译，我们就会看到具体的字符了。记住这个读和写的流程。

---------------------------------------------------------------------------------------------------------------------------

public class Demo2\_FileWriter {

public static void main(String[] args) throws IOException {

FileWriter fw = new FileWriter("yyy.txt"); //没有文件这种写的路径都是自动进行创建的，但是会进行清空，不想清空的话就在后边加上true

fw.write("大家要努力,要坚持!!!"); //注意这里是可以直进行书写的，但是字节就是不可以的需要变成自己数组，这里的方法很多可以通过提示知道的。

fw.write(97);//直接的写上编码。

fw.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

### 21.03\_IO流(字符流的拷贝★★★★★★★★)

---------------------------------------------------------------------------------------------------------------------------

public static void demo1() throws FileNotFoundException, IOException {

**FileReader fr = new FileReader("xxx.txt");**

**FileWriter fw = new FileWriter("zzz.txt");**

**int c;**

**while((c = fr.read()) != -1) {**

**fw.write(c);**

**}**

**fr.close();**

**fw.close();** //Writer类中有一个2k的小缓冲区（一个字符两个字节）,如果不关流,就会将内容写到缓冲区里,关流会将缓冲区内容刷新,再关闭

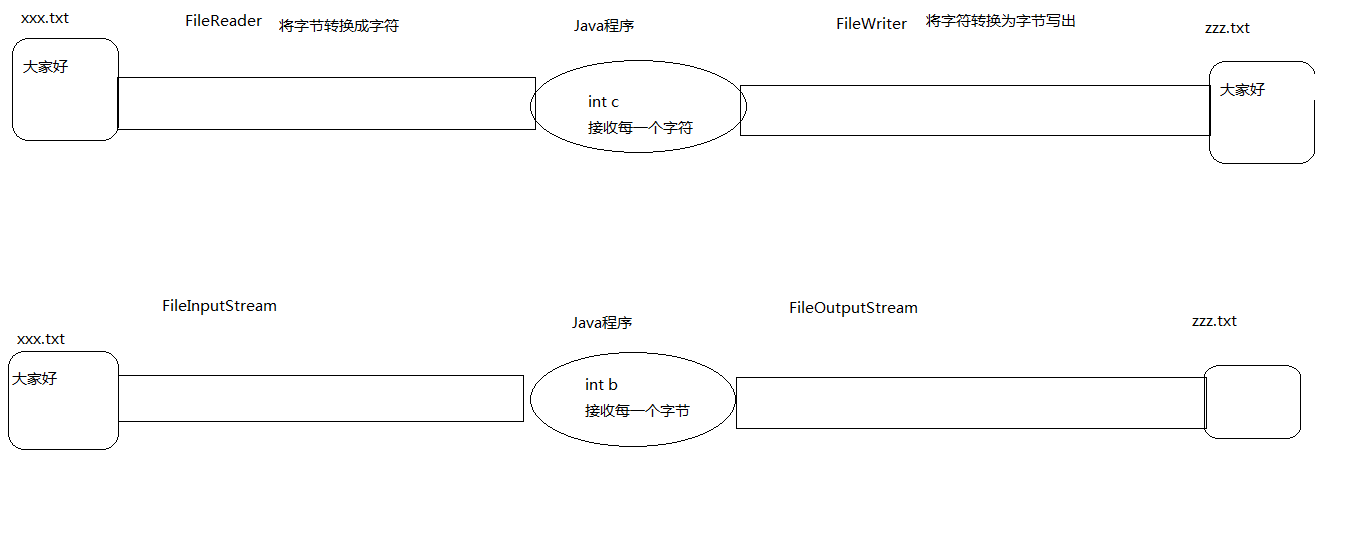
}

---------------------------------------------------------------------------------------------------------------------------

### 21.04\_IO流(什么情况下使用字符流★★★★)

**字符流也可以拷贝文本文件, 但不推荐使用. 因为读取时会把字节转为字符, 写出时还要把字符转回字节.**

**程序需要读取一段文本, 或者需要写出一段文本的时候可以使用字符流读取的时候是按照字符的大小读取的,不会出现半个中文写出的时候可以直接将字符串写出,不用转换为字节数组**



### 21.05\_IO流(字符流是否可以拷贝非纯文本的文件)

不可以拷贝非纯文本的文件，因为在读的时候会将字节转换为字符,在转换过程中,**可能找不到对应的字符,就会用?代替**,写出的时候会将字符转换成字节写出去。如果是?,直接写出,这样写出之后的文件就乱了,看不了了

## 字符流的拷贝

### 21.06\_IO流(自定义字符数组的拷贝)

字符中是没有像字节一样的available方法的。

因为在文件中是可以知道字节大小的，**但是不知道字符数组的大小，因为采用什么码表我们是不知道的，所以就不可能有available方法。**

---------------------------------------------------------------------------------------------------------------------------

public static void demo3() throws FileNotFoundException, IOException {

FileReader fr = new FileReader("xxx.txt");

FileWriter fw = new FileWriter("yyy.txt");

**char[] arr = new char[1024];**

int len;

while((len = fr.read(arr)) != -1) { //将文件上的数据读取到字符数组中

fw.write(arr,0,len); //将字符数组中的数据写到文件上

}

fr.close();

fw.close();

}

---------------------------------------------------------------------------------------------------------------------------

### 21.07\_IO流(带缓冲的字符流)

BufferedReader的read()方法读取字符时会一次读取若干字符到缓冲区, 然后逐个返回给程序, 降低读取文件的次数, 提高效率

BufferedWriter的write()方法写出字符时会先写到缓冲区, 缓冲区写满时才会写到文件, 降低写文件的次数, 提高效率

---------------------------------------------------------------------------------------------------------------------------

BufferedReader br = new BufferedReader(new FileReader("aaa.txt")); //创建字符输入流对象,关联aaa.txt

BufferedWriter bw = new BufferedWriter(new FileWriter("bbb.txt")); //创建字符输出流对象,关联bbb.txt

int ch;

while((ch = br.read()) != -1) { //read一次,会先将缓冲区读满,从缓冲去中一个一个的返给临时变量ch

bw.write(ch); //write一次,是将数据装到字符数组,装满后再一起写出去

}

br.close(); //关流

bw.close();

---------------------------------------------------------------------------------------------------------------------------

### 21.08\_IO流(readLine()和newLine()方法)

BufferedReader的readLine()方法可以读取一行字符(不包含换行符号)，读取一个文本行。通过下列字符之一即可认为某行已终止：换行 ('\n')、回车 ('\r') 或回车后直接跟着换行。返回：**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** 。

BufferedWriter的newLine()可以输出一个跨平台的换行符号"\r\n"

**是以上两个的特有的方法。**

---------------------------------------------------------------------------------------------------------------------------

BufferedReader br = new BufferedReader(new FileReader("aaa.txt"));

BufferedWriter bw = new BufferedWriter(new FileWriter("bbb.txt"));

String line;

while((line = br.readLine()) != null) {

bw.write(line);

