1.File类

1.1File类概述和构造方法【应用】

- File类介绍
  - 它是文件和目录路径名的抽象表示
  - 文件和目录是可以通过File封装成对象的
  - 对于File而言,其封装的并不是一个真正存在的文件,仅仅是一个路径名而已.它可以是存在的,也可以是不存在的.将来是要通过具体的操作把这个路径的内容转换为具体存在的
- File类的构造方法

方法名	说明
File(String pathname)	通过将给定的路径名字符串转换为抽象路径名来创建新的 File实例
File(String parent, String child)	从父路径名字符串和子路径名字符串创建新的 File实例
File(File parent, String child)	从父抽象路径名和子路径名字符串创建新的 File实例

- 示例代码

```
public class FileDemo01 {
    public static void main(String[] args) {
        //File(String pathname): 通过将给定的路径名字符串转换为抽象路径名来创建新的 File实例
        File f1 = new File("E:\\itcast\\java.txt");
        System.out.println(f1);

        //File(String parent, String child): 从父路径名字符串和子路径名字符串创建新的 File实例
        File f2 = new File("E:\\itcast","java.txt");
        System.out.println(f2);

        //File(File parent, String child): 从父抽象路径名和子路径名字符串创建新的 File实例
        File f3 = new File("E:\\itcast");
        File f4 = new File(f3,"java.txt");
        System.out.println(f4);
    }
}
```

1.2绝对路径和相对路径【理解】

- 绝对路径

  是一个完整的路径,从盘符开始
- 相对路径

  是一个简化的路径,相对当前项目下的路径
- 示例代码

```
public class FileDemo02 {
    public static void main(String[] args) {
        // 是一个完整的路径,从盘符开始
        File file1 = new File("D:\\itheima\\a.txt");

        // 是一个简化的路径,从当前项目根目录开始
        File file2 = new File("a.txt");
        File file3 = new File("模块名\\a.txt");
    }
}
```

1.3File类创建功能【应用】

- 方法分类

方法名	说明
public boolean createNewFile()	当具有该名称的文件不存在时，创建一个由该抽象路径名命名的新空文件
public boolean mkdir()	创建由此抽象路径名命名的目录
public boolean mkdirs()	创建由此抽象路径名命名的目录，包括任何必需但不存在的父目录

- 示例代码

```
public class FileDemo02 {
    public static void main(String[] args) throws IOException {
        //需求1：我要在E:\\itcast目录下创建一个文件java.txt
        File f1 = new File("E:\\itcast\\java.txt");
        System.out.println(f1.createNewFile());
        System.out.println("--------");

        //需求2：我要在E:\\itcast目录下创建一个目录JavaSE
        File f2 = new File("E:\\itcast\\JavaSE");
        System.out.println(f2.mkdir());
        System.out.println("--------");

        //需求3：我要在E:\\itcast目录下创建一个多级目录JavaWEB\\HTML
        File f3 = new File("E:\\itcast\\JavaWEB\\HTML");
//        System.out.println(f3.mkdir());
        System.out.println(f3.mkdirs());
        System.out.println("--------");

        //需求4：我要在E:\\itcast目录下创建一个文件javase.txt
        File f4 = new File("E:\\itcast\\javase.txt");
//        System.out.println(f4.mkdir());
        System.out.println(f4.createNewFile());
    }
}
```

1.4File类删除功能【应用】

- 方法分类

方法名	说明
public boolean delete()	删除由此抽象路径名表示的文件或目录

- 示例代码

```
public class FileDemo03 {
    public static void main(String[] args) throws IOException {
//        File f1 = new File("E:\\itcast\\java.txt");
        //需求1：在当前模块目录下创建java.txt文件
        File f1 = new File("myFile\\java.txt");
//        System.out.println(f1.createNewFile());

        //需求2：删除当前模块目录下的java.txt文件
        System.out.println(f1.delete());
        System.out.println("--------");

        //需求3：在当前模块目录下创建itcast目录
        File f2 = new File("myFile\\itcast");
//        System.out.println(f2.mkdir());

        //需求4：删除当前模块目录下的itcast目录
        System.out.println(f2.delete());
        System.out.println("--------");

        //需求5：在当前模块下创建一个目录itcast,然后在该目录下创建一个文件java.txt
        File f3 = new File("myFile\\itcast");
//        System.out.println(f3.mkdir());
        File f4 = new File("myFile\\itcast\\java.txt");
//        System.out.println(f4.createNewFile());

        //需求6：删除当前模块下的目录itcast
        System.out.println(f4.delete());
        System.out.println(f3.delete());
    }
}
```

