|  |  |  |  |
| --- | --- | --- | --- |
| **a实验名称** | **综合实验——文件搜索程序** | | |
| **学号** | **1120161793** | **姓名** | **张嘉琦** |
| <张嘉琦>  **一、小组成员**  组长：07111603班 张嘉琦 1120161793  组员：07111603班 张 圳 1120161795  07111601班 艾若琳 1120161768  07111601班 高钒骐 1120161650  **二、实验目的**  1.熟悉相关的操作系统原理  2.提高编程水平  3.提升团队协作的能力  **三、实验内容**  编写一个进行文件搜索的程序。选择文件夹后，该程序能够在此文件夹中根据给定的字符串或其他条件进行搜索，并将搜索结果按照特定方式呈现出来。还可以设置条件对搜索的结果进行筛选或按照特定的方式进行排序。搜索结果可以点击或进行其他相关的文件操作，还可以将搜索结果录入数据库或者进行统计处理。该程序主要涉及到操作系统中的文件操作，此外还带有一定的进程通信的内容。  **四、实验环境及配置方法**  操作系统：Windows 10  处理器：Intel® Core™ i5-6300HQ CPU @ 2.30GHz  内存：8GB  编译器：Eclipse  编程语言：Java  </张嘉琦>  **五、实验方法和实验步骤**  <张圳>  1.程序总体架构  程序的总体架构如图所示：    工程中包括七个Java包，它们及每个包中的类如下：   * application：Application.java * filesearch：FileSearch.java * fileutil：FileUtil.java * sortutil：DefaultSortUtil.java、SortUtil.java * searchhistorydb：ConnectionInstance.java、InsertIntoDB.java * resultvisual：ResultVisual.java * gui：GUI.java、MenuBar.java、MessegeBox.java、Search.java、SortBox.java、Table.java、TableRender.java   其中，前六个包与程序运行原理相关，是该程序的后端；最后一个包是程序的界面相关，是该程序的前端。程序架构的难点在于前端与后端的结合。  2.程序后端  (1).Application类    Application.java是项目的main方法入口类，该类调用GUI的静态方法getGUIInstance获取一个GUI实例。采用单例模式，确保自始至终只有一个GUI实例，这对于项目的性能提升有一定程度上的帮助。  (2).Search类    该类的主要功能是调用FileSearch类进行搜索、将搜索结果进行展示、记录每次搜索消耗的时间、搜索结果的数目、调用InsertIntoDB类保存搜索记录、保存每次过滤和排序功能使用的数据。  (2).FileSearch类    该类是项目的核心功能实现的部分，使用了Java8新特性NIO中的walkFileTree的方式进行遍历搜索。方法接收待搜索文件夹的路径和搜索模式两个参数，搜索模式支持完全匹配、部分匹配、正则匹配和后缀名匹配。  (3).FileUtil类    这是一个文件操作相关的类，提供了文件或文件夹的复制、删除、打开和重命名的方法。  (4).DefaultSortUtil类    该类主要是对本项目中排序和过滤算法需要经常使用到的字段和数据结构进行初始化和存储，比如将文件的分类存储在HashMap中，在排序和筛选时的速度会很快。例如，程序中存在对文件格式进行的操作，因此需要对文件名进行分析，抽取出文件的文件格式进行分析和分类，传统的方式为O(n)的时间复杂度，在数据量较大时会减慢性能，因此这里采用了时间复杂度仅仅为O(1)的HashMap数据结构进行处理。  (5).SortUtil类    该类为项目中的排序操作提供比较器。其中有按名称排序、按文件路径排序、按文件大小排序、按修改时间排序和按文件格式排序五种。但初始设置时出现了问题，这是因为HashMap在进行存储时存在一个“键值”，该数据结构是按键进行存储的，数据结构中不同的条目必须有不同的键值。因此在比较的时候如果仅按大小比较，相同大小的文件就像被“吃掉”了一样，无法显示。我们发现文件的路径（包括文件名本身）是唯一的键值，因此在代码中添加了比较器相等时的操作，最后按照文件的路径进行排序。例如，如果两个文件的大小一样，则按文件名排序，如果依旧一样则按路径进行排序。  (6).ConnectionInstance类    该类提供一个数据库的连接实例，也使用了单例模式，同GUI类似。这里采用的数据库是MySQL。  (7).InsertIntoDB类    本项目中使用数据库，主要是用来保存用户的搜索记录，即搜索时间、搜索的关键字和搜索模式，同时记录了用户最终的筛选结果的详细信息和每种类型的文件数量。  (8).ResultVisual类    该类用于将用户的搜索结果进行可视化展示，即柱状图或饼状图的形式。这里会展示用户的搜索结果中每种类型的文件（如jpg、png等）或大类（如图片类、视频类等）的具体数量。默认展示数量最多的前6种/类以及每种类型文件的占比，总计四种可视化结果。详细的数据用户可以在数据库中查看更加细致的情况。  </张圳>  3. 程序前端  <张嘉琦>  (1).GUI类  该类是GUI界面的主体，包括了主界面所需的各种信息。程序的前端主要用Java的Swing库进行绘制。  当main方法进行调用时，程序创建一个新的GUI类，接下来开始绘制界面。界面采用Windows的窗口风格进行渲染，基本主体为一个JFrame。设置窗口的初始大小以及相关参数，接下来向窗口上添加控件。绘制出的界面如图所示：    首先添加菜单栏，调用MenuBar类中的addMenuBar方法，在界面最上方添加菜单栏。  接下来是窗口的整体布局，这里采用了一个纵向的盒状容器嵌套多个横向盒状容器的方法。因为盒状容器方便控制，且缩放后不易变形。于是我们新建一个mainBox用作窗口的总框架，在上面依次添加容器。  首先添加“选择菜单栏”横向盒状容器，该容器中含有一个按钮和一个文本框。该文本框在未选择文件夹时会出现红色字样（如图所示），选择文件夹后会出现选择出的文件夹的路径。  接下来依次是“搜索框”、“筛选框”、“排序框”、显示列表和“信息框”，每个框都是一个横向的盒状容器，通过创建、设置大小、设置位置的方式即可添加入窗口。每个控件都经过创建、设置属性、设置字体再进行添加。  接下来我们需要对每个控件添加监听器，即动作响应函数，这涉及到界面交互的问题。例如搜索按钮，当未选择文件夹或未填写搜索框时，搜索和清空按钮不可被点击。在搜索的输入框和筛选的输入框按回车键均可以进行搜索，但是如果没有选择文件夹或者没有输入搜索内容则会发出错误信息。  需要说明的是：这里的搜索包含三个部分，第一个是输入框的搜索内容，这个搜索框的搜索模式可以通过菜单栏的选项进行修改，默认是部分匹配；第二个是筛选框中“不含的字符串”；第三个是下拉框中的文件格式。例如，我可以在搜索框输入“12”、在筛选框输入“3”、在文件格式中选择“图片格式”，从而搜索到文件夹中所有文件名含“12”但不含“3”的图片。该筛选框的设计思路来自网络上的搜索引擎，很多时候不同领域对某一个词汇有不同的解释，我们想过滤掉某些领域的搜索结果，就可以使用到筛选栏。正则表达式也能够产生同样的效果，但正则表达式对用户的要求较高，很多普通用户无法掌握正则表达式的写法，因此将功能拆分出来是存在一定必要性的。  另外需要说明的就是展示搜索结果的表格。我们希望该表格不能被编辑但可以被选中，但是原始的JTable不允许这样的操作，因此我们编写了继承自该类的Table类。