JAVASE-DAY05/DAY06 晚课

1. File 类

File 类的每一个实例都可以表示硬盘(文件系统)中的一个文件或者目录(实际上表示的是一个抽象的路径)

File 类只能用于表示文件(目录)的信息（名称，大小等），换句话说，File 只能访问文件的相关属性，不能对文章的内容进行操作。

使用 File 可以做到

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

获取 File 文件的路径

```
Java
package cn.tedu.demo;

import java.io.File;
import java.io.IOException;
import java.sql.SQLOutput;

public class FileDemo {
    public static void main(String[] args) throws IOException {
        //使用 File 访问当前项目目录下的 demo。txt 文件
        /**
         *创建 File 时要指定路径，而路径通常使用相对路径
         * 相对路径的好处在于有良好的跨平台性
         * 相对路径： ./demo.txt
         * 可以使用./开头表示当前目录
         * 这个"当前目录"是哪里取决于程序运行环境而定，在 Idea 中，这里
就是指
         * 当前程序所在的项目目录。
         *
```

```
         * 绝对路径：
/Users/xiujia.zhao/IdeaProjects/demo01/./demo/demo.txt
         * 绝对路径移植性差
         * 相同操作系统需要路径存在，不同的操作系统就无法使用
         */
        File  file = new File("./demo.txt");
        System.out.println(file.getAbsolutePath()); //查看绝对路径
        System.out.println(file.getPath()); //查看相对路径
        System.out.println(file.getParentFile()); //查看上一级路径
    }
}
```

对文件操作常用方法

- 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。demo/a/b/c
- 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

1. 创建多级文件夹

```
Java
package cn.tedu.demo;

import java.io.File;
```

```
public class MkdirsDemo {
    public static void main(String[] args) {
        File dir = new File("./demo/a/b/c/d/e");
        if(dir.exists()){
            System.out.println("已存在");
        }else{
            System.out.println("创建文件夹");
            /*
             * 如果使用 mkdir 创建多级目录,会无效
             * 只要是多级目录必须使用 mkdirs 方法
             * mkdirs 会将所有不存在的父目录一同创建出来
             */
            dir.mkdirs();
        }

    }
}
```

2. 删除多级文件夹
   a. 例如： 删除 demo/a/b/c/d
   b. 所用方法：
      i. delete() 删除文件夹，必须是空文件夹
      ii. File[] listFiles（） 获取当前目录下的所有子项
   c. 思路：
      i. demo--a
      ii. 先取出 demo 下的所有子项，然后遍历删除 a
         1. a--b
      iii. 再取出 a 下的所有子项，删除 b
         1. b--c
      iv. 再取出 b 下的所有子项，删除 c
         1. C--d
      v. 再取出 c 下的所有子项， 删除 d

```
Java
package cn.tedu.demo;
```

```
import java.io.File;

public class DeleteFiles {
    public static void main(String[] args) {
        File dir = new File("./demo");
        deteleDirs(dir);
    }
    public static void  deteleDirs(File dir){
        //先获取当前文件夹下的所有子项
        File [] dirs=dir.listFiles();
        System.out.println("dirs 的长度："+dirs.length);
        //遍历 dirs 数组，获取子项
        for (int i = 0; i <dirs.length ; i++) {
            File d=dirs[i];
            System.out.println("路径:"+d.getPath());
            //再获取当前文件夹下的所有子项---直接调用 deteleDirs 方法
            deteleDirs(d);
        }
        dir.delete();
    }
}
```

3. 文件过滤器

- File[] listFiles(FileFilter filter) 该方法在获取该目录中子项的过程中利用参数给定的过滤器将满足其要求的子项返回，其余的忽略。

```
Java
package cn.tedu.demo;

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

public class FileterDemo {
    public static void main(String[] args) {
        //获取当前项目下的所有文件
        File dir = new File("./src/cn/tedu");

        //判断是否是一个文件夹
        if(dir.isDirectory()){
            /*
```

