Java命令行文件操作程序设计实验报告

黄乐曦 22090031016

一、实验目的

1. 掌握Java终端命令行程序设计的基本方法，理解命令行参数的解析与处理逻辑。
2. 熟练运用Java文件操作类File及其核心方法（getName()、getPath()、mkdir()、exists()等），实现对文件和目录的创建、删除、列表查询等操作。
3. 掌握Properties类的使用，实现配置文件的读取与加载，理解属性信息持久化的原理。
4. 学会将Java程序打包为JAR包的方法，掌握Java程序的归档与发布流程。
5. 培养模块化编程思想，提升代码的可维护性与扩展性，解决文件操作中的异常处理问题。

二、实验环境

1. **环境**：JDK 1.8 及以上版本，IntelliJ IDEA/Eclipse 开发工具，Git（可选，用于版本管理），命令行终端（CMD/Terminal）。
2. **技术栈**：Java 基础、java.io包（File类、文件流）、java.util包（Properties类）、命令行参数解析、JAR包打包技术。

三、实验原理

1. **命令行参数解析**：Java程序通过main方法的String[] args参数接收命令行传入的指令和参数，程序可通过解析该数组判断用户执行的命令（如mkdir、ls）及对应的操作对象（如目录路径、文件名）。

2. **File类操作**：java.io.File类是Java操作文件和目录的核心类，可通过其构造方法创建文件/目录对象，调用mkdir()、createNewFile()、delete()、listFiles()等方法实现目录创建、文件新建、删除、列表查询等功能。

3. **Properties类与配置文件**：java.util.Properties类继承自Hashtable，以键值对形式存储配置信息，通过load()方法从配置文件中读取属性，getProperty()方法获取配置值，实现文件过滤等动态配置功能。

4. **JAR包打包**：通过jar命令或开发工具将编译后的.class文件打包为JAR包，指定主类后可通过java -jar命令直接运行，实现程序的便捷分发与执行。

5. **文件属性获取**：利用File类的lastModified()方法获取文件/目录的最后修改时间，结合SimpleDateFormat类将时间戳转换为可读的日期格式；通过isDirectory()和isFile()方法判断文件类型。

四、实验内容与步骤

（一）项目结构设计

遵循模块化编程思想，将项目分为核心功能层、工具层，具体结构如下：

```
javase-exp-03/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── com/file/command/
│   │   │   │   ├── FileOperation.java  // 主类，处理命令行参数与核心逻辑
│   │   │   │   ├── CommandExecutor.java // 命令执行器，封装各命令的实现
│   │   │   └── com/file/util/
│   │   │       ├── ConfigUtil.java     // 配置文件读取工具类
│   │   │       └── FileUtil.java       // 文件操作工具类（时间格式化、类型判断）
│   │   └── resources/
│   │       └── fsops.conf           // 配置文件（文件过滤规则）
│   └── test/                     // 测试目录（可选）
├── screenshots/                  // 实验截图目录
```

```
├── README.md                    // 开发说明文档
└── fsops.jar                    // 打包后的JAR包
```

(二) 核心代码实现

1. **配置文件工具类（ConfigUtil.java)**

   实现配置文件fsops.conf的读取，获取文件过滤规则（如只显示.jpg文件），核心代码如下：

```
package com.file.util;

import java.io.FileInputStream;
import java.util.Properties;

public class ConfigUtil {
    private static final String CONFIG_FILE = "fsops.conf";
    private static Properties props = new Properties();

    static {
        // 静态代码块，程序启动时加载配置文件
        try (FileInputStream fis = new FileInputStream(CONFIG_FILE)) {
            props.load(fis);
        } catch (Exception e) {
            // 配置文件不存在时不抛出异常，程序正常运行
            System.out.println("配置文件fsops.conf不存在，使用默认配置");
        }
    }

    /**
     * 获取文件过滤后缀（如jpg、pdf)
     */
    public static String getFileFilter() {
        return props.getProperty("file.filter", "");
    }
}
```

