【济南中心】JAVA 编程阶梯：基础篇之第二十章

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

* 使用后，释放资源

IO 流之 FileInputStream：字节流

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
                }

                fis.close();
                                        //关闭流释放资源
```

问题：既然 read（ ）读取的是一个字节，为什么返回的是 int，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,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

IO 流之 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
                //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();
```

例子：FileOutputStream 的构造方法写出数据如何实现数据的追加写入

```
FileOutputStream fos = new FileOutputStream("bbb.txt",true);        //如果没有 bbb.txt,会创建出一个
                //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();
```

例子：拷贝图片

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();
```

字节数组拷贝之 available()方法

A:案例演示

* int read(byte[] b):一次读取一个字节数组

* 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

```
   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();
```

定义小数组的标准格式：

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) {
                        //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();
```

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

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

* 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

字节流读写中文：

* 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

* 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());

流的标准处理异常代码 1.6 版本及其以前：

try finally 嵌套

```
FileInputStream fis = null;
                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)
                                        fis.close();
                        }finally {
                                if(fos != null)
                                        fos.close();
                        }
                }
```

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);
        }
}
```

* 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

图片加密：

给图片加密

```
BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("a.jpg"));[/align]                BufferedOutputStream bos = new
BufferedOutputStream(new FileOutputStream("b.jpg"));

        int b;
        while((b = bis.read()) != -1) {
                bos.write(b ^ 123);
        }

        bis.close();
[align=left]bos.close();
```

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

```
Scanner sc = new Scanner(System.in);
        System.out.println("请输入一个文件路径");
        String line = sc.nextLine();                    //将键盘录入的文件
路径存储在 line 中
        File file = new File(line);                     //封装成File
对象
        FileInputStream fis = new FileInputStream(file);
        FileOutputStream fos = new FileOutputStream(file.getName());
```

```
int len;
byte[] arr = new byte[8192];                                    //定义缓冲区
while((len = fis.read(arr)) != -1) {
        fos.write(arr,0,len);
}

fis.close();
fos.close();
```

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

```
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();
```