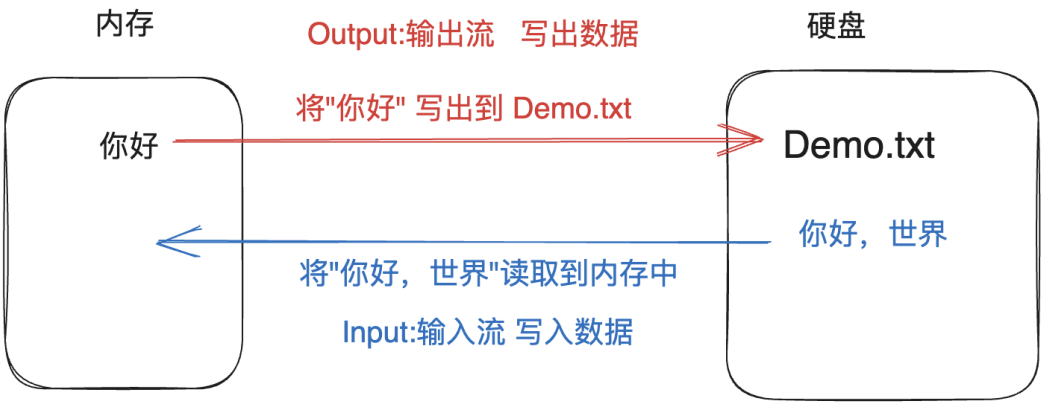
```
            FileFilter filter=new FileFilter(){
                //只获取以 .java 为后缀的文件
                @Override
                public boolean accept(File pathname) {
                    return pathname.getName().endsWith(".java");
                }
            };
            //获取当前项目下的所有子项
            //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[] dirs = dir.listFiles(filter);
            */
             //使用 Lambda 表达式
            File[] dirs = dir.listFiles(f->f.getName().endsWith(".java"));
            System.out.println(dirs.length);
            for (File sub:dirs) {
                System.out.println(sub.getName());
            }
        }
    }
}
```

2. I O 流

I/O 这里的 I 和 O 是指输入和输出

- 输入：Input 用来读取数据
- 输出：Output 用来写出数据

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



IO 流的分类

按方向分为：输入流和输出流

- 输入流：从本地文件读取数据到内存
- InputStream 是所有字节输入流的父类，规定了读取字节的相关方法
- 输出流：从内存写出数据到本地文件
- OutputStream 是所有字节输出流的父类，规定了写出字节的相关方法

按照使用方式分为：低级流和高级流

- 节点流（低级流）：低级流是直接与底层 I/O 设备（如文件、网络套接字）进行交互的流。它们是基于字节（byte）的，用于读取和写入字节数据。低级字节流的常见例子是
- 处理流（高级流）：不能独立存在,必须连接在其它流上,是对一个已存在的流的连接和封装，通过所封装的流的功能调用实现数据读写。

按照读写单位分为：字节流和字符流

- 字节流：读写单位为字节，顶级父类为 InputStream，OutputStream
- 字符流：读写单位为字符（所有的字符流都是高级流），顶级父类为 Reader，Writer

Java 定义了两个父类

- OutputStream:所有字节输出流的超类,其中定义了写出数据的方法.
  - void write(int d) 写出一个字节，写的是给定的 int 的"低八位"
  - void 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

- 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
  - int read（） 一次读取一个字节，以 int 形式返回，该 int 值的"低八位"有效，如果返回-1.则表示读到文件的末尾 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。

文件流

文件流是一对低级流，用于读写文件的流。

- **文件输出流**：FileOutputStream 用于写出文件数据，以字节形式写出数据(一个字节 8 个位)
- 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) 写出一个字节，写的是给定的 int 的"低八位"
  - void 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

```
Java
```

```
package cn.tedu.io;

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

public class OutputStreamDemo {
    public static void main(String[] args) throws IOException {
        //FileOutputStream("./fos.dat");会判断文件是否为空，如果为空
则会创建这个文件
        FileOutputStream fos = new FileOutputStream("./fos.dat");
        /*
           void write(int d) 用来向文件中写入 1 字节
           写入的是 int 值对应的 2 进制的"低八位"
           int--4 字节    1 字节-8 位
            int 型 1 的 2 进制 二进制数的最右边（低位）表示最小的权值,
而最左边（高位）表示最大的权值。
            fos.write(1)
            00000000 00000000 00000000 00000001
                                       ^^^^^^^^
                                       低八位写出
            fos.dat 文件中的内容  00000001
            fos.write(2)
            2 的 2 进制
            00000000 00000000 00000000 00000010
                                       ^^^^^^^^
            fos.dat 文件中的内容 00000001 00000010
         */
        fos.write(1);
        fos.write(2);

    }
}
```

int 型 1 的二进制表示示例：00000000 00000000 00000000 00000001

在一个 32 位的 int 数据类型中，使用 32 个比特（bits）来表示整数值。二进制数的最右边（低位）表示最小的权值，而最左边（高位）表示最大的权值。

- **文件输入流**：FileInputStream 用于读取文件数据，以字节形式读取文件
- **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（） 一次读取一个字节，以 int 形式返回，该 int 值的"低八位"有效，如果返回 -1.则表示读到文件的末尾 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
package cn.tedu.io;

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

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

        FileInputStream fis = new FileInputStream("./fos.dat");
        /*
         * fis 对象是按照顺序读取文件内容
         * 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.
           fos.dat 中的数据 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