//bw.write("\r\n"); //只支持windows系统

bw.newLine(); //跨平台的

}

br.close();

bw.close();

---------------------------------------------------------------------------------------------------------------------------

## 字符流的练习

### 21.09\_IO流(将文本反转)

将一个文本文档上的文本反转,第一行和倒数第一行交换,第二行和倒数第二行交换

---------------------------------------------------------------------------------------------------------------------------

分析:

1,创建输入输出流对象

2,创建集合对象

3,将读到的数据存储在集合中

4,倒着遍历集合将数据写到文件上

5,关流

注意事项:

**流对象尽量晚开早关**

public class Test1 {

public static void main(String[] args) throws IOException {

//改写后是尽量晚开早关

// 1,创建输入输出流对象

BufferedReader br = new BufferedReader(new FileReader("zzz.txt"));

//2,创建集合对象

**ArrayList<String> list = new ArrayList<>();**

**//3,将读到的数据存储在集合中**

String line;

while((line = br.readLine()) != null) {

list.add(line);

}

br.close(); //关流

//4,倒着遍历集合将数据写到文件上

BufferedWriter bw = new BufferedWriter(new FileWriter("revzzz.txt"));

for(int i = list.size() - 1; i >= 0; i--) {

//bw.write(list.get(i));

bw.newLine();

}

//5,关流

bw.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

### 21.10\_IO流(LineNumberReader)

LineNumberReader是BufferedReader的子类, 具有相同的功能, 并且可以统计行号

注意是一种对象了,有setget方法,是被装饰了,被封装了.

调用getLineNumber()方法可以获取当前行号

调用setLineNumber()方法可以设置当前行号

---------------------------------------------------------------------------------------------------------------------------

LineNumberReader lnr = new LineNumberReader(new FileReader("aaa.txt"));

String line;

lnr.setLineNumber(100); //设置起始行号，下边是从101开始的。

while((line = lnr.readLine()) != null) {

System.out.println(lnr.getLineNumber() + ":" + line); //获取行号

}

lnr.close();

---------------------------------------------------------------------------------------------------------------------------

## 21.11\_IO流(装饰设计模式)

装饰设计模式的好处是:

**耦合性不强,被装饰的类的变化与装饰类的变化无关,一般是继承关系就是为了少写代码.有一些功能也是不需要增强的.**

---------------------------------------------------------------------------------------------------------------------------

public class Demo6\_Wrap {

public static void main(String[] args) {

HeiMaStudent hms = new HeiMaStudent(new Student());

hms.code();

}

}

interface Coder {

public void code();

}

class Student implements Coder {

public void code() {

System.out.println("javase");

System.out.println("javaweb");

}

}

class HeiMaStudent implements Coder {

**//1,获取被装饰类的引用**

private Student s; //获取学生引用

**//2,在构造方法中传入被装饰类的对象**

public HeiMaStudent(Student s) {

this.s = s;

}

**//3,对原有的功能进行升级**

@Override

public void code() {

s.code();

System.out.println("ssh");

System.out.println("数据库");

System.out.println("大数据");

System.out.println("...");

}

}

---------------------------------------------------------------------------------------------------------------------------

## 21.12\_IO流(使用指定的码表读写字符)

**总结:主要是针对文本文件,而且是不相同的情况下才采用转换流.**

FileReader是使用默认码表读取文件, 如果需要使用指定码表读取, 那么可以使用**InputStreamReade**r(字节流,编码表)

FileWriter是使用默认码表写出文件, 如果需要使用指定码表写出, 那么可以使用**OutputStreamWriter**(字节流,编码表)（utf-8怎么都行，但是GBK必须大写！）

---------------------------------------------------------------------------------------------------------------------------

BufferedReader br = //高效的用指定的编码表读

new BufferedReader(new InputStreamReader(new FileInputStream("UTF-8.txt"), "UTF-8"));

BufferedWriter bw = //高效的用指定的编码表写

new BufferedWriter(new OutputStreamWriter(new FileOutputStream("GBK.txt"), "GBK"));

int ch;

while((ch = br.read()) != -1) {

bw.write(ch);

}

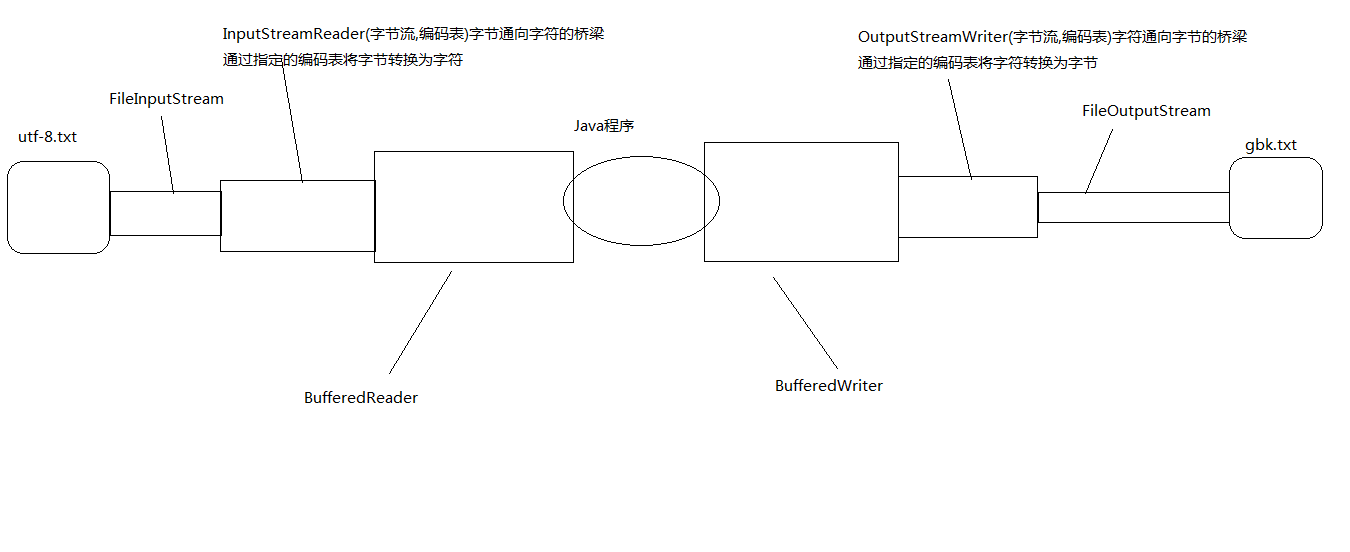
br.close();

bw.close();

---------------------------------------------------------------------------------------------------------------------------

### 21.13\_IO流(转换流图解)

画图分析转换流



## 字符流的使用

### 21.14\_IO流(获取文本上字符出现的次数)

获取一个文本上每个字符出现的次数,将结果写在times.txt上

---------------------------------------------------------------------------------------------------------------------------

public class Test2 {

/\*\*

获取一个文本上每个字符出现的次数,将结果写在times.txt上

1,创建带缓冲区的输入流对象

2,创建双列集合对象,目的是把字符当作键,把字符出现的次数当作值

3,通过读取不断向集合中存储,存储的时候要判断,如果不包含这个键就将键和值为1存储,如果包含就将键和值加1存储

4,关闭输入流

5,创建输出流对象

6,将结果写出

7,关闭输出流

@throws IOException

\*/

public static void main(String[] args) throws IOException {

**//1,创建带缓冲区的输入流对象**

BufferedReader br = new BufferedReader(new FileReader("zzz.txt"));

