**面向对象的程序设计及JAVA语言**

**大作业报告**

|  |  |
| --- | --- |
| **作业选题：** | **文献题录分析软件** |
| **专业：** | **数据科学与大数据技术** |
| **指导老师：** | **陈果** |
| **组内成员：** | **那铭心（921127940133） 沈贝宁（921127940137）**  **顾翔（921127940122） 季诚（921127940125）** |

**南京理工大学网络空间安全学院**

**2023年 6 月**

**目录**

[一、设计思路与小组分工 1](#_Toc137137617)

[二、数据结构及其说明 2](#_Toc137137618)

[ 散列映射（HashMap） 3](#_Toc137137619)

[ 链表（LinkedList） 4](#_Toc137137620)

[三、用户界面截图与说明 4](#_Toc137137621)

[ 开始界面 4](#_Toc137137622)

[ 统计功能界面 7](#_Toc137137623)

[ 分析功能界面 16](#_Toc137137624)

[四、数据库各表截图与说明 22](#_Toc137137625)

[ Paper Information数据表 22](#_Toc137137626)

[ Author Address Frequency数据表 23](#_Toc137137627)

[ Author Frequency数据表 23](#_Toc137137628)

[ Keywords Frequency数据表 24](#_Toc137137629)

[五、主要类代码及说明 25](#_Toc137137630)

[ createPaperInfo类 25](#_Toc137137631)

[ DeleteAllTables类 27](#_Toc137137632)

[ EstablishConnection类 29](#_Toc137137633)

[ GenericCreateTable类 29](#_Toc137137634)

[ GenericQuery类 32](#_Toc137137635)

[ analyticalFunction包中的WordsFreq类和GetResult类 34](#_Toc137137636)

[ statisticalFunction包中的WordsFreq类和GetResult类 37](#_Toc137137637)

[ FrequencyCount类 38](#_Toc137137638)

[ CooccurrenceCount类 40](#_Toc137137639)

[ TopNDistribution类 41](#_Toc137137640)

一、设计思路与小组分工

因为我们要开发一款文献题录分析软件，需要同时包含用户界面互动与数据操作，所以我们考虑使用MVC结构来设计程序的结构。  
 MVC（Model-View-Controller）是软件工程中的一种软件架构模式，它把软件系统分为模型、视图和控制器三个基本部分：  
 MVC架构中的模型（M）：模型负责封装应用的状态，实现应用的功能。我们将功能分别封装在不同的包中便于理解与后续维护。

整个文献题录分析软件功能实现一共分为4个package，最重要的一个是genericOperation包，这里定义了一系列通用操作，包括与Access数据库建立连接、创建及删除数据表、查询数据表相关信息；其余3个包分别用于实现将题录文件导入数据库、统计功能和分析功能，它们的具体实现都依赖于genericOperation包。

首先，我们需要实现导入题录功能，将从知网上获取的.net文件导入Access数据库中。代码为importPaper包中的CreatePaperInfo类中，通过把.net文件题录信息遍历、正则化匹配、按行插入到数据表，实现了在数据库中建立一个名为Paper information的数据表来存储这些题录信息。

其次，针对统计功能，包括关键词频次统计、作者统计、机构统计以及关键词修正、机构修正。这些功能的具体实现代码封装在statisticalFunction包中。

针对分析功能，分析功能包括词共现分析、查询关键词对应的前10名作者、期刊、机构分布；作者合著分析；机构合作分析、查询机构对应的前10名作者、期刊、前30名关键词分布。实现代码封装在analyticalFunction包中。  
 MVC架构中的视图（V）：视图是将模型的内容呈现出来，用户可以通过对视图的操作进而操作模型，封装的是对数据模型Model的显示。我们借助Java的java.awt和javax.swing等实现用户图形界面。代码封装在userInterface包的GraphicView.java文件中。

用户界面整体上包含引导界面与功能界面这两个面板。用户在引导界面进行文件的选择，并点击“题录分析”，同时跳转到功能界面；功能界面是一个水平拆分窗格，左区域为菜单面板作为功能选择区域，用户可以在此区域看到导入的题录文件、选择使用统计功能或分析功能，或是重新选择题录文件进行另一次题录分析；右区域是具体的功能实现界面，使用了CardLayout布局：在未选择功能前，会提示用户选择功能；在选择功能后，会切换显示对应功能的界面。统计功能和分析功能界面均使用BorderLayout布局，查询结果（以表格的方式呈现）放在CENTER位置，提示引导用户使用查询功能的界面放在NORTH位置。  
 MVC架构中的控制器（C）。因为Model和View是严格分开的，甚至两者之间互相不知道彼此，因此需要控制器Controller作为中介连接他们两个，实现用户与界面的交互。这部分代码封装在userInterface包的Controller.java文件中。为了达到软件的设计要求，我们定义了15个控制器类，它们都是ActionListener、ItemListener等监视器接口的实现类。用户在界面执行诸如点击按钮、在下拉列表选择项的操作都会触发控制器执行相应代码，响应用户的请求。