1.5File类判断和获取功能【应用】

- 判断功能

方法名	说明
public boolean isDirectory()	测试此抽象路径名表示的File是否为目录
public boolean isFile()	测试此抽象路径名表示的File是否为文件
public boolean exists()	测试此抽象路径名表示的File是否存在

- 获取功能

方法名	说明
public String getAbsolutePath()	返回此抽象路径名的绝对路径名字符串
public String getPath()	将此抽象路径名转换为路径名字符串
public String getName()	返回由此抽象路径名表示的文件或目录的名称
public File[] listFiles()	返回此抽象路径名表示的目录中的文件和目录的File对象数组

- 示例代码

```
public class FileDemo04 {
    public static void main(String[] args) {
        //创建一个File对象
        File f = new File("myFile\\java.txt");

//        public boolean isDirectory(): 测试此抽象路径名表示的File是否为目录
//        public boolean isFile(): 测试此抽象路径名表示的File是否为文件
//        public boolean exists(): 测试此抽象路径名表示的File是否存在
        System.out.println(f.isDirectory());
        System.out.println(f.isFile());
        System.out.println(f.exists());

//        public String getAbsolutePath(): 返回此抽象路径名的绝对路径名字符串
//        public String getPath(): 将此抽象路径名转换为路径名字符串
//        public String getName(): 返回由此抽象路径名表示的文件或目录的名称
        System.out.println(f.getAbsolutePath());
        System.out.println(f.getPath());
        System.out.println(f.getName());
        System.out.println("--------");

//        public File[] listFiles(): 返回此抽象路径名表示的目录中的文件和目录的File对象数组
        File f2 = new File("E:\\itcast");
        File[] fileArray = f2.listFiles();
        for(File file : fileArray) {
//            System.out.println(file);
//            System.out.println(file.getName());
            if(file.isFile()) {
                System.out.println(file.getName());
            }
```

```
        }
    }
}
```

1.6File类练习一【应用】

- 案例需求

  在当前模块下的aaa文件夹中创建一个a.txt文件

- 实现步骤
  - 创建File对象,指向aaa文件夹
  - 判断aaa文件夹是否存在,如果不存在则创建
  - 创建File对象,指向aaa文件夹下的a.txt文件
  - 创建这个文件

- 代码实现

```
public class Test1 {
    public static void main(String[] args) throws IOException {
        //练习一：在当前模块下的aaa文件夹中创建一个a.txt文件
       /* File file = new File("filemodule\\aaa\\a.txt");
        file.createNewFile();*/
        //注意点:文件所在的文件夹必须要存在.

        //1.创建File对象,指向aaa文件夹
        File file = new File("filemodule\\aaa");
        //2.判断aaa文件夹是否存在,如果不存在则创建
        if(!file.exists()){
            //如果文件夹不存在,就创建出来
            file.mkdirs();
        }
        //3.创建File对象,指向aaa文件夹下的a.txt文件
        File newFile = new File(file,"a.txt");
        //4.创建这个文件
        newFile.createNewFile();
    }
}
```

1.7File类练习二【应用】

- 案例需求

  删除一个多级文件夹

- 实现步骤
  - 定义一个方法,接收一个File对象
  - 遍历这个File对象,获取它下边的每个文件和文件夹对象
  - 判断当前遍历到的File对象是文件还是文件夹
  - 如果是文件,直接删除
  - 如果是文件夹,递归调用自己,将当前遍历到的File对象当做参数传递
  - 参数传递过来的文件夹File对象已经处理完成,最后直接删除这个空文件夹

