day05

File类(续)

访问一个目录中的所有子项

listFiles方法可以访问一个目录中的所有子项

```
package file;

import java.io.File;

/**
 * 访问一个目录中的所有子项
 */
public class ListFilesDemo1 {
    public static void main(String[] args) {
        //获取当前目录中的所有子项
        File dir = new File(".");
        /*
            boolean isFile()
            判断当前File表示的是否为一个文件
            boolean isDirectory()
            判断当前File表示的是否为一个目录
         */
        if(dir.isDirectory()){
            /*
                File[] listFiles()
                将当前目录中的所有子项返回。返回的数组中每个File实例表示其中的一个子项
             */
            File[] subs = dir.listFiles();
            System.out.println("当前目录包含"+subs.length+"个子项");
            for(int i=0;i<subs.length;i++){
                File sub = subs[i];
                System.out.println(sub.getName());
            }
        }
    }
}
```

获取目录中符合特定条件的子项

重载的listFiles方法:File[] listFiles(FileFilter)

该方法要求传入一个文件过滤器，并仅将满足该过滤器要求的子项返回。

```
package file;

import java.io.File;
import java.io.FileFilter;
```

```java
/**
 * 有条件的获取一个目录中的子项
 */
public class ListFilesDemo2 {
    public static void main(String[] args) {
        //获取当前目录下的所有文本文件(文件名是以".txt"结尾的)
        File dir = new File(".");
        if(dir.isDirectory()){
//            FileFilter fileFilter = new FileFilter() {
//                public boolean accept(File f) {
//                    return f.getName().endsWith(".txt");
//                }
//            };
//            /*
//                重载的listFiles方法要求传入一个文件过滤器
//                该方法会将File对象表示的目录中所有满足过滤器条件的子项返回
//             */
//            File[] subs = dir.listFiles(fileFilter);


            File[] subs = dir.listFiles(f->f.getName().endsWith(".txt"));

            for(File sub : subs){
                System.out.println(sub.getName());
            }
        }
    }
}
```

JAVA IO

基本概念

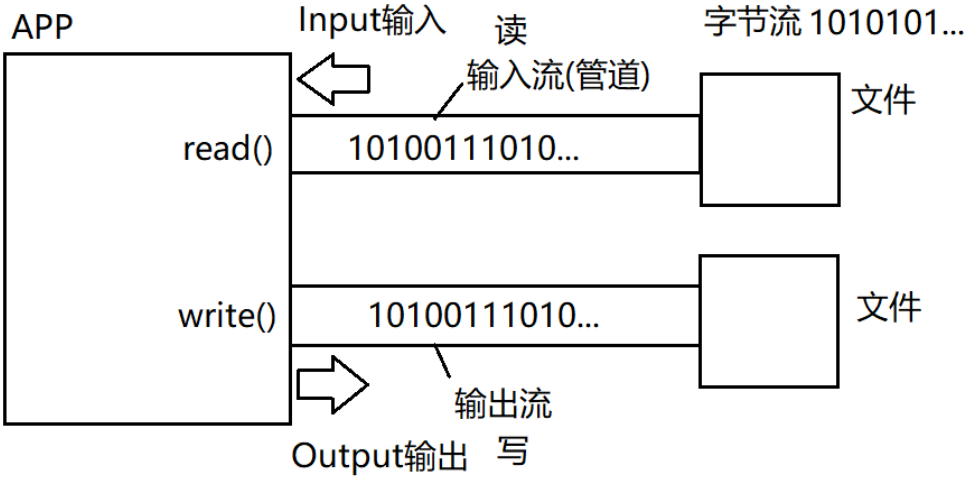
I/O 这里的I和O指的是输入与输出

- 输入:Input　用来读取数据的
- 输出:Output　用来写出数据的

流的概念

java将输入与输出比喻为"流"，英文:**Stream**.

就像生活中的"电流","水流"一样,它是以同一个方向顺序移动的过程.只不过这里流动的是字节(2进制数据).所以在IO中有输入流和输出流之分,我们理解他们是连接程序与另一端的"管道",用于获取或发送数据到另一端.