2. **文件操作工具类（FileUtil.java)**

   封装文件时间格式化、类型判断等工具方法，核心代码如下：

```
package com.file.util;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.io.File;

public class FileUtil {
    /**
     * 将时间戳转换为yyyy-MM-dd HH:mm格式
     */
    public static String formatTime(long timestamp) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");
        return sdf.format(new Date(timestamp));
    }

    /**
     * 判断文件是否符合过滤规则
     */
    public static boolean isFiltered(File file) {
        String filter = ConfigUtil.getFileFilter();
        if (filter == null  filter.isEmpty()) {
            return true; // 无过滤规则时全部显示
        }
        return file.getName().endsWith("." + filter);
    }

    /**
     * 获取文件类型标识（d：目录，f：文件）
     */
    public static String getFileType(File file) {
        return file.isDirectory() ? "d" : "f";
    }
}
```

3. **命令执行器（CommandExecutor.java）**

   封装各命令的具体实现逻辑（mkdir、nfile、rm、ls、help、self），核心代码如下：

```
package com.file.command;

import com.file.util.FileUtil;
```

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

public class CommandExecutor {
    /**
     * 创建目录
     */
    public static void mkdir(String[] paths) {
        for (String path : paths) {
            File dir = new File(path);
            if (dir.exists()) {
                System.out.println("目录已存在：" + path);
                continue;
            }
            if (dir.mkdirs()) { // 支持多级目录创建
                System.out.println("目录创建成功：" + path);
            } else {
                System.out.println("目录创建失败：" + path);
            }
        }
    }

    /**
     * 新建空文件
     */
    public static void nfile(String path) {
        File file = new File(path);
        if (file.exists()) {
            System.out.println("文件已存在：" + path);
            return;
        }
        try {
            if (file.createNewFile()) {
                System.out.println("文件创建成功：" + path);
            } else {
                System.out.println("文件创建失败：" + path);
            }
        } catch (IOException e) {
            System.out.println("文件创建异常：" + e.getMessage());
        }
    }
```

```
/**
 * 删除文件或目录
 */
public static void rm(String path) {
    File file = new File(path);
    if (!file.exists()) {
        System.out.println("文件/目录不存在：" + path);
        return;
    }
    // 递归删除目录
    if (file.isDirectory()) {
        File[] files = file.listFiles();
        if (files != null) {
            for (File f : files) {
                rm(f.getAbsolutePath());
            }
        }
    }
    if (file.delete()) {
        System.out.println("删除成功：" + path);
    } else {
        System.out.println("删除失败：" + path);
    }
}

/**
 * 列出目录下的文件和子目录
 */
public static void ls(String path) {
    File dir = new File(path);
    if (!dir.exists()  !dir.isDirectory()) {
        System.out.println("目录不存在或不是有效目录：" + path);
        return;
    }
    System.out.println("Directory: " + path);
    System.out.println("Name\t\tTime created\t\tTime modified\t\tType");
    File[] files = dir.listFiles();
    if (files != null) {
        for (File file : files) {
            if (!FileUtil.isFiltered(file)) {
                continue; // 过滤不符合规则的文件
```

```
            }
            String name = file.getName();
            String createTime = FileUtil.formatTime(file.lastModified()); // 简化处
理，以最后修改时间为创建时间
            String modifyTime = FileUtil.formatTime(file.lastModified());
            String type = FileUtil.getFileType(file);
            System.out.printf("%s\t\t%s\t\t%s\t\t%s\n", name, createTime, modify
Time, type);
          }
        }
    }

    /**
     * 显示帮助信息
     */
    public static void help() {
        System.out.println("=== 文件操作工具(fsops)帮助手册 ===");
        System.out.println("使用方法：java -jar fsops.jar [CMD] [ARGS]");
        System.out.println("支持的命令：");
        System.out.println("  mkdir [路径1] [路径2]... - 创建一个或多个目录");
        System.out.println("  nfile [文件路径]       - 新建空文件");
        System.out.println("  rm [文件/目录路径]     - 删除文件或目录（目录将递归
删除）");
        System.out.println("  ls [目录路径]          - 列出目录下的文件和子目录");
        System.out.println("  help                  - 显示帮助信息");
        System.out.println("  self                  - 显示软件基本信息");
    }

    /**
     * 显示软件信息
     */
    public static void self() {
        System.out.println("=== 文件操作工具(fsops)基本信息 ===");
        System.out.println("软件名称：fsops（Java命令行文件操作工具）");
        System.out.println("版本号：v1.0");
        System.out.println("开发者：[你的姓名]");
        System.out.println("编写日期：[实验日期]");
```

