day05

块读写的文件复制操作

int read(byte[] data)
一次性从文件中读取给定的字节数组总长度的字节量，并存入到该数组中。
返回值为实际读取到的字节量。若返回值为-1则表示读取到了文件末尾。

块写操作
void write(byte[] data)
一次性将给定的字节数组所有字节写入到文件中

void write(byte[] data,int offset,int len)
一次性将给定的字节数组从下标offset处开始的连续len个字节写入文件

```
package io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
/**
 * 通过提高每次读写的数据量，减少实际读写的次数，可以提高读写效率。
 * 单字节读写是一种随机读写形式。而一组一组字节的读写是块读写形式。
 */
public class CopyDemo2 {
    public static void main(String[] args) throws IOException {
        //使用块读写形式完成文件复制
        //创建文件输入流读取原文件
        FileInputStream fis = new FileInputStream("wnwb.exe");
        //创建文件输出流写复制文件
        FileOutputStream fos = new FileOutputStream("wnwb_cp.exe");

        /*
            流提供了块读写的方法
            int read(byte[] data)
            一次性从文件中读取给定的字节数组总长度的字节量，并存入到该数组中。
            返回值为实际读取到的字节量。若返回值为-1则表示读取到了文件末尾。

            文件数据
            11001100 11110000 10101010 00001111 00110011
            ^^^^^^^^ ^^^^^^^^ ^^^^^^^^
```

```
            int d;
            byte[] data = new byte[3];
            [00000000 00000000 00000000]
            第一次调用
            d = fis.read(data);
            [11001100 11110000 10101010]
            d = 3 本次读取到了3个字节

            文件数据
            11001100 11110000 10101010
00001111 00110011

^^^^^^^^ ^^^^^^^^
            第二次调用
            d = fis.read(data);//仅读取了最后
两个字节
            [00001111 00110011 10101010]//前
两个字节为本次读取的内容
             ^^^^^^^^ ^^^^^^^^
            d = 2 本次读取到了2个字节

            文件数据
            11001100 11110000 10101010
00001111 00110011 文件末尾!

              ^^^^^^^^
            第三次调用
            d = fis.read(data);//一个字节都没
有读取到
```

```
            [00001111 00110011 10101010]数组
没变化
            d = -1 文件末尾



            块写操作
            void write(byte[] data)
            一次性将给定的字节数组所有字节写入到文
件中

            void write(byte[] data,int
offset,int len)
            一次性将给定的字节数组从下标offset处
开始的连续len个字节写入文件

         */

        int len;//记录每次实际读取的字节量
        /*
            00000000  1byte   8位2进制称为1字
节
            1024byte  1kb
            1024kb    1mb
            1024mb    1gb
         */
        byte[] data = new
byte[1024*10];//10kb
        long start =
System.currentTimeMillis();
```

```
        while((len = fis.read(data))!=-1){
            fos.write(data,0,len);//读取多少
就写多少
        }
        long end =
System.currentTimeMillis();
        System.out.println("复制完毕!耗时:"+
(end-start)+"ms");
        fis.close();
        fos.close();

    }
}
```

写文本数据

String提供方法:
byte[] getBytes(String charsetName)
将当前字符串转换为一组字节

参数为字符集的名字，常用的是UTF-8。 其中中文字3字节表示1个，英文1字节表示1个。

```
package io;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
import
java.io.UnsupportedEncodingException;

/**
 * 向文件中写入文本数据
 */
public class WriteStringDemo {
    public static void main(String[] args)
throws IOException {
        FileOutputStream fos = new
FileOutputStream("demo.txt");
        String str = "super idol的笑容都没你的
甜,";
        /*
            支持中文的常见字符集有:
            GBK:国标编码。英文每个字符占1个字节,
中文每个字符占2个字节
            UTF-8:内部是unicode编码，在这个基础
上不同了少部分2进制信息作为长度描述
                英文每个字符占1字节
                中文每个字符占3字节

            String提供了将字符串转换为一组字节的
方法
            byte[] getBytes(String
charsetName)
            参数为字符集的名字，名字不缺分大小写,
但是拼写错误会引发异常:
            UnsupportedEncodingException
            不支持      字符集    异常
```

```
             */
            byte[] data = str.getBytes("UTF-
8");
            fos.write(data);

            fos.write("八月正午的阳光，都没你耀
眼。".getBytes("UTF-8"));

            System.out.println("写出完毕!");
            fos.close();
        }
}
```