超类

- **java.io.InputStream**是所有字节输入流的超类，里面定义了所有字节输入流都必须具备的读取字节的方法
  - int read()

    读取一个字节，以int形式返回，该int值的”低八位”有效，若返回值为-1则表示EOF

  - int read(byte[] data)

    尝试最多读取给定数组的length个字节并存入该数组，返回值为实际读取到的字节量。
- **java.io.OutputStream**是所有字节输出流的超类，里面定义了所有字节输出流都必须具备的写出字节的方法
  - void write(int d)

    写出一个字节,写的是给定的int的”低八位”

  - void write(byte[] data)

    将给定的字节数组中的所有字节全部写出

  - void write(byte[] data,int offset,int len)

    将给定的字节数组中从offset处开始的连续len个字节写出

文件流

概念

文件流是用来链接我们的程序与文件之间的"管道",用来读写文件数据的流。

文件流分为

- 文件输入流java.io.FileInputStream:读取文件数据的流
- 文件输出流java.io.FileOutputStream:写入文件数据的流
- 文件流是继承自InputStream和OutputStream

文件输出流

java.io.FileOutputStream使用文件输出流向文件中写入数据

构造器

```
FileOutputStream(String path)
创建文件输出流对指定的path路径表示的文件进行写操作，如果该文件不存在则将其创建出来

FileOutputStream(File file)
创建文件输出流对指定的file对象表示的文件进行写操作，如果该文件不存在则将其创建出来
```

例

```
package io;

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

/**
 * JAVA IO
 *
 */
public class FOSDemo {
    public static void main(String[] args) throws IOException {
        /*
            FileOutputStream(String path)
            FileOutputStream(File file)
         */
        //向当前项目目录下的文件fos.dat中写入数据
        FileOutputStream fos = new FileOutputStream("./fos.dat");
//        File file = new File("./fos.dat");
//        FileOutputStream fos = new FileOutputStream(file);
        /*
            void write(int d)
            用来向文件中写入1个字节

            计算机底层只有2进制。1和0

                          8421
         0000 0          0001    1
         0001 1          0010    2
         0010 2          0100    4
         0011 3          1000    8
         0100 4
         0101 5          1110    14
         0110 6
         0111 7        01111111  +127       1字节  1byte
         1000 8        10000000  -
         1001 9
         1010 10
         1011 11
         1100 12
```

```
            1101 13
            1110 14
            1111 15


            write方法会将给定的int值对应的2进制的"低八位"写出
            fos.write(1)

            int型1的2进制:
            00000000 00000000 00000000 00000001
                                       ^^^^^^^^
                                       写出的数据
            fos.dat文件中内容:
            00000001

            write(2)
            2的2进制
            00000000 00000000 00000000 00000010
                                       ^^^^^^^^
                                       写出的数据

            fos.dat文件中内容:
            00000001 00000010
            */
            fos.write(1);
            fos.write(2);

            //当IO操作完毕后要关闭
            fos.close();


        }
}
```

文件输入流

java.io.FileInputStream使用文件输入流向从文件中读取数据

构造器

```
FileInputStream(String path)
基于给定的路径对应的文件创建文件输入流

FileInputStream(File file)
基于给定的File对象所表示的文件创建文件输入流
```

例

```
package io;

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

```java
/**
 * 文件输入流
 * 用于从文件中读取字节的流
 */
public class FISDemo {
    public static void main(String[] args) throws IOException {
        /*
            FileInputStream(String path)
            FileInputStream(File file)
         */
        FileInputStream fis = new FileInputStream("fos.dat");
        /*
            int read()
            读取1个字节，以int形式返回该字节内容。int值只有"低八位"有数据，高24位
            全部补0.
            有一个特殊情况:如果返回的int值为整数-1，则表示EOF。
            EOF:end of file 文件末尾

            fos.dat文件数据
            00000001 00000010

            第一次调用:
            int d = fis.read();

            00000001 00000010
            ^^^^^^^^
            读取的字节

            返回值d的二进制样子:
            00000000 00000000 00000000 00000001
            |-----自动补充的24个0------| 读取的字节


            第二次调用:
            d = fis.read();

            00000001 00000010
                     ^^^^^^^^
                     读取的字节

            返回值d的二进制样子:
            00000000 00000000 00000000 00000010
            |-----自动补充的24个0------| 读取的字节


