day12【File类、递归、IO流、字节流、字符流】

今日内容

- File类
- 递归
- IO流
- 字节流
- 字符流

教学目标

- [ ] 能够说出File对象的创建方式
- [ ] 能够使用File类常用方法
- [ ] 能够辨别相对路径和绝对路径
- [ ] 能够遍历文件夹
- [ ] 能够解释递归的含义
- [ ] 能够使用递归的方式计算5的阶乘
- [ ] 能够说出使用递归会内存溢出隐患的原因
- [ ] 能够说出IO流的分类和功能
- [ ] 能够使用字节输出流写出数据到文件
- [ ] 能够使用字节输入流读取数据到程序
- [ ] 能够理解读取数据read(byte[])方法的原理
- [ ] 能够使用字节流完成文件的复制
- [ ] 能够使用FileWriter写数据的5个方法
- [ ] 能够说出FileWriter中关闭和刷新方法的区别
- [ ] 能够使用FileWriter写数据实现换行和追加写
- [ ] 能够使用FileReader读数据一次一个字符
- [ ] 能够使用FileReader读数据一次一个字符数组

第一章 File类

知识点-- File类的概述和构造方法

目标

- 理解File类的概述和创建File类对象

路径

- File类的概述

- File类的构造方法

讲解

File类的概述

`java.io.File` 类是文件和目录路径名的抽象表示，主要用于文件和目录的创建、查找和删除等操作。

File类的构造方法

- `public 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。
- `public 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。
- `public 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。

- 构造举例，代码如下：

```
public class Test {
    public static void main(String[] args) {
        /*
            File类的概述
            java.io.File 类是文件和目录路径名的抽象表示，主要用于文件和目录的创建、查找和删除等操作。

            File类的构造方法
            - public 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。
            - public 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。
            - public 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。
            注意:
                无论文件路径或者文件夹路径是否真实存在,都不会影响File对象的创建
         */
        // File对象表示的路径是真实存在的
        // 创建File对象,表示H:\aaa\hb.jpg文件路径
        File file1 = new File("H:\\aaa\\hb.jpg");
        File file2 = new File("H:\\aaa","hb.jpg");

        File parent = new File("H:\\aaa");
        File file3 = new File(parent, "hb.jpg");

        System.out.println("file1:"+file1);// file1:H:\aaa\hb.jpg
        System.out.println("file2:"+file2);// file2:H:\aaa\hb.jpg
        System.out.println("file3:"+file3);// file3:H:\aaa\hb.jpg

        System.out.println("============================");
        // File对象表示的路径不是真实存在的
        File file4 = new File("H:\\aaa\\b.txt");
        File file5 = new File("H:\\aaa\\ccc");
        System.out.println("file4:"+file4);
        System.out.println("file5:"+file5);
    }
}
```

> 小贴士：
>
> 1. 一个File对象代表硬盘中实际存在的一个文件或者目录。
> 2. 无论该路径下是否存在文件或者目录，都不影响File对象的创建。

小结

略

知识点-- File类常用方法

目标

- 掌握File类常用方法的使用

路径

- 获取功能的方法
- 绝对路径和相对路径
- 判断功能的方法
- 创建删除功能的方法

讲解

获取功能的方法

- `public String getAbsolutePath()`：返回此File的绝对路径名字符串。
- `public String getPath()`：将此File转换为路径名字符串。
- `public String getName()`：返回由此File表示的文件或目录的名称。
- `public long length()`：返回由此File表示的文件的长度。 不能获取目录的长度。

  方法演示，代码如下：

```
public class Test1获取功能方法 {
    public static void main(String[] args) {

        // 创建File对象,表示G:\szitheima98\day12\aaa\hb.jpg文件的路径
        File file1 = new File("day12\\aaa\\hb.jpg");
        File file2 = new File("G:\\szitheima98\\day12\\aaa\\hb.jpg");

        // 获取绝对路径和构造路径
        System.out.println("file1表示的文件的绝对路径:
"+file1.getAbsolutePath());// G:\szitheima98\day12\aaa\hb.jpg
        System.out.println("file1表示的文件的构造路径: "+file1.getPath());//
day12\aaa\hb.jpg
        System.out.println("file2表示的文件的绝对路径:
"+file2.getAbsoluteFile());// G:\szitheima98\day12\aaa\hb.jpg
        System.out.println("file2表示的文件的构造路径: "+file2.getPath());//
G:\szitheima98\day12\aaa\hb.jpg

        // 获取文件名
        System.out.println("file1表示的文件的文件名: "+file1.getName());//
hb.jpg
        // 获取文件的字节大小
```

```
        System.out.println("file1表示的文件的字节大小: "+file1.length());//
24666

        System.out.println("==================================");

        // 创建File对象,表示G:\szitheima98\day12\aaa文件夹的路径
        File file3 = new File("day12\\aaa");
        System.out.println("file3表示的文件夹的绝对路径:
"+file3.getAbsolutePath());//  G:\szitheima98\day12\aaa
        System.out.println("file3表示的文件夹的构造路径: "+file3.getPath());//
day12\aaa
        System.out.println("file3表示的文件夹的名称: "+file3.getName());// aaa
        System.out.println("file3表示的文件夹的字节大小: "+file3.length());//
0

    }
}
```

> API中说明：length()，表示文件的长度。但是File对象表示目录，则返回值未指定。

绝对路径和相对路径

- **绝对路径**：从盘符开始的路径，这是一个完整的路径。
- **相对路径**：相对于项目目录的路径，这是一个便捷的路径，开发中经常使用。

```
  public static void main(String[] args) {
        /*
            - 绝对路径：从盘符开始的路径，这是一个完整的路径。
            - 相对路径：相对于项目目录的路径，这是一个便捷的路径，开发中经常使用。

           举例:
            生活中的例子:假设你在深圳市宝安区中粮商务公园3栋1701教室  你女朋友在深圳市宝安区
中粮商务公园楼下
            绝对路径: 中国广东省深圳市宝安区中粮商务公园3栋1701教室
            相对路径: 3栋1701教室

            程序中的例子:
                绝对路径: G:\szitheima0414\day15\aaa\a.txt
                相对路径: day15\aaa\a.txt

           注意: 开发中使用相对路径
         */
        // 绝对路径
        File file1 = new File("G:\\szitheima0414\\day15\\aaa\\a.txt");
        System.out.println("file1的绝对路径:"+file1.getAbsoluteFile());//
G:\szitheima0414\day15\aaa\a.txt
        System.out.println("file1的构造路径:"+file1.getPath());//
G:\szitheima0414\day15\aaa\a.txt

        // 相对路径
        File file2 = new File("day15\\aaa\\a.txt");
        System.out.println("file2的绝对路径:"+file2.getAbsoluteFile());//
G:\szitheima0414\day15\aaa\a.txt
        System.out.println("file2的构造路径:"+file2.getPath());// day15\aaa\a.txt
    }
```

判断功能的方法

- `public boolean exists()` ：此File表示的文件或目录是否实际存在。
- `public boolean isDirectory()` ：此File表示的是否为目录。
- `public boolean isFile()` ：此File表示的是否为文件。

方法演示，代码如下：

```
public class Test2判断功能方法 {
    public static void main(String[] args) {
        /*
            判断功能的方法
                - public boolean exists() ：此File表示的文件或目录是否实际存在。
                - public boolean isDirectory() ：此File表示的是否为目录。
                - public boolean isFile() ：此File表示的是否为文件。
            注意:
                只对路径存在的文件或者文件夹进行判断,如果路径不存在,那么结果都是false
         */
        // 文件存在
        // 创建File对象,表示G:\szitheima98\day12\aaa\hb.jpg文件的路径
        File file1 = new File("day12\\aaa\\hb.jpg");
        System.out.println("file1表示的路径是否真实存在:"+file1.exists());// true
        System.out.println("file1表示的路径是否是文件夹路
径:"+file1.isDirectory());// false
        System.out.println("file1表示的路径是否是文件路径:"+file1.isFile());// true

        // 文件不存在
        File file2 = new File("day12\\aaa\\hb1.jpg");
        System.out.println("file2表示的路径是否真实存在:"+file2.exists());// false
        System.out.println("file2表示的路径是否是文件夹路
径:"+file2.isDirectory());// false
        System.out.println("file2表示的路径是否是文件路径:"+file2.isFile());// false

        // 文件夹存在
        // 创建File对象,表示G:\szitheima98\day12\aaa文件夹的路径
        File file3 = new File("day12\\aaa");
        System.out.println("file3表示的路径是否真实存在:"+file3.exists());// true
        System.out.println("file3表示的路径是否是文件夹路
径:"+file3.isDirectory());// true
        System.out.println("file3表示的路径是否是文件路径:"+file3.isFile());// false

        // 文件夹不存在
        File file4 = new File("day12\\zzz");
        System.out.println("file4表示的路径是否真实存在:"+file4.exists());// false
        System.out.println("file4表示的路径是否是文件夹路
径:"+file4.isDirectory());// false
        System.out.println("file4表示的路径是否是文件路径:"+file4.isFile());// false

    }
}
```

创建删除功能的方法

- `public boolean createNewFile()` ：当且仅当具有该名称的文件尚不存在时，创建一个新的空文件。
- `public boolean delete()` ：删除由此File表示的文件或目录。
- `public boolean mkdir()` ：创建由此File表示的目录。

- `public boolean mkdirs()` ：创建由此File表示的目录，包括任何必需但不存在的父目录。

方法演示，代码如下：

```
public class Test3创建和删除功能方法 {
    public static void main(String[] args) throws IOException {
        /*
            创建删除功能的方法
                - public boolean createNewFile() ：当且仅当具有该名称的文件尚不存在时，创建一个新的空文件。
                - public boolean delete() ：删除由此File表示的文件或目录。注意不能删除非空文件夹
                - public boolean mkdir() ：创建由此File表示的目录。
                - public boolean mkdirs() ：创建由此File表示的目录，包括任何必需但不存在的父目录。
         */
        // 创建文件:
        // 创建File对象,表示文件路径
        // File file1 = new File("day12\\aaa\\b.txt");
        // System.out.println("文件是否存在:"+file1.exists());// false
        // file1.createNewFile();
        // System.out.println("文件是否存在:"+file1.exists());// true


        // File file2 = new File("day12\\aaa\\bbb");
        // System.out.println("文件是否存在:"+file2.exists());// false
        // file2.createNewFile();
        // System.out.println("文件是否存在:"+file2.exists());// true

        //System.out.println("============================");
        // 删除文件
        //File file1 = new File("day12\\aaa\\b.txt");
        //System.out.println("文件是否删除成功:"+file1.delete());// true

        // 删除文件夹
        // File file2 = new File("day12\\aaa");
        // System.out.println("文件夹是否删除成功:"+file2.delete());// false

        // File file2 = new File("day12\\aaa\\ccc");
        // System.out.println("文件夹是否删除成功:"+file2.delete());// true

        System.out.println("============================");
        // 创建文件夹
        // File file3 = new File("day12\\aaa\\ccc");;
        // System.out.println("是否创建文件夹成功:"+file3.mkdir());// true
        // System.out.println("是否创建文件夹成功:"+file3.mkdir());// false

        File file4 = new File("day12\\aaa\\ddd\\aaa\\bbb\\ccc");
        //System.out.println("是否创建文件夹成功:"+file4.mkdir());// false
        System.out.println("是否创建文件夹成功:"+file4.mkdirs());// true
    }
}
```

> API中说明：delete方法，如果此File表示目录，则目录必须为空才能删除。

小结

略

知识点-- File类遍历目录方法

目的

- 掌握File类遍历目录方法的使用

路径

- File类遍历目录方法

讲解

- `public String[] list()`：返回一个String数组，存储的是File目录中所有的子文件或子目录的名称。
- `public File[] listFiles()`：返回一个File数组，存储的是File目录中所有的子文件或子目录的路径的File对象。

```
public class Test {
    public static void main(String[] args) {
        /*
            - public String[] list() ：返回一个String数组，存储的是File目录中所有的子文件或子目录的名称。
            - public File[] listFiles() ：返回一个File数组，存储的是File目录中所有的子文件或子目录的路径的File对象。

            注意： 如果文件夹路径不存在或者没有访问权限,以上2个方法获取的时候会返回null,容易出现空指针异常
                    所以大家在遍历获取到的子文件和子目录之前,应该做非空判断
         */
        // 文件夹路径存在
        // 创建File对象,表示一个文件夹路径
        File file1 = new File("day12\\aaa");

        // 获取所有的子文件或子目录的名称。
        String[] arr1 = file1.list();
        // 循环遍历
        if (arr1 != null){
            for (String path : arr1) {
                System.out.println(path);
            }
        }

        System.out.println("======================");
        // 所有的子文件或子目录路径的File对象
        File[] arr2 = file1.listFiles();
        // 循环遍历
        if (arr2 != null) {
            for (File filePath : arr2) {
                System.out.println(filePath);
            }
        }

        System.out.println("======================");
```

```
        // 注意: 如果文件夹路径不存在,获取的时候会返回null,容易出现空指针异常
        // 文件夹路径不存在
        File file2 = new File("day12\\bbb");

        // 获取所有的子文件或子目录的名称。
        String[] arr3 = file2.list();
        System.out.println("arr3:"+arr3);// null

        File[] arr4 = file2.listFiles();
        System.out.println("arr4:"+arr4);// null
        // 循环遍历
        //for (String path : arr3) {// 报空指针异常
        //    System.out.println(path);
        //}

        // 注意: 如果文件夹路径没有访问权限,获取的时候会返回null,容易出现空指针异常
        File file3 = new File("G:\\System Volume Information");
        System.out.println(file3.listFiles());// null
        System.out.println(file3.list());// null
    }
}
```

> 小贴士:
>
> 调用listFiles方法的File对象，表示的必须是实际存在的目录，否则返回null，无法进行遍历。

小结

略

第二章 递归

知识点--递归的概述

目标

- 理解递归的概述

路径

- 递归的概述
- 案例演示

讲解

递归的概述

- 生活中的递归: 放羊--赚钱--盖房子--娶媳妇--生娃--放羊--赚钱--盖房子--娶媳妇--生娃--放羊...
- 程序中的递归: 指在当前方法内调用自己的这种现象。
- 递归的注意事项:
  - 递归要有出口(结束方法),否则会报栈内存溢出错误StackOverflowError
  - 递归的出口不能太晚了

案例演示

```
public class Test {

    // 旗帜变量
    static int count = 0;

    public static void main(String[] args) {
        /*
            递归的概述:
                生活中的递归: 放羊-->赚钱-->盖房子-->娶媳妇-->生娃--->放羊-->赚钱-->盖房子-->娶媳妇-->生娃.....
                程序中的递归: 指在当前方法内调用自己的这种现象。
                递归的注意事项:
                    1.如果无限递归,就会出现栈内存溢出错误StackOverflowError
                    2.递归一定要有出口,并且出口不能太晚,否则还是会出现栈内存溢出错误StackOverflowError
                    解决办法: 就是合理递归
         */
        method();// 调用method()方法
    }

    public static void method(){
        if (count == 5){// 出口
            return;// 当count加到5的时候,就结束递归
        }
        count++;
        System.out.println("执行了吗...");
        method();
    }
}
```

小结

略

实操-- 递归累和

需求

- 计算1 ~ n的累加和

分析

- num的累加和 = num + (num-1)的累和，所以可以把累加和的操作定义成一个方法，递归调用。

实现

代码实现

```
public class Test1 {
    public static void main(String[] args) {
        /*
            练习一:使用递归计算1 ~ n的和
                分析:
```

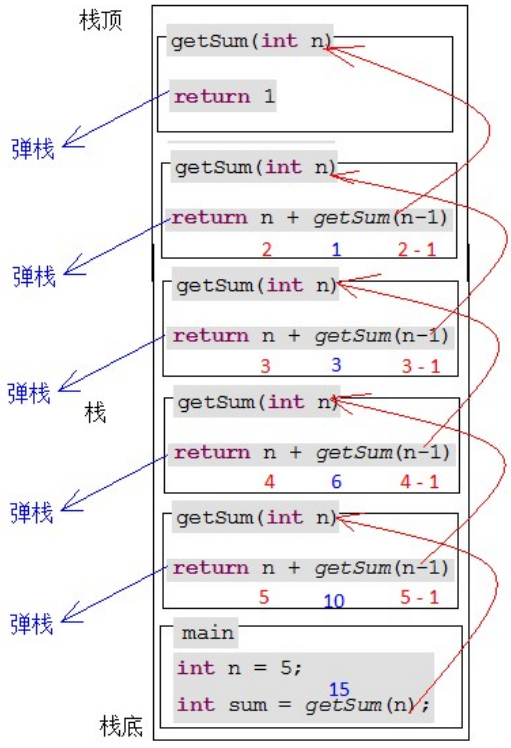
```
                        1 的累加和 = 1                          1的累加和=1
                        2 的累加和 = 1 + 2                      2的累加和=2+1的累加和
                        3 的累加和 = 1 + 2 + 3                  3的累加和=3+2的累加和
                        4 的累加和 = 1 + 2 + 3 + 4              4的累加和=4+3的累加和
                        .....
                        n 的累加和                              n的累加和=n+(n-1)的累加
和
         */
        // 调用getSum方法计算5的累加和
        int sum = getSum(5);
        System.out.println("5的累加和:"+sum);// 15
    }

    /**
     * 计算一个数的累加和
     * @param n
     * @return
     */
    public static int getSum(int n){
        // 出口
        if(n == 1){
            return 1;
        }
        return n + getSum(n-1);// 规律
    }
}
```

代码执行图解

```
public class DiGuiDemo {
    public static void main(String[] args) {
        //计算 1~num的和，使用递归完成
        int n = 5;
        int sum = getSum(n);
        System.out.println(sum);

    }
    public static int getSum(int n) {
        if(n == 1){
            return 1;
        }
        return n + getSum(n-1);
    }
}
```



小结

略

实操-- 递归求阶乘

需求

- 计算n的阶乘

分析

- **阶乘**：所有小于及等于该数的正整数的积。

```
n的阶乘：n! = n * (n-1) *...* 3 * 2 * 1
```

n的阶乘 = n * (n1)的阶乘，所以可以把阶乘的操作定义成一个方法，递归调用。

```
推理得出：n! = n * (n-1)!
```

实现

代码实现：

```
public class Test2 {
    public static void main(String[] args) {
        /*
            递归求阶乘:
                规律:
                    1! = 1                                          1的阶乘 : 1
                    2! = 2 * 1                                      2的阶乘 : 2 * 1的
阶乘
                    3! = 3 * 2 * 1                                  3的阶乘 : 3 * 2的
阶乘
                    4! = 4 * 3 * 2 * 1                              4的阶乘 : 4 * 3的
阶乘
                    5! = 5 * 4 * 3 * 2 * 1                          5的阶乘 : 5 * 4的
阶乘
                    ....
                    num! = num * (num-1) * (num-2) *...* 1          num的阶乘 : num *
num-1的阶乘
         */
        int res = jieCheng(5);
        System.out.println("5的阶乘:"+res);// 5的阶乘:120
    }

    /**
     * 计算一个数的阶乘
     * @param num
     * @return
     */
    public static int jieCheng(int num){
        // 出口
        if (num == 1){
            return 1;
        }
        return num *  jieCheng(num-1); // 计算阶乘的规律
    }
}
```

小结

实操-- 文件搜索

需求

- 输出day12\src目录中的所有.java文件的绝对路径。

分析

1. 目录搜索，无法判断多少级目录，所以使用递归，遍历所有目录。
2. 遍历目录时，获取的子文件，通过文件名称，判断是否符合条件。

实现

```
public class Test3_文件搜索 {
    public static void main(String[] args) {
        /*
            需求:
                输出day12\\src目录中的所有.java文件的绝对路径。
            分析:
                1.定义一个方法,用来获取一个目录中所有符合条件的文件(子文件,子子文件,子子子文件...)
                2.在方法中,获取该目录下的所有子文件和子目录
                3.在方法中,循环遍历获取到的所有字文件和子目录
                4.在方法中,遍历的时候,需要判断遍历出来的是文件还是目录
                5.如果是文件,就判断该文件是否以.java结尾,如果是就获取其绝对路径打印输出
                6.如果是文件夹,就递归调用该方法
         */
        File file = new File("day12\\src");
        findFile(file);
    }

    /**
     * 定义一个方法,用来获取一个目录中所有符合条件的文件(子文件,子子文件,子子子文件...)
     * @param file
     */
    public static void findFile(File file) {
        // 在方法中,获取该目录下的所有子文件和子目录
        File[] files = file.listFiles();

        // 在方法中,循环遍历获取到的所有字文件和子目录
        if (files != null) {
            // 在方法中,遍历的时候,需要判断遍历出来的是文件还是目录
            for (File file1 : files) {
                // 如果是文件,就判断该文件是否以.java结尾,如果是就获取其绝对路径打印输出
                if (file1.isFile() && file1.getName().endsWith(".java")){
                    System.out.println(file1.getAbsolutePath());
                }
                // 如果是文件夹,就递归调用该方法
                if (file1.isDirectory()){
                    findFile(file1);
                }
            }
        }
    }
```

```
}
```

小结

略

第三章 IO概述

目标

- 理解IO的概述以及IO的分类

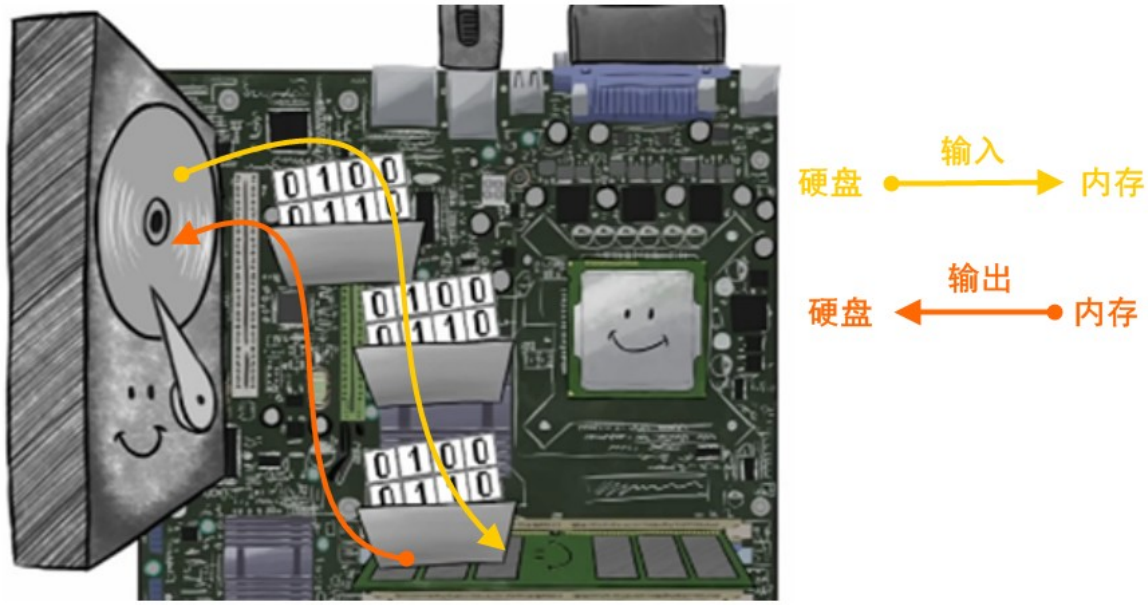
路径

- IO的概述
- IO的分类
- IO的顶层父类
- 注意事项

讲解

IO的概述

- I : Input 输入 从其他存储设备读数据到内存中就是输入
- O : Output 输出 从内存中写数据到其他存储设备



IO的分类

根据数据的流向分为：**输入流**和**输出流**。

- **输入流**： 把数据从`其他设备`上读取到`内存`中的流。
    - 字节输入流:以字节为基本单位,读数据
    - 字符输入流:以字符为基本单位,读数据
- **输出流**： 把数据从`内存`中写出到`其他设备`上的流。

    - 字节输出流:以字节为基本单位,写出数据
    - 字符输出流:以字符为基本单位,写出数据

根据数据的类型分为：**字节流**和**字符流**。

- **字节流** ： 以字节为单位，读写数据的流。
    - 字节输入流:以字节为基本单位,读数据
    - 字节输出流:以字节为基本单位,写出数据
- **字符流** ： 以字符为单位，读写数据的流。
    - 字符输入流:以字符为基本单位,读数据
    - 字符输出流:以字符为基本单位,写出数据

IO的顶层父类

- 字节输入流:顶层父类 InputStream 抽象类
- 字节输出流:顶层父类 OutputStream 抽象类
- 字符输入流:顶层父类 Reader 抽象类
- 字符输出流:顶层父类 Writer 抽象类

注意事项

- utf8编码一个中文占3个字节,gbk编码一个中文占2个字节
- 如果存储和解析的编码不一致就会乱码
- idea默认编码是utf8

小结

略

第四章 字节流

知识点-- 字节输出流【OutputStream】

目标

- 理解OutputStream类的概述以及常用方法

路径

- OutputStream类的概述
- OutputStream类的常用方法

讲解

OutputStream类的概述

`java.io.OutputStream`抽象类是表示字节输出流的所有类的超类，将指定的字节信息写出到目的地。它定义了字节输出流的基本共性功能方法。

OutputStream类的常用方法

- `public void close()` ： 关闭此输出流并释放与此流相关联的任何系统资源。
- `public void write(byte[] b)`： 将 b.length字节从指定的字节数组写入此输出流。

- `public void write(byte[] b, int off, int len)`：从指定的字节数组写入 len字节，从偏移量 off开始输出到此输出流。
- `public abstract void write(int b)`：将指定的字节输出流。

> 小贴士：
>
> close方法，当完成流的操作时，必须调用此方法，释放系统资源。

小结

略

知识点-- FileOutputStream类

目标

- 掌握FileOutputStream类的使用

路径

- FileOutputStream类的概述
- FileOutputStream类的构造方法
- FileOutputStream类的写出数据
- 数据追加续写
- 写出换行

讲解

FileOutputStream类的概述

`java.io.FileOutputStream`类是OutputStream类的子类,用来表示是文件输出流，用于将数据写出到文件。

FileOutputStream类的构造方法

- `public 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。
- `public FileOutputStream(String name)`： 创建文件输出流以指定的名称写入文件。

当你创建一个流对象时，必须传入一个文件路径。该路径下，如果没有这个文件，会创建该文件。如果有这个文件，会清空这个文件的数据。

- 构造举例，代码如下：

```
public class FileOutputStreamConstructor throws IOException {
    public static void main(String[] args) {
        // a.txt文件不存在
        FileOutputStream fos1 = new FileOutputStream(new
File("day12\\bbb\\a.txt"));// 会自动创建
        // b.txt文件存在,并且有内容
        FileOutputStream fos2 = new FileOutputStream(new
File("day12\\bbb\\b.txt"));

        System.out.println("============================");
        FileOutputStream fos3 = new FileOutputStream("day12\\bbb\\a.txt");
    }
}
```