我们给此表格设置排序器和我们自己编写的渲染器，再在排序器上添加我们自己编写的过滤器。JTable默认的排序器是添加在表头的，当且仅当用户点击表头时才能够对内部的信息进行排序，这不满足我们的需求。因此我们重新写了排序的内容，使得点击表格外的按钮也能够对表格进行排序。但是这会导致两个排序的结果冲突，因此我们将表头禁用，无法点击。但是我们不能将排序器删除，因为过滤器必须放置在排序器上方。过滤器的基本原理是通过输入一个正则表达式，用该正则表达式对表格中的内容进行过滤。我们上方使用的“筛选”和“文件格式”就是用这种方式进行过滤的。  JTable的模式是设置一个“表格模型”，表格模型的内容并不会呈现出来，会通过渲染器、排序器和过滤器进行展示。也就是说，展示出来的结果并不是实际数据结构中的存储内容。在实际过滤的过程中，表格模型中的内容从来没有发生过变化。一定程度上这可以提升性能，不需要每次改变筛选内容就重新进行一次搜索。但为了代码结构的简洁，每次点击搜索按钮（或在输入框按回车键）都视为重新进行了一次搜索。  在表格的项目上点击鼠标右键即可弹出一个小的菜单栏。监听器发现点击了鼠标右键后会获取被点击的行号和列号，如果是在路径列则会弹出和文件夹操作相关的菜单栏，否则弹出和文件操作的菜单栏。在表格的项目上双击则会打开文件，如果点击的是路径栏则会打开该文件所在的文件夹。  需要说明的是，实际上我们发现，搜索消耗的时间远远小于展示搜索结果消耗的时间，甚至存在数量级上的差别。也就是说，可能我们修改搜索的方式并不能提升太多搜索性能，大多数时间都消耗在了我们没有办法修改的展示搜索结果上了。  </张嘉琦>  <高钒骐>  (2).MenuBar类  该类用于创建菜单栏，该部分的代码虽然较多，但重复性较高，基本的流程是：创建菜单栏、菜单及菜单项；将菜单栏添加到窗口中；将菜单添加到菜单栏中；将菜单项添加到菜单中；设置菜单项的字体、快捷键等属性；设置每个菜单项的监听器。对每个菜单项都重复这一过程，最后即可写出最终的代码。  菜单栏中的条目如下：   * 文件：选择文件夹、创建柱状图、创建饼图、退出； * 编辑：打开文件、复制文件、删除文件、重命名文件、打开文件夹、复制文件夹、删除文件夹、重命名文件夹； * 搜索：搜索、部分匹配、正则匹配、完全匹配、后缀名匹配、按路径排序、按名字排序、按大小排序、按修改日期排序、按文件格式排序、反序； * 帮助：关于、帮助。   菜单栏项目的监听器调用的方法与其他地方调用的基本一致。需要特殊指出的是，菜单栏项目部分内容保持着和其他部分的一致性。例如排序方式的五个按钮是一个单选框，在菜单栏进行选择后主界面也会相应地变化。  “编辑”菜单下的八个按钮是要选择表格内的信息后才允许点击的，会对文件或文件夹进行不同的操作。其中复制和重命名都会额外弹出窗口。  </高钒骐>  <艾若琳>  (3).TableRender类  该类继承自JtableRender类，是表格的渲染器，这里的渲染器取代了默认的表格渲染器。首先获取文件类型并根据相应的类型以及行数填充背景，如给可执行文件填充紫色、给图片文件填充黄色等。奇数行染深色，偶数行染浅色，这样的结果比较美观。并且设置表格项目被选中时的前景色和背景色，这里选择的结果是黑底白字。    此外，为了提高该程序运行的效率，我们事先识别了文件的类型并将其填充到了文件名的开头，这样我们可以简单地利用文件名开头进行染色了。但是在展示时必须将文件类型进行剔除，同样写在了本段代码中。  (4).Table类  该类继承自JTable类，取代了原本的JTable。设置该类的目的是我们希望表格无法被编辑。但是原本的JTable类只能将表格直接禁用，导致表格不仅无法编辑而且无法被选中。因此我们编写了继承类，设置表格可以被选中行但不能够被编辑，从而达到我们的要求。  (5).SortBox类  该类主要是填写主界面的排序横向盒状容器。