            第三次调用:
            d = fis.read();

            00000001 00000010
                             ^^^^^^^^
                             文件末尾了

            返回值d的二进制样子:
            11111111 11111111 11111111 11111111
```

```
            |----------32位2进制都是1-----------|
         */
        int d = fis.read();
        System.out.println(d);//1
        d = fis.read();
        System.out.println(d);//2
        d = fis.read();//文件只有2个字节，因此第三次读取已经是文件末尾EOF
        System.out.println(d);//-1

        fis.close();
    }
}
```

文件复制

复制文件的原理就是使用文件输入流从原文件中陆续读取出每一个字节，然后再使用文件输出流将字节陆续写入到另一个文件中完成的。

示例

第一次读取

第二次读取

第三次读取

循环进行上述操作，直到某次fis.read()方法返回值为-1，表示读取到了文件末尾，那么就不再写出即可。

例

```
package io;

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

/**
 * 文件的复制
 */
public class CopyDemo {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("image.png");
        FileOutputStream fos = new FileOutputStream("image_cp.png");
```

```
        int d;
//        while(true){
//            d = fis.read();
//            if(d==-1){
//                break;
//            }
//            fos.write(d);
//        }

        while((d = fis.read())!= -1){
            fos.write(d);
        }

        System.out.println("复制完毕!");
        fis.close();
        fos.close();
    }
}
```

效率问题

上述案例在复制文件时的读写效率是很低的。因为硬盘的特性，决定着硬盘的读写效率差，而单字节读写就是在频繁调用硬盘的读写，从而产生了"木桶效应"。

为了解决这个问题，我们可以采取使用块读写的方式来复制文件，减少硬盘的实际读写的次数，从而提高效率。

块读写

- 块读:一次性读取一组字节的方式

  InputStream中定义了块读的方法

  ```
  int read(byte[] data)
  一次性读取给定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数。如果返回值为-1表示本次没有读取到任何字节已经是流的末尾了
  ```

- 块写:一次性写出一组字节的方式

  OutputStream中定义了块写的方法

  ```
  void write(byte[] data)
  一次性将给定数组中所有字节写出
  void write(byte[] data,int offset,int len)
  一次性将data数组中从下标offset处开始的连续len个字节一次性写出
  ```

例

改为块读写形式后，复制效率得到了明显的提升

```
package io;

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

/**
 * 提高每次读写的数据量减少读写次数，可以提高读写效率
 *
 * 块读取:一次性读取一组字节的方式
 * 块写:一次性写出一组字节
 */
public class CopyDemo2 {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("image.png");
        FileOutputStream fos = new FileOutputStream("image_cp2.png");
        /*
            在InputStream中定义了块读取的方法
            int read(byte[] data)
            一次性读取给定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数。如果返回值为-1表示本次没有读取到任何字节已经是流的末尾
了


            文件内容(6字节):
            00110011 11001100 10101010 01010101 11110000 00001111


            byte[] data = new byte[4];//4个字节的数组
            data:{00000000,00000000,00000000,00000000} 2进制表示

            第一次调用:int d = fis.read(data);
            一次性从文件中读取4(data数组的长度为4)个字节
            00110011 11001100 10101010 01010101 11110000 00001111
            ^^^^^^^^ ^^^^^^^^ ^^^^^^^^ ^^^^^^^^
            |----------读取的数据---------------|

            data:{00110011,11001100,10101010,01010101}
                  数组里的4个字节为本次读取到的全部数据
            d:4  返回值d为整数4，表示本次实际读取到了4个字节