**//2,创建双列集合对象,目的是把字符当作键,把字符出现的次数当作值**

HashMap<Character, Integer> hm = new HashMap<>();

**//3,通过读取不断向集合中存储,存储的时候要判断,如果不包含这个键就将键和值为1存储,如果包含就将键和值加1存储**

int c;

while((c = br.read()) != -1) {

char ch = (char)c;

/\*if(!hm.containsKey(ch)) {

hm.put(ch, 1);

}else {

hm.put(ch, hm.get(ch) + 1);

}\*/

hm.put(ch, !hm.containsKey(ch)? 1 : hm.get(ch) + 1);

}

**//4,关闭输入流**

br.close();

**//5,创建输出流对象**

BufferedWriter bw = new BufferedWriter(new FileWriter("times.txt"));

**//6,将结果写出**

for (Character key : hm.keySet()) {

bw.write(key + "=" + hm.get(key));

}

bw.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

### 21.15\_IO流(试用版软件)

当我们下载一个试用版软件,没有购买正版的时候,每执行一次就会提醒我们还有多少次使用机会用学过的IO流知识,模拟试用版软件,试用10次机会,执行一次就提示一次您还有几次机会,如果次数到了提示请购买正版

---------------------------------------------------------------------------------------------------------------------------

public class Test4 {

/\*\*

@throws IOException

分析:

1,创建带缓冲的输入流对象,因为要使用readLine方法,可以保证数据的原样性

2,将读到的字符串转换为int数

3,对int数进行判断,如果大于0,就将其--写回去,如果不大于0,就提示请购买正版

4,在if判断中要将--的结果打印,并将结果通过输出流写到文件上

\*/

public static void main(String[] args) throws IOException {

**//1,创建带缓冲的输入流对象,因为要使用readLine方法,可以保证数据的原样性**

BufferedReader br = new BufferedReader(new FileReader("config.txt"));

**//2,将读到的字符串转换为int数**

String line = br.readLine();

int times = Integer.parseInt(line); //将数字字符串转换为数字

**//3,对int数进行判断,如果大于0,就将其--写回去,如果不大于0,就提示请购买正版**

if(times > 0) {

**//4,在if判断中要将--的结果打印,并将结果通过输出流写到文件上**

System.out.println("您还有" + times-- + "次机会");

FileWriter fw = new FileWriter("config.txt");

fw.write(times + "");

fw.close();

}else {

System.out.println("您的试用次数已到,请购买正版");

}

**//关闭流**

br.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

## 21.16\_File类(递归)

5的阶乘

**注意:**

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

构造方法是否可以递归调用?构造方法不能使用递归调用。

递归调用是否必须有返回值?不一定(可以有,也可以没有)。

递归要有出口!

---------------------------------------------------------------------------------------------------------------------------

public class Demo8\_Digui {

/\*\*

@param args

递归:方法自己调用自己

5!

5 4 3 2 1

5 fun(4)(代表4!)

4 fun(3)(代表3!)

3 fun(2)(代表2!)

2 fun(1)(代表1!)

\*/

public static void main(String[] args) {

/\*int result = 1;

for(int i = 1; i <= 5; i++) {

result = result i;

}

System.out.println(result);\*/

System.out.println(fun(3000));

}

public static int fun(int num) {

if(num == 1) {

return 1;

}else {

return num fun(num - 1);

}

}

}

---------------------------------------------------------------------------------------------------------------------------

## 21.17\_File类(练习)

需求: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.java文件名

---------------------------------------------------------------------------------------------------------------------------

public class Test5 {

/\*\*

分析:

从键盘接收一个文件夹路径

1,如果录入的是不存在,给与提示

2,如果录入的是文件路径,给与提示

3,如果是文件夹路径,直接返回

打印出该文件夹下所有的.java文件名

1,获取到该文件夹路径下的所有的文件和文件夹,存储在File数组中

2,遍历数组,对每一个文件或文件夹做判断

3,如果是文件,并且后缀是.java的,就打印

4,如果是文件夹,就递归调用

\*/

public static void main(String[] args) {

File dir = getDir();

printJavaFile(dir);

}

/\*

获取键盘录入的文件夹路径

1,返回值类型File

2,不需要有参数

\*/

public static File getDir() {

Scanner sc = new Scanner(System.in); //创建键盘录入对象

System.out.println("请输入一个文件夹路径");

while(true) {

String line = sc.nextLine(); //将键盘录入的文件夹路径存储

File dir = new File(line); //封装成File对象

if(!dir.exists()) {

System.out.println("您录入的文件夹路径不存在,请重新录入");

}else if(dir.isFile()) {

System.out.println("您录入的是文件路径,请重新录入文件夹路径");

}else {

return dir;

}

}

}

/\*

获取文件夹路径下的所.java文件

1,返回值类型 void

2,参数列表File dir

\*/

public static void printJavaFile(File dir) {

//1,获取到该文件夹路径下的所有的文件和文件夹,存储在File数组中

File[] subFiles = dir.listFiles();

//2,遍历数组,对每一个文件或文件夹做判断

for (File subFile : subFiles) {

//3,如果是文件,并且后缀是.java的,就打印

if(subFile.isFile() && subFile.getName().endsWith(".java")) {

System.out.println(subFile);

//4,如果是文件夹,就递归调用

}else if (subFile.isDirectory()){

printJavaFile(subFile);

}

}

}

}

---------------------------------------------------------------------------------------------------------------------------

## 21.18\_IO流(总结)

1.会用BufferedReader读取GBK码表和UTF-8码表的字符

2.会用BufferedWriter写出字符到GBK码表和UTF-8码表的文件中

3.会使用BufferedReader从键盘读取一行

# 22 其他流

## 序列流

### 22.01\_IO流(序列流)(了解)

1.什么是序列流

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.