- 代码实现

```
public class Test2 {
    public static void main(String[] args) {
        //练习二：删除一个多级文件夹
        //delete方法
        //只能删除文件和空文件夹.
        //如果现在要删除一个有内容的文件夹?
        //先删掉这个文件夹里面所有的内容.
        //最后再删除这个文件夹

        File src = new File("C:\\Users\\apple\\Desktop\\src");
        deleteDir(src);
    }

    //1.定义一个方法,接收一个File对象
    private static void deleteDir(File src) {
        //先删掉这个文件夹里面所有的内容.
        //递归 方法在方法体中自己调用自己.
        //注意：可以解决所有文件夹和递归相结合的题目
        //2.遍历这个File对象,获取它下边的每个文件和文件夹对象
        File[] files = src.listFiles();
        //3.判断当前遍历到的File对象是文件还是文件夹
        for (File file : files) {
            //4.如果是文件,直接删除
            if(file.isFile()){
                file.delete();
            }else{
                //5.如果是文件夹,递归调用自己,将当前遍历到的File对象当做参数传递
                deleteDir(file);//参数一定要是src文件夹里面的文件夹File对象
            }
        }
        //6.参数传递过来的文件夹File对象已经处理完成,最后直接删除这个空文件夹
        src.delete();
    }

}
```

1.8File类练习三【应用】

- 案例需求

  统计一个文件夹中每种文件的个数并打印

  打印格式如下：

  txt:3个

  doc:4个

  jpg:6个

  ...

- 实现步骤
  - 定义一个方法,参数是HashMap集合用来统计次数和File对象要统计的文件夹

- 遍历File对象,获取它下边的每一个文件和文件夹对象
- 判断当前File对象是文件还是文件夹
- 如果是文件,判断这种类型文件后缀名在HashMap集合中是否出现过
  - 没出现过,将这种类型文件的后缀名存入集合中,次数存1
  - 出现过,获取这种类型文件的后缀名出现的次数,对其+1,在存回集合中
- 如果是文件夹,递归调用自己,HashMap集合就是参数集合,File对象是当前文件夹对象

- 代码实现

```
public class Test3 {
    public static void main(String[] args) {
        //统计一个文件夹中,每种文件出现的次数.
        //统计 --- 定义一个变量用来统计. ---- 弊端:同时只能统计一种文件
        //利用map集合进行数据统计,键 --- 文件后缀名  值 ----  次数

        File file = new File("filemodule");
        HashMap<String, Integer> hm = new HashMap<>();
        getCount(hm, file);
        System.out.println(hm);
    }

    //1.定义一个方法,参数是HashMap集合用来统计次数和File对象要统计的文件夹
    private static void getCount(HashMap<String, Integer> hm, File file) {
        //2.遍历File对象,获取它下边的每一个文件和文件夹对象
        File[] files = file.listFiles();
        for (File f : files) {
            //3.判断当前File对象是文件还是文件夹
            if(f.isFile()){
                //如果是文件,判断这种类型文件后缀名在HashMap集合中是否出现过
                String fileName = f.getName();
                String[] fileNameArr = fileName.split("\\.");
                if(fileNameArr.length == 2){
                    String fileEndName = fileNameArr[1];
                    if(hm.containsKey(fileEndName)){
                        //出现过,获取这种类型文件的后缀名出现的次数,对其+1,在存回集合中
                        Integer count = hm.get(fileEndName);
                        //这种文件又出现了一次.
                        count++;
                        //把已经出现的次数给覆盖掉.
                        hm.put(fileEndName,count);
                    }else{
                        // 没出现过,将这种类型文件的后缀名存入集合中,次数存1
                        hm.put(fileEndName,1);
                    }
                }
            }else{
              //如果是文件夹,递归调用自己,HashMap集合就是参数集合,File对象是当前文件夹对象代码实
现

                getCount(hm,f);
            }
        }
    }
```

```
}
```

