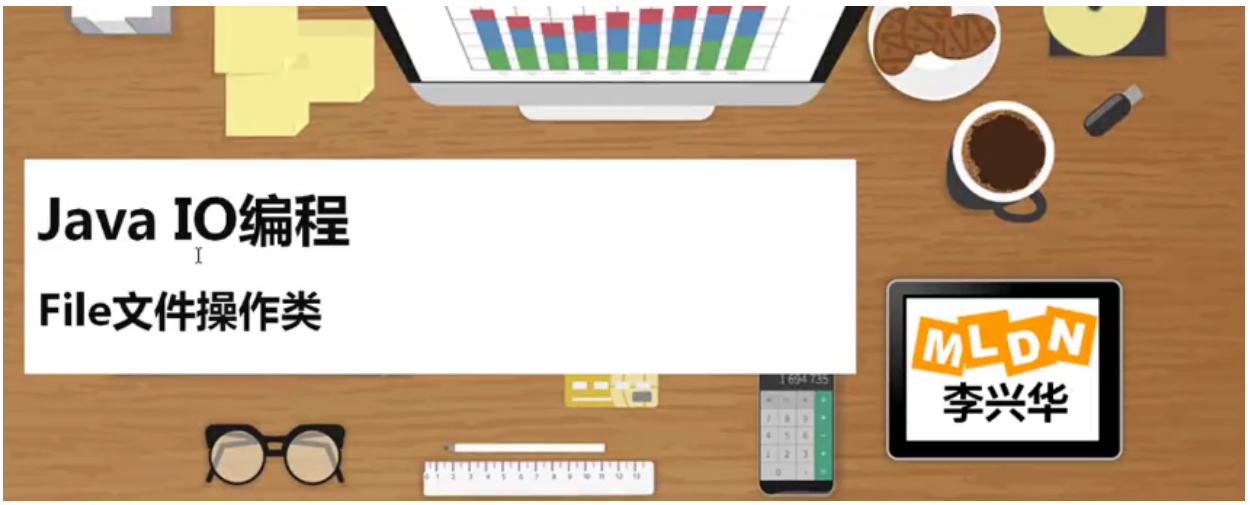

博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

在Java语言里面提供有对于文件操作系统操作的支持，而这个支持就在java.io.File类中进行了定义，也就是说在整个java.io包里面，File类是唯一 一个与文件本身操作（创建、删除、重命名等）有关的类，而如果要想进行File类的操作，必须要提供有完整的路径，而后可以调用相应的方法进行处理。

■File类的基本使用

打开JDK文档可以发现，File类时Comparable接口的子类，所以File类的对象是可以进行排序处理的。而在F进行ile类处理的时候需要为其设置访问路径，那么对于路径的配置主要通过File类的构造方法处理：

·**构造方法**：public File(String pathname)，设置要操作完整路径；

·**构造方法**：public File(File parent,String child)，设置父路径与子目录；

如果现在要想进行文件的基本操作，可以使用如下的方法：

·**创建新文件**：public **boolean** createNewFile()throws **IOException**；

·**判断文件是否存在**：public boolean exists()；

·**删除文件**：public boolean delete()；

范例：使用File类创建一个文件（d:\mldn.txt）

```
import java.io.File;

public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		File file = new File("d:\\mldn.txt");
		if (file.exists()) {
			file.delete();// 删除文件
		} else { // 文件不存在
			System.out.println(file.createNewFile()); // 创建新的文件
```

```
        }
    }
}
```

通过代码可以发现，File类实现的就是文件本身的处理。

■File类操作深入

现在已经实现了文件的基础操作，但是对于这个操作里面也是存在有一些问题的，下面针对于之前的代码进行优化处理。

1、在实际的软件项目开发和运行的过程之中，往往都会在Windows中进行项目的开发，而在项目部署的时候基于Linux或Unix系统来进行项目发布以保证生产环节的安全性；

在不同的操作系统之中会存在有不同的路径分隔符：Windows分隔符“\”、Linux分隔符"/"，所以在最初进行开发的时候就必须考虑不同系统环境下的分隔符的问题，所以为了解决此问题，File类提供有一个常量：**public static final String separator**；

范例：正常的路径编写

```
File file = new File("d:" + File.separator + "mldn.txt");
```

但是随着系统的适应性的不断加强，对于当前的路径操作，也可以随意使用了。

```
File file = new File("d:/mldn.txt");
```

```
//完整代码
import java.io.File;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "mldn.txt");
        if (file.exists()) {
            file.delete();// 删除文件
        } else { // 文件不存在
            System.out.println(file.createNewFile()); // 创建新的文件
        }
    }
}
```

2、在使用File类进行文件处理的时候需要注意的是：程序 -> JVM -> 操作系统函数 -> 文件处理，所在进行同一文件的反复删除或创建的时候有可能会出现有延迟的问题，所以这个时候最好的方案是别重名；

3、在进行文件创建的时候有一个重要的前提：文件的父路径必须首先存在。

- **如何获取父路径：public File getParentFile();**
- **创建多级目录：public boolean mkdirs();**

```
import java.io.File;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
```

```
            File file = new File("d:" + File.separator + "hello" + File.separator + "demo" +
File.separator + "message"
                        + File.separator + "mldn.txt");
            if (!file.getParentFile().exists()) {      // 父路径不存在
                file.getParentFile().mkdirs() ; // 创建父路径
            }
            if (file.exists()) {
                file.delete();// 删除文件
            } else { // 文件不存在
                System.out.println(file.createNewFile()); // 创建新的文件
            }
        }
}
```

这种判断并且建立父目录的操作在很多的情况下可能只需要一次，但是如果将这个判断一直都停留在代码里面，那么就会造成时间复杂度的提升，所以这个时候如果要想提升性能，请先保证目录以及创建。

■获取文件信息

除了可以进行文件的操作之外也可以通过File类来获取一些文件本身提供的信息，可以获取如下内容：

·文件是否可读：public boolean canRead();

·文件是否可写：public boolean canWrite();

·获取文件长度：public long length()，该方法返回的是long数据类型、字节长度；

·最后一次修改日期时间：public long lastModified();

·判断是否为目录：public boolean isDirectory();

·判断是否为文件：public boolean isFile();

```
//my.jpg可以修改为改盘的任一个文件，须带上后缀名
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;
class MathUtil {
        private MathUtil() {
        }

        public static double round(double num, int scale) {
                return Math.round(Math.pow(10, scale) * num) / Math.pow(10, scale);
        }
}
public class JavaAPIDemo {
        public static void main(String[] args) throws Exception {
                File file = new File("d:" + File.separator + "my.jpg");
                System.out.println("文件是否可读：" + file.canRead());
                System.out.println("文件是否可写：" + file.canWrite());
```

```
            System.out.println("文件大小：" + MathUtil.round(file.length() / (double) 1024 /
1024, 2));
            System.out.println(
                        "最后的修改时间：" + new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date(file.lastModified())));
            System.out.println("是目录吗？" + file.isDirectory());
            System.out.println("是文件吗？" + file.isFile());
      }
}
```

既然可以判断给定的路径是文件还是目录，那么就可以进一步的判断，如果发现是目录，则应该列出目录中的全部内容：

·**列出目录内容**：public File[] listFiles();

```
import java.io.File;
public class JavaAPIDemo {
      public static void main(String[] args) throws Exception {
            File file = new File("d:" + File.separator);
            if (file.isDirectory()) {       // 当前是一个目录
                  File result [] = file.listFiles() ; // 列出目录中的全部内容
                  for (int x = 0 ; x < result.length ; x ++) {
                        System.out.println(result[x]);
                  }
            }
      }
}
```

这些信息的获得都是文件或目录本身的操作，都是不涉及到文件内容处理的。

■File操作案例：列出指定目录中的全部文件

现在可以由开发者任意设置一个目录的路径，而后将这个目录中所有的文件的信息全部列出，包括子目录中的所有文件，在这样的处理情况下最好的做法就是利用递归的形式来完成。

范例：程序实现

```
import java.io.File;
public class JavaAPIDemo {
      public static void main(String[] args) throws Exception {
            File file = new File("D:" + File.separator) ;      // 是一个目录
            long start = System.currentTimeMillis();
            listDir(file) ;
            long end = System.currentTimeMillis();
            System.out.println("时间是："+(end-start));
      }
      public static void listDir(File file) {
            if (file.isDirectory()) {       // 是一个目录
                  File results [] = file.listFiles() ; // 列出目录中的全部内容
                  if (results != null) {
                        for (int x = 0 ; x < results.length ; x ++) {
```

```
                        listDir(results[x]) ; // 继续依次判断
                    }
                }
            }
            System.out.println(file);  // 获得完整路径
        }
}
```

如果现在将路径输出变为删除操作，那么就彻底删除路径。

```
//慎重，小心使用 mldnjava问目录，若该目录不存在，也可以使用改盘的其他文件夹
import java.io.File;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        File file = new File("D:" + File.separator + "mldnjava") ;       // 是一个目录
        listDir(file) ;
    }
    public static void listDir(File file) {
        if (file.isDirectory()) {       // 是一个目录
            File results [] = file.listFiles() ; // 列出目录中的全部内容
            if (results != null) {
                for (int x = 0 ; x < results.length ; x ++) {
                    listDir(results[x]) ; // 继续依次判断
                }
            }
        }
        file.delete() ;
    }
}
```

■File操作案例：批量修改文件名称

编写程序，程序运行时输入目录名称，并把该目录下的所有文件名后缀修改为.txt。

对于这些的操作必须设置一些假设的约定，能够重命名的文件都是有后缀的，如果没有后缀的路径，则为其追加路径，如果有后缀的路径，则必须以最后一个“.”进行截取。

```
import java.io.File;
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        File file = new File("D:" + File.separator + "test") ; // 是一个目录
        long start = System.currentTimeMillis() ;
        renameDir(file) ;
        long end = System.currentTimeMillis() ;
        System.out.println("本次操作所花费的时间：" + (end - start));
    }
    public static void renameDir(File file) {
        if (file.isDirectory()) {       // 是一个目录
            File results [] = file.listFiles() ; // 列出子目录中的内容
            if (results != null) {
                for (int x = 0 ; x < results.length ; x ++) {
```

```
                        renameDir(results[x]) ;
                    }
                }
            } else {
                if (file.isFile()) { // 如果是文件则必须进行重命名
                    String fileName = null ;
                    if (file.getName().contains(".")) {
                        fileName =
file.getName().substring(0,file.getName().lastIndexOf(".")) + ".txt" ;
                    } else {
                        fileName = file.getName() + ".txt" ;
                    }
                    File newFile = new File(file.getParentFile(),fileName) ; // 新的文件名称
                    file.renameTo(newFile) ; // 重命名
                }
            }
        }
}
```

在面试的过程之中经常会出现给你一个路径而后让你进行名称或文件的批量修改操作，那么就采用以上的代码结构即可。