按排序方式建立五个按钮，分别为按名称排序、按地址排序、按大小排序、按修改时间排序和按类型排序。再增加一个反序按钮，用于将排序结果反序输出。并且注意这些按钮的启用状态需要与菜单栏的按钮保持同步。  接下来对每个按钮添加监听器，当每个按钮被点击时，调用Search类中不同的排序函数。    (6).Search类  该类作为后端与前端的一个接口程序出现，主要用于调用查找方法和进行排序。    在这段代码开始前后分别计算了时间作为最终的搜索时间并在主界面进行显示。    在显示过程中先获取文件信息表。    在文件列表没有被读取完全的时候获取列表中每一个文件的信息：    在SortBox的结果更改时设置排序器改变搜索表的排序并且更新结果表。  (7).MessegeBox类  该类用于向主界面添加搜索时间和搜索条目信息，为两个Jlabel。    </艾若琳>  **六、实验结果和分析**  <张嘉琦>  1.程序运行结果  因为功能较多不一一进行展示，这里只选择几个主要的功能示意。选择我电脑的D盘，搜索“12”。得到的结果如图：    可以看到搜索得到了1919个结果，并且搜索到的结果按文件格式进行了染色。接下来我们点击“FORMAT”按钮，即按文件格式进行排序，结果如图：    可以看到可执行文件被染成了紫色，压缩文件被染成了绿色，文档文件被染成了青色，图片文件被染成了黄色……接下来我们设置不含的字符串“3”，并且只搜索图片，结果如图所示：    只有图片被遗留了下来。  接下来展示结果四种类型的结构可视化：（具体类型/大类——柱状图/饼图）          最后是对MySQL中的历史记录表（search\_history）以及搜素结果可视化的具体类(specific\_kind)/大类(kinds)表进行结果展示：          </张嘉琦>  <张圳>  2.可改进的地方  我们发现，实际程序的运行时间中，90%以上的时间被消耗在结果的展示上。我们可以在这里着手进行修改，采用多线程的方式，利用“消费者-生产者”模型，创建一个搜索线程和一个展示线程。搜索线程将搜索到的每一个符合要求的结果放入临界区（例如Java中的阻塞队列），展示线程从临界区中取出结果进行展示。多线程的方法可以使得程序的性能得到一定程度上的提升。  **七、讨论、心得**  在最开始实现本项目的核心基础功能——搜索功能时，我们尝试过很多不同的方法。最开始的思考方向是在Windows自带的文件搜索功能以及搜索软件Everything，得到了两种解决方案：第一，通过提前对电脑上的磁盘建立索引，例如Everything通过读取对USN日志操作来建立索引，一旦建立好索引后，后续查找的速度就会非常快。但是这种方法每次建立索引的时间比较长，对代码的要求比较高，并且用Java语言实现相对困难；第二，采用最简单的遍历和筛选来实现搜索功能，这种方法简洁明了，非常直观。虽然每次搜索耗费的时间相比第一种方法更长，但是对于一般的需求已经可以接受。并且这种搜索方式便于扩展，因此我们采用了第二种方案。通过查阅大量资料，我们采用了Java8新特性NIO中的walkFileTree方法来实现。  在编写程序中仍然遇到了许多问题，例如对筛选的结果进行过滤和排序时，Java的Swing库自带的JTable功能并不好用，需要自己重新编写，并将原本的功能禁用；对表格进行渲染上色需要自己编写渲染类继承自JTable，并且当渲染条目过多时，原本的if…else方法必须进行重构，否则会造成卡顿，最后采用了哈希树和switch语句才避免了这个问题……最大的问题出现在团队成员的交流中，由于需求不明确，设计不完整，在编写代码时会遇到许多问题，写出的代码不易读、不易扩展，为团队间交流造成了许多阻碍。最终通过查找资料和团队成员讨论，通过阅读Java相关方法的API文档才得以解决各种问题。实验过程中团队成员收获颇深，无论是对操作系统运行方式的理解，还是自身的编码能力，抑或是团队协作软件工程的经验都是一种提高。  </张圳> | | | |