            第二次调用:d = fis.read(data);
            一次性从文件中读取4(data数组的长度为4)个字节
            00110011 11001100 10101010 01010101 11110000 00001111 文件末尾了
                                               ^^^^^^^^ ^^^^^^^^ ^^^^^^^^
^^^^^^^^
                                              |---读取的数据-----|

            data:{11110000,00001111,10101010,01010101}
```

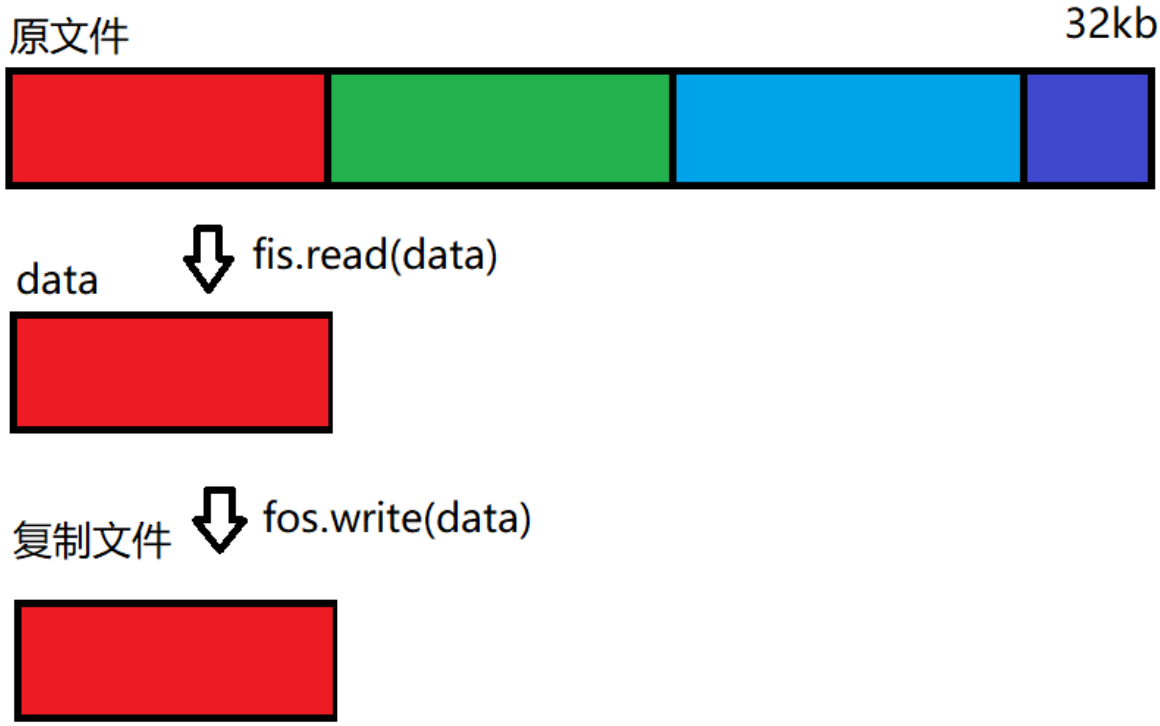
```
                |-本次实际读取字节-| |---旧数据-------|
            d:2  返回值d为整数2，表示本次实际读取到了2个字节



             第二次调用:d = fis.read(data);
            一次性从文件中读取4(data数组的长度为4)个字节
            00110011 11001100 10101010 01010101 11110000 00001111 文件末尾
                                                                ^^^^^^^^
^^^^^^^^ ^^^^^^^^ ^^^^^^^^
                                                                已经是文件末尾

            data:{11110000,00001111,10101010,01010101}
                 |-------------旧数据-----------------|
            d:-1  返回值d为整数-1，表示已经是末尾了，本次没有读取任何数据



