

# FileDemo

```
package file;

import java.io.File;

/**
 * io:input output
 *      输入    输出
 *
 * java.io.File:
 * File的每一个实例用于表示硬盘上的一个文件或目录。实际上表示的是一个抽象路径。
 *
 * File可以:
 * 1: 访问其表示的文件或目录的属性信息(名字, 大小, 修改时间等)
 * 2: 操作(创建, 删除)文件或目录
 * 3: 访问一个目录中的所有子项
 *
 * 但是File不能访问文件数据!
 */
public class FileDemo {
    public static void main(String[] args) {
        /**
         * 创建File时常用的构造方法:File(String path)
         * path指定该文件的路径。
         * 路径推荐用相对路径, 有更好的跨平台性。
         * 其中常用的是"./"即当前目录。
         * 在idea执行当前程序时, "./"表示的位置就是当前程序所在的项目目录:如JSD2205_SE这个
         * 目录
         */
        File file = new File("./demo.txt");

        /**
         * File提供了一组方法可以获取其表示的文件或目录的属性信息
         * String getName()
         * 返回当前文件或目录的名字

         * long length()
         * 返回当前文件的大小, 单位是字节。

         * boolean canRead()
         * 返回一个boolean表达该文件是否可读

         * boolean canWrite()
         * 返回一个boolean表达该文件是否可写
         */
        String name = file.getName();
        System.out.println("名字:" + name);

        long length = file.length(); //注意: 是long类型!
        System.out.println(length + "字节"); //文件不存在返回0字节

        boolean cr = file.canRead();
        System.out.println("是否可读:" + cr);
    }
}
```

```
        boolean cw = file.canWrite();  
        System.out.println("是否可写:" + cw);  
    }  
}
```

# CreateNewFileDemo

```
package file;

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

/**
 * create:创建
 * new:新的
 * file:文件
 * exists:存在
 * 使用File新建一个空文件
 */
public class CreateNewFileDemo {
    public static void main(String[] args) throws IOException {
        //在当前目录下新建一个文件:test.txt
        File file = new File("./test.txt");

        /**
         * File的方法:
         * boolean exists()
         * 返回当前File对象表示的文件或目录是否真实存在, true:存在, false:不存在
         *
         * boolean createNewFile()
         * 将当前File对象表示的文件在硬盘该位置上实际创建出来
         */
        if (file.exists()) {
            System.out.println("该文件已存在!");
        } else {
            file.createNewFile();
            System.out.println("该文件已创建!");
        }
    }
}
```

# Test1

---

```
package file;

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

/**
 * 在当前目录下创建100个文件，名字为test1.txt - test100.txt
 */
public class Test1 {
    public static void main(String[] args) throws IOException {
        for (int i = 1; i <= 100; i++) {
            File file = new File("./test" + i + ".txt");
            if (!file.exists()) {
                file.createNewFile();
                System.out.println(file.getName() + " 创建成功");
            } else {
                System.out.println(file.getName() + " 已存在");
            }
        }
        System.out.println("创建完毕!");
    }
}
```

# DeleteFileDemo

```
package file;

import java.io.File;

/**
 * delete:删除
 * 删除一个文件
 */
public class DeleteFileDemo {
    public static void main(String[] args) {
        //将当前目录下的test.txt文件删除
        //相对路径中最开始的"./"可以忽略不写，默认就是从"./"开始的
        File file = new File("./test.txt");
        if (file.exists()) {
            file.delete(); //彻底删除，不放回收站
            System.out.println("该文件已删除!");
        } else {
            System.out.println("该文件不存在!");
        }

        //删除当前目录中的test1.txt - test100.txt
        for (int i = 1; i <= 100; i++) {
            File f = new File("./test" + i + ".txt");
            if (f.exists()) {
                f.delete();
                System.out.println("该文件已删除!");
            } else {
                System.out.println("该文件不存在!");
            }
        }
        System.out.println("删除完毕!");
    }
}
```