整个软件的GUI界面操作流程图如图1、图2所示，图1展示了用户在引导界面的操作；图2展示了用户在功能界面的操作，包括统计功能和分析功能两个方面。

手机屏幕的截图

低可信度描述已自动生成

**图1. 引导界面操作**

图示

描述已自动生成

**图2. 功能界面操作**

**小组分工**：成员顾翔从CNKI网站上获取了2022年的所有文献题录文件，并完成了软件设计要求中的导入题录功能，进行软件用户界面的设计和完成最终的小组汇报；成员沈贝宁在成员顾翔工作的基础上，参考UI设计图，完成了统计功能、分析功能代码的编写和用户界面的实现；成员季诚熟读、理解成员沈贝宁的代码后，为代码加上了注释说明、对UI进行了美化，并且撰写完成实验报告；成员那铭心根据实验报告的内容制作了汇报PPT。

二、数据结构及其说明

### 散列映射（HashMap）

HashMap<K, V>是Java中的一个泛型类，它实现了Map接口，用于存储键值对的集合，K代表键，V代表值。在HashMap内部结构中，使用哈希表数据结构来存储数据，通过哈希算法将键映射到对应的存储位置。这样可以快速地插入、删除和查找键值对，使得HashMap在大多数情况下具有很高的性能。

在软件制作过程中，HashMap是频次统计功能实现的重要数据结构。将词语作为键K，词语对应的频次作为值V构造键值对存储在HashMap中，在遍历文献的过程中不断更新HashMap：使用getOrDefault方法，如果没有该键就将该键的值置为1，否则将该键对应的值增加1；最终可以获得频次统计结果，利用get方法能够很方便的获取某个词语的频次；也可以遍历整个HashMap将频次统计结果存入一个二维数组中进行下一步的处理。图3展示了HashMap的结构图：

图示

描述已自动生成

**图3. HashMap**

### 链表（LinkedList）

LinkedList<E>也是Java中的一个泛型类，它实现了List接口，并且使用链表数据结构来存储元素：每个节点包含该位置的元素<E>和指向前一个节点和后一个节点的引用。这种结构使得在插入和删除元素时具有较低的时间复杂度。

本程序中使用LinkedList的主要原因是Collections类提供的静态方法sort(List<E>)可以对List类进行排序，只需要将LinkedList类的实例上转型为List即可。在实现TopN词频统计与分析时，需要动态添加元素并用自定义的方法进行排序。具体实现时，我们自定义类WordsFreq实现Comrapable<T>泛型接口，在comparaTo方法中自定义了词频排序规则。之后把获得的频次信息全部转换为WordsFreq类的实例作为List的元素，调用Collections.sort(List<E>)对List进行排序。图4展示了LinkedList的结构图：

图表

描述已自动生成

**图4. LinkedList**

三、用户界面截图与说明

### 开始界面

运行程序，首先会弹出下图界面：

文本

描述已自动生成

**图5. 引导界面**

图5是“引导界面”，也是运行程序后出现的第一个界面startPanel。

startPanel采用BorderLayout布局，JLabel标签“文献题录分析系统”放在NORTH位置；“开始题录分析”按钮放在SOUTH位置；WEST位置嵌套一个布局BorderLayout的面板tipPanel，tipPanel的CENTER位置是一个存放提示信息“请选择需要导入的文件的文献题录文件”和“浏览”按钮的Box盒式容器；startPanel布局的CENTER位置是一个用于展示选择文件复选框的面板fileSelectedPanel。

以下是操作演示：

用户首先需要点击“浏览”按钮，选择需要导入的文件。用户点击“浏览”按钮后，因为“浏览”按钮注册了监视器FileChooseController，用户的点击触发ActionEvent事件，弹出文件选择对话框如图6所示，用户可以在此界面选择需要进行题录分析的后缀名为net的文献题录文件。

图形用户界面, 文本, 电子邮件

描述已自动生成

**图6. 文件选择界面**

选择文件后，监视器FileChooseController将用户选择的文件显示在引导界面，并带有复选框，如图7所示。此时用户仍可以借助文件前的复选框选择要分析的文件。选择完毕后，用户可以点击“开始题录分析”按钮开始题录分析。这里我们选择了2022年的所有8个题录文件进行题录分析。