2.使用方式

整合两个: SequenceInputStream(InputStream, InputStream)

---------------------------------------------------------------------------------------------------------------------------

FileInputStream fis1 = new FileInputStream("a.txt"); //创建输入流对象,关联a.txt

FileInputStream fis2 = new FileInputStream("b.txt"); //创建输入流对象,关联b.txt

SequenceInputStream sis = new SequenceInputStream(fis1, fis2);//将两个流整合成一个流

FileOutputStream fos = new FileOutputStream("c.txt"); //创建输出流对象,关联c.txt

int b;

while((b = sis.read()) != -1) { fos.write(b); //写到指定文件上

}

sis.close();

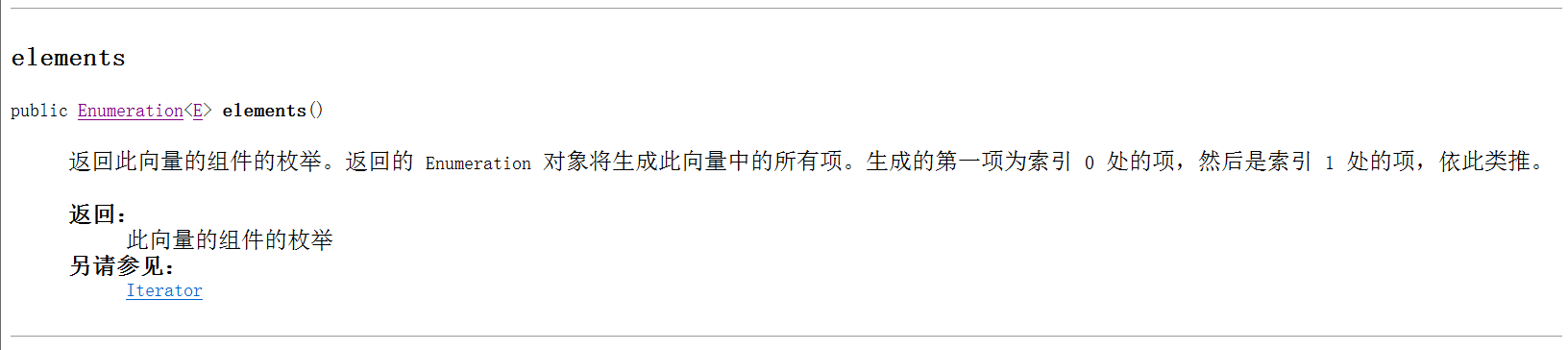
fos.close();

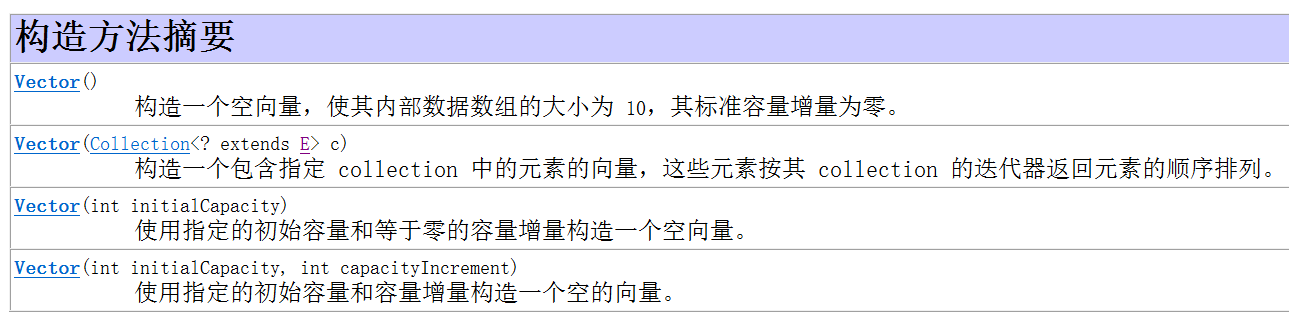
---------------------------------------------------------------------------------------------------------------------------

### 22.02\_IO流(序列流整合多个)(了解)

整合多个: SequenceInputStream(Enumeration)







---------------------------------------------------------------------------------------------------------------------------

FileInputStream fis1 = new FileInputStream("a.txt"); //创建输入流对象,关联a.txt

FileInputStream fis2 = new FileInputStream("b.txt"); //创建输入流对象,关联b.txt

FileInputStream fis3 = new FileInputStream("c.txt"); //创建输入流对象,关联c.txt

Vector<InputStream> v = new Vector<>(); //创建vector集合对象

v.add(fis1); //将流对象添加

v.add(fis2);

v.add(fis3);

Enumeration<InputStream> en = v.elements(); //获取枚举引用

SequenceInputStream sis = new SequenceInputStream(en); //传递给SequenceInputStream构造

FileOutputStream fos = new FileOutputStream("d.txt");

int b;

while((b = sis.read()) != -1) {

fos.write(b);

}

sis.close();

fos.close();

**解析**：1，SequenceInputStream([Enumeration](mk:@MSITStore:A:\\Java%20software\\[Java参考文档].JDK_API_1_6_zh_CN.CHM::/java/io/../../java/util/Enumeration.html" \o "java.util 中的接口)<? extends [InputStream](mk:@MSITStore:A:\\Java%20software\\[Java参考文档].JDK_API_1_6_zh_CN.CHM::/java/io/../../java/io/InputStream.html" \o "java.io 中的类)> e)这是SequenceInputStream的构造方法，参数需要一个枚举。

2，集合Vector中就有一个element方法，其中返回的就是一个个枚举，集合中又是可以存储任何对象的当然包括流对象，那么就好办了。

3，将流对象放在集合中，然后通过调用集合的方法返回枚举，这个枚举就是SequenceInputStream的参数。

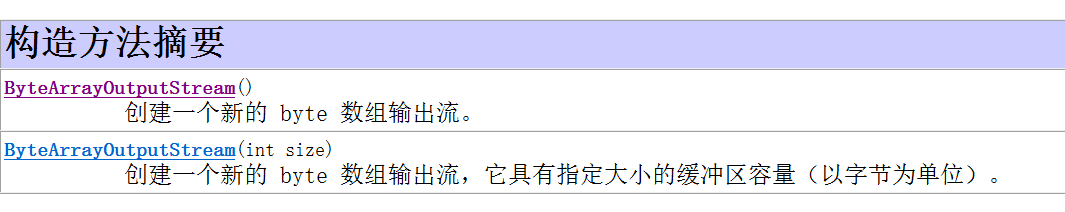
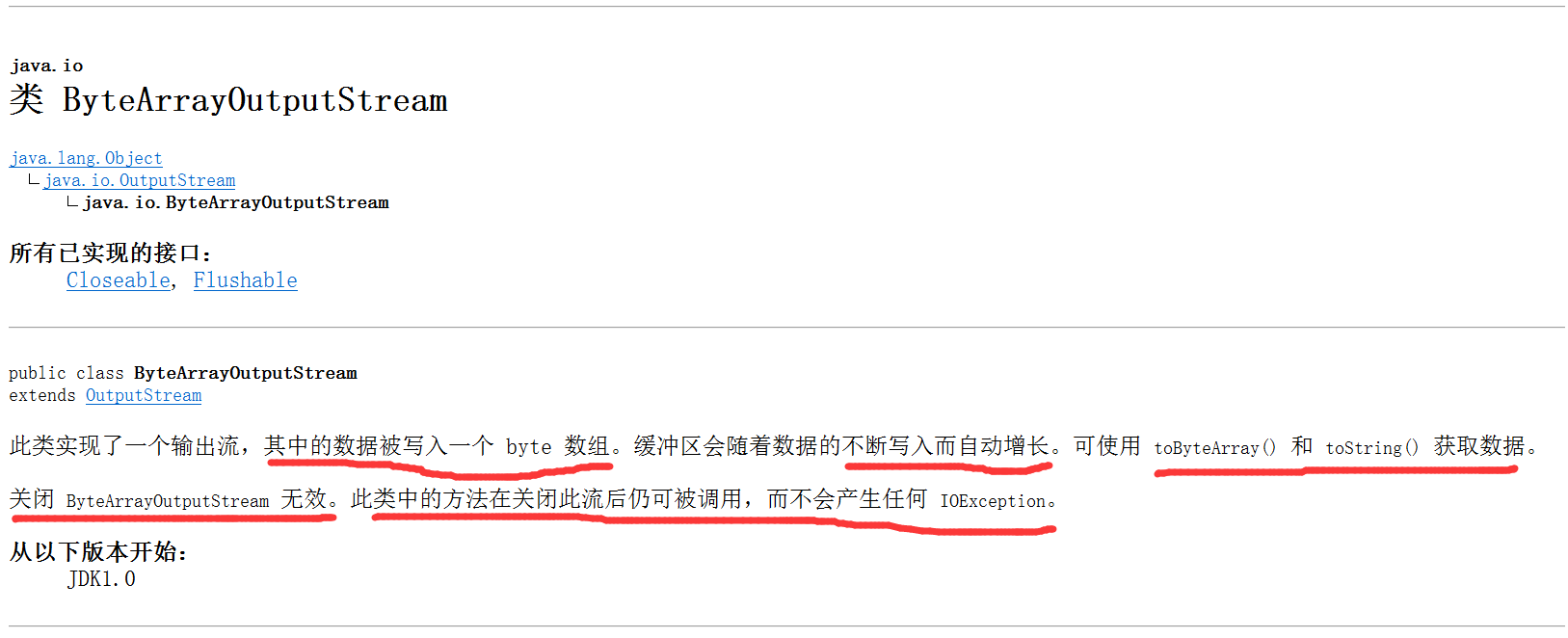
---------------------------------------------------------------------------------------------------------------------------