2.字节流

2.1 IO流概述和分类【理解】

- IO流介绍
  - IO：输入/输出(Input/Output)
  - 流：是一种抽象概念,是对数据传输的总称.也就是说数据在设备间的传输称为流,流的本质是数据传输
  - IO流就是用来处理设备间数据传输问题的.常见的应用: 文件复制; 文件上传; 文件下载
- IO流的分类
  - 按照数据的流向
    - 输入流：读数据
    - 输出流：写数据
  - 按照数据类型来分
    - 字节流
      - 字节输入流
      - 字节输出流
    - 字符流
      - 字符输入流
      - 字符输出流
- IO流的使用场景
  - 如果操作的是纯文本文件,优先使用字符流
  - 如果操作的是图片、视频、音频等二进制文件,优先使用字节流
  - 如果不确定文件类型,优先使用字节流.字节流是万能的流

2.2字节流写数据【应用】

- 字节流抽象基类
  - InputStream：这个抽象类是表示字节输入流的所有类的超类
  - OutputStream：这个抽象类是表示字节输出流的所有类的超类
  - 子类名特点：子类名称都是以其父类名作为子类名的后缀
- 字节输出流
  - FileOutputStream(String name)：创建文件输出流以指定的名称写入文件
- 使用字节输出流写数据的步骤
  - 创建字节输出流对象(调用系统功能创建了文件,创建字节输出流对象,让字节输出流对象指向文件)
  - 调用字节输出流对象的写数据方法
  - 释放资源(关闭此文件输出流并释放与此流相关联的任何系统资源)
- 示例代码

```
public class FileOutputStreamDemo01 {
    public static void main(String[] args) throws IOException {
```

```
        //创建字节输出流对象
        /*
            注意点:
                    1.如果文件不存在,会帮我们创建
                    2.如果文件存在,会把文件清空
        */
        //FileOutputStream(String name): 创建文件输出流以指定的名称写入文件
        FileOutputStream fos = new FileOutputStream("myByteStream\\fos.txt");

        //void write(int b): 将指定的字节写入此文件输出流
        fos.write(97);
//        fos.write(57);
//        fos.write(55);

        //最后都要释放资源
        //void close(): 关闭此文件输出流并释放与此流相关联的任何系统资源。
        fos.close();
    }
}
```

2.3字节流写数据的三种方式【应用】

- 写数据的方法分类

方法名	说明
void write(int b)	将指定的字节写入此文件输出流 一次写一个字节数据
void write(byte[] b)	将 b.length字节从指定的字节数组写入此文件输出流 一次写一个字节数组数据
void write(byte[] b, int off, int len)	将 len字节从指定的字节数组开始，从偏移量off开始写入此文件输出流 一次写一个字节数组的部分数据

- 示例代码

```
public class FileOutputStreamDemo02 {
    public static void main(String[] args) throws IOException {
        //FileOutputStream(String name): 创建文件输出流以指定的名称写入文件
        FileOutputStream fos = new FileOutputStream("myByteStream\\fos.txt");
        //FileOutputStream(File file): 创建文件输出流以写入由指定的 File对象表示的文件
//        FileOutputStream fos = new FileOutputStream(new File("myByteStream\\fos.txt"));

        //void write(int b): 将指定的字节写入此文件输出流
//        fos.write(97);
//        fos.write(98);
//        fos.write(99);
//        fos.write(100);
//        fos.write(101);

//        void write(byte[] b): 将 b.length字节从指定的字节数组写入此文件输出流
//        byte[] bys = {97, 98, 99, 100, 101};
```

```
        //byte[] getBytes(): 返回字符串对应的字节数组
        byte[] bys = "abcde".getBytes();
//        fos.write(bys);

        //void write(byte[] b, int off, int len): 将 len字节从指定的字节数组开始，从偏移量off开始写入此文件输出流
//        fos.write(bys,0,bys.length);
        fos.write(bys,1,3);

        //释放资源
        fos.close();
    }
}
```

