【实验7-1】 批量操作文件功能

### 任务介绍

**1．任务描述**

在日常工作中，经常会遇到批量操作系统文件的事情，通常情况下，只能手动重复的完成批量文件的操作，这样很是费时费力。本案例要求编写一个文件管理器，实现文件的批量操作。文件管理器具体功能要求如下：

1. 用户输入指令1，代表“指定关键字检索文件”，此时需要用户输入检索的目录和关键字，系统在用户指定的目录下检索出文件名中包含关键字的文件，并将其绝对路径展示出来。
2. 用户输入指令2，代表“指定后缀名检索文件”，此时需要用户输入检索的目录和后缀名（多个后缀名用逗号分隔），系统在用户指定的目录下检索出指定后缀名的文件，并将其绝对路径展示出来。
3. 用户输入指令3，代表“复制文件/目录”，此时需要用户输入源目录和目标目录，程序执行后会将源目录下的内容复制到目标目录下。
4. 用户输入指令4，代表“删除文件/目录”，此时需要用户输入需要删除掉的文件目录，程序执行后会将目录以及目录下的内容全部删除。
5. 用户输入指令5，代表“退出”，即退出该系统。

**2．运行结果**

任务运行结果如图7-1所示。

手机屏幕截图

描述已自动生成

图7-1 运行结果

源路径G:\itcast下的文件，如图7-2所示。

社交网站的手机截图

描述已自动生成

图7-2源目录

### 任务目标

* 学会分析“模拟文件管理器”程序的实现思路。
* 能够根据思路独立完成“模拟文件管理器”程序的源代码编写、编译及运行。
* 掌握File类的常用方法。
* 掌握文件遍历的方法和文件名过滤器FilenameFilter的用法。

### 实现思路

（1）根据任务介绍和运行结果分析可知，首先需要创建一个文件管理器类。可以在类中使用while循环实现控制台中操作指令的多次输入，并使用switch语句根据控制台输入的操作指令来判断执行什么操作。

（2）执行指令1时，代表指定关键字检索文件，即在目录下查找包含关键字的文件。执行时先要从控制台获取目录和关键字，然后将其传到后台的方法中，后台可将传入的关键字利用过滤器将其制定成“规则”，通过递归的方式遍历文件夹，在每个子文件夹下调用过滤器，来获取符合规则的文件路径的集合，并将集合返回，最后打印出来；

（3）执行指令2时，代表指定后缀名检索文件，即在目录下查找名称结尾是指定后缀的文件。执行时可以先从控制台获取目录和后缀名，然后将后缀名拆分成数组，并将数组和目录传到后台方法中。后台可用过滤器将后缀名数组循环遍历，制定成“规则”，通过递归的方式遍历文件夹，在每个子文件夹下调用过滤器，来获取符合规则的文件路径的集合，并将集合返回，最后并打印出来；

（4）执行指令3时，代表将源目录的内容复制到目标目录中。执行时可以先从控制台获取源目录和目标目录，然后将其传入后台。后台主要是通过递归的方法执行文件和文件夹复制的。每次调用此方法时，首先会获取当前目录下的文件和文件夹列表，然后循环列表并判断当前元素是文件还是文件夹。如果是文件夹，就利用File类的mkdirs()方法在目标目录中创建此文件夹，再调用自身；如果是文件，由于不一定是纯文本文件，所以可以采用字节流将文件写入到目标目录；

（5）执行指令4时，执行代表将输入的文件删除。执行是先从控制台获取文件地址，后台先判断File对象是否存在，再使用foreach循环遍历删除文件夹下的所有文件，最后删除该文件夹；

（6）执行指令5时，执行退出该程序的操作，该操作可以通过System.exit(0)来实现。

### 实现代码

（1）定义DocumentManager类，具体如文件7-1所示。

文件7-1 DocumentManager.java

1. package chapter0705;
2. import java.io.File;
3. import java.util.ArrayList;
4. import java.util.Scanner;
5. public class DocumentManager {
6. public static void main(String[] args) throws Exception {
7. Scanner sc = new Scanner(System.in);
8. System.out.println("--1:指定关键字检索文件 2:指定后缀名检索文件 "
9. + "3:复制文件/目录 4:删除文件/目录 5:退出--");
10. while(true){
11. System.out.print("请输入指令：");
12. int command = sc.nextInt();
13. switch (command) {
14. case 1:
15. searchByKeyWorld();//指定关键字检索文件
16. break;
17. case 2:
18. searchBySuffix();//指定后缀名检索文件
19. break;
20. case 3:
21. copyDirectory();//复制文件/目录
22. break;
23. case 4:
24. deleteDir();//删除文件/目录
25. break;
26. case 5:
27. exit();//退出
28. break;
29. default:
30. System.out.println("您输入的指令错误！");
31. break;
32. }
33. }
34. }
35. // \*\*\*\*\*\*\*\*\*1.指定关键字检索文件\*\*\*\*\*\*\*\*\*
36. private static void searchByKeyWorld() {
37. Scanner sc = new Scanner(System.in);
38. System.out.print("请输入要检索的目录位置：");
39. String path = sc.next();//从控制台获取路径
40. File file = new File(path);
41. if (!file.exists() || !file.isDirectory()) {//判断目录是否存在，是否是目录
42. System.out.println(path + " (不是有效目录)");
43. return;
44. }
45. System.out.print("请输入搜索关键字：");
46. String key = sc.next();//获取关键字
47. //在输入目录下获取所有包含关键字的文件路径
48. ArrayList<String> list = FileUtils.listFiles(file,key);
49. for (Object obj : list) {
50. System.out.println(obj);//将路径打印到控制台
51. }
52. }
53. // \*\*\*\*\*\*\*\*\*2.指定后缀名检索文件\*\*\*\*\*\*\*\*//
54. private static void searchBySuffix() {
55. Scanner sc = new Scanner(System.in);
56. System.out.print("请输入要检索的目录位置：");
57. String path = sc.next();//从控制台获取路径
58. File file = new File(path);
59. if (!file.exists() || !file.isDirectory()) {//判断目录是否存在，是否是目录
60. System.out.println(path + " (不是有效目录)");
61. return;
62. }
63. System.out.print("请输入搜索后缀：");
64. String suffix = sc.next();
65. String[] suffixArray = suffix.split(",");//获取后缀字符串
66. //在输入目录下获取所有指定后缀名的文件路径
67. ArrayList<String> list = FileUtils.listFiles(file, suffixArray);
68. for (Object obj : list) {
69. System.out.println(obj);//将路径打印到控制台
70. }
71. }
72. // \*\*\*\*\*\*\*\*\*3.复制文件/目录\*\*\*\*\*\*\*\*\*\*//
73. private static void copyDirectory() throws Exception {
74. Scanner sc = new Scanner(System.in);
75. System.out.print("请输入源目录：");
76. String srcDirectory = sc.next();//从控制台获取源路径
77. File srcFile = new File(srcDirectory);
78. //判断是否存在 是否是目录
79. if (!srcFile.exists() || !srcFile.isDirectory()) {
80. System.out.println("无效目录！");
81. return;
82. }
83. System.out.print("请输入目标位置：");
84. String destDirectory = sc.next();//从控制台获取目标路径
85. File destFile = new File(destDirectory);
86. //判断是否存在 是否是目录
87. if (!destFile.exists() || !destFile.isDirectory()) {
88. System.out.println("无效位置！");
89. return;
90. }
91. //将源路径中的内容复制到目标路径下
92. FileUtils.copySrcPathToDestPath(srcFile, destFile);
93. }
94. // \*\*\*\*\*\*\*\*\*4.删除文件/目录\*\*\*\*\*\*\*\*\*\*//
95. public static void deleteDir() {
96. Scanner sc = new Scanner(System.in);
97. System.out.print("请输入需要删除的源目录：");
98. String delpath = sc.next();// 从控制台获取源路径
99. File dir = new File(delpath); // 创建一个代表目录的File对象
100. if (dir.exists()) { // 判断传入的File对象是否存在
101. File[] files = dir.listFiles(); // 得到File数组
102. for (File file : files) { // 遍历所有的子目录和文件
103. if (file.isDirectory()) {
104. deleteDir(); // 如果是目录，递归调用deleteDir()
105. } else {
106. // 如果是文件，直接删除
107. file.delete();
108. }
109. }
110. // 删除完一个目录里的所有文件后，就删除这个目录
111. dir.delete();
112. }
113. }
114. // \*\*\*\*\*\*\*\*\*5.退出\*\*\*\*\*\*\*\*\*\*//
115. private static void exit() {
116. System.out.println("您已退出系统，谢谢使用！");
117. System.exit(0);
118. }
119. }

在文件7-26中，第10~33行代码，通过while循环等待指令输入，并通过switch来判断执行什么动作。第36~52行执行了根据指定的关键字检索文件的动作，其中第41行代码用来判断了输入的目录是否有效。第48行代码用于调用后台方法，获取符合规则的列表。第54~71行代码执行指定后缀名检索文件，其中第65行代码将获得的后缀名字符串拆分成数组，并利用第67行代码传入后台，获取符合规则的列表。第73~93行代码的作用是将源目录下的内容复制到目标目录中，其中第79和89行代码用于判断目录是否有效，第92行代码用于调用后台方法，实现源目录内容复制的功能。第93-113行代码的作用是将控制台输入的目标文件删除，其中第100行代码是判断File对象是否存在，第102-109行代码使用foreach循环删除子目录与文件。