## 内存流(★★★★★)

### 22.03\_IO流(内存输出流)(掌握)

1.什么是内存输出流

**该输出流可以向内存中写数据, 把内存当作一个缓冲区, 写出之后可以一次性获取出所有数据,就是外部可以将数据全部的写入,底层也是运用的数组,自动的生长创建新的数组,等待装入完毕之后一次性的写出操作**。聊天应用。这个是在内存中开辟空间且在内存直接的输出展示，是一种方法，所以就不需要关流操作。



2.使用方式

创建对象: new ByteArrayOutputStream()

写出数据: write(int), write(byte[])

获取数据: toByteArray() toString()方法

---------------------------------------------------------------------------------------------------------------------------

FileInputStream fis = new FileInputStream("a.txt");

ByteArrayOutputStream baos = new ByteArrayOutputStream(); **//在内存中创建了可以增长的内存数组**

int b;

while((b = fis.read()) != -1) {

baos.write(b); //将读取到的数据逐个写到内存中

}

//byte[] newArr = baos.toByteArray(); 将内存缓冲区中所有的字节存储在newArr中

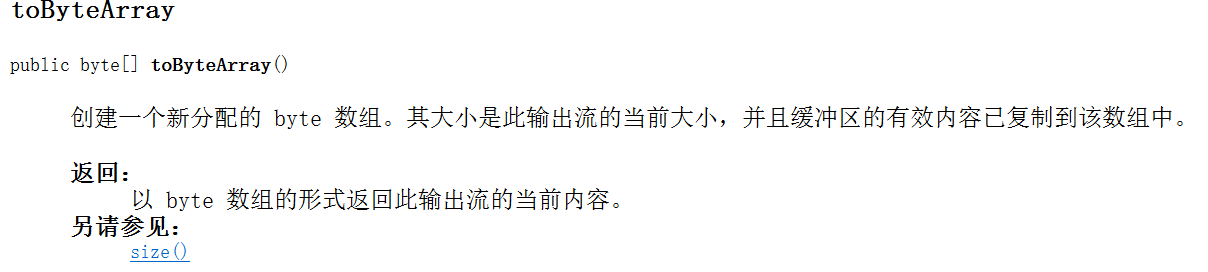
//System.out.println(new String(newArr));

System.out.println(baos); //将缓冲区的内容转换为了字符串,在输出语句中可以省略调用toString方法

fis.close();

**注意：1，toString方法和toByteArray方法是有区别的，后边的是可以放不同的编码表，而前边是采用的平台默认码表。**

---------------------------------------------------------------------------------------------------------------------------



**注意：读取中文的文本出现乱码怎么解决？**

解决方案：1,字符流读取

2,ByteArrayOutputStream

### 22.04\_IO流(内存输出流之黑马面试题)(掌握)

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)

---------------------------------------------------------------------------------------------------------------------------

分析:

1,reda(byte[] b)是字节输入流的方法,创建FileInputStream,关联a.txt

2,创建内存输出流,将读到的数据写到内存输出流中

3,创建字节数组,长度为5（长度为5就说明是不能读取中文的，应该还要采取其他的办法，应该是第二步）

4,将内存输出流的数据全部转换为字符串打印

5,关闭输入流

public static void main(String[] args) throws IOException {

FileInputStream fis = new FileInputStream("a.txt");

ByteArrayOutputStream baos = new ByteArrayOutputStream();

byte[] arr = new byte[5];

int len;

while((len = fis.read(arr)) != -1) {

baos.write(arr, 0, len);

//System.out.println(new String(arr,0,len));

}

System.out.println(baos);

fis.close();

}

---------------------------------------------------------------------------------------------------------------------------

## 随机访问流(多线程下载中用到)

### 22.05\_IO流(随机访问流概述和读写数据)(了解)

A:随机访问流概述

RandomAccessFile概述

**RandomAccessFile类不属于流，是Object类的子类。但它融合了InputStream和OutputStream的功能。是io包下边的。**

**支持对随机访问文件的读取和写入。就是雌雄同体的**

B:read(),write(),seek()

注意：1，元数据指的就是文件的详细信息。

---------------------------------------------------------------------------------------------------------------------------

public class Demo08\_RandomAccessFile {

public static void main(String[] args) throws IOException {

RandomAccessFile raf = new RandomAccessFile("g.txt", "rw");

//raf.write(97);

//int x = raf.read();