2.4字节流写数据的两个小问题【应用】

- 字节流写数据如何实现换行
  - windows:\r\n
  - linux:\n
  - mac:\r
- 字节流写数据如何实现追加写入
  - public FileOutputStream(String name,boolean append)
  - 创建文件输出流以指定的名称写入文件。如果第二个参数为true， 则字节将写入文件的末尾而不是开头
- 示例代码

```
public class FileOutputStreamDemo03 {
    public static void main(String[] args) throws IOException {
        //创建字节输出流对象
//        FileOutputStream fos = new FileOutputStream("myByteStream\\fos.txt");
        FileOutputStream fos = new FileOutputStream("myByteStream\\fos.txt",true);

        //写数据
        for (int i = 0; i < 10; i++) {
            fos.write("hello".getBytes());
            fos.write("\r\n".getBytes());
        }

        //释放资源
        fos.close();
    }
}
```

2.5字节流写数据加异常处理【应用】

- 异常处理格式
  - try-catch-finally

```
try{
    可能出现异常的代码;
}catch(异常类名 变量名){
    异常的处理代码;
}finally{
    执行所有清除操作;
}
```

  - finally特点
    - 被finally控制的语句一定会执行，除非JVM退出

- 示例代码

```
public class FileOutputStreamDemo04 {
    public static void main(String[] args) {
        //加入finally来实现释放资源
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream("myByteStream\\fos.txt");
            fos.write("hello".getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if(fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

2.6字节流读数据(一次读一个字节数据)【应用】

- 字节输入流
  - FileInputStream(String name)：通过打开与实际文件的连接来创建一个FileInputStream,该文件由文件系统中的路径名name命名
- 字节输入流读取数据的步骤
  - 创建字节输入流对象
  - 调用字节输入流对象的读数据方法
  - 释放资源
- 示例代码

```
public class FileInputStreamDemo01 {
    public static void main(String[] args) throws IOException {
        //创建字节输入流对象

        //FileInputStream(String name)
```

```
        FileInputStream fis = new FileInputStream("myByteStream\\fos.txt");

        int by;
        /*
            fis.read(): 读数据
            by=fis.read(): 把读取到的数据赋值给by
            by != -1: 判断读取到的数据是否是-1
         */
        while ((by=fis.read())!=-1) {
            System.out.print((char)by);
        }

        //释放资源
        fis.close();
    }
}
```

2.7字节流复制文件【应用】

- 案例需求

  把“E:\itcast\窗里窗外.txt”复制到模块目录下的“窗里窗外.txt” (文件可以是任意文件)

- 实现步骤
  - 复制文本文件，其实就把文本文件的内容从一个文件中读取出来(数据源)，然后写入到另一个文件中(目的地)
  - 数据源：

    E:\itcast\窗里窗外.txt --- 读数据 --- InputStream --- FileInputStream
  - 目的地：

    myByteStream\窗里窗外.txt --- 写数据 --- OutputStream --- FileOutputStream

- 代码实现

```
public class CopyTxtDemo {
    public static void main(String[] args) throws IOException {
        //根据数据源创建字节输入流对象
        FileInputStream fis = new FileInputStream("E:\\itcast\\窗里窗外.txt");
        //根据目的地创建字节输出流对象
        FileOutputStream fos = new FileOutputStream("myByteStream\\窗里窗外.txt");

        //读写数据，复制文本文件(一次读取一个字节，一次写入一个字节)
        int by;
        while ((by=fis.read())!=-1) {
            fos.write(by);
        }

        //释放资源
        fos.close();
        fis.close();
    }
}
```

2.8字节流读数据(一次读一个字节数组数据)【应用】

- 一次读一个字节数组的方法
  - public int read(byte[] b)：从输入流读取最多b.length个字节的数据
  - 返回的是读入缓冲区的总字节数,也就是实际的读取字节个数
- 示例代码

```
public class FileInputStreamDemo02 {
    public static void main(String[] args) throws IOException {
        //创建字节输入流对象
        FileInputStream fis = new FileInputStream("myByteStream\\fos.txt");

        byte[] bys = new byte[1024]; //1024及其整数倍
        int len;
        //循环读取
        while ((len=fis.read(bys))!=-1) {
            System.out.print(new String(bys,0,len));
        }