文件输出流-追加模式

重载的构造方法可以将文件输出流创建为追加模式

- FileOutputStream(String path,boolean append)
- FileOutputStream(File file,boolean append)

当第二个参数传入true时，文件流为追加模式，即:指定的文件若存在，则原有数据保留，新写入的数据会被顺序的追加到文件中

```
package io;

import java.io.FileOutputStream;
import java.io.IOException;
```

```java
/**
 * 文件流的追加写模式
 */
public class FileAppendDemo {
    public static void main(String[] args)
throws IOException {
        /*
            FileOutputStream默认创建方式为覆盖
模式，即:如果连接的文件存在，
            则会将该文件原有数据全部删除。然后将通
过当前流写出的内容保存到文件中。

            重载的构造方法允许我们再传入一个
boolean型参数，如果这个值为true，则
            文件流为追加模式，即:若连接文件时该文
件存在，原有数据全部保留，通过当前
            流写出的数据会顺序的追加到文件中。
         */
        FileOutputStream fos = new
FileOutputStream(
                "demo.txt",true
        );
        fos.write("热爱105°的
你，".getBytes("UTF-8"));
        fos.write("滴滴清纯的蒸馏
水。".getBytes("UTF-8"));
        System.out.println("写出完毕!");
        fos.close();
    }
}
```

读取文本数据

```
package io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 * 从文件中读取文本数据
 */
public class ReadStringDemo {
    public static void main(String[] args)
throws IOException {
        FileInputStream fis = new
FileInputStream("fos.txt");

        byte[] data = new byte[1024];
        int len = fis.read(data);//块读操作
        System.out.println("实际读取到
了"+len+"个字节");
        /*
            String提供了将字节数组转换为字符串的
构造方法:
            String(byte[]data,String
charsetName)
```

```
                将给定的字节数组中所有字节按照指定的字符集转换为字符串

                String(byte[]data,int offset,int len,String charsetName)
                将给定的字节数组从下标offset处开始的连续len个字节按照指定的字符集转换为字符串
                 */
                String line = new String(data,0,len,"UTF-8");
                System.out.println(line);
                System.out.println(line.length());

                fis.close();

            }
}
```