//System.out.println(x);

raf.seek(0); //在指定位置设置指针

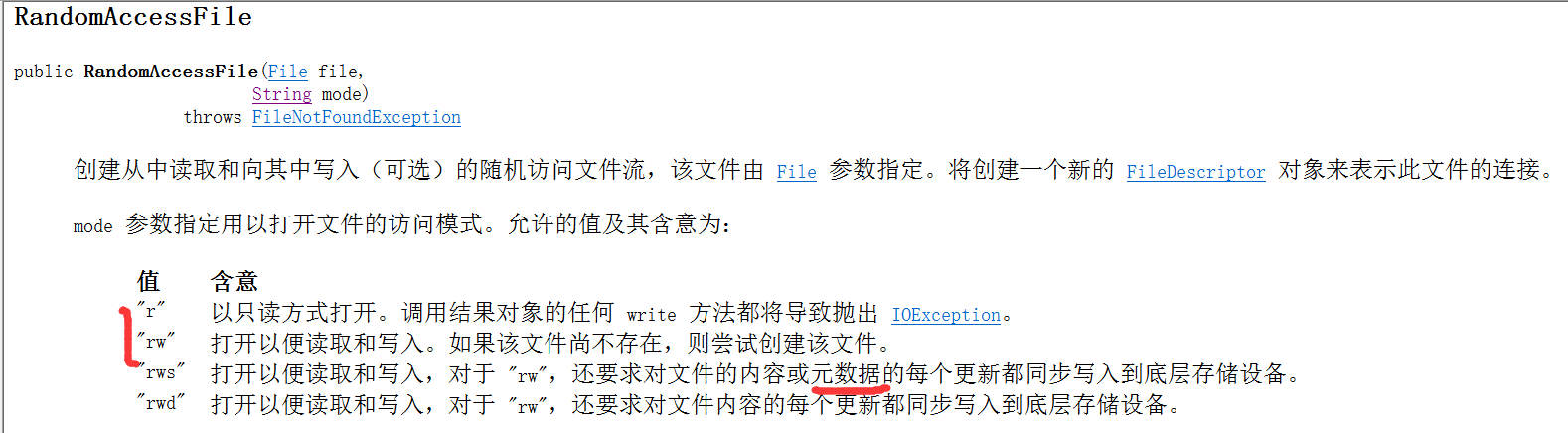
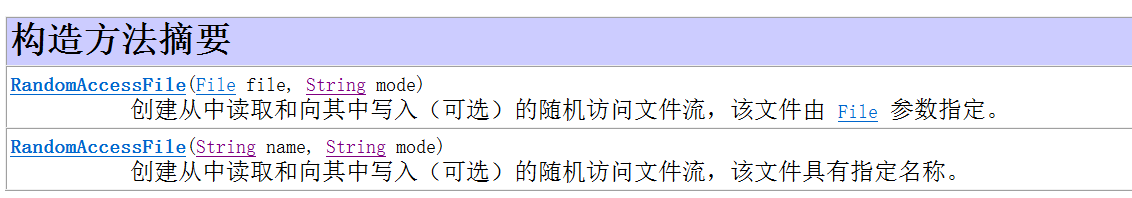
raf.write(98);

raf.close();

}

}

---------------------------------------------------------------------------------------------------------------------------



## 对象操作流

### 22.06\_IO流(对象操作流ObjecOutputStream)(了解)

1.什么是对象操作流

**该流可以将一个对象写出, 或者读取一个对象到程序中. 也就是执行了序列化和反序列化的操作.**

2.使用方式

写出: new ObjectOutputStream(OutputStream), writeObject()

---------------------------------------------------------------------------------------------------------------------------

public static void demo1() throws IOException, FileNotFoundException {

Person p1 = new Person("张三", 23);

Person p2 = new Person("李四", 24);

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream ("e.txt"));

oos.writeObject(p1);

oos.writeObject(p2);

oos.close();

}

注意：

1，对象必须实现Serializable接口，这个类中是没有任何方法的，只是一个制定对着的接口罢了。

2，writeObject();方法是将对象变成二进制的文件贮存在文本中永久保存的，所以写出来的文件我们看不懂的，因为将对象用码表翻译成文本，也有识别的也有不能识别的，就会出现乱码文件，我们不需要看懂只需要存储即可，用的时候再翻译过来就行了。

3，应用：打游戏时候角色属性的存储。

---------------------------------------------------------------------------------------------------------------------------

### 22.07\_IO流(对象操作流ObjectInputStream)(了解)

读取: new ObjectInputStream(InputStream), readObject()

---------------------------------------------------------------------------------------------------------------------------

public class Demo3\_ObjectInputStream {

public static void main(String[] args) throws IOException, ClassNotFoundException {

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("e.txt"));

Person p1 = (Person) ois.readObject(); //强转，上边的是Object你给的是p1

Person p2 = (Person) ois.readObject();

//Person p3 = (Person) ois.readObject(); //当文件读取到了末尾时出现EOFException（end of file）

System.out.println(p1);

System.out.println(p2);

ois.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

### 22.08\_IO流(对象操作流优化)(了解)

将对象存储在集合中写出，一次写和一次读就是优化了。

---------------------------------------------------------------------------------------------------------------------------

Person p1 = new Person("张三", 23);

Person p2 = new Person("李四", 24);

Person p3 = new Person("马哥", 18);

Person p4 = new Person("辉哥", 20);

ArrayList<Person> list = new ArrayList<>();

list.add(p1);

list.add(p2);

list.add(p3);

list.add(p4);

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("f.txt"));

oos.writeObject(list); //写出集合对象

oos.close();

-----------------------------------

读取到的是一个集合对象

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("f.txt"));

ArrayList<Person> list = (ArrayList<Person>)ois.readObject(); //泛型在运行期会被擦除,索引运行期相当于没有泛型

//想去掉黄色可以加注解 @SuppressWarnings("unchecked")

for (Person person : list) {

System.out.println(person);

}

ois.close();

---------------------------------------------------------------------------------------------------------------------------

### 22.09\_IO流(加上id号)(了解)

注意

1，要写出的对象必须实现Serializable接口才能被序列化

2，不用必须加id号

3，关于id号就是你不存档直接的去读档那么就会产生两个随机的id号，肯定是不同的，就是说两者不一样，若是先存后读那么就不用发生不一样的问题了，是一样的。

---------------------------------------------------------------------------------------------------------------------------

private static final long serialVersionUID = 2L;

---------------------------------------------------------------------------------------------------------------------------

## 数据输出流

### 22.10\_IO流(数据输入输出流)(了解)

1.什么是数据输入输出流

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.

2.使用方式

DataOutputStream(OutputStream), writeInt(), writeLong()

--------------------------------------------------------------------------------------------------------------------------- DataOutputStream dos = new DataOutputStream(new FileOutputStream("b.txt"));

dos.**writeInt**(997);

dos.writeInt(998);

dos.writeInt(999);

dos.close();

DataInputStream(InputStream), readInt(), readLong()

DataInputStream dis = new DataInputStream(new FileInputStream("b.txt"));

int x = dis.**readInt**();

int y = dis.readInt();

int z = dis.readInt();

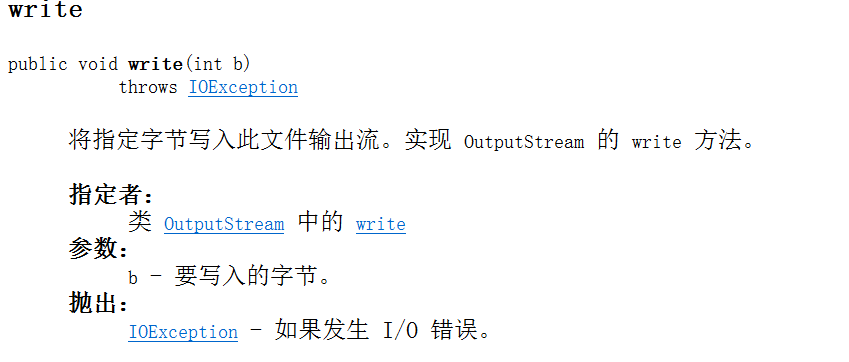
System.out.println(x);