文本

描述已自动生成

**图7.选择文件后的引导界面**

“开始题录分析”按钮注册了switch2FunctionController监听器，用户点击“开始题录分析”按钮，触发ActionEvent事件，在actionPerformed方法中，监视器会做出如下响应：

1. 把所有选中的文件读入到一个链表，并传递给主要功能实现面板；
2. 判断用户选择的文件是否有效（后缀名是否为net）和是否选择了题录文件；
3. 如果2成立，便会调用DeleteAllTables类的静态方法delete方法删除Database中的全部数据表，然后调用CreatePaperInfo类静态方法importPaperInfo构建数据库，将.net文件中所有文献的信息录入数据库中的Paper Information表中，最后切换到主要功能实现界面；
4. 重新实例化startPanel，恢复到初始状态。

如图8所示是点击“开始题录分析”按钮后进入的功能界面。

文本

中度可信度描述已自动生成

**图8. 功能界面**

这个界面是一个继承自JSplitPane水平拆分窗格的类实例，分为左侧的菜单栏和右侧的功能展示区域。

menuPanel面板是左侧窗格（图8绿色区域），其中嵌套了一个中间容器menuBox，容纳了所有menuPanel上展示的组件，包括一些提示性标签，用户导入的文件名，两个分别注册了Switch2StartController和ExportController的交互按钮以及一个注册了panelDisplaycontroller监视器的功能选择下拉列表。

displayPanel是右侧窗格（图8紫色区域），布局为CardLayout。其中添加了三个面板，它们分别为：initialPanel、statisticalPanel和analyticalPanel。initialPanel是从引导界面切换到主界面时的初始展示面板也就是上图显示的“功能区（请在左侧下拉列表中选择一个功能）”；statisticalPanel是统计功能的展示面板；analyticalPanel是分析功能的展示面板。

### 统计功能界面

* + **关键词频次统计**

用户在下拉列表中选择“统计”功能，触发ItemEvent事件，右侧功能展示界面的面板切换至statisticalPanel（统计功能的展示面板），如图9所示:

图形用户界面

描述已自动生成

**图9. 统计功能实现面板**

此时右侧功能区为statisticalPanel（统计功能的展示面板），布局是BorderLayout。staGuidePanel作为引导用户使用统计功能的面板放在statisticalPanel的NORTH位置（图9黄色区域）；staShowPanel作为展示用户使用统计功能的结果的面板放在statisticalPanel的CENTER位置（图9蓝色区域）。

staGuidePanel包含一个中间容器guideBox，该guideBox容器中存放了三个Box类的容器box1、box2、box3，这三个容器容纳了所有staGuidePanel上展示的组件，引导用户进行统计查询。box1存储了标签“统计信息名”和下拉列表，下拉列表中有“关键词”、“作者”、“关键词”三个选择，初始状态下拉列表没有选择，注册了staInfoSelectController监视器。box2存储了标签“单个查询”、文本框和单个查询的“开始查询”按钮，文本框初始状态下不可编辑，“开始查询”按钮注册了staInquiryController监视器；box3存储了标签“频次TOP N查询”、文本框和TopN查询的“开始查询”按钮，文本框初始状态下不可编辑，“开始查询”按钮也注册了staInquiryController监视器，与单个查询的“开始查询”按钮注册的是同一个监视器实例。

而staShowPanel主体是一个tableHintBox，包含用户查询结果表格及其标题，其中表格组件被放入JScrollPane实例上，在查询内容行数过多屏幕无法全部显示时用户可以通过滚动条查看查询内容；另外，如果用户进行的是“关键词”或是“机构”字段的相关查询，在staShowPanel的SOUTH位置还会展示modificationPanel面板，实现查询结果修正功能。

以下是操作演示：

图形用户界面

描述已自动生成

**图10. 选择频次统计字段**

如图10，此时用户可以下拉“统计信息名”下拉列表，进行选择，以“关键词”为例，触发ItemEvent事件，监视器使文本框可编辑，且两个文本框注册staEditController监视器，以监视文本框是否获得焦点：如果用户没有在文本框中输入内容且文本框内容为空，文本框会出现灰色的提示文字“输入待查询的关键词名”，若用户开始在文本框中输入内容，灰色的提示文字会消失。