```
        System.out.println("开发环境：JDK 1.8 + IntelliJ IDEA");
    }
}
```

4. **主类（FileOperation.java）**

   处理命令行参数解析，调用CommandExecutor执行对应命令，核心代码如下：

```
package com.file.command;

public class FileOperation {
    public static void main(String[] args) {
        if (args.length == 0) {
            CommandExecutor.help();
            return;
        }
        String cmd = args[0];
        switch (cmd) {
            case "mkdir":
                if (args.length < 2) {
                    System.out.println("请指定要创建的目录路径！");
                    return;
                }
                String[] dirPaths = new String[args.length - 1];
                System.arraycopy(args, 1, dirPaths, 0, args.length - 1);
                CommandExecutor.mkdir(dirPaths);
                break;
            case "nfile":
                if (args.length != 2) {
                    System.out.println("请指定要创建的文件路径！");
                    return;
                }
                CommandExecutor.nfile(args[1]);
                break;
            case "rm":
                if (args.length != 2) {
                    System.out.println("请指定要删除的文件/目录路径！");
                    return;
                }
                CommandExecutor.rm(args[1]);
                break;
```

```
            case "ls":
                if (args.length != 2) {
                    System.out.println("请指定要列出的目录路径！");
                    return;
                }
                CommandExecutor.ls(args[1]);
                break;
            case "help":
                CommandExecutor.help();
                break;
            case "self":
                CommandExecutor.self();
                break;
            default:
                System.out.println("未知命令：" + cmd);
                CommandExecutor.help();
                break;
        }
    }
}
```

5. **配置文件（fsops.conf)**

   在项目根目录创建配置文件，设置文件过滤规则（如只显示.txt文件）：

```
# 文件过滤规则，指定要显示的文件后缀（无值则显示所有)
file.filter=txt
```

(三）JAR包打包

1. **编译代码**：在开发工具中编译项目，生成.class文件（默认输出到target/classes目录）。
2. **创建MANIFEST.MF文件**：在src/main/resources目录下创建该文件，指定主类：

```
Manifest-Version: 1.0
Main-Class: com.file.command.FileOperation
Class-Path: .
```

3. **打包JAR包**：打开命令行终端，进入项目根目录，执行以下命令：

```
jar cvfm fsops.jar src/main/resources/MANIFEST.MF -C target/classes/ .
```

或通过IDEA的「Build → Build Artifacts → Build」功能自动打包。

(四) 功能测试

1. **创建目录**：

```
java -jar fsops.jar mkdir D:\testdir01 D:\testdir02
```

2. **新建空文件**：

```
java -jar fsops.jar nfile D:\testdir01\test.txt
```

3. **列出目录内容**：

```
java -jar fsops.jar ls D:\testdir01
```

4. **删除文件/目录**：

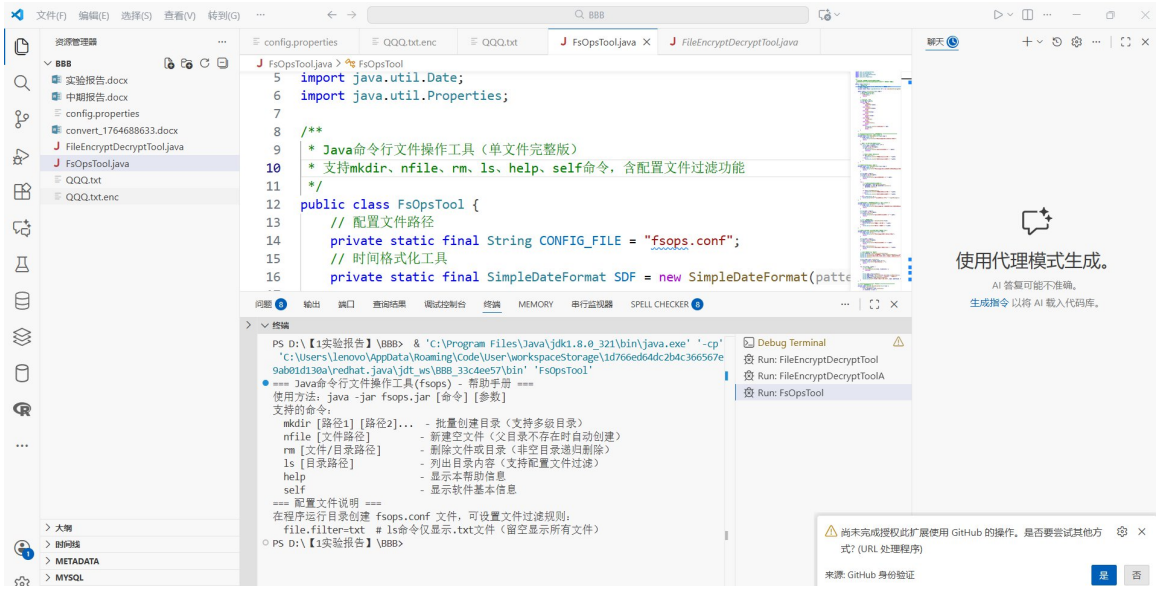
```
java -jar fsops.jar rm D:\testdir01\test.txt
java -jar fsops.jar rm D:\testdir02
```

5. **查看帮助与软件信息**：