System.out.println(y);

System.out.println(z);

dis.close();

---------------------------------------------------------------------------------------------------------------------------



---------------------------------------------------------------------------------------------------------------------------

public class Demo09\_Data {

/\*\*

00000000 00000000 00000011 11100101 int类型997

11100101

00000000 00000000 00000000 11100101

读的时候是只读一个字节就会将int类型数字前边的三个字节进行砍掉那么就只剩下11100101，将这个数字写到文本中，读的时候又将前边补上三个字节的零。那么就会变数字了。

\*/

public static void main(String[] args) throws IOException {

//demo1();

//demo2();

DataInputStream dis = new DataInputStream(new FileInputStream("h.txt"));

int x = dis.readInt();

int y = dis.readInt();

int z = dis.readInt();

System.out.println(x);

System.out.println(y);

System.out.println(z);

dis.close();

}

public static void demo2() throws FileNotFoundException, IOException {

FileInputStream fis = new FileInputStream("h.txt");

int x = fis.read();

int y = fis.read();

int z = fis.read();

System.out.println(x);

System.out.println(y);

System.out.println(z);

fis.close();

}

public static void demo1() throws FileNotFoundException, IOException {

FileOutputStream fos = new FileOutputStream("h.txt");

fos.write(997);

fos.write(998);

fos.write(999);

fos.close();

}

}

---------------------------------------------------------------------------------------------------------------------------

## 打印流

### 22.11\_IO流(打印流的概述和特点)(掌握)

什么是打印流



**PrintStream 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, 而且可以使用自动刷出的模式**

**System.out就是一个PrintStream, 其默认向控制台输出信息**

---------------------------------------------------------------------------------------------------------------------------

public static void demo1() {

System.out.println("aaa");

PrintStream ps = System.out; //获取标注输出流

ps.println(97); //底层通过Integer.toString()将97转换成字符串并打印

ps.write(97); //查找码表,找到对应的a并打印

Person p1 = new Person("张三", 23);

ps.println(p1); //默认调用p1的toString方法

Person p2 = null; //打印引用数据类型,如果是null,就打印null,如果不是null就打印对象的toString方法

ps.println(p2);

ps.close();

}

---------------------------------------------------------------------------------------------------------------------------

2.使用方式

打印: print(), println()

自动刷出: PrintWriter(OutputStream out, boolean autoFlush, String encoding)

**PrintStream和PrintWriter分别是打印的字节流和字符流，打印流只操作数据目的（也就是说仅仅只是将缓冲区的文件打印出来而并不针对数据源进行任何的操作）**

---------------------------------------------------------------------------------------------------------------------------

PrintWriter pw = new PrintWriter(new FileOutputStream("g.txt"), true);

pw.write(97);

pw.print("大家好");

pw.println("你好"); //自动刷出,只针对的是println方法

pw.close();

注意：1，只是针对println有用，而并不针对其他的，若是将println放在其他的后边就会将其他的也打印出来；2，其他的必须得进行关流；3，这个并没有什么用处的。

---------------------------------------------------------------------------------------------------------------------------

public static void main(String[] args) throws IOException {

PrintWriter pw = new PrintWriter("f.txt");

pw.println(97); //打印的将是97

pw.write(97); //打印的将是a

//pw.close(); //上边都是写到缓冲区中的，若是将流关掉就不会输出。

}

---------------------------------------------------------------------------------------------------------------------------

### 22.12\_IO流(标准输入输出流概述和输出语句)

1.什么是标准输入输出流(掌握)

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

---------------------------------------------------------------------------------------------------------------------------

public static void demo1() throws IOException {

InputStream is = System.in;

int x = is.read();

System.out.println(x); //这里若是在控制台输入48将会读到52，因为只读了一个字节4，码表翻译过来就是52.

is.close(); //无论底下创建几个输出流，他的流已经被关闭了,键盘输入一般是不用关流的

InputStream is2 = System.in;

int y = is2.read();

System.out.println(y);

}

---------------------------------------------------------------------------------------------------------------------------

2.修改标准输入输出流(了解)

修改输入流: System.setIn(InputStream)

修改输出流: System.setOut(PrintStream)

---------------------------------------------------------------------------------------------------------------------------

System.setIn(new FileInputStream("a.txt")); //修改标准输入流

System.setOut(new PrintStream("b.txt")); //修改标准输出流

InputStream in = System.in; //获取标准输入流

PrintStream ps = System.out; //获取标准输出流

int b;

while((b = in.read()) != -1) { //从a.txt上读取数据

ps.write(b); //将数据写到b.txt上

}

//System.out.println();也是一个输出流,不用关,因为没有和硬盘上的文件产生关联的管道

in.close();

ps.close();

注意：1，平时是不这么用的，用fileinputstream就行了，这种方法知道就行了。

---------------------------------------------------------------------------------------------------------------------------

### 22.13\_IO流(修改标准输入输出流拷贝图片)(了解)

---------------------------------------------------------------------------------------------------------------------------

System.setIn(new FileInputStream("IO图片.png")); //改变标准输入流

System.setOut(new PrintStream("copy.png")); //改变标准输出流

InputStream is = System.in; //获取标准输入流

PrintStream ps = System.out; //获取标准输出流

int len;

byte[] arr = new byte[1024 8];

while((len = is.read(arr)) != -1) {

ps.write(arr, 0, len);

}

is.close();

ps.close();

注意：1，开发中绝对的不推荐使用的，只是了解的。

---------------------------------------------------------------------------------------------------------------------------

### 22.14\_IO流(两种方式实现键盘录入)(了解)

A:BufferedReader的readLine方法。

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

B:Scanner

---------------------------------------------------------------------------------------------------------------------------

public static void main(String[] args) throws IOException {

/\*BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); //InputStreamReader转换流

String line = br.readLine();

System.out.println(line);

br.close();\*/

Scanner sc = new Scanner(System.in); //System.in可以是很多的，不仅仅止于此

String line = sc.nextLine();

System.out.println(line);

sc.close();

}

注意：1，推荐使用Scanner方法，因为里边的方法还有很多的，而上边的只是简单的一种读取。

---------------------------------------------------------------------------------------------------------------------------

