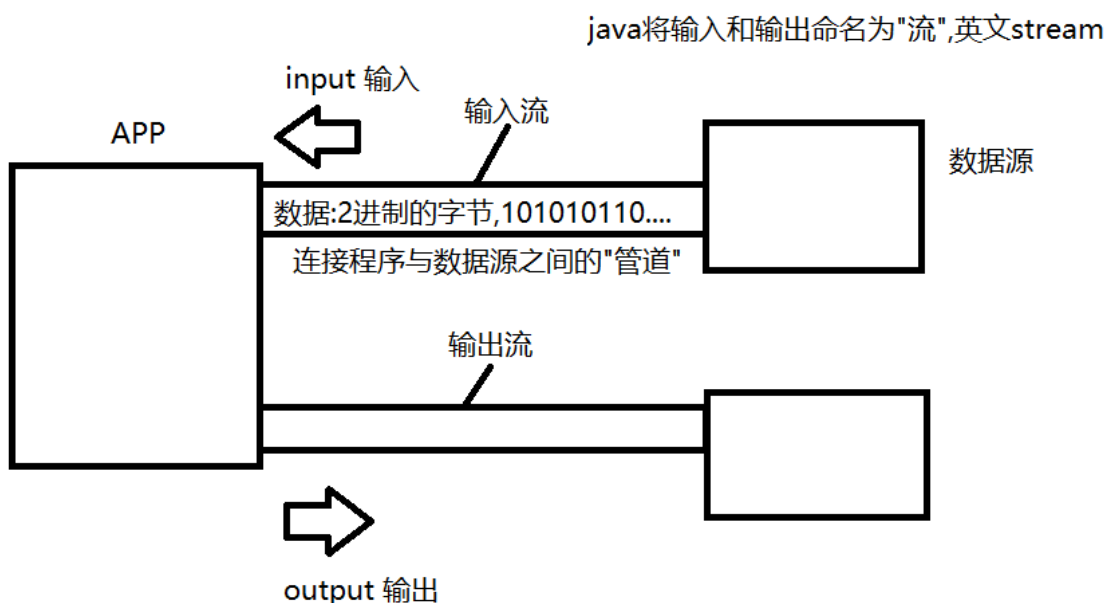


day02

JAVA IO

- 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.



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
- 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.

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,用于读写文件数据的流.用于连接程序与文件(硬盘)的"管道".负责读写文件数据.