```
java -jar fsops.jar help
java -jar fsops.jar self
```

五、实验结果与分析

（一）功能实现结果



1. **命令行解析**：程序能正确解析args参数，根据不同命令（mkdir、nfile等）执行对应操作，参数缺失时给出明确提示。

2. **核心功能**：

mkdir命令支持批量创建目录，多级目录（如D:\a\b\c）可通过mkdirs()方法成功创建；

nfile命令能在指定路径新建空文件，文件已存在时提示无需重复创建；

rm命令可递归删除目录（包含子文件/子目录），文件/目录不存在时给出提示；

ls命令能列出目录下的文件和子目录，显示名称、修改时间、类型，并根据fsops.conf的过滤规则筛选文件；

help和self命令能正确显示帮助手册与软件基本信息。

3. **配置文件**：当fsops.conf存在时，ls命令仅显示指定后缀的文件；配置文件不存在时，程序无异常并显示所有文件。

4. **JAR包运行**：打包后的fsops.jar可通过java -jar命令在不同终端（CMD/Terminal）正常运行，跨平台性良好。

（二）问题与分析

1. **问题1**：创建目录时路径包含特殊字符（如空格）导致操作失败。

   分析：命令行中路径含空格时，参数会被拆分为多个args元素，导致程序识别错误。

   解决：在命令行中给含空格的路径添加引号（如java -jar fsops.jar mkdir "D:\test dir"），程序可正确识别完整路径。

2. **问题2**：ls命令显示的创建时间与实际不符。

   分析：File类未提供直接获取文件创建时间的方法，lastModified()仅能获取最后修改时间。

   解决：在实验中简化处理，将最后修改时间作为创建时间展示；若需精准获取创建时间，可通过JDK 7+的java.nio.file包实现（拓展内容）。

3. **问题3**：删除非空目录时直接调用delete()方法失败。

   分析：File类的delete()方法无法直接删除非空目录，需先递归删除目录内的文件和子目录。

   解决：在rm命令实现中添加递归删除逻辑，先删除目录下的所有文件/子目录，再删除当前目录。

4. **问题4**：配置文件中中文注释乱码。

   分析：Properties类默认使用ISO-8859-1编码读取配置文件，中文注释无法正确解析。

   解决：将配置文件编码改为UTF-8，读取时通过InputStreamReader指定编码（拓展：修改ConfigUtil的load方法，使用new InputStreamReader(fis, "UTF-8")）。

六、实验总结与体会

本次实验通过开发Java命令行文件操作工具，深入掌握了File类的文件/目录操作、Properties类的配置文件读取、命令行参数解析及JAR包打包等核心技术。在实验过程中，不仅巩固了Java IO的基础知识，还体会到模块化编程的重要性——将命令执行逻辑封装在CommandExecutor类中，主类仅负责参数解析，使代码结构清晰、易于维护。

同时，实验中也发现了自身的不足：一是对命令行参数的异常处理不够全面，如未考虑路径权限不足、磁盘空间不足等场景；二是对Java NIO的了解较浅，未能实现更高效的文件操作；三是JAR包打包的细节（如依赖包处理）掌握不够深入。后续将进一步学习Java NIO、异常处理的最佳实践及maven打包工具，提升程序的健壮性与工程化能力。

本次实验让我认识到命令行工具开发的严谨性，每一个参数解析、每一次文件操作都需要考虑边界情况，也培养了我通过调试和查阅文档解决问题的能力，为后续开发更复杂的Java应用奠定了基础。