## 关于Properties

### 22.15\_IO流(Properties的概述和作为Map集合的使用)(了解)

A:Properties的概述

**Properties是一个双列集合并不是流**。主要应用于配置文件，里边的东西只能放String，所以不加泛型，**Properties是Hashtable的子类。有put()和get()方法**

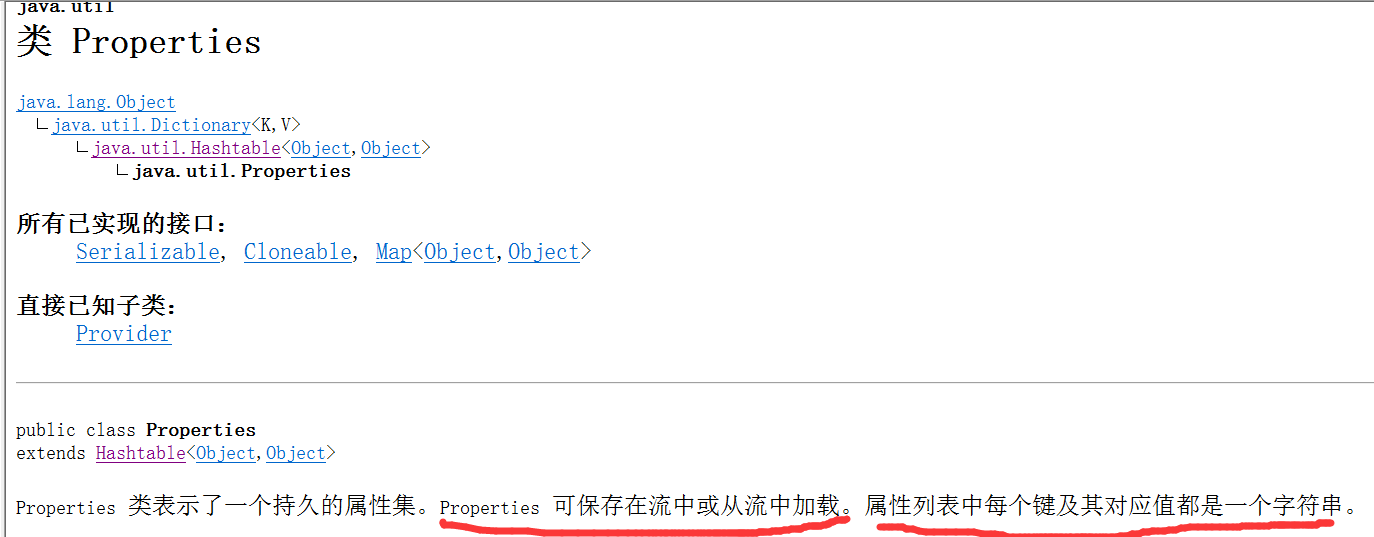
Properties 类表示了一个持久的属性集。

Properties 可保存在流中或从流中加载。

属性列表中每个键及其对应值都是一个字符串（也可以放其他的类型但是很少）。

B:案例演示

Properties作为Map集合的使用

---------------------------------------------------------------------------------------------------------------------------

public static void demo1() {

Properties prop = new Properties();

prop.put("abc", 123);

System.out.println(prop);

}

---------------------------------------------------------------------------------------------------------------------------

### 22.16\_IO流(Properties的特殊功能使用)(了解)

A:Properties的特殊功能

public Object setProperty(String key,String value)

public String getProperty(String key)

public Enumeration<String> PropertyNames()

B:案例演示

Properties的特殊功能

---------------------------------------------------------------------------------------------------------------------------

public static void demo2() {

Properties prop = new Properties();

prop.setProperty("name", "张三");

prop.setProperty("tel", "18912345678");

//System.out.println(prop);

Enumeration<String> en = (Enumeration<String>) prop.propertyNames();

while(en.hasMoreElements()) {

String key = en.nextElement(); //获取Properties中的每一个键

String value = prop.getProperty(key); //根据键获取值

System.out.println(key + "="+ value);

}

}

---------------------------------------------------------------------------------------------------------------------------

### 22.17\_IO流(Properties的load()和store()功能)(了解)

A:Properties的load()和store()功能

B:案例演示

Properties的load()输入和store()输出功能

---------------------------------------------------------------------------------------------------------------------------

public static void main(String[] args) throws FileNotFoundException, IOException {

//demo1();

//demo2();

Properties prop = new Properties();

prop.load(new FileInputStream("config.properties")); //将文件上的键值对读取到集合中

prop.setProperty("tel", "18912345678");

prop.store(new FileOutputStream("config.properties"), null);//第二个参数是对列表参数的描述,可以给值,也可以给null

System.out.println(prop);

}

---------------------------------------------------------------------------------------------------------------------------

# 23递归练习

## 23.01\_File类递归练习(统计该文件夹大小)

需求:1,从键盘接收一个文件夹路径,统计该文件夹大小

体会：

1，递归遇到一个文件夹中包含了文件夹和文件的时候，若是文件夹在前边的话那额就是先进行一个文件夹的彻底判断，然后再回撤到循环中进行下一个文件的判断，流程必须能说出来的。

2，方法名称应该是见名知意，合理得当。

3，方法书写明确返回值和参数类型的。

---------------------------------------------------------------------------------------------------------------------------

/\*\*

需求:1,从键盘接收一个文件夹路径,统计该文件夹大小

从键盘接收一个文件夹路径

1,创建键盘录入对象

2,定义一个无限循环

3,将键盘录入的结果存储并封装成File对象

4,对File对象判断

5,将文件夹路径对象返回

统计该文件夹大小

1,定义一个求和变量

2,获取该文件夹下所有的文件和文件夹listFiles();

3,遍历数组

4,判断是文件就计算大小并累加

5,判断是文件夹,递归调用

\*/

public class Test\_1 {

public static void main(String[] args) {

File file=getDir();

System.out.println(getFilesNum(file));

//File dir = new File("F:\\day06");

//System.out.println(dir.length()); //直接获取文件夹的结果是0

}

/\*

方法定义:1，返回值是文件夹类型

2，参数不参与方法，所以是无参数

\*/

public static File getDir() { // 创建键盘录入方法

Scanner sc = new Scanner(System.in);

System.out.println("请输入一个文件夹路径"); // 1，创建键盘录入对象

while (true) { // 2，有可能输入的是错误的所以应该是定义成无限循环的

String str = sc.nextLine(); //任何输入都是可以转换成字符串的，并且输入应该作为截断

File dir = new File(str); // 3，将输入转换为文件夹

if (!dir.exists()) { // 4，判断路径的文件对象是不是文件夹

System.out.println("您输入了不存在的的路径或者根本就不是路径，请重新输入");

} else if (dir.isFile()) {

System.out.println("您输入的是文件");

} else {

return dir; // 4，返回文件夹路径对象

}

}

}

/\*

方法定义：1，返回值是目的就是文件的大小，定义成long类型

2，参数应该是文件夹类型

\*/

public static long getFilesNum(File dir){

long length = 0; //1，定义一个变量为零

File[] subFiles = dir.listFiles(); //2将路径转换成一个文件夹集合,有文件和文件夹

for (File subFile : subFiles) { //3，对集合进行遍历判断，若文件夹是文件直接进行求其大小

if (subFile.isFile()) {

length = length + subFile.length();

}else{

length = length + getFilesNum(subFile); //4，递归调用

}

}

return length; //5，返回值不可缺少的

}

}

---------------------------------------------------------------------------------------------------------------------------