用户可以进行单个查询，点击文本框，以输入“美国”为例，点击“开始查询”按钮，触发ActionEvent，staInquiryController监视器会创建FrequencyCount类实例table并调用createFreqTables方法生成关键词频次表。之后调用重载的freqInquiry实例方法freqInquiry(String)获得关键词为“美国”的频次查询结果。最后，actionPerformed方法会调用staShowPanel的相应重载初始化方法init(String, String)，将查询结果展示在staShowPanel中，如图11所示，美国作为关键词一共出现了37次。

图形用户界面, 应用程序

描述已自动生成

**图11. 关键词为“美国”的频次查询界面**

如果查询结果为空即所请求的词语不存在，则会弹出消息对话框提示读者，如图12所示：

图形用户界面, 应用程序

描述已自动生成

**图12. 查询词语不存在时提示对话框**

用户也可以进行频次TOP N查询，调用重载的freqInquiry(int)方法即可获得查询结果。图13展示了关键词词频TOP N查询的结果图（以N为50为例）。

图形用户界面, 应用程序, 表格

描述已自动生成

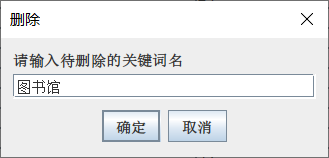
**图13. 关键词Top 50查询结果**

* **统计关键词修正**

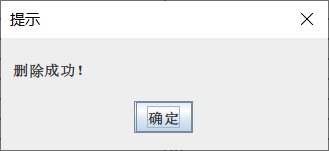
该功能由修正面板modificationPanel中的三个按钮组件实现。modificationPanel位于staShowPanel的SOUTH位置，展示统计结果表格的容器则tableHintBox位于其CENTER位置。修正面板中，包含“删除”“删除所在行”“合并”三个按钮，三个按钮分别注册了delController、delsController、combineController监视器。

以下是操作演示：

要实现删除某个词语，用户需要点击右下角处的“删除”按钮，点击“删除”按钮，会触发ActionEvent事件，监视器会弹出名为“删除”的用户输入对话框。用户在对话框中输入需要删除的关键词，以“图书馆”为例，点击“确定”后，表格中“图书馆”所在行被成功删除；另外考虑到用户需求我们设计了删除多行的按钮，用户只要选中希望删除的行，再点击“删除所选行”按钮即可实现删除。演示结果如图14、15、16、17所示。



**图14. 以删除“图书馆”关键词为例**



**图15. “图书馆”成功删除后的消息对话框**

查询结果staShowPanel显示“图书馆”所在行确实已被删除，如图16所示。

图形用户界面, 应用程序

描述已自动生成

**图16. 删除功能演示（“图书馆”成功被删除）**

图形用户界面, 应用程序, 表格

描述已自动生成 图形用户界面, 应用程序

描述已自动生成

**图17. 多行删除功能演示**

要实现将某个词语的频次合并到另一词语中，需要点击“合并”按钮，监视器combineController会弹出“合并”对话框，用户需要在对话框中输入“合并源”和“合并目标”，如图18所示。

图形用户界面, 应用程序

描述已自动生成

**图18. 合并功能对话框**

以“美国”为“合并源”，“区块链”为“合并目标”，点击确定后的效果图如图19所示：

图形用户界面, 应用程序, 表格

描述已自动生成

**图19. 合并结果演示**

可以看出原查询结果中的“美国”行被删除，且“区块链”的出现频次修改为“美国”和“区块链”的出现频次之和，且根据TOP N排序，“区块链”所在行被移动到了第一位。

* **作者统计**

需要用户点击右上角的“统计信息名”下拉列表，选择“作者”，其余功能实现与关键词频次统计相似。以统计作者出现频次TOP 80为例进行演示如图20所示：

图形用户界面, 应用程序, 表格

描述已自动生成

**图20. 作者频次Top80查询结果**

* **机构统计**

需要用户点击右上角的“统计信息名”下拉列表，选择“机构”，其余功能实现类似于关键词频次统计功能。以查询机构名为“渤海大学”为例进行演示如图21所示，渤海大学仅出现了1次。

图形用户界面, 应用程序, Word

描述已自动生成

**图21. 机构为“渤海大学”的频次查询结果**

* **机构修正**

类似于关键词修正功能，用户点击“删除”和“合并”按钮即可实现相应功能。这里首先查询Top 100机构出现频次，然后点击“删除”按钮选择删除“北京大学信息管理系”；点击“合并”按钮选择合并源为“北京大学图书馆”，合并目标为“湘潭大学公共管理学院”。软件运行界面分别如图22、图23、图24所示。