        //释放资源
        fis.close();
    }
}
```

2.9字节流复制文件【应用】

- 案例需求

  把“E:\itcast\mn.jpg”复制到模块目录下的“mn.jpg” (文件可以是任意文件去)
- 实现步骤
  - 根据数据源创建字节输入流对象
  - 根据目的地创建字节输出流对象
  - 读写数据，复制图片(一次读取一个字节数组，一次写入一个字节数组)
  - 释放资源
- 代码实现

```
public class CopyJpgDemo {
    public static void main(String[] args) throws IOException {
        //根据数据源创建字节输入流对象
        FileInputStream fis = new FileInputStream("E:\\itcast\\mn.jpg");
        //根据目的地创建字节输出流对象
        FileOutputStream fos = new FileOutputStream("myByteStream\\mn.jpg");

        //读写数据，复制图片(一次读取一个字节数组，一次写入一个字节数组)
        byte[] bys = new byte[1024];
        int len;
        while ((len=fis.read(bys))!=-1) {
            fos.write(bys,0,len);
        }
```

```
        //释放资源
        fos.close();
        fis.close();
    }
}
```

3.字节缓冲流

3.1字节缓冲流构造方法【应用】

- 字节缓冲流介绍
  - lBufferOutputStream：该类实现缓冲输出流.通过设置这样的输出流,应用程序可以向底层输出流写入字节,而不必为写入的每个字节导致底层系统的调用
  - lBufferedInputStream：创建BufferedInputStream将创建一个内部缓冲区数组.当从流中读取或跳过字节时,内部缓冲区将根据需要从所包含的输入流中重新填充,一次很多字节
- 构造方法：

方法名	说明
BufferedOutputStream(OutputStream out)	创建字节缓冲输出流对象
BufferedInputStream(InputStream in)	创建字节缓冲输入流对象

- 示例代码

```
public class BufferStreamDemo {
    public static void main(String[] args) throws IOException {
        //字节缓冲输出流：BufferedOutputStream(OutputStream out)

        BufferedOutputStream bos = new BufferedOutputStream(new
                            FileOutputStream("myByteStream\\bos.txt"));
        //写数据
        bos.write("hello\r\n".getBytes());
        bos.write("world\r\n".getBytes());
        //释放资源
        bos.close();


        //字节缓冲输入流：BufferedInputStream(InputStream in)
        BufferedInputStream bis = new BufferedInputStream(new
                           FileInputStream("myByteStream\\bos.txt"));

        //一次读取一个字节数据
//        int by;
//        while ((by=bis.read())!=-1) {
//            System.out.print((char)by);
//        }

        //一次读取一个字节数组数据

        byte[] bys = new byte[1024];
```

```
        int len;
        while ((len=bis.read(bys))!=-1) {
            System.out.print(new String(bys,0,len));
        }

        //释放资源
        bis.close();
    }
}
```

3.2字节缓冲流复制视频【应用】

- 案例需求

  把“E:\itcast\字节流复制图片.avi”复制到模块目录下的“字节流复制图片.avi”

- 实现步骤
  - 根据数据源创建字节输入流对象
  - 根据目的地创建字节输出流对象
  - 读写数据，复制视频
  - 释放资源
- 代码实现

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

        //复制视频
//        method1();
         method2();

    }

    //字节缓冲流一次读写一个字节数组
    public static void method2() throws IOException {
        BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("E:\\itcast\\字节流复制图片.avi"));
        BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("myByteStream\\字节流复制图片.avi"));

        byte[] bys = new byte[1024];
        int len;
        while ((len=bis.read(bys))!=-1) {
            bos.write(bys,0,len);
        }

        bos.close();
        bis.close();
    }

    //字节缓冲流一次读写一个字节
    public static void method1() throws IOException {

        BufferedInputStream bis = new BufferedInputStream(new
```

```
FileInputStream("E:\\itcast\\字节流复制图片.avi"));
        BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("myByteStream\\字节流复制图片.avi"));

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

        bos.close();
        bis.close();
    }

}
```