            OutputStream中定义了块写方法
            void write(byte[] data)
            一次性将给定数组中所有字节写出
         */
        /*
            00000000  1byte  1字节
            1024byte  1kb
            1024kb    1mb
            1024mb    1gb
            1024gb    1tb
            1024tb    1pb
         */
        byte[] data = new byte[1024*10];//10kb
        int d;//记录每次实际读取的数据量
        long start = System.currentTimeMillis();
        while( (d = fis.read(data)) !=-1){
            fos.write(data);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制完毕!耗时:"+(end-start)+"ms");
        fis.close();
        fos.close();
    }
}
```

问题

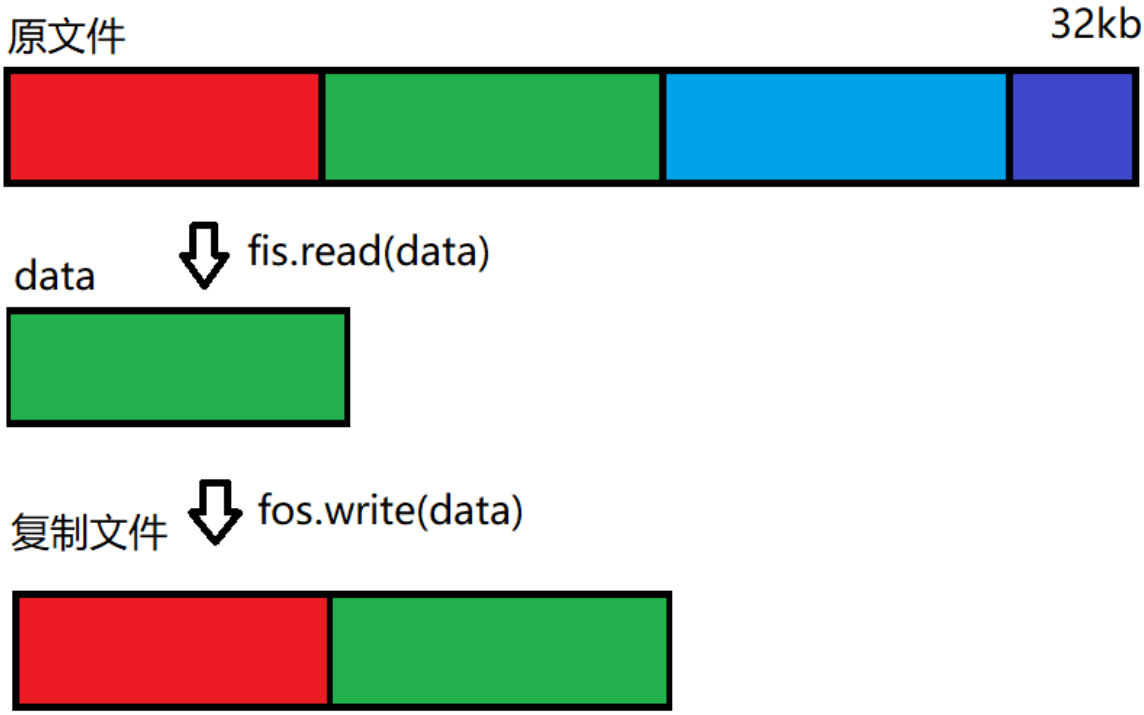
速度问题解决了，但是复制后的文件比原文件大一些。这是文件不一定是10240的倍数，这会导致最后一次读取时是读不够10240的字节数的，那么data数组中就不是所有数据都是新数据了。此时如果在写出时将data数组所有内容写出就会出现文件最后多出很多旧的数据。

示例

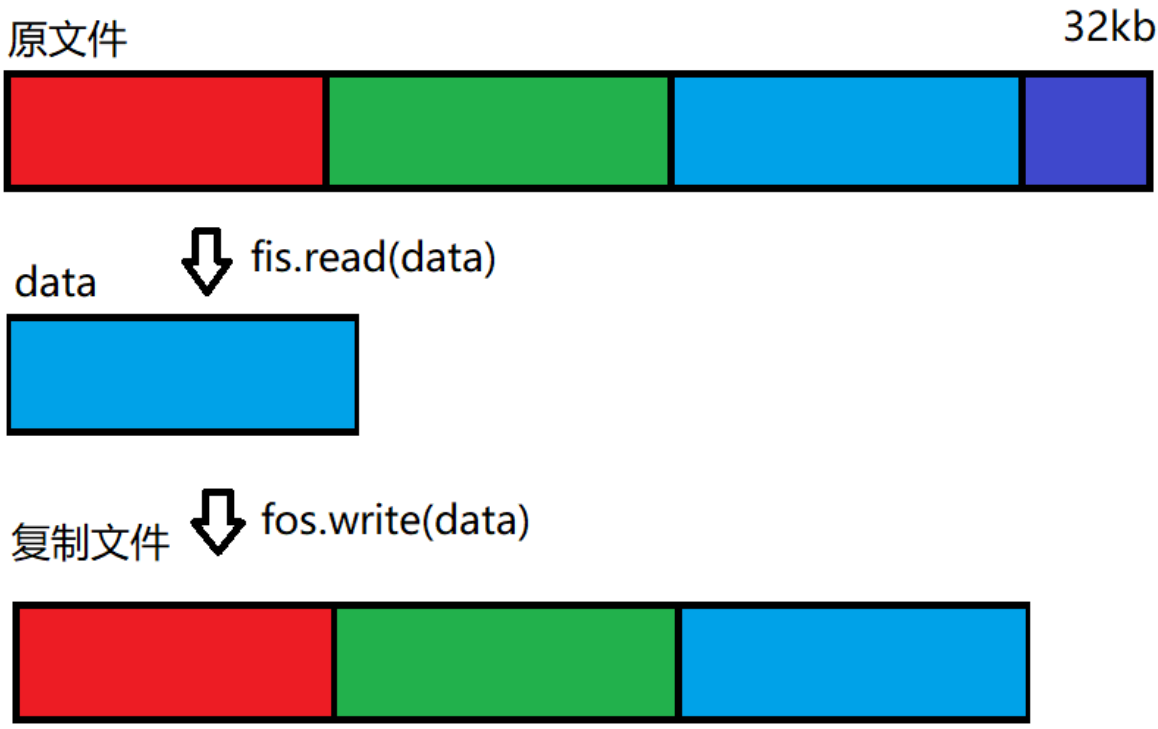
第一次操作



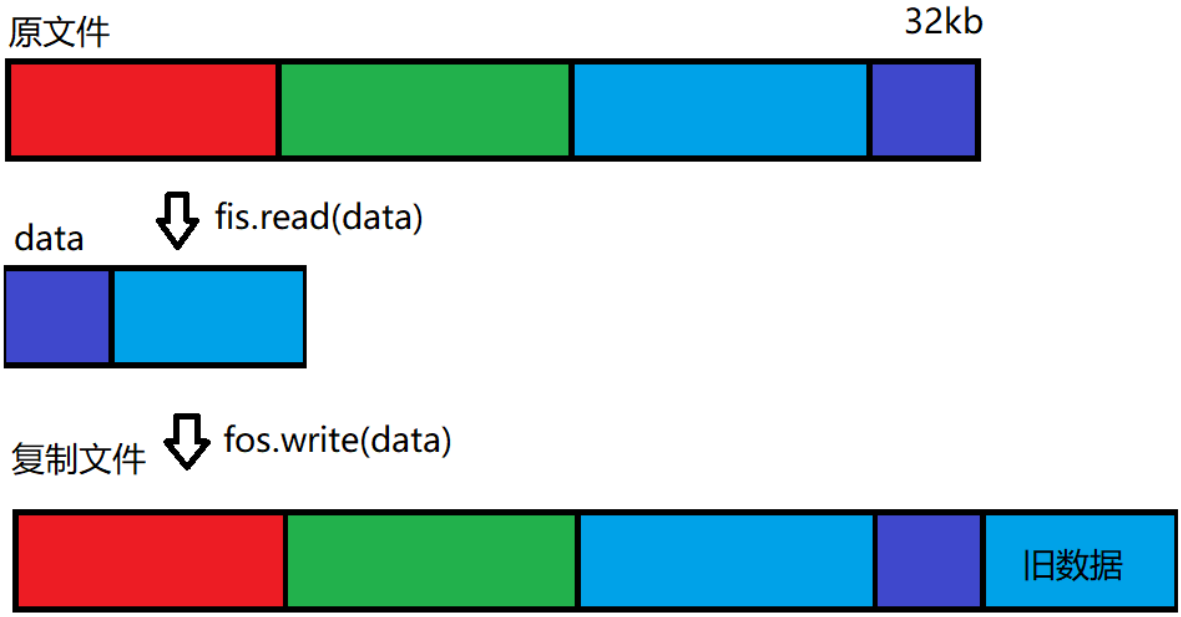
第二次操作



第三次读取操作



第四次操作



解决办法

使用OutputStream的另一个块写操作