高级流

流连接示意图

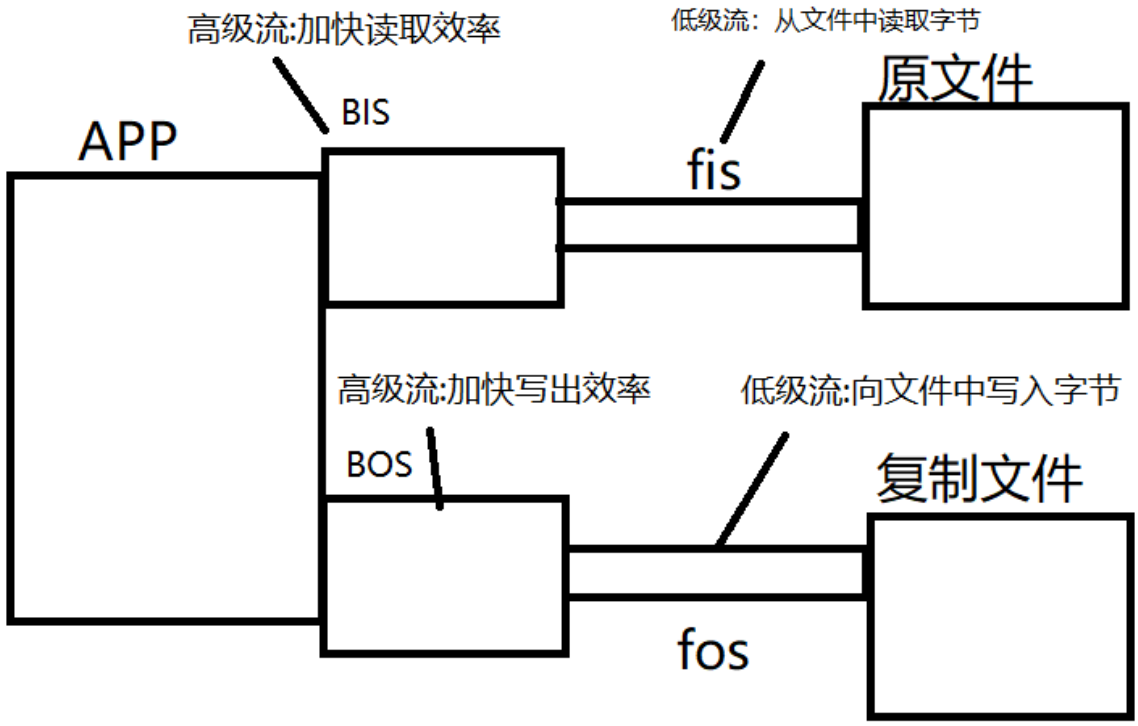
缓冲流

java.io.BufferedOutputStream和BufferedInputStream.

缓冲流是一对高级流,作用是提高读写数据的效率.

缓冲流内部有一个字节数组,默认长度是8K.缓冲流读写数据时一定是将数据的读写方式转换为块读写来保证读写效率.

使用缓冲流完成文件复制操作




```
package io;

import java.io.*;

/**
 * java将流分为节点流与处理流两类
```

```
 * 节点流:也称为低级流，是真实连接程序与另一端的"管道",负责实际读写数据的流。
 *         读写一定是建立在节点流的基础上进行的。
 *         节点流好比家里的"自来水管"。连接我们的家庭与自来水厂，负责搬运水。
 * 处理流:也称为高级流，不能独立存在，必须连接在其他流上，目的是当数据经过当前流时
 *         对其进行某种加工处理，简化我们对数据的同等操作。
 *         高级流好比家里常见的对水做加工的设备，比如"净水器"，"热水器"。
 *         有了它们我们就不必再自己对水进行加工了。
 * 实际开发中我们经常会串联一组高级流最终连接到低级流上，在读写操作时以流水线式的加工
 * 完成复杂IO操作。这个过程也称为"流的连接"。
 *
 * 缓冲流，是一对高级流，作用是加快读写效率。
 * java.io.BufferedInputStream和java.io.BufferedOutputStream
 *
 */
public class CopyDemo3 {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("ppt.pptx");
        BufferedInputStream bis = new BufferedInputStream(fis);
        FileOutputStream fos = new FileOutputStream("ppt_cp.pptx");
```

```
        BufferedOutputStream bos = new
BufferedOutputStream(fos);
        int d;
        long start =
System.currentTimeMillis();
        while((d = bis.read())!=-1){//使用缓
冲流读取字节
            bos.write(d);//使用缓冲流写出字节
        }
        long end =
System.currentTimeMillis();
        System.out.println("耗时:"+(end-
start)+"ms");
        bis.close();//关闭流时只需要关闭高级流即
可，它会自动关闭它连接的流
        bos.close();
    }
}
```

缓冲输出流写出数据时的缓冲区问题

通过缓冲流写出的数据会被临时存入缓冲流内部的字节数组,直到数组存满数据才会真实写出一次

```
package io;

import java.io.BufferedOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
/**
 * 缓冲输出流写出数据的时效性问题(缓冲区问题)
 */
public class BOSDemo {
    public static void main(String[] args)
throws IOException {
        FileOutputStream fos = new
FileOutputStream("bos.txt");
        BufferedOutputStream bos = new
BufferedOutputStream(fos);

        String line = "画画的baby,画画的baby,
奔驰的小野马和带刺的玫瑰~";
        byte[] data = line.getBytes("UTF-
8");
        bos.write(data);
        /*
            void flush()
            将缓冲流的缓冲区中已缓存的数据一次性写
出。
            注:频繁调用该方法会提高写出的次数降低
写出效率。但是可以换来写出
                数据的即时性。
         */
        bos.flush();
        System.out.println("写出完毕!");
        /*
            缓冲流的close操纵中会自动调用一次
flush,确保所有缓存的数据会被写出
```

```
            */
            bos.close();

        }
}
```