图形用户界面, 应用程序

描述已自动生成

**图22. 机构Top100频次查询结果**

图形用户界面, 应用程序, 表格, Excel

描述已自动生成

**图23. 删除“北京大学信息管理系”演示界面**

图形用户界面, 应用程序

描述已自动生成

**图24. 把“北京大学图书馆”合并到“湘潭大学公共管理学院”的演示界面**

### 分析功能界面

* **关键词共现分析**

用户在左侧菜单面板下拉列表中选择“分析”功能，监视器会切换右侧的功能显示面板到分析功能的展示面板analyticalPanel。如图25所示：



**图25. 合并结果演示**

与statisticalPanel一致，analyticalPanel同样使用BorderLayOut布局：

处于NORTH位置的是anaGuidePanel分析功能引导面板，在这个面板中，我们同样使用嵌套容器的方法进行布局。整体上是一个Box类的中间容器guideBox。guideBox容器又分为两部分，上半部分是共现查询的引导组件，下半部分是分布查询的引导组件。

处于CENTER位置的是anaShowPanel分析功能展示面板。同样，它的布局组织形式与“统计”功能中的staShowPanel相似。当用户选择“共现查询”或“分布查询”时，由于“分布查询”可能涉及到多个表格的同时展示，anaShowPanel的布局形式会相应做出改变。

当用户选择“共现查询”时，anaShowPanel里的组件仍然是表格及其标题的组合tableHintBox，与统计功能展示面板staShowPanel无异。

当用户选择“分布查询”时，tableHintBox上添加了tablePanel而不是直接一个带滚动条的表格组件；tablePanel展示了查询结果的若干个表格。tablePanel的布局为GridLayout（1行x列），其中x的含义为查询结果包含x个表格，这取决于用户选择了哪些分布信息进行查询。

以下是操作演示：

用户在“选择查询共现的字段”下拉列表内选择“关键词”，由于下拉列表注册了监视器AnaInfoSelectController，触发ItenEvent事件，与统计功能类似，相关文本框出现了提示性文本。此时以N=100为例，点击“TOP N查询”后的“开始查询”按钮（注册了AnaInquiryController监视器），查询结果如图26所示：

图形用户界面, 应用程序, 表格

描述已自动生成

**图26. 关键词共现Top 100查询结果**

用户也可以指定两个关键词，查询它们的共现信息。以“智慧图书馆”和“智慧服务”为例，查询结果如图27所示，这两个关键词共现了12次。

图形用户界面, 应用程序

描述已自动生成

**图27. 关键词“智慧图书馆”和“智慧服务”的共现频次查询**

* **查询关键词对应的前10名作者、期刊、机构分布**

用户需要在“选择需查询分布的字段”下拉列表内选择“关键词”并在文本框内输入需查询分布的关键词名。以“图书馆”为例，在文本框内输入“图书馆”并点击右侧的“开始查询”按钮，此时监视器会弹出图28所示的“选择待查询的分布信息”对话框，用户可以勾选复选框选择自己想要查询的分布字段。

图形用户界面, 文本, 应用程序

描述已自动生成

**图28. 选择查询分布字段对话框**

这里选择所有复选框，点击“确定”，查询结果如图29：

图形用户界面, 表格

描述已自动生成

**图29. 关键词“图书馆”的前十名作者、期刊、机构分布**

* + **作者合著分析**

作者合著分析：用户指定数目，输出TOP N个作者的合著信息；包括：作者1；作者2；合著频次；以N=100为例。查询结果如图30。

图形用户界面, 表格

描述已自动生成

**图30. Top100作者合著查询结果**

用户也可以指定两个作者的共现信息，这里以“曹树金”和“王志红”这两人的合著为例，查询结果如图31，他们共合著了6篇论文。

图形用户界面, 应用程序

描述已自动生成

**图31. “曹树金”和“王志红”两人合著查询结果**

* + **机构合作分析**

机构合作分析：用户指定数目，输出TOP N个机构的合作信息；包括：机构1；机构2；合作频次。以N=50为例。查询结果如图32所示：

图形用户界面, 应用程序

描述已自动生成

**图32. 机构合作Top50查询结果**

用户也可以指定查询两个机构的合作次数查询，这里以“北京外国语大学”和“国家信息中心”为例，查询结果如图33所示，两者一共合作了两次。

图形用户界面, 应用程序

描述已自动生成

**图33. 机构“北京外国语大学”和“国家信息中心”合作查询结果**

* + **查询机构对应的前10名作者、期刊、前30名关键词分布**

用户需要在“选择需查询分布的字段”中，下拉列表内选择“机构”并在文本框内输入需查询分布的机构名。以“武汉大学信息管理学院”为例，在文本框内输入“武汉大学信息管理学院”并点击右侧的“开始查询”按钮，会弹出下图“选择带查询的分布信息”对话框，如图34所示。