# MkdirDemo

```
package file;

import java.io.File;

/**
 * make: 做
 * directory: 目录(windows下习惯叫作"文件夹")
 *
 * 创建一个空的目录
 */
public class MkdirDemo {
    public static void main(String[] args) {
        //在当前目录下新建一个目录demo
        //File dir = new File("./demo");
        //创建f目录，以下是f目录所在的结构：
        File dir = new File("./a/b/c/d/e/f");

        /**
         * File常用方法：
         * boolean mkdir()
         * 将当前File表示的目录在硬盘上实际创建出来。
         *
         * boolean mkdirs()
         * 将当前File表示的目录在硬盘上实际创建出来。
         *
         * mkdir和mkdirs最大的区别是：
         * mkdir方法在创建目录时要求该目录所在的目录必须存在，否则会创建失败。
         * 而mkdirs则会将所有不存在的父目录一同创建出来。
         * 因此实际开发中绝大多数都是直接使用mkdirs这个方法创建目录。
         */
        if (dir.exists()) {
            System.out.println("该目录已存在!");
        } else {
            dir.mkdirs();
            System.out.println("该目录已创建!");
        }
    }
}
```

# DeleteDirDemo

---

```
package file;

import java.io.File;

/**
 * 删除一个目录
 */
public class DeleteDirDemo {
    public static void main(String[] args) {
        File dir = new File("./demo");
        if (dir.exists()) {
            //delete在删除目录时要求该目录必须是一个【空目录】
            dir.delete();
            System.out.println("目录已删除");
        } else {
            System.out.println("目录不存在");
        }
    }
}
```

# ListFilesDemo

```
package file;

import java.io.File;

/**
 * 获取一个目录中的所有子项
 */
public class ListFilesDemo {
    public static void main(String[] args) {
        //获取当前项目目录中的所有子项
        File dir = new File("./");

        /**
         list:列表

         File常用方法:
         boolean isFile()
         判断当前File对象表示的是否为一个文件

         boolean isDirectory()
         判断当前File对象表示的是否为一个目录

         File[] listFiles()
         获取当前File对象表示的目录中的所有子项, 包括文件和目录。
         返回的数组中每一个元素(每一个File对象)表示该目录中的一个子项。
         */
        if (dir.isDirectory()) { //布尔值表达式, 直接写
            File[] subs = dir.listFiles();
            System.out.println("一共有:" + subs.length + "个子项");
            for (int i = 0; i < subs.length; i++) {
                File sub = subs[i];
                System.out.println(sub.getName());
            }
        }
    }
}
```



# Test2

---

```
package file;

import java.io.File;

/**
 * 向控制台打桩输出file包中所有子项的名字和占用的空间大小
 * 提示：
 * ./src/file
 * File类的getName()和length()方法
 */
public class Test2 {
    public static void main(String[] args) {
        File dir = new File("./src/file");
        if (dir.isDirectory()) {
            File[] subs = dir.listFiles();
            for (int i = 0; i < subs.length; i++) {
                File sub = subs[i];
                System.out.println(sub.getName() + ":" + sub.length() + "字节");
            }
        }
    }
}
```

# ListFilesDemo2

```
package file;

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

/**
 * 有条件的获取一个目录中的部分子项
 */
public class ListFilesDemo2 {
    public static void main(String[] args) {
        /**
         * 获取file目录中名字含有字母"s"的所有子项
         *
         * Filter:过滤器
         * accept:接受
         * contains:包含
         */
        FileFilter filter = new FileFilter() {
            //定义是否接受参数file所表示元素的条件
            public boolean accept(File file) {
                String name = file.getName();
                //return name.indexOf("s")>=0;
                //indexOf()返回第一次出现的位置,范围0到数组长度-1,若没出现则返回-1
                return name.contains("s");
                //String的contains方法用于判断当前字符串是否包含给定内容
            }
        };

        File dir = new File("./src/file");
        if (dir.isDirectory()) {
            /**
             * File重写的:File[] listFiles(FileFilter filter)
             * 该方法返回当前File对象表示的目录中所有满足给定过滤器要求的子项。
             * listFiles会将这个目录中每个子项所对应的File对象逐一传给过滤器filter重写的
             accept方法,
             并将accept方法返回true的所有子项返回。
             */
            File[] subs = dir.listFiles(filter);
            System.out.println("共:" + subs.length + "个子项");
            for (int i = 0; i < subs.length; i++) {
                System.out.println(subs[i].getName());
            }
        }
    }
}
```

# Test3

```
package file;

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

/**
 * 获取file包中大小超过1000字节的所有子项
 * 输出这些子项的名字和实际占用的字节数
 */
public class Test3 {
    public static void main(String[] args) {
        File dir = new File("./src/file");
        if (dir.isDirectory()) {
            File[] subs = dir.listFiles(new FileFilter() {
                public boolean accept(File file) {
                    return file.length() > 1000;
                }
            });
            for (int i = 0; i < subs.length; i++) {
                File sub = subs[i];
                System.out.println(sub.getName() + ":" + sub.length() + "字节");
            }
        }
    }
}
```