文件输出流:java.io.FileOutputStream

```
package io;

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

/**
 * JAVA IO
 * IO:Input,Output 即: 输入与输出
 *
 * JAVA IO用于我们程序可以和外界交换数据。用于与外界的数据进行读写操作的。
 *
 * java中将输入与输出比喻为"流":stream
 * 如何理解流:讲流想象为一个连接我们程序和另一端的"管道",在其中按照同一方向顺序移动的数据。
 * 有点像"水管"中向着统一方向流动的水。
 * 输入流:从外界向我们的程序中移动的方向, 因此是用来获取数据的流, 作用就是:读取操作
 * 输出流:写出操作
 * 注意:流是单向的, 输入永远用来读, 输出永远用来写。将来我们在实际开发中希望与程序交互的另一端
 * 互相发送数据时, 我们只需要创建一个可以连接另一端的"流", 进行读写操作完成。
 *
 * java定义了两个超类, 来规范所有的字节流
 * java.io.InputStream:所有字节输入流的超类(抽象类), 里面定义了读取字节的相关方法。
 * 所有字节输入流都继承自它
 * java.io.OutputStream:所有字节输出流的超类(抽象类), 里面定义了写出字节的相关方法。
 * 所有的字节输出流都继承自它
 *
 *
 * 文件流
 * java.io.FileInputStream和java.io.FileOutputStream
 * 作用是真实连接我们程序和文件之间的"管道"。其中文件输入流用于从文件中读取字节。而文件输出流则
 * 用于向文件中写入字节。
 *
 * 文件流是节点流
 * JAVA IO将流划分为两类:节点流和处理流
 * 节点流:俗称"低级流", 特点:真实连接我们程序和另一端的"管道", 负责实际读写数据的流
 * 文件流就是典型的节点流, 真实连接我们程序与文件的"管道", 可以读写文件数据了。
 * 处理流:俗称"高级流"
 * 特点:
 * 1:不能独立存在(单独实例化进行读写操作不可以)
 * 2:必须连接在其他流上, 目的是当数据"流经"当前流时, 可以对其做某种加工操作, 简化我们的工作、
 * 流的连接:实际开发中经常会串联一组高级流最终到某个低级流上, 对数据进行流水线式的加工读写。
 *
 */
public class FOSDemo {
    public static void main(String[] args) throws IOException {
        //需求:向当前目录的文件fos.dat中写入数据
        /**
         * 在创建文件输出流时, 文件输出流常见的构造器:
         * FileOutputStream(String filename)
         * FileOutputStream(File file)
         * 上述两种构造器都会在创建时将该文件创建出来(如果该文件不存在才会这样做), 自动创建
         * 该文件的前提是该文件所在的目录必须存在, 否则会抛出异常。
         */
    }
}
```

```

//      File file = new File("./fos.dat");
//      FileOutputStream fos = new FileOutputStream(file);
/*
    一个小技巧:在指定相对路径时,如果是从"当前目录"./开始的,那么"./"是可以忽略不写
的
    因为在相对路径中,默认就是从"./"开始
*/
//      FileOutputStream fos = new FileOutputStream("./fos.dat");
FileOutputStream fos = new FileOutputStream("fos.dat");//与上面一句位置相同

/*
    OutputStream(所有字节输出流的超类)中定义了写出字节的方法:
    write(int d)
    write(byte[] data)
    write(byte[] data,int offset,int len)

    其中:
    void write(int d)
    写出一个字节,将给定的参数int值对应的2进制的"低八位"写出。

    文件输出流继承OutputStream后就重写了该方法,作用是将该字节写入到文件中。
*/

/*
    向文件中写入1个字节
    fow.write(1)
    将int值的1对应的2进制的"低八位"写如到文件第一个字节位置上

    1个int值占4个字节,每个字节是一个8为2进制
    int 1的2进制样子:
    00000000 00000000 00000000 00000001
                                ^^^^^^^^^^
                                写出的字节

    write方法调用后,fos.dat文件中就有了1个字节,内容为:
    00000001

    再次调用:
    fos.write(2)
    int 2的2进制样子:
    00000000 00000000 00000000 00000010
                                ^^^^^^^^^^
                                写出的字节

    write方法调用后,fos.dat文件中就有了2个字节,内容为:
    00000001 00000010
    上次写的    本次写的
*/

fos.write(1);
fos.write(2);

System.out.println("写出完毕!");
//注意! 流使用完后要关闭,来释放底层资源
fos.close();

}
}

```

文件输入流

```
package io;

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

/**
 * 使用文件输入流读取文件中的数据
 */
public class FISDemo {
    public static void main(String[] args) throws IOException {
        //将fos.dat文件中的字节读取回来
        /**
         fos.dat文件中的数据:
         00000001 00000010
         */
        FileInputStream fis = new FileInputStream("fos.dat");
        /**
         java.io.InputStream(所有字节输入流的超类)定义着读取字节的相关方法
         int read()
         读取1个字节并以int型整数返回读取到的字节内容，返回的int值中对应的2进制的"低八位"
         就是读取到的数据。如果返回的int值为整数-1(这是一个特殊值，32位2进制全都是1)表达
         的
         是流读取到了末尾了。

         int read(byte[] data)

         文件输入流重写了上述两个方法用来从文件中读取对应的字节。
         */

        /**
         fos.dat文件中的数据:
         00000001 00000010
         ^^^^^^^^^^
         第一次读取的字节

         当我们第一次调用:
         int d = fis.read();//读取的是文件中第一个字节

         该int值d对应的2进制:
         00000000 00000000 00000000 00000001
         |-----自动补充24个0-----| ^^^^^^^^^^
                                         读取到的数据

         而该2进制对应的整数就是1.
         */
        int d = fis.read();//读取到的就是整数1
        System.out.println(d);
        /**
         fos.dat文件中的数据:
         00000001 00000010
         ^^^^^^^^^^
         第二次读取的字节
        */
    }
}
```

```

        当我们第二次调用：
        d = fis.read();//读取的是文件中第二个字节

        该int值d对应的2进制：
        00000000 00000000 00000000 00000010
        |-----自动补充24个0-----|  ^^^^^^^^^^
                                           读取到的数据

        而该2进制对应的整数就是2。

    */
    d = fis.read();//2
    System.out.println(d);

    /*
    fos.dat文件中的数据：
    00000001 00000010 文件末尾
    ^^^^^^^^^^
    没有第三个字节

    当我们第三次调用：
    d = fis.read();//读取到文件末尾了！

    该int值d对应的2进制：
    11111111 11111111 11111111 11111111
    该数字是正常读取1个字节永远表达不了的值。并且-1的2进制格式好记。因此用它表达读取
    到了末尾。

    */
    d = fis.read();//-1
    System.out.println(d);

    fis.close();
}
}