图形用户界面, 文本, 应用程序

描述已自动生成

**图34. 选择查询分布的字段为作者和关键词**

这里选择“关键词”和“作者”复选框，点击“确定”，查询结果如图35所示：

图形用户界面

中度可信度描述已自动生成

**图35. 武汉大学信息管理学院机构的作者Top10和关键词Top30分布查询结果**

四、数据库各表截图与说明

### Paper Information数据表

如图36所示，Paper Information数据表共有3540行，说明导入的文献题录共有3540条；共有ID、Reference Type、Title、Author、Author Address等17个字段：ID标识该文献索引，作为数据表的主键；其余列诸如Reference Type、Title、Author、Author Address等记录了文献的信息，每一个信息都以分号结束。

该表是将.net文件中各文献信息导入数据库后的表。用户选择文件后，点击“开始题录分析”，switch2FunctionController监视器会调用CreatePaperInfo类中importPaperInfo方法创建Paper Information数据表。

电脑屏幕截图

中度可信度描述已自动生成

**图36. Paper Information数据表**

### Author Address Frequency数据表

如图37所示Author Address Frequency数据表共有2088行，表示所有题录文件中出现了2088个机构；共有ID、Author Address、Frequency三个字段：ID列标识了记录索引，作为数据表的主键；Author Address列记录了机构名，Frequency列记录了机构对应的频次。

当用户选择“统计”功能，统计信息名选择“机构”，无论进行单个查询还是TopN查询，点击“开始查询”后, staInquiryController监视器都会创建FrequencyCount实例，并调用createFreqTables方法生成Author Address Frequency数据表。

表格

描述已自动生成

**图37. Author Address Frequency数据表**

### Author Frequency数据表

如图38所示，Author Frequency数据表共有5371行，表示所有题录文献共出现了5371个不同的作者；共有ID、Author、Frequency三个字段：ID列是文献索引，作为主键；Author列记录了作者名，Frequency列则是作者对应的频次。

同样，当用户选择“统计”功能，统计信息名选择“作者”，无论进行单个查询还是TopN查询，点击“开始查询”后, staInquiryController监视器都会创建FrequencyCount实例，并调用createFreqTables方法生成频次表。

表格

描述已自动生成

**图38. Author Frequency数据表**

### Keywords Frequency数据表

如图39所示，Keywords Frequency数据表共有8769行，表示所有题录文献共出现了8769个不同关键词；共有ID、Keywords、Frequency三个字段：ID列记录了文献的索引，作为主键；Keywords列记录了关键词，Frequency列记录了该关键词对应的频次。

当用户选择“统计”功能，统计信息名选择“机构”，无论进行单个查询还是TopN查询，点击开始查询后，staInquiryController监视器创建FrequencyCount实例，并调用createFreqTables方法生成频次表。

表格

描述已自动生成

**图39. Keywords Frequency数据表**

以上各表均可以通过主要功能界面的“导出数据表”按钮导出到用户指定目录。

五、主要类代码及说明

这里将介绍软件所有功能实现涉及到的类。

### createPaperInfo类

这个类实现了将.net后缀的题录文件导入到数据库的功能，定义了一个静态方法 importPaperInfo(GenericCreateTable, List<String>)，它会利用GenericOperation包中的GnericCreateTable类创建所有题录文件形成的数据表；类中另外还有一个私有静态方法String[] getInfo(StringBuffer, String)，它使用正则表达式，实现了对题录文件的信息捕获。

具体说明见代码注释：

package importPaper;

import java.io.*\**;

import java.util.*\**;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import genericOperation.GenericCreateTable;

public class CreatePaperInfo { *// 该类用于根据用户选择的题录文件创建Paper Information数据表*

    public static void importPaperInfo(GenericCreateTable table, List<File> files) { *// fileNames列表存储了所有用户选择的题录文件名*

        for (File file : files) {

            try {

                Reader in = new FileReader(file);

                BufferedReader br = new BufferedReader(in); *// 创建缓冲流进行读文件*

                String line; *// line是从文件中读出来的每一行内容*

                StringBuffer rawInfoBlock = new StringBuffer(); *// rawInfoBlock包含一篇论文的所有信息*

                while ((line = br.readLine()) != null) {

                    if (line.equals("")) { *// 如果line为空，说明一篇论文已经读完，需要将其插入到table中*

                        String[] infoBlock = getInfo(rawInfoBlock, "\\{([^\\}]+)\\}:\\s\*([^\\{]+)"); *// 调用静态方法getInfo获取一篇论文的infoBlock*