```
void write(byte[] data,int offset,int len)
将给定数组data从offset处开始的连续len个字节一次性写出
```

例

```
package io;

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

```java
/**
 * 提高每次读写的数据量减少读写次数，可以提高读写效率
 *
 * 块读取:一次性读取一组字节的方式
 * 块写:一次性写出一组字节
 */
public class CopyDemo2 {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("image.png");
        FileOutputStream fos = new FileOutputStream("image_cp2.png");
        byte[] data = new byte[1024*10];//10kb
        int d;//记录每次实际读取的数据量
        long start = System.currentTimeMillis();
        while( (d = fis.read(data)) !=-1){
            fos.write(data,0,d);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制完毕!耗时:"+(end-start)+"ms");
        fis.close();
        fos.close();
    }
}
```

写出文本数据

文件中只能保存2进制信息，因此我们要想写出文本数据，需要先将字符串转换为2进制。

文字编码

String提供了将字符串转换为一组字节的方法

```
byte[] getBytes(Charset cs)
将当前字符串按照指定的字符集转换为一组字节

例如:
String line = "在小小的花园里面";
byte[] data = line.getBytes(StandardCharsets.UTF_8);//将字符串按照UTF-8编码转换为一组字节
```

写入文本文件

```java
package io;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 写出文本数据
 */
```

```
public class WriteStringDemo {
    public static void main(String[] args) throws IOException {
         //向文件fos1.txt中写入一行中文
        FileOutputStream fos = new FileOutputStream("fos1.txt");
        String line = "在小小的花园里面";
        byte[] data = line.getBytes(StandardCharsets.UTF_8);
        fos.write(data);

        fos.write("，挖呀挖呀挖~".getBytes(StandardCharsets.UTF_8));

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

    }
}
```

读取文本数据

先将文件中的字节读取出来，然后再将这些字节按照对应的字符集转换为字符串即可

文本解码

String的构造器提供了对字节解码为字符串的操作

```
String(byte[] data,Charset cn)
将data数组中的所有字节按照指定的字符集转换为字符串
```

例

```
package io;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 读取文本数据
 */
public class ReadStringDemo {
    public static void main(String[] args) throws IOException {
        /*
            1:先从fos.txt中读取所有的字节
            2:再将这些字节转换为字符串
         */
        //1
        File file = new File("fos.txt");
        long len = file.length();//文件名长度

        FileInputStream fis = new FileInputStream(file);
        byte[] data = new byte[(int)len];//创建一个与文件长度等长的字节数组
        fis.read(data);//一口气将文件所有字节读入到data数组中(块读)
```

```
        //2将data数组中所有字节按照UTF-8编码还原为字符串
        String line = new String(data, StandardCharsets.UTF_8);
        System.out.println(line);

        fis.close();
    }
}
```

总结:

File类

File类的每一个实例可以表示硬盘(文件系统)中的一个文件或目录(实际上表示的是一个抽象路径)

使用File可以做到:

- 1:访问其表示的文件或目录的属性信息,例如:名字,大小,修改时间等等
- 2:创建和删除文件或目录
- 3:访问一个目录中的子项

常用构造器:

- File(String pathname)
- File(File parent,String name)可参考文档了解

常用方法:

- length()：返回一个long值，表示占用的磁盘空间，单位为字节。
- canRead()：File表示的文件或目录是否可读
- canWrite()：File表示的文件或目录是否可写
- isHidden()：File表示的文件或目录是否为隐藏的
- createNewFile()：创建一个新文件，如果指定的文件所在的目录不存在会抛出异常 java.io.FileNotFoundException
- mkdir：创建一个目录
- mkdirs：创建一个目录，并且会将所有不存在的父目录一同创建出来，推荐使用。
- delete()：删除当前文件或目录，如果目录不是空的则删除失败。
- exists()：判断File表示的文件或目录是否真实存在。true:存在 false:不存在
- isFile()：判断当前File表示的是否为一个文件。
- isDirectory()：判断当前File表示的是否为一个目录
- listFiles()：获取File表示的目录中的所有子项
- listFiles(FileFilter filter)：获取File表示的目录中满足filter过滤器要求的所有子项

JAVA IO必会概念:

- java io可以让我们用标准的读写操作来完成对不同设备的读写数据工作.
- java将IO按照方向划分为输入与输出,参照点是我们写的程序.
- **输入**:用来读取数据的,是从外界到程序的方向,用于获取数据.
- **输出**:用来写出数据的,是从程序到外界的方向,用于发送数据.

java将IO比喻为"流",即:stream. 就像生活中的"电流","水流"一样,它是以同一个方向顺序移动的过程.只不过这里流动的是字节(2进制数据).所以在IO中有输入流和输出流之分,我们理解他们是连接程序与另一端的"管道",用于获取或发送数据到另一端.