（2）定义FileUtils类，具体如文件7-2所示。

文件7-2 FileUtils.java

1. package cn.itcast.chapter07.task03;
2. import java.io.File;
3. import java.io.FileInputStream;
4. import java.io.FileOutputStream;
5. import java.io.FilenameFilter;
6. import java.util.ArrayList;
7. public class FileUtils {
8. /\*\*
9. \* 指定关键字检索文件
10. \* @param file File对象
11. \* @param key 关键字
12. \* @return 包含关键字的文件路径
13. \*/
14. public static ArrayList<String> listFiles(File file, final String key){
15. FilenameFilter filter = new FilenameFilter() { // 创建过滤器对象
16. public boolean accept(File dir, String name) {// 实现accept()方法
17. File currFile = new File(dir, name);
18. // 如果文件名包含关键字返回true，否则返回false
19. if (currFile.isFile() && name.contains(key)) {
20. return true;
21. }
22. return false;
23. }
24. };
25. //递归方式获取规定的路径
26. ArrayList<String> arraylist = fileDir(file, filter);
27. return arraylist;
28. }
29. /\*\*
30. \* 指定后缀名检索文件
31. \* @param file File对象
32. \* @param suffixArray 后缀名数组
33. \* @return 指定后缀名的文件路径
34. \*/
35. public static ArrayList<String> listFiles(File file, final String[] suffixArray) {
36. FilenameFilter filter = new FilenameFilter() { // 创建过滤器对象
37. public boolean accept(File dir, String name) {// 实现accept()方法
38. File currFile = new File(dir, name);
39. if (currFile.isFile()) {
40. for (String suffix : suffixArray) {
41. if (name.endsWith("." + suffix)) {
42. return true;
43. }
44. }
45. }
46. return false;
47. }
48. };
49. //递归方式获取规定的路径
50. ArrayList<String> arraylist = fileDir(file, filter);
51. return arraylist;
52. }
53. /\*\*
54. \* 递归方式获取规定的路径
55. \* @param dir File对象
56. \* @param filter 过滤器
57. \* @return 过滤器过滤后的文件路径
58. \*/
59. public static ArrayList<String> fileDir(File dir, FilenameFilter filter){
60. ArrayList<String> arraylist = new ArrayList<String>();
61. File[] lists = dir.listFiles(filter); // 获得过滤后的所有文件数组
62. for (File list : lists) {
63. //将文件的绝对路径放到集合中
64. arraylist.add(list.getAbsolutePath());
65. }
66. File[] files = dir.listFiles(); // 获得当前目录下所有文件的数组
67. for (File file : files) { // 遍历所有的子目录和文件
68. if (file.isDirectory()) {
69. // 如果是目录，递归调用fileDir()
70. ArrayList<String> every = fileDir(file, filter);
71. arraylist.addAll(every);//将文件夹下的文件路径添加到集合中
72. }
73. }//此时的集合中有当前目录下的文件路径，和当前目录的子目录下的文件路径
74. return arraylist;
75. }
76. /\*\*
77. \* 复制文件/目录
78. \* @param srcFile 源目录
79. \* @param destFile 目标目录
80. \*/
81. public static void copySrcPathToDestPath(File srcDir, File destDir)
82. throws Exception {
83. File[] files = srcDir.listFiles();// 子文件目录
84. for (int i = 0; i < files.length; i++) {
85. File copiedFile = new File(destDir, files[i].getName());
86. if (files[i].isDirectory()) {// 如果是目录
87. if(!copiedFile.mkdirs()){//创建文件夹
88. System.out.println("无法创建："+copiedFile);
89. return;
90. }
91. // 调用递归，获取子文件夹下的文件路径
92. copySrcPathToDestPath(files[i], copiedFile);
93. } else {// 复制文件
94. FileInputStream input = new FileInputStream(files[i]);//获取输入流
95. FileOutputStream output = new FileOutputStream(copiedFile);//获取输出流
96. byte[] buffer = new byte[1024];//创建缓冲区
97. int n = 0;
98. //循环读取字节
99. while ((n = input.read(buffer))!= -1) {
100. output.write(buffer, 0, n);
101. }
102. input.close();//关闭输入流
103. output.close();//关闭输出流
104. }
105. }
106. }
107. }

在文件7-27中，执行指令1时调用第14~28行的代码，其中第15~24行代码用于创建过滤器，第26行代码用于调用递归方法。执行指令2时，将调用第35~52行代码，其中第36~48行代码用于创建过滤器，其中第40~44行代码通过循环，将所有后缀名都配成了过滤规则。第50行代码调用了递归方法，此方法的代码是59~75行，方法中第61~65行代码获取当前目录下经过过滤后的文件数组，通过循环取其路径存入集合中。第66~73行代码将当前目录下内容循环，如果是文件夹则调用自身。执行指令3时调用第81~106行代码，第83行代码先获取了子文件列表，然后将列表中文件进行循环，如果循环获取的当前目录是文件夹，则执行第87~92行的代码，创建此文件夹，之后调用自身；当前目录是文件时，通过第94~103行代码，将此文件用字节流形式写入并关闭资源。