```

文件复制

```

package io;

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

/**
 * 利用文件输入流与输出流实现文件的复制操作
 */
public class CopyDemo {
    public static void main(String[] args) throws IOException {
        //用文件输入流读取待复制的文件
        //    FileInputStream fis = new FileInputStream("image.jpg");
        FileInputStream fis = new FileInputStream("01.rmvb");
        //用文件输出流向复制文件中写入复制的数据
        //    FileOutputStream fos = new FileOutputStream("image_cp.jpg");
        FileOutputStream fos = new FileOutputStream("01_cp.rmvb");
        /*
        原文件image.jpg中的数据
        10100011 00111100 00001111 11110000....
        */
    }
}

```

AAAAAAAA

读取该字节

第一次调用：

```
int d = fis.read();
```

d的2进制:00000000 00000000 00000000 10100011

读到的字节

fos向复制的文件image_cp.jpg中写入字节

第一次调用：

```
fos.write(d);
```

作用:将给定的int值d的2进制的"低八位"写入到文件中

d的2进制:00000000 00000000 00000000 10100011

写出字节

调用后image_cp.jpg文件数据：

10100011

```
*/
```

```
/*
```

循环条件是只要文件没有读到末尾就应该复制

如何直到读取到末尾了呢？

前提是:要先尝试读取一个字节，如果返回值是-1就说明读到末尾了

如果返回值不是-1，则说明读取到的是一个字节的内容，就要将他写入到复制文件中

```
*/
```

```
int d;//先定义一个变量，用于记录每次读取到的数据
```

```
long start = System.currentTimeMillis();//获取当前系统时间
```

```
while ((d = fis.read()) != -1) {
```

```
    fos.write(d);
```

```
}
```

```
long end = System.currentTimeMillis();
```

```
System.out.println("复制完毕!耗时:" + (end - start) + "ms");
```

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

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

```
}
```

```
}
```

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.FileOutputStream;
```

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

```
/**
```

```
 * 通过提高每次读写的数据，减少读写次数可以提高读写效率。
```

```
*/
```

```

public class CopyDemo2 {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("01.rmvb");
        FileOutputStream fos = new FileOutputStream("01_cp.rmvb");
        /*
            块读:一次性读取一组字节
            块写:一次性将写出一组字节

            java.io.InputStream上定义了块读字节的方法:
            int read(byte[] data)
            一次性读取给定字节数组length个字节并从头开始装入到数组中。返回值为实际读取到的字
            节量

            如果返回值为-1则表示流读取到了末尾。
            文件流重写了该方法,作用是块读文件里的数据。

            java.io.OutputStream上定义了块写字节的方法:
            void write(byte[] data)
            一次性将给定的字节数组中所有的字节写出。

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

            原文件数据(假设文件共6个字节):
            11110000 00001111 01010101 11111111 00000000 10101010

            byte[] buf = new byte[4]; //创建一个长度为4的字节数组
            buf默认的样子(每个元素若以2进制表现):{00000000,00000000,00000000,00000000}
            int len; //记录每次实际读取的字节数

            当第一次调用:
            len = fis.read(buf);
            由于字节数组buf的长度为4.因此可以一次性最多从文件中读取4个字节并装入到buf数组中
            返回值len表示的整数是这次实际读取到了几个字节。

            原文件数据(假设文件共6个字节):
            11110000 00001111 01010101 11111111 00000000 10101010
            ^^^^^^^^^ ^^^^^^^^^ ^^^^^^^^^ ^^^^^^^^^
            第一次读取的4个字节

            buf:{11110000, 00001111, 01010101, 11111111}
            len:4 表示本次读取到了4个字节

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

            原文件数据(假设文件共6个字节):
            11110000 00001111 01010101 11111111 00000000 10101010 文件末尾了
            ^^^^^^^^^ ^^^^^^^^^ ^^^^^^^^^
            ^^^^^^^^^

            本次实际只能读取到2个字节

            buf:{00000000, 10101010, 01010101, 11111111}
            |本次新读的2字节数据| |---上次的旧数据---|
            len:2表示本次实际只读取到了2个字节。它的意义就是告诉你buf数组中前几个字节是本次真
            实
            读取到的数据
        */
    }
}

```

```

        第三次调用:
        len = fis.read(buf);
        原文件数据(假设文件共6个字节):
        11110000 00001111 01010101 11111111 00000000 10101010 文件末尾了
                                                ^^^^^^^^^^
    ^^^^^^^^^^ ^^^^^^^^^^ ^^^^^^^^^^
        buf:{00000000, 10101010, 01010101, 11111111} 没有任何变化!
        len:-1 表示本次读取时已经是文件末尾了!!