                        table.insertToTable(infoBlock); *// 将infoBlock插入表中*

                        rawInfoBlock = new StringBuffer(); *// rawInfoBlock重新实例化，接收下一篇论文的信息*

                    } else {

                        rawInfoBlock.append(line); *// 在rawInfoBlock后追加新读入的行*

                        if (rawInfoBlock.charAt(rawInfoBlock.length() - 1) != ';') { *// 为了以后更容易进行查询，在每个字段值读入后都会加上分号*

                            rawInfoBlock.append(";");

                        }

                    }

                }

                br.close();

                in.close();

            } catch (Exception e) {

                e.printStackTrace();

            }

        }

    }

    private static String[] getInfo(StringBuffer rawString, String regex) {*// 该静态方法利用正则表达式，捕获rawInfoBlock中的论文相关信息*

        Pattern pattern = Pattern.compile(regex);

        Matcher matcher = pattern.matcher(rawString);

        LinkedHashMap<String, String> infoMap = new LinkedHashMap<>(); *// infoMap用于存储该论文每个字段对应的字段值*

        String[] infoKeys = { "Reference Type", "Title", "Author", "Author Address", "Journal", "Year", "Volume",

                "Issue", "Pages", "Keywords", "Abstract", "ISBN/ISSN", "Notes", "URL", "DOI", "Database Provider" };

*// infoKeys记录了Paper Information数据表中的所有可能的字段名*

        for (String infoKey : infoKeys) {

            infoMap.put(infoKey, "");

        }

        while (matcher.find()) {

            String infoKey = matcher.group(1); *// 捕获的字段名*

            String infoValue = matcher.group(2).trim(); *// 捕获的字段值，应该去除首尾的无效空格*

            infoMap.put(infoKey, infoValue);

        }

        String[] infoBlock = new String[16];

        int i = 0;

        for (String infoValue : infoMap.values()) {

            infoBlock[i++] = infoValue;

        }

        return infoBlock; *// infoBlock数组的每个元素代表相应字段值，如果该字段值不存在，用""存储*

    }

}

### DeleteAllTables类

这个类是用于删除数据库中所有的表，当用户在引导界面点击“开始题录分析”按钮后会调用这个类的静态方法delete对用户上一次题录分析得到的数据表进行删除。这个操作是可行的，因为在用户完成上一次题录分析后会提醒用户是否需要导出分析过程中得到的数据表。

具体说明见代码注释：

package genericOperation;

import java.sql.*\**;

public class DeleteAllTables {  *// 该类用于删除所有数据表*

    public static void delete() {

        Connection con;

        Statement stmt;

        try {

            EstablishConnection.loadDriver();

            con=EstablishConnection.connectDatebase("D:/Temp/Database.accdb");  *// 调用EstablishConnection类中的connectDatabase静态方法建立连接*

            DatabaseMetaData metaData = con.getMetaData();

            ResultSet resultSet = metaData.getTables(null, null, null, new String[]{"TABLE"});  *// resultSet包含数据库中所有数据表的名称*

            while (resultSet.next()) {

                String tableName = resultSet.getString("TABLE\_NAME");

                if (tableName.contains(" ") || tableName.contains("/")) {  *// 表名中可能含有空格和正斜杠，需要将其格式化，否则SQL语句解析异常*

                    tableName = "[" + tableName + "]";

                }

                String sql = "drop table " + tableName;  *// 创建SQL语句删除表*

                stmt = con.createStatement();

                stmt.executeUpdate(sql);  *// 执行该语句，数据表被删除*

            }

        }

        catch(SQLException e) {

            e.printStackTrace();

        }

    }

}

### EstablishConnection类

这个类用于连接数据库，定义了两个静态方法loadDriver()和Connection connectDatebase(String)，分别加载驱动和连接数据库。

具体说明见代码注释：

package genericOperation;

import java.sql.*\**;

public class EstablishConnection {  *// 该类用于Java与Access数据库建立连接*

    public static void loadDriver() {

        try {

            Class.forName("com.hxtt.sql.access.AccessDriver");  *// 加载JDBC-Access数据库驱动程序*

        }

        catch (ClassNotFoundException e) {}

    }

    public static Connection connectDatebase(String databasePath) {  *// dataPath指定数据库文件路径，调用该静态方法即可建立连接*