FileOutputStream类的写出数据

1. **写出字节**：`write(int b)` 方法，每次可以写出一个字节数据，代码使用演示：

```
public class FOSWrite {
    public static void main(String[] args) throws IOException {
       // 1.创建FileOutputStream字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day12\\bbb\\c.txt");
        // 2.写出一个字节数据
        fos.write(97);// 97-->文件中存储的是97的二进制数--->打开文件解析(文本中都是字
符)--->a
        // 3.关闭流,释放资源
        fos.close();
    }
}
写出结果:
a
```

> 小贴士：
>
> 1. 虽然参数为int类型四个字节，但是只会保留一个字节的信息写出。
> 2. 流操作完毕后，必须释放系统资源，调用close方法，千万记得。

1. **写出字节数组**：`write(byte[] b)`，每次可以写出数组中的数据，代码使用演示：

```
public class FOSWrite {
    public static void main(String[] args) throws IOException {
       // 1.创建FileOutputStream字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day12\\bbb\\d.txt");

        // 2.写整个数组的字节数据
        byte[] bys = {97, 98, 99, 100, 48, 49, 50};
        fos.write(bys);

        // 3.关闭流,释放资源
        fos.close();
    }
}
写出结果:
abcd012
```

1. **写出指定长度字节数组**：`write(byte[] b, int off, int len)`,每次写出从off索引开始，len个字节，代码使用演示：

```
public class FOSWrite {
    public static void main(String[] args) throws IOException {
         // 案例3: 写指定数组范围的字节数据到day12\\bbb\\e.txt文件中
        // 1.创建FileOutputStream字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day12\\bbb\\e.txt");

        // 2.写指定数组范围的字节数据
        byte[] bys = {97, 98, 99, 100, 48, 49, 50};
        fos.write(bys,1,3);

        // 3.关闭流,释放资源
        fos.close();
    }
}
写出结果:
bcd
```

数据追加续写

经过以上的演示，每次程序运行，创建输出流对象，都会清空目标文件中的数据。如何保留目标文件中数据，还能继续添加新数据呢?

- `public FileOutputStream(File file, boolean append)`： 创建文件输出流以写入由指定的File对象表示的文件。
- `public FileOutputStream(String name, boolean append)`： 创建文件输出流以指定的名称写入文件。

这两个构造方法，参数中都需要传入一个boolean类型的值，`true` 表示追加数据，`false` 表示清空原有数据。这样创建的输出流对象，就可以指定是否追加续写了，代码使用演示：

```
public class Test {
    public static void main(String[] args) throws Exception {
        // 1.创建FileOutputStream字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day12\\bbb\\f.txt",true);//
true 追击续写
        // 2.写出一个字节数据
        fos.write(98);
        // 3.关闭流,释放资源
        fos.close();
    }
}
文件操作前：a
文件操作后：ab
```

写出换行

Windows系统里，换行符号是`\r\n` 。把

代码使用演示：

```
public class FOSWrite {
    public static void main(String[] args) throws IOException {
          // 需求: 把这首诗写入day15\\aaa\\d.txt文件中
```

```
        // String类 byte[] getBytes();把一个字符串转换为一个字节数组
        // 创建字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day15\\aaa\\d.txt");
        // 写出数据
        fos.write("吟诗一首".getBytes());
        fos.write("\r\n".getBytes());
        fos.write("看这风景美如画".getBytes());
        fos.write("\r\n".getBytes());
        fos.write("吟诗一首赠天下".getBytes());
        fos.write("\r\n".getBytes());
        fos.write("奈何本人没文化".getBytes());
        fos.write("\r\n".getBytes());
        fos.write("只能卧槽浪好大".getBytes());

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

> - 回车符`\r`和换行符`\n`：
>   - 回车符：回到一行的开头（return）。
>   - 换行符：下一行（newline）。
> - 系统中的换行：
>   - Windows系统里，每行结尾是 `回车+换行` ，即`\r\n`；
>   - Unix系统里，每行结尾只有 `换行` ，即`\n`；
>   - Mac系统里，每行结尾是 `回车` ，即`\r`。从 Mac OS X开始与Linux统一。

小结

略

知识点-- 字节输入流【InputStream】

目标

- 理解InputStream类的概述以及常用方法

路径

- InputStream类的概述
- InputStream类的常用方法

讲解

InputStream类的概述

`java.io.InputStream`抽象类是表示字节输入流的所有类的超类，可以读取字节信息到内存中。它定义了字节输入流的基本共性功能方法。

InputStream类的常用方法

- `public void close()` ：关闭此输入流并释放与此流相关联的任何系统资源。
- `public abstract int read()`： 从输入流读取数据的下一个字节。
- `public int read(byte[] b)`： 从输入流中读取一些字节数，并将它们存储到字节数组 b中 。

> 小贴士：
>
> close方法，当完成流的操作时，必须调用此方法，释放系统资源。

小结

略

知识点-- FileInputStream类

目标

- 掌握FileInputStream类的使用

路径

- FileInputStream类的概述
- FileInputStream类的构造方法
- FileInputStream类的读取数据

讲解

FileInputStream类的概述

`java.io.FileInputStream`类是InputStream类的子类，用来表示文件输入流，从文件中读取字节。

FileInputStream类的构造方法

- `FileInputStream(File file)`： 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文件对象 file命名。
- `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。

当你创建一个流对象时，必须传入一个文件路径。该路径下，如果没有该文件,会抛出`FileNotFoundException`

- 构造举例，代码如下：

```
public class FileInputStreamConstructor throws IOException{
    private static void method01() throws FileNotFoundException {
        // 数据源文件路径存在,正常
        FileInputStream fis1 = new FileInputStream("day10\\eee\\a.txt");

        // 数据源文件路径不存在,报文件找不到异常
        FileInputStream fis2 = new FileInputStream("day10\\eee\\utf8.txt");
    }
}
```

FileInputStream类的读取数据

1. **读取字节**：`read`方法，每次可以读取一个字节的数据，提升为int类型，读取到文件末尾，返回`-1`，代码使用演示：

```
public class FISRead {
    public static void main(String[] args) throws IOException{
        // 使用文件名称创建流对象
```

```
        FileInputStream fis = new FileInputStream("read.txt");
        // 读取数据，返回一个字节
        int read = fis.read();
        System.out.println((char) read);
        read = fis.read();
        System.out.println((char) read);
        read = fis.read();
        System.out.println((char) read);
        read = fis.read();
        System.out.println((char) read);
        read = fis.read();
        System.out.println((char) read);
        read = fis.read();
        System.out.println((char) read);
        // 读取到末尾,返回-1
        read = fis.read();
        System.out.println( read);
        // 关闭资源
        fis.close();
    }
}
输出结果:
a
b
c
d
e
-1
```

循环改进读取方式，代码使用演示：

```
public class FISRead {
    public static void main(String[] args) throws IOException{
        // 使用文件名称创建流对象
        FileInputStream fis = new FileInputStream("read.txt");
        // 定义变量，保存数据
        int b ;
        // 循环读取
        while ((b = fis.read())!=-1) {
            System.out.println((char)b);
        }
        // 关闭资源
        fis.close();
    }
}
输出结果:
a
b
c
d
e
```

> 小贴士：
>
> 1. 虽然读取了一个字节，但是会自动提升为int类型。
> 2. 流操作完毕后，必须释放系统资源，调用close方法，千万记得。

1. **使用字节数组读取**：`read(byte[] b)`，每次读取b的长度个字节到数组中，返回读取到的有效字节个数，读取到末尾时，返回`-1`，代码使用演示：