        */
        /*
        00000000 8位2进制 1byte 1字节
        1024byte = 1kb
        1024kb = 1mb
        1024mb = 1gb
        1024gb = 1tb
        */
        /*
        编译完该句代码:byte[] buf = new byte[10240];

        在实际开发中,有时候用一个计算表达式更能表现这个值的含义时,我们不妨使用计算表达式
        long t = 864000000;
        long t = 60 * 60 * 24 * 1000;

        */
        byte[] buf = new byte[1024 * 10]; //10kb
        int len; //记录每次实际读取到的字节数
        long start = System.currentTimeMillis();
        while ((len = fis.read(buf)) != -1) {
            fos.write(buf, 0, len);
        }
        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.nio.charset.StandardCharsets;

/**
 * 使用文件输出流向文件中写入文本数据
 */
public class WriteStringDemo {
    public static void main(String[] args) throws IOException {

```



```

    /*
        1: 创建一个文件输出流
        2: 将写出的文字先转换为2进制(一组字节)
        3: 关闭流

        文件流有两种创建方式:
        1: 覆盖模式, 对应的构造器:
            FileOutputStream(String filename)
            FileOutputStream(File file)
        所谓覆盖模式: 文件流在创建是若发现该文件已存在, 则会将该文件原内容全部删除。然后在陆续将通过该流写出的内容保存到文件中。
    */
    FileOutputStream fos = new FileOutputStream("fos.txt", true);
    String line = "让我再看你一遍, 从南到北。";
    /*
        String提供了将内容转换为一组字节的方法:getBytes()
        java.nio.charset.StandardCharsets
    */
    byte[] data = line.getBytes(StandardCharsets.UTF_8);
    fos.write(data);

    line = "像是北五环路蒙住的双眼。";
    data = line.getBytes(StandardCharsets.UTF_8);
    fos.write(data);

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

/**
 * 使用文件输出流向文件中写入文本数据
 */
public class WriteStringDemo {
    public static void main(String[] args) throws IOException {
        /*
            1: 创建一个文件输出流
            2: 将写出的文字先转换为2进制(一组字节)
            3: 关闭流
        */
    }
}

```

文件流有两种创建方式：

1:覆盖模式，对应的构造器：

```
FileOutputStream(String filename)
```

```
FileOutputStream(File file)
```

所谓覆盖模式:文件流在创建是若发现该文件已存在，则会将该文件原内容全部删除。然后在陆续将通过该流写出的内容保存到文件中。

2:追加模式，对应的构造器

```
FileOutputStream(String filename,boolean append)
```

```
FileOutputStream(File file,boolean append)
```

当第二个参数为true时，那么就是追加模式。

所谓追加模式:文件流在创建时若发现该文件已存在，则原内容都保留。通过当前流陆续写出的内容都会被陆续追加到文件末尾。

```
*/
FileOutputStream fos = new FileOutputStream("fos.txt",true);

String line = "斯国一!";
byte[] data = line.getBytes(StandardCharsets.UTF_8);
fos.write(data);

line = "奥里给!";
data = line.getBytes(StandardCharsets.UTF_8);
fos.write(data);

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;
import java.nio.charset.StandardCharsets;

/**
 * 从文件中读取文本数据
 */
public class ReadStringDemo {
    public static void main(String[] args) throws IOException {
        /*
        1:创建一个文件输入流
        2:从文件中将字节都读取回来
        3:将读取到的字节转换回字符串
        */
        FileInputStream fis = new FileInputStream("fos.txt");

        byte[] data = new byte[1024]; //1kb
        int len = fis.read(data); //块读操作，返回值表达实际读取到了多少字节
        System.out.println("实际读取了:"+len+"个字节");
    }
}
```

```

    /*
    String提供了构造方法可以将一个字节数组还原为字符串
    String(byte[] data,Charset charset)
    将给定的字节数组data中所有字节按照给定的字符集转换为字符串。

    String(byte[] data,int offset,int len,Charset charset)
    将给定的字节数组data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,StandardCharsets.UTF_8);
    System.out.println(line.length()); //输出字符串长度
    System.out.println(line);

    fis.close();
}
}

```

####

总结

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)`: 向文件中块写数据。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。