因此流的读写是顺序读写的，只能顺序向后写或向后读，不能回退。

Java定义了两个超类(抽象类):

- **java.io.InputStream**:所有字节输入流的超类,其中定义了读取数据的方法.因此将来不管读取的是什么设备(连接该设备的流)都有这些读取的方法,因此我们可以用相同的方法读取不同设备中的数据

  常用方法:

  `int read()`：读取一个字节，返回的`int`值低8位为读取的数据。如果返回值为整数`-1`则表示读取到了流的末尾

  `int read(byte[] data)`：块读取，最多读取`data`数组总长度的数据并从数组第一个位置开始存入到数组中，返回值表示实际读取到的字节量，如果返回值为`-1`表示本次没有读取到任何数据，是流的末尾。

- **java.io.OutputStream**:所有字节输出流的超类,其中定义了写出数据的方法.

  常用方法:

  void write(int d)：写出一个字节，写出的是给定的int值对应2进制的低八位。

  void write(byte[] data)：块写，将给定字节数组中所有字节一次性写出。

  void write(byte[]data,int off,int len)：块写，将给定字节数组从下标off处开始的连续len个字节一次性写出。

java将流分为两类:节点流与处理流:

- **节点流**:也称为**低级流**.

  节点流的另一端是明确的,是实际读写数据的流,读写一定是建立在节点流基础上进行的.

- **处理流**:也称为**高级流**.

  处理流不能独立存在,必须连接在其他流上,目的是当数据流经当前流时对数据进行加工处理来简化我们对数据的该操作.

实际应用中,我们可以通过串联一组高级流到某个低级流上以流水线式的加工处理对某设备的数据进行读写,这个过程也成为流的连接,这也是IO的精髓所在.

文件流

文件流是一对低级流，**用于读写文件的流。**

java.io.FileOutputStream文件输出流，继承自java.io.OutputStream

常用构造器

覆盖模式对应的构造器

覆盖模式是指若指定的文件存在，文件流在创建时会先将该文件原内容清除。

- FileOutputStream(String pathname)：创建文件输出流用于向指定路径表示的文件做写操作
- FileOutputStream(File file)：创建文件输出流用于向File表示的文件做写操作。

注:如果写出的文件不存在文件流自动创建这个文件，但是如果该文件所在的目录不存在会抛出异常:java.io.FileNotFoundException

追加写模式对应的构造器

> 追加模式是指若指定的文件存在，文件流会将写出的数据陆续追加到文件中。

- FileOutputStream(String pathname,boolean append)：如果第二个参数为true则为追加模式，false则为覆盖模式
- FileOutputStream(File file,boolean append)：同上

常用方法:

> `void write(int d)`：向文件中写入一个字节，写入的是`int`值2进制的低八位。
>
> `void write(byte[] data)`：向文件中块写数据。将数组`data`中所有字节一次性写入文件。
>
> `void write(byte[] data,int off,int len)`：向文件中快写数据。将数组`data`中从下标`off`开始的连续`len`个字节一次性写入文件。

java.io.FileInputStream文件输入流，继承自java.io.InputStream

常用构造器

FileInputStream(String pathname) 创建读取指定路径下对应的文件的文件输入流，如果指定的文件不存在则会抛出异常java.io.FileNotFoundException

FileInputStream(File file) 创建读取File表示的文件的文件输入流，如果File表示的文件不存在则会抛出异常java.io.IOException。

常用方法

> `int read()`：从文件中读取一个字节，返回的`int`值低八位有效，如果返回的`int`值为整数`-1`则表示读取到了文件末尾。
>
> `int read(byte[] data)`：块读数据，从文件中一次性读取给定的`data`数组总长度的字节量并从数组第一个元素位置开始存入数组中。返回值为实际读取到的字节数。如果返回值为整数`-1`则表示读取到了文件末尾。