```
public class FISRead {
    public static void main(String[] args) throws IOException{
        // 使用文件名称创建流对象.
        FileInputStream fis = new FileInputStream("read.txt"); // 文件中为abcde
        // 定义变量，作为有效个数
        int len ;
        // 定义字节数组，作为装字节数据的容器
        byte[] b = new byte[2];
        // 循环读取
        while (( len= fis.read(b))!=-1) {
            // 每次读取后,把数组变成字符串打印
            System.out.println(new String(b));
        }
        // 关闭资源
        fis.close();
    }
}

输出结果:
ab
cd
ed
```

错误数据`d`，是由于最后一次读取时，只读取一个字节`e`，数组中，上次读取的数据没有被完全替换，所以要通过`len`，获取有效的字节，代码使用演示：

```
public class FISRead {
    public static void main(String[] args) throws IOException{
        // 使用文件名称创建流对象.
        FileInputStream fis = new FileInputStream("read.txt"); // 文件中为abcde
        // 定义变量，作为有效个数
        int len ;
        // 定义字节数组，作为装字节数据的容器
        byte[] b = new byte[2];
        // 循环读取
        while (( len= fis.read(b))!=-1) {
            // 每次读取后,把数组的有效字节部分，变成字符串打印
            System.out.println(new String(b, 0, len));//  len 每次读取的有效字节个数
        }
        // 关闭资源
        fis.close();
    }
}

输出结果:
ab
cd
e
```

> 小贴士：
>
> 使用数组读取，每次读取多个字节，减少了系统间的IO操作次数，从而提高了读写的效率，建议开发中使用。

小结

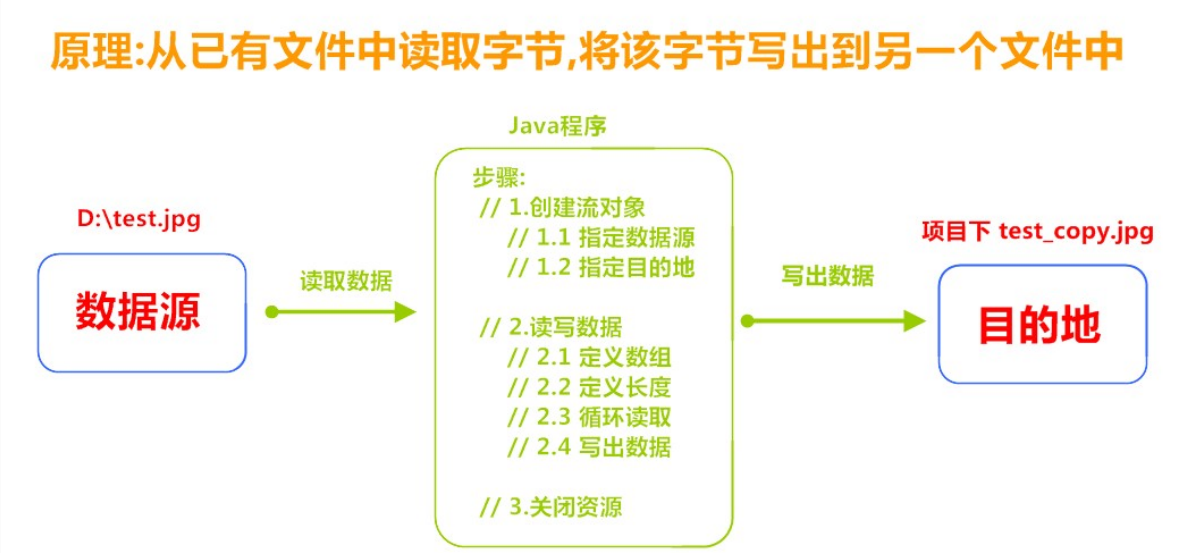
略

实操-- 字节流练习：图片复制

需求

- 使用字节流拷贝一张图片

分析



一次读写一个字节拷贝文件思路:

1.创建字节输入流对象,关联数据源文件路径
2.创建字节输出流对象,关联目的地文件路径
3.定义一个变量,用来存储读取到的字节数据
4.循环读取
5.在循环中,写出数据
6.关闭流,释放资源

一次读写一个字节数组拷贝文件

1.创建字节输入流对象,关联数据源文件路径
2.创建字节输出流对象,关联目的地文件路径
3.定义一个字节数组,用来存储读取到的字节数据
3.定义一个变量,用来存储读取到的字节个数
4.循环读取
5.在循环中,写出数据
6.关闭流,释放资源

实现

复制图片文件，代码使用演示:

```
public class Test {
    public static void main(String[] args) throws Exception{
        // 需求:使用字节流拷贝一张图片
        // IO流核心代码: 6步
        // 案例2:一次读写一个字节数组拷贝文件
        // 创建字节输入流对象,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day12\\aaa\\hb.jpg");
        // 创建字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day12\\ccc\\hb1.jpg");
```

```
        // 定义字节数组,用来存储读取到的字节数据
        byte[] bys = new byte[8192];
        // 定义int变量,用来存储读取到的字节个数
        int len;
        // 循环读取数据
        while ((len = fis.read(bys)) != -1) {
            fos.write(bys,0,len);
        }
        // 关闭流释放资源
        fos.close();
        fis.close();

        // 案例1:一次读写一个字节拷贝文件
        /*// 创建字节输入流对象,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day12\\aaa\\hb.jpg");
        // 创建字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day12\\ccc\\hb.jpg");
        // 定义变量,用来存储读取到的字节数据
        int len;
        // 循环读取数据
        while ((len = fis.read()) != -1) {
            // 写出数据
            fos.write(len);
        }
        // 关闭流释放资源
        fos.close();
        fis.close();*/
    }
}
```

> 小贴士：
>
> 流的关闭原则：先开后关，后开先关。

小结

略

第五章 字符流

当使用字节流读取文本文件时，可能会有一个小问题。就是遇到中文字符时，可能不会显示完整的字符，那是因为一个中文字符可能占用多个字节存储。所以Java提供一些字符流类，以字符为单位读写数据，专门用于处理文本文件。

知识点-- 字符输入流【Reader】

目标

- 字符输入流Reader类的概述和常用方法

目标

- 字符输入流Reader类的概述
- 字符输入流Reader类的常用方法

讲解

字符输入流Reader类的概述

`java.io.Reader`抽象类是表示用于读取字符流的所有类的超类，可以读取字符信息到内存中。它定义了字符输入流的基本共性功能方法。

字符输入流Reader类的常用方法

- `public void close()`：关闭此流并释放与此流相关联的任何系统资源。
- `public int read()`：从输入流读取一个字符。
- `public int read(char[] cbuf)`：从输入流中读取一些字符，并将它们存储到字符数组 cbuf 中。

小结

略

知识点-- FileReader类

目标

- 掌握FileReader类的使用

路径

- FileReader类的概述
- FileReader类的构造方法
- FileReader类读取数据

讲解

FileReader类的概述

`java.io.FileReader`类是读取字符文件的便利类。构造时使用系统默认的字符编码和默认字节缓冲区。

FileReader类的构造方法

- `FileReader(File file)`：创建一个新的 FileReader，给定要读取的File对象。
- `FileReader(String fileName)`：创建一个新的 FileReader，给定要读取的文件的名称。

当你创建一个流对象时，必须传入一个文件路径。类似于FileInputStream。

- 构造举例，代码如下：

```
public class FileReaderConstructor throws IOException{
    public static void main(String[] args) {
        FileReader fr1 = new FileReader("day12\\ddd\\a.txt");
        FileReader fr2 = new FileReader(new File("day12\\ddd\\a.txt"));
    }
}
```

FileReader类读取数据

1. **读取字符**：`read`方法，每次可以读取一个字符的数据，提升为int类型，读取到文件末尾，返回`-1`，循环读取，代码使用演示：

```
public class FRRead {
    public static void main(String[] args) throws IOException {
         // 字符流读文件数据:  不会乱码
        // 1.创建字符输入流对象,关联数据源文件路径
        FileReader fr = new FileReader("day12\\ddd\\a.txt");
        // 2.定义一个int类型的变量,用来存储读取到的字符数据
        int len;
        // 3.循环读取
        while ((len = fr.read()) != -1) {
            System.out.println((char)len);
        }
        // 4.关闭流,释放资源
        fr.close();

        System.out.println("==========================================");
        // 字节流读文件数据:  文件中有中文,就会乱码
        // 1.创建字节输入流对象,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day12\\ddd\\a.txt");
        // 定义一个int类型的变量,用来存储读取到的字节数据
        int b;

        // 循环读取
        while ((b = fis.read()) != -1){// 读取一个字节数据,赋值给b,然后拿b与-1进行比较
            System.out.println((char)b);
        }

        // 3.关闭流,释放资源
        fis.close();

    }
}
```

> 小贴士：虽然读取了一个字符，但是会自动提升为int类型。

1. **使用字符数组读取**：`read(char[] cbuf)`，每次读取多个字符到数组中，返回读取到的有效字符个数，读取到末尾时，返回`-1`，代码使用演示：

```
public class FRRead {
    public static void main(String[] args) throws IOException {
       // 1.创建字符输入流对象,关联数据源文件路径
        FileReader fr = new FileReader("day12\\ddd\\a.txt");
        // 2.定义一个字符数组,用来存储读取到的字符数据
        char[] chs = new char[3];
        // 3.定义一个int类型的变量,用来存储读取到的字符个数
        int len;
        // 4.循环读取
        while ((len = fr.read(chs)) != -1){
            System.out.println(new String(chs,0,len));
        }
        // 5.关闭流,释放资源
        fr.close();
    }
}
```

小结

略

知识点--字符输出流【Writer】

目标

- 字符输出流Writer类的概述和常用方法

目标

- 字符输出流Writer类的概述
- 字符输出流Writer类的常用方法

讲解

字符输出流Writer类的概述

`java.io.Writer`抽象类是表示用于写出字符流的所有类的超类，将指定的字符信息写出到目的地。它定义了字节输出流的基本共性功能方法。

字符输出流Writer类的常用方法

- `public abstract void close()`： 关闭此输出流并释放与此流相关联的任何系统资源。
- `public abstract void flush()`： 刷新此输出流并强制任何缓冲的输出字符被写出。
- `public void write(int c)`： 写出一个字符。
- `public void write(char[] cbuf)`： 将 b.length字符从指定的字符数组写出此输出流。
- `public abstract void write(char[] b, int off, int len)`： 从指定的字符数组写出 len 字符，从偏移量 off开始输出到此输出流。
- `public void write(String str)`： 写出一个字符串。
- `public void write(String str,int off,int len)`： 写出一个字符串的一部分。

小结

略

知识点--FileWriter类

目标

- 掌握FileWriter类的使用

路径

- FileWriter类的概述
- FileWriter类的构造方法
- FileWriter类写出数据
- 关闭和刷新

讲解

FileWriter类的概述

`java.io.FileWriter`类是写出字符到文件的便利类。构造时使用系统默认的字符编码和默认字节缓冲区。

FileWriter类的构造方法

- `FileWriter(File file)`： 创建一个新的 FileWriter，给定要读取的File对象。
- `FileWriter(String fileName)`： 创建一个新的 FileWriter，给定要读取的文件的名称。
- `FileWriter(File file,boolean append)`： 创建一个新的 FileWriter，给定要读取的File对象。
- `FileWriter(String fileName,boolean append)`： 创建一个新的 FileWriter，给定要读取的文件的名称。

当你创建一个流对象时，必须传入一个文件路径，类似于FileOutputStream。

- 构造举例，代码如下：

```
public class FileWriterConstructor {
    public static void main(String[] args) throws IOException {
        //FileWriter fs1 = new FileWriter("day12\\ddd\\b.txt");
        FileWriter fs2 = new FileWriter("day12\\ddd\\b.txt",true);// 追加续写
    }
}
```

FileWriter类写出数据

写出字符：`write(int b)`方法，每次可以写出一个字符数据，代码使用演示：

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
         // 案例1: 一次写出一个字符
        // 1.创建字符输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day12\\ddd\\c.txt");
        // 2.写出数据
        fw.write('a');
        fw.write('b');
        // 3.关闭流,释放资源
        fw.close();
    }
}
```

> 小贴士：
>
> 1. 虽然参数为int类型四个字节，但是只会保留一个字符的信息写出。
> 2. 未调用close方法，数据只是保存到了缓冲区，并未写出到文件中。

1. **写出字符数组**： `write(char[] cbuf)` 和 `write(char[] cbuf, int off, int len)` ，每次可以写出字符数组中的数据，用法类似FileOutputStream，代码使用演示：

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
     // 案例2: 一次写出一个字符数组
        // 1.创建字符输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day12\\ddd\\d.txt");
        // 2.写出数据
```

```
        char[] chs = {'a','a','c'};
        fw.write(chs);

        // 3.关闭流,释放资源
        fw.close();
    }
     private static void method03() throws IOException {
        // 案例3: 一次写出指定范围的字符数组数据
        // 1.创建字符输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day12\\ddd\\e.txt");
        // 2.写出数据
        char[] chs = {'a','a','c','d'};
        fw.write(chs,1,2);

        // 3.关闭流,释放资源
        fw.close();
    }
}
```

1. **写出字符串**：`write(String str)` 和 `write(String str, int off, int len)` ，每次可以写出字符串中的数据，更为方便，代码使用演示：

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
         // 案例5: 一次写出一个字符串数据
        // 1.创建字符输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day12\\ddd\\g.txt");
        // 2.写出数据
        String str = "itheima中国itcast";
        fw.write(str,7,2);

        // 3.关闭流,释放资源
        fw.close();
    }

    private static void method04() throws IOException {
        // 案例4: 一次写出一个字符串数据
        // 1.创建字符输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day12\\ddd\\f.txt");
        // 2.写出数据
        String str = "itheima中国itcast";
        fw.write(str);

        // 3.关闭流,释放资源
        fw.close();
    }
}
```

1. **续写和换行**：操作类似于FileOutputStream。

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
        // 创建字符输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day12\\ddd\\h.txt");
        // 写出数据
        fw.write("戒槟榔\r\n");
```

```
        fw.write("槟榔加烟，法力无边\r\n");
        fw.write("槟榔生吞，道法乾坤\r\n");
        fw.write("槟榔配酒，永垂不朽\r\n");
        fw.write("槟榔加烟又加酒，阎王在向你招手\r\n");
        // 关闭资源
        fw.close();
    }
}
```

> 小贴士：字符流，只能操作文本文件，不能操作图片，视频等非文本文件。
>
> 当我们单纯读或者写文本文件时 使用字符流 其他情况使用字节流

关闭和刷新

因为内置缓冲区的原因，如果不关闭输出流，无法写出字符到文件中。但是关闭的流对象，是无法继续写出数据的。如果我们既想写出数据，又想继续使用流，就需要 `flush` 方法了。

- `flush` ：刷新缓冲区，流对象可以继续使用。
- `close` ：关闭流，释放系统资源。关闭前会刷新缓冲区。

代码使用演示：

```
public class FWWrite {
    public static void main(String[] args) throws IOException {
          // 创建字符输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day12\\ddd\\h.txt");
        // 写出数据
        fw.write("戒槟榔\r\n");
        fw.write("槟榔加烟，法力无边\r\n");
        fw.write("槟榔生吞，道法乾坤\r\n");
        fw.write("槟榔配酒，永垂不朽\r\n");
        fw.write("槟榔加烟又加酒，阎王在向你招手\r\n");

        // 问题: 以上代码无法把数据写入文件中
        // 原因:因为内置缓冲区的原因，如果不关闭输出流或者不刷新缓存区，那么无法写出字符到文件中。
        // 解决: 关闭流,或者刷新缓存区
        //fw.close();
        //fw.flush();

        // close和flush的区别:
        /*fw.close();
        fw.write("好诗");// 报异常,IOException: Stream closed
        fw.close();*/

        fw.flush();
        fw.write("好诗");
        fw.flush();
    }
}
```

> 小贴士：即便是flush方法写出了数据，操作的最后还是要调用close方法，释放系统资源。

小结

略

总结

必须完成作业:针对IO流

1.使用字节流一次读写一个字节拷贝文件

2.使用字节流一次读写一个字节数组拷贝文件

3.使用字符流一次读写一个字符拷贝文件

4.使用字符流一次读写一个字符数组拷贝文件

注意: 字节流可以拷贝一切文件,字符流只能拷贝文本文件

拔高作业: 针对File和递归

5.统计非空文件夹的大小

6.删除非空文件夹

- 能够说出File对象的创建方式
  File(String pathname)
  File(String parent,String child)
  File(File parent,String child)
- 能够使用File类常用方法
 - public String getAbsolutePath() ：返回此File的绝对路径名字符串。 *****
- public String getPath() ：将此File转换为路径名字符串。
- public String getName() ：返回由此File表示的文件或目录的名称。 *****
- public long length() ：返回由此File表示的文件的长度。 不能获取目录的长度。 *****
- public boolean exists() ：此File表示的文件或目录是否实际存在。
- public boolean isDirectory() ：此File表示的是否为目录。 *****
- public boolean isFile() ：此File表示的是否为文件。 *****
- public boolean createNewFile() ：当且仅当具有该名称的文件尚不存在时，创建一个新的空文件。
- public boolean delete() ：删除由此File表示的文件或目录。
- public boolean mkdir() ：创建由此File表示的目录。 *****
- public boolean mkdirs() ：创建由此File表示的目录，包括任何必需但不存在的父目录。 *****
- public String[] list() ：返回一个String数组，存储的是File目录中所有的子文件或子目录的名称。 *****
- public File[] listFiles() ：返回一个File数组，存储的是File目录中所有的子文件或子目录的路径的File对象。 *****

- 能够辨别相对路径和绝对路径
  绝对路径:以盘符开始的完整路径
  相对路径:相对于项目路径而言的路径
- 能够遍历文件夹
  listFiles()\ list() 循环遍历
- 能够解释递归的含义
  方法内部自己调用自己
- 能够使用递归的方式计算5的阶乘

```
public int jieCheng(int n){
    if(n == 1){return 1;}// 出口
   return n*jieCheng(n-1);// 规律
}
```

- 能够说出使用递归会内存溢出隐患的原因
  递归的次数太多,造成方法无法弹栈,导致栈内存溢出
- 能够说出IO流的分类和功能
  字节流:以字节为单位,进行读写数据

字节输入流,字节输出流
字符流: 以字符为单位,进行读写数据
字符输入流,字符输出流

- 能够使用字节输出流写出数据到文件
  write(int b)
  write(byte[] bys)
  write(byte[] bys,int off,int len)
- 能够使用字节输入流读取数据到程序
  int read();返回读取到的字节数据,读到文件末尾返回-1
  int read(byte[] bys);读取到的字节数据存储在参数数组中,返回读取到的字节数个数,读到文件末尾返回-1
- 能够理解读取数据read(byte[])方法的原理
  读取到的字节数据存储在参数数组中,返回读取到的字节数个数,读到文件末尾返回-1
- 能够使用字节流完成文件的复制
  创建输入流对象,关联数据源文件路径
  创建输出流对象,关联目的地文件路径
  定义一个int类型的变量,用来存储读取到的数据
  循环读取数据
  在循环中,写出数据到目的地文件中
  关闭流,释放资源

  创建输入流对象,关联数据源文件路径
  创建输出流对象,关联目的地文件路径
  定义一个数组,用来存储读取到的数据
  定义一个int类型的变量,用来存储读取到的数据个数
  循环读取数据
  在循环中,写出数据到目的地文件中
  关闭流,释放资源
- 能够使用FileWriter写数据的5个方法
  write(int c)
  write(char[] chs)
  write(char[] chs,int off,int len)
  write(String str)
  write(String str,int off,int len)
- 能够说出FileWriter中关闭和刷新方法的区别
  关闭: 关闭流,刷新缓冲区,关闭后流不能再使用
  刷新: 不管流,刷新缓存区,刷新后流还可以再使用
- 能够使用FileWriter写数据实现换行和追加写
  write("\r\n")
  FileWriter(String path,boolean append)
- 能够使用FileReader读数据一次一个字符
  int read()
- 能够使用FileReader读数据一次一个字符数组
  int read(char[] chs)