        Connection con = null;

        try {

            con = DriverManager.getConnection("jdbc:access:/" + databasePath);

        }

        catch (SQLException e) {}

        return con;

    }

}

### GenericCreateTable类

用于创建数据库中的表，定义了三个方法：

* + 构造方法GenericCreateTable(String, String[])，以表名和字段名实例化对象，相当于创建了一个数据表模板；
  + boolean initialTable()方法首先询问数据库是否存在相同表名的数据表，如果存在返回false；否则会创建由构造方法指定的数据表，然后返回true；
  + insertToTable(String[])方法将单个文献的信息按行插入表中。

具体说明见代码注释：

package genericOperation;

import java.sql.*\**;

public class GenericCreateTable {  *// 该类用于创建数据表*

    private String tableName, formatTableName;  *// tableName是表名，formatTableName指格式化后的表名（表名中如果有空格等字符，需要将表名放在中括号，否则SQL语句无法正常解析）*

    private String[] infoNames;  *// infoNames指数据表的字段名称*

    public GenericCreateTable(String tableName, String[] infoNames){  *// 以表名和字段名来构造GenericCreateTable实例*

*this*.tableName=tableName;

        formatTableName = tableName;

        if (tableName.contains(" ") || tableName.contains("/")) {  *// 表名中可能含有空格和正斜杠，需要将其格式化，否则SQL语句解析异常*

            formatTableName = "[" + tableName + "]";

        }

*this*.infoNames=infoNames;

    }

    public boolean initialTable() {  *// 创建数据表，如果表已存在，则会返回false，否则返回true*

        Connection con;

        Statement statement;

        StringBuffer formatInfoNames=new StringBuffer();

        try {

            EstablishConnection.loadDriver();

            con=EstablishConnection.connectDatebase("D:/Temp/Database.accdb");  *// 调用EstablishConnection类中的connectDatabase静态方法建立连接*

            DatabaseMetaData metaData = con.getMetaData();

            ResultSet rs = metaData.getTables(null, null, tableName, null);

            if (rs.next()) {  *// 说明表已存在，不会进行任何操作*

                return false;

            }

            statement = con.createStatement();

            for (String infoName: infoNames) {  *// 遍历infoNames中的每一个字段名构造SQL语句，与表名一样，字段名也需要格式化*

                if (infoName.contains(" ") || infoName.contains("/")) {

                    formatInfoNames.append(", [" + infoName + "] VARCHAR(100)");

                }

                else {

                    if (infoName.equals("Frequency")) {  *// 频次列的字段类型为整型int*

                        formatInfoNames.append(", " + infoName + " int");

                    }

                    else{

                        formatInfoNames.append(", " + infoName + " VARCHAR(100)");

                    }

                }

            }

            statement.executeUpdate("create table " + formatTableName + " (ID int primary key" + formatInfoNames + ")");  *// 执行SQL语句，创建相应表*

            con.close();

        }

        catch (SQLException e) {

            e.printStackTrace();

        }

        return true;

    }

    public void insertToTable(String[] infoBlock) {  *// 该实例方法用于将infoBlock（代表一行数据）插入数据表中；即每次插入一条记录*

        Connection con;

        StringBuffer formatInsertion=new StringBuffer();

        try {

            EstablishConnection.loadDriver();

            con=EstablishConnection.connectDatebase("D:/Temp/Database.accdb");  *// 调用EstablishConnection类中的connectDatabase静态方法建立连接*

            Statement stmt = con.createStatement();

            ResultSet rs = stmt.executeQuery("select count(\*) from " + formatTableName);

            rs.next();

            int id = rs.getInt(1) + 1;  *// rs.getInt(1)获得的就是当前数据表最后一个记录的ID字段值，将它加1就是现在待插入的记录的ID字段值*

            for(int i=0; i<infoBlock.length; i++) {

                formatInsertion.append(", ?");

            }

            String sqlStr="insert into " + formatTableName + " values (?" + formatInsertion + ")";

            PreparedStatement preparedStmt = con.prepareStatement(sqlStr);  *// 使用预处理语句进行记录插入*

            preparedStmt.setInt(1, id);

            int location=2;

            for (String infoValue:infoBlock) {  *// 遍历infoBlock，在相应位置插入字段值*

                preparedStmt.setString(location, infoValue);

                location+=1;

            }

            preparedStmt.executeUpdate();

            con.close();

        }

        catch (SQLException e) {

            e.printStackTrace();

        }

    }

}

### GenericