一、File

1 File类概述

- 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

2 获取文件及其属性

```
length(): 返回文件的长度, 单位是字节数(如果File是目录, 则返回0)
exists(): 判断当前文件或目录是否存在, 存在则返回true
isFile(): 判断当前file是否为文件, 是文件则返回true
isDirectory(): 判断当前file是否为目录, 是目录返回true
getName(): 获取当前File文件或目录的名字
getParent(): 获取当前File父目录的路径
getAbsolutePath(): 获取当前File文件或目录的完整路径
```

- **代码案例**

```java
package cn.tedu.file;

import java.io.File;

/**
 * 这个案例是学习File绑定指定目标文件
 */
public class FileDemo {
    public static void main(String[] args) {
        //访问当前项目下的demo目录中的demo.txt文件
        /*
         * 路径一般分为两种,绝对路径和相对路径
         * ①绝对路径: 从根目录一直定位到目标文件的路径 绝对路径一般不用
         * 选中指定文件,右键→Copy Path→Absolute Path,复制绝对路径
         * D:\Dates\IDEASPACE\JSDVN2302SE-TEACHER\demo\demo.txt
         * 在java中"\"(反斜杠)是特殊字符,是转义字符
         * ②相对路径 相对于特殊节点所在位置的路径
         * 在idea中使用"./"表示当前项目的目录
         * 在本项目中,"./"就表示"D:\Dates\IDEASPACE\JSDVN2302SE-TEACHER\"
         */
        File file = new File("./demo/demo.txt");
        //获取文件名字(绑定的字符串的名字)
        //变量名.soutv 快速生成输出语句
        String name = file.getName();
        System.out.println("name = " + name);
        //获取文件的长度(字节) 一个因为字母是1个字节 一个中文是3个字节
        long length = file.length();
        System.out.println("length = " + length+"个字节");
    }
}
```

3 创建文件

```
createNewFile(): 创建指定路径和名称的文件, 如果文件不存在, 则创建并返回true, 否则就不创建并返回false
```

- 代码案例

```java
package cn.tedu.file;

import java.io.File;
import java.io.IOException;

/**
 * 这个案例是学习使用File创建文件
 */
public class CreateNewFileDemo {
    public static void main(String[] args) throws IOException {
        //idea中,相对路径中"./"可以不写,会默认识别
        //File file = new File("demo/new.txt");
        File file = new File("./demo/new.txt");
        /*
         * boolean exists()
         * 判断调用的File实例是否存在,存在返回true,不存在返回false
         * File可能是目录也可能是文件
         */
        if (file.exists()) {
            System.out.println("文件已存在!");
        } else {
            //方法报红线错误,按alt+enter(回车),然后直接再按回车
            file.createNewFile();//create 创建 new 新的 file 文件
            System.out.println("该文件创建完毕!!!");
        }
    }
}
```

4 删除文件

```
delete(): 删除文件或删除空目录, 删除成功返回true(非空目录删除会失败)
```

- 代码案例

```java
package cn.tedu.file;

import java.io.File;
import java.io.IOException;

/**
 * 这个案例是学习使用File删除文件
 */
public class DeleteFileDemo {
    public static void main(String[] args) throws IOException {
        //idea中,相对路径中"./"可以不写,会默认识别
        File file = new File("./demo/new.txt");
        if (file.exists()) {
            file.delete(); //delete删除
            System.out.println("文件删除成功!!!");
        } else {
            System.out.println("文件不存在!!!不可删除!!!");
        }
    }
}
```

5 创建目录

```
mkdir(): 创建指定路径和名称的目录，如果目录不存在，则创建并返回true，否则就不创建并返回false
mkdirs(): 创建指定路径和名称的多级目录，如果目录不存在，则创建并返回true，否则就不创建并返回false
```

- 代码案例

```java
package cn.tedu.file;

import java.io.File;

/**
 * 此案例学习使用File创建目录
 */
public class MKDirDemo {
    public static void main(String[] args) {
        //目录是没有后缀名
        File dir = new File("./demo/h/e/l/l/o");
        if (dir.exists()) {
            System.out.println("该目录已存在!");
        } else {
            /*
             * mkdir() 创建目录时要求该目录所在的目录必须存在,否则无法创建
             * mkdirs() 创建目录时,会将不存在的目录以一同创建出来(推荐)
             */
            dir.mkdirs(); //make 制作 dir目录
            System.out.println("目录制作完毕!!");
        }
    }
}
```

6 删除目录

```
delete(): 删除文件或删除空目录，删除成功返回true(非空目录删除会失败)
```

- 代码案例

```java
package cn.tedu.file;

import java.io.File;

/**
 * 此案例学习使用File删除目录
 */
public class DeleteDirDemo {
    public static void main(String[] args) {
        //目录是没有后缀名
        File dir = new File("./demo/h/e/l/l/o");
        if (dir.exists()) {
            /*
             * delete方法,只能删除一个文件或者目录
             * 原因是delete不能删除非空目录
             */
            dir.delete();
            System.out.println("删除目录成功!");
        } else {
            System.out.println("目录不存在!!删除失败!!!");
        }
    }
```

```
}
```

7 获取目录中的子项

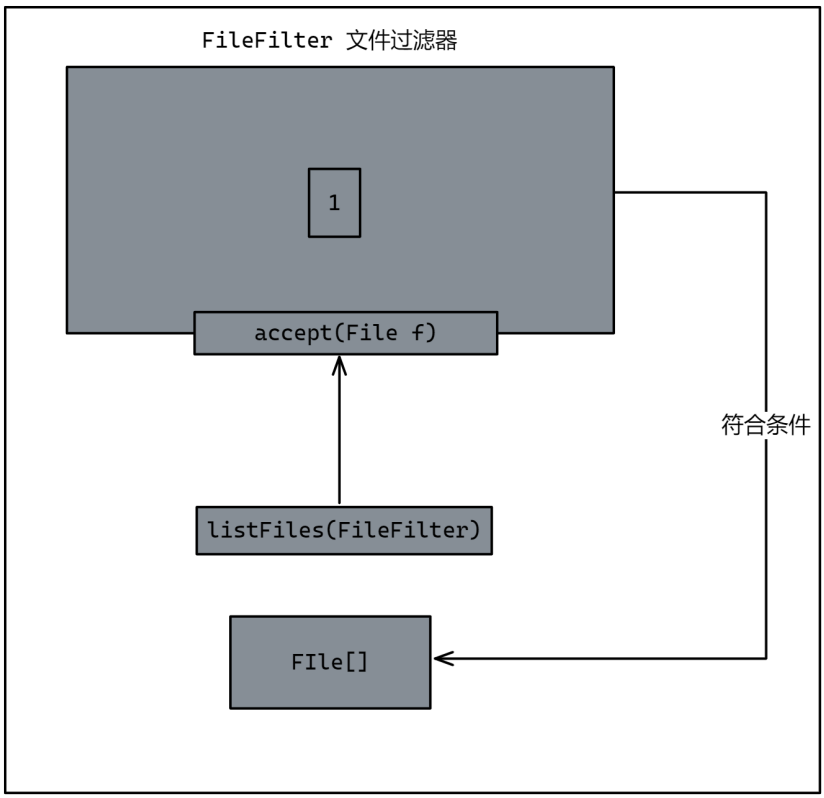
- 代码案例

```
package cn.tedu.file;

import java.io.File;

/**
 * 此案例学习使用File获取指定目录下的一层子项
 */
public class ListFilesDemo {
    public static void main(String[] args) {
        File dir = new File("./demo");
        /*
         * boolean isDirectory()
         * 判断当前调用的File是否表示的是一个目录,是则返回true,不是返回false
         */
        if (dir.isDirectory()) {
            /*
             * File[] listFiles()
             * 将当前调用的File表示的目录中的一层子项,各自实例化为File,并存储到File数组中
             */
            File[] subs = dir.listFiles();
            System.out.println("demo目录下,有" + subs.length + "个子项!");
            //subs.fori 自动生成根据下标遍历数组的代码结构
            for (int i = 0; i < subs.length; i++) {
                System.out.println(subs[i].getName());
            }
        } else {
            System.out.println("是文件!!!!");
        }
    }
}
```

8 获取目录中的符合过滤条件子项



- **代码案例**

```java
package cn.tedu.file;

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

/**
 * 通过该案例学习通过File的listFiles方法获取符合过滤条件的目录中的所有子项
 * 获取一个目录中的符合过滤条件的子项
 * 重载的lsitFiles方法
 * File[] listFiles(FileFilter filter)
 * 该方法要求传入一个文件过滤器(FileFilter),然后该方法会自动根据文件过滤器的要求,将符合条件的子项返回
 */
public class ListFilesDemo2 {
    public static void main(String[] args) {
        //指定当前项目的目录,只需要写一个"."就可以
        File dir = new File(".");
        //创建文件过滤器实例,定义过滤规则(一般会使用匿名内部类创建,并且重写accept方法)
        FileFilter fileFilter = new FileFilter() {
            /**
             * accept方法就是用于定义过滤规则
             * @param file 默认的,表示要过滤的一个文件
             * @return 布尔值, 如果返回true, 表示当前过滤的文件符合条件, 反之则不符合
             */
            @Override
            public boolean accept(File file) {
                //定义过滤规则(要求获取子项名字中包含"o")
                //获取过滤的文件的名字
                String fileName = file.getName();
                //contains判断是否包含,包含就返回true,不包含返回false
                return fileName.contains("o");
            }
        };
        //将文件过滤器传入到listFiles方法中
```

```
        File[] subs = dir.listFiles(fileFilter);
        for (int i = 0; i < subs.length; i++) {
            System.out.println(subs[i].getName());
        }
    }
}
```

9 递归遍历目录

递归（recursion）是一种常见的解决问题的方法，即把问题逐渐简单化。

递归的基本思想就是“自己调用自己”，一个使用递归技术的方法将会直接或者间接的调用自己。

```
m();
...
public void m(){
  ...
  m();
  ...
}
```

需要注意的是: 递归方法一定要有出口, 否则将会一直自己调用自己, 变成死循环, 严重时将会导致内存溢出!

9.1 需求:

- 遍历指定File(目录)下的所有子目录和子文件, 输出该目录下的所有目录和文件名

```
思路: 声明一个diGui目录的方法, 接收一个File类型的对象, 方法内部实现如下:
1.判断当前File是否为文件(防止第一次传入的是文件)
  1.1.如果file是文件, 输出: "文件不支持递归!"
  1.2.如果file是目录, 则继续执行第2步
2.获取当前目录下的所有子目录及子文件对象组成的File数组
3.遍历当前目录下的所有子目录及子文件对象
4.判断当前遍历的是目录还是文件
  4.1.如果当前遍历的是文件, 输出该文件的路径+名称
  4.2.如果当前遍历的是目录, 输出当前目录的路径+名称
  并以此目录作为根, 接着遍历该目录下的所有子目录和子文件, 输出该目录下的所有目录和文件名
```

9.2 统计一个目录下所有文件的大小之和

- **代码案例**

```
package cn.tedu.file;

import java.io.File;

/**
 * 此案例使用递归(recursion),借助listFiles实现获取指定目录中的所有子项
 */
public class ListDirDemo {
    public static void main(String[] args) {
        File dir = new File("./demo");
        //调用recursionDir,递归遍历上面的目录
        recursionDir(dir);
    }

    /**
     * 递归遍历指定的目录
     *
     * @param dir 要遍历的目录
```

```
     */
    private static void recursionDir(File dir) {
        //1.判断当前File是否为文件(防止第一次传入的是文件)
        /*
         * boolean isFile()
         * 判断当前调用者是否是一个文件,是则返回true,不是返回false
         */
        if (dir.isFile()) {
            //1.1.如果file是文件, 输出: "文件不支持递归!"
            System.out.println("文件是不支持递归的!!!");
            //return在此处直接结束当前方法
            return;
        } else {
            //1.2.如果file是目录, 则继续执行第2步
            //2.获取当前目录下的所有子目录及子文件对象组成的File数组
            File[] subs = dir.listFiles();
            //3.遍历当前目录下的所有子目录及子文件对象
            for (int i = 0; i < subs.length; i++) {
                //4.判断当前遍历的是目录还是文件
                if (subs[i].isFile()) {
                    //4.1.如果当前遍历的是文件, 输出该文件的路径+名称
                    System.out.println("文件:" + subs[i]);
                } else {
                    //4.2.如果当前遍历的是目录, 输出当前目录的路径+名称
                    System.out.println("目录:" + subs[i]);
                    //并以此目录为根,再次执行recursionDir方法的逻辑,遍历该目录的子
目录和子文件
                    recursionDir(subs[i]);
                }
            }
        }
    }
}
```

9.3 删除一个目录及其中所有子项

- 代码案例

```
package cn.tedu.file;

import java.io.File;

/**
 * 此案例使用递归(recursion),删除指定目录及其中所有子项
 */
public class DeleteDirDGDemo {
    public static void main(String[] args) {
        File dir = new File("./demo");
        recursionDeleteDir(dir);
    }

    /**
     * 递归遍历删除指定的目录
     *
     * @param dir 要遍历删除的目录
     */
    private static void recursionDeleteDir(File dir) {
        //1.判断当前File是否为文件(防止第一次传入的是文件)
        /*
         * boolean isFile()
         * 判断当前调用者是否是一个文件,是则返回true,不是返回false
         */
```

```
        if (dir.isFile()) {
            //1.1.如果file是文件, 输出: "文件不支持递归!"
            System.out.println("文件是不支持递归的!!!");
            //return在此处直接结束当前方法
            return;
        } else {
            //1.2.如果file是目录, 则继续执行第2步
            //2.获取当前目录下的所有子目录及子文件对象组成的File数组
            File[] subs = dir.listFiles();
            //3.遍历当前目录下的所有子目录及子文件对象
            for (int i = 0; i < subs.length; i++) {
                //4.判断当前遍历的是目录还是文件
                if (subs[i].isFile()) {
                    //4.1.如果当前遍历的是文件, 则删除并输出该文件的路径+名称
                    System.out.println("文件:" + subs[i]);
                    subs[i].delete();
                } else {
                    //4.2.并以此目录为根,再次执行recursionDir方法的逻辑,遍历该目录的子目录和子文件
                    recursionDeleteDir(subs[i]);
                }
            }
            //5.删除dir目录
            dir.delete();
            System.out.println("成功删除" + dir.getName() + "目录!");
        }
    }
}
```

二、Lambda表达式

- JDK8之后,java支持了lambda表达式这个特性.
- lambda可以用更精简的代码创建匿名内部类.但是该匿名内部类实现的接口只能有一个抽象方法,否则无法使用!
- lambda表达式是编译器认可的,最终会将其改为内部类编译到class文件中
- **代码案例**

```
package cn.tedu.lambda;

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

/**
 * 通过此案例学习Lambda表达式的使用
 * JDK8之后,java支持了lambda表达式这个特性.
 * lambda可以用更精简的代码创建匿名内部类.但是该匿名内部类实现的接口只能有一个抽象方法,否则无法使用!
 * 语法:
 * (参数列表) ->{
 * 方法体
 * }
 */
public class LambdaDemo {
    public static void main(String[] args) {
        //①不使用λ表达式的匿名内部类写法
        FileFilter f1 = new FileFilter() {
            @Override
            public boolean accept(File file) {
                return file.getName().contains("o");
```

```
            }
        };
        //②使用λ表达式忽略接口名和方法名
        /*
         * 1) 将new FileFilter() {}删除
         * 2) public boolean accept
         * 3) 在方法参数和方法体之间添加'->'
         */
        FileFilter f2 = (File file) -> {
            return file.getName().contains("o");
        };
        //③使用λ表达式忽略参数类型
        FileFilter f3 = (file) -> {
            return file.getName().contains("o");
        };
        //④如果要重写的方法中只有一个形参时,那么参数的'()'也可以省略
        FileFilter f4 = file -> {
            return file.getName().contains("o");
        };
        //⑤如果方法体中只有一句代码,那么可以将方法体的'{}'省略,如果代码包含
    return,return也需要一同省略
        FileFilter f5 = file -> file.getName().contains("o");

        File dir = new File(".");
        File[] subs = dir.listFiles(file -> file.getName().contains("o"));
        for (int i = 0; i < subs.length; i++) {
            System.out.println(subs[i].getName());
        }
    }
}
```