day01

File类

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

但是File不能访问文件数据.

```
public class FileDemo {
    public static void main(String[] args) {
        //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。
            "./"是相对路径中使用最多的，表示"当前目录"，而当前目录是哪里
            取决于程序运行环境而定，在idea中运行java程序时，这里指定的
            当前目录就是当前程序所在的项目目录。
         */
//        File file = new File("c:/xxx/xxx/xx/xxx.txt");
        File file = new File("./demo.txt");
        //获取名字
        String name = file.getName();
        System.out.println(name);
        //获取文件大小(单位是字节)
        long len = file.length();
        System.out.println(len+"字节");
        //是否可读可写
        boolean cr = file.canRead();
        boolean cw = file.canWrite();
        System.out.println("是否可读:"+cr);
        System.out.println("是否可写:"+cw);
        //是否隐藏
        boolean ih = file.isHidden();
        System.out.println("是否隐藏:"+ih);

    }

}
```

创建一个新文件

createNewFile()方法，可以创建一个新文件

```
package file;

import java.io.File;
```

```
import java.io.IOException;

/**
 * 使用File创建一个新文件
 */
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");
        //boolean exists()判断当前File表示的位置是否已经实际存在该文件或目录
        if(file.exists()){
            System.out.println("该文件已存在!");
        }else{
            file.createNewFile();//将File表示的文件创建出来
            System.out.println("文件已创建!");
        }

    }
}
```

删除一个文件

delete()方法可以将File表示的文件删除

```
package file;

import java.io.File;

/**
 * 使用File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"./"可以忽略不写，默认就是从当前目录开始的。
         */
        File file = new File("test.txt");
        if(file.exists()){
            file.delete();
            System.out.println("文件已删除!");
        }else{
            System.out.println("文件不存在!");
        }
    }
}
```

创建目录

mkDir():创建当前File表示的目录

mkDirs():创建当前File表示的目录，同时将所有不存在的父目录一同创建

```
package file;
```

```
import java.io.File;

/**
 * 使用File创建目录
 */
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");
        File dir = new File("./a/b/c/d/e/f");

        if(dir.exists()){
            System.out.println("该目录已存在!");
        }else{
//            dir.mkdir();//创建目录时要求所在的目录必须存在
            dir.mkdirs();//创建目录时会将路径上所有不存在的目录一同创建
            System.out.println("目录已创建!");
        }
    }
}
```

删除目录

delete()方法可以删除一个目录，但是只能删除空目录。

```
package file;

import java.io.File;

/**
 * 删除一个目录
 */
public class DeleteDirDemo {
    public static void main(String[] args) {
        //将当前目录下的demo目录删除
        File dir = new File("demo");
//        File dir = new File("a");
        if(dir.exists()){
            dir.delete();//delete方法删除目录时只能删除空目录
            System.out.println("目录已删除!");
        }else{
            System.out.println("目录不存在!");
        }
    }
}
```

访问一个目录中的所有子项

listFiles方法可以访问一个目录中的所有子项

```
package file;

import java.io.File;

/**
```

```
 * 访问一个目录中的所有子项
 */
public class ListFilesDemo1 {
    public static void main(String[] args) {
        //获取当前目录中的所有子项
        File dir = new File(".");
        /*
            boolean isFile()
            判断当前File表示的是否为一个文件
            boolean isDirectory()
            判断当前File表示的是否为一个目录
         */
        if(dir.isDirectory()){
            /*
                File[] listFiles()
                将当前目录中的所有子项返回。返回的数组中每个File实例表示其中的一个子项
             */
            File[] subs = dir.listFiles();
            System.out.println("当前目录包含"+subs.length+"个子项");
            for(int i=0;i<subs.length;i++){
                File sub = subs[i];
                System.out.println(sub.getName());
            }
        }
    }
}
```

获取目录中符合特定条件的子项

重载的listFiles方法:File[] listFiles(FileFilter)

该方法要求传入一个文件过滤器，并仅将满足该过滤器要求的子项返回。

```
package file;

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

/**
 * 重载的listFiles方法，允许我们传入一个文件过滤器从而可以有条件的获取一个目录
 * 中的子项。
 */
public class ListFilesDemo2 {
    public static void main(String[] args) {
        /*
            需求:获取当前目录中所有名字以"."开始的子项
         */
        File dir = new File(".");
        if(dir.isDirectory()){
//            FileFilter filter = new FileFilter(){//匿名内部类创建过滤器
//                public boolean accept(File file) {
//                    String name = file.getName();
//                    boolean starts = name.startsWith(".");//名字是否以"."开始
//                    System.out.println("过滤器过滤:"+name+",是否符合要
求:"+starts);
//                    return starts;
```

```
//                }
//            };
//            File[] subs = dir.listFiles(filter);//方法内部会调用accept方法


            File[] subs = dir.listFiles(new FileFilter(){
                public boolean accept(File file) {
                    return file.getName().startsWith(".");
                }
            });
            System.out.println(subs.length);
        }
    }
}
```

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

```
package lambda;

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

/**
 * JDK8之后java支持了lambda表达式这个特性
 * lambda表达式可以用更精简的语法创建匿名内部类，但是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，最终会被改为内部类形式编译到class文件中。
 *
 * 语法:
 * (参数列表)->{
 *     方法体
 * }
 */
public class LambdaDemo {
    public static void main(String[] args) {
        //匿名内部类形式创建FileFilter
        FileFilter filter = new FileFilter() {
            public boolean accept(File file) {
                return file.getName().startsWith(".");
            }
        };

        FileFilter filter2 = (File file)->{
            return file.getName().startsWith(".");
        };

        //lambda表达式中参数的类型可以忽略不写
```

```
        FileFilter filter3 = (file)->{
            return file.getName().startsWith(".");
        };

        /*
            lambda表达式方法体中若只有一句代码，则{}可以省略
            如果这句话有return关键字，那么return也要一并省略!
         */
        FileFilter filter4 = (file)->file.getName().startsWith(".");
    }
}
```

总结:

File类

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

常用构造器:

- File(String pathname)
- File(File parent,String name)可参考文档了解

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。
- 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