         */
```

```
        int d= fis.read();  //1
        System.out.println(d);
        /*
        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
         */
        d = fis.read(); //2
        System.out.println(d);
        /*
         d = fis.read();
         00000001 00000010
                           ^^^^^^^^
                           文件末尾了
         */
        d=fis.read();//-1
        System.out.println(d);

        fis.close();
    }
}
```

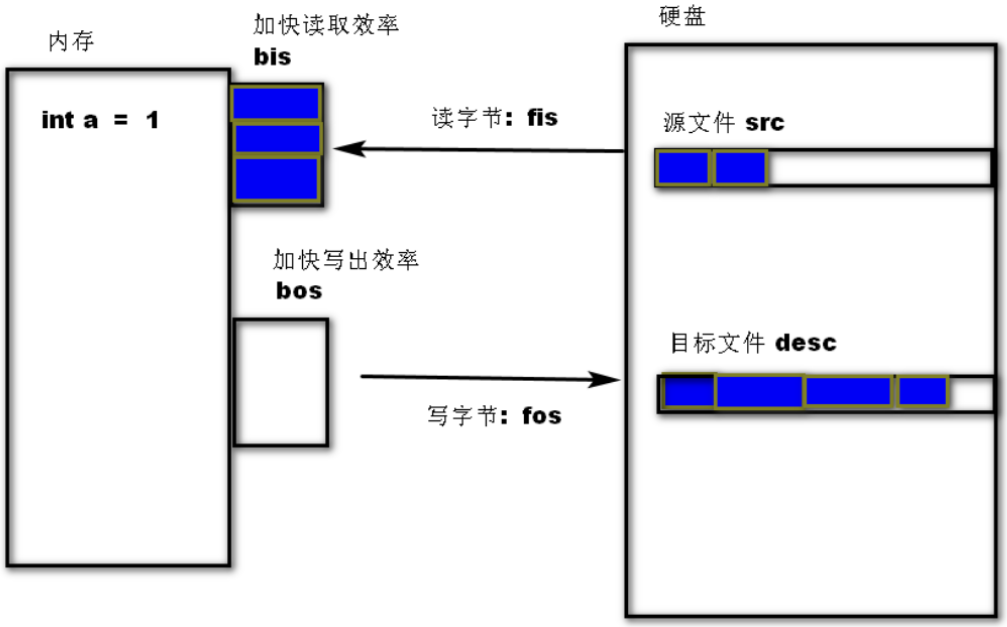
- int read()和 int read(byte[] data)的区别

	方法功能	方法返回值	方法参数
int read()	一次读取一个字节，以 int 形式返回	每次读取到的一个字节 （以 int 形式返回）	不带参，默认读取 1 字节
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	实际读取到的字节量 (以 int 形式返回)	自定义字节数组，每次读取字节数组长度的数据

	到的字节量。		

缓冲流

- BufferedInputStream 缓冲字节输入流，内部维护着一个缓冲区(字节数组)，使用该流在读取一个字节时，该流会尽可能多的一次性读取若干字节并存入缓冲区，然后逐一的将字节返回，直到缓冲区中的数据被全部读取完毕，会再次读取若干字节。
    - 好处：减少读取的次数，提高了读取效率
- BufferedOutputStream 缓冲字节输出流，缓冲输出流内部也维护着一个缓冲区，每当我们向该流写数据时，都会先将数据存入缓冲区，当缓冲区已满时，缓冲流会将数据一次性全部写出。
    - 好处：减少写出的次数，提高了写出效率



缓冲流和字节数组的区别是都是缓存区，但是一个是自己定义的，可以更改大小，另一个是缓冲流内部维护的。无法更改大小。

- **缓冲区 flush 方法的作用**

```
Java
void flush()
```

使用缓冲输出流可以提高写出效率，但是这也存在着一个问题，就是写出数据缺乏即时性。有时我们需要在执行完某些写出操作后，就希望将这些数据确实写出，而非在缓冲区中保存直到缓冲区满后才写出。

- 清空缓冲区，将缓冲区中的数据强制写出

```Java
package cn.tedu.io;

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

public class BufferedOutputStreamDemo {
    public static void main(String[] args) throws IOException {
        BufferedOutputStream bos =
                new BufferedOutputStream(new
FileOutputStream("./fos.dat"));
        String str = "今天是周五了";
        bos.write(str.getBytes());
        /**
         * void  flush();
         * 优点:强制将当前缓冲区的字节一次性写出
         * 同时清空缓冲区,即时的写出
         * 缺点:实际上也会增加写出的次数,降低写出效率
         */
//        bos.flush();
        System.out.println("写出完毕");
        bos.close(); //close 方法中自动调用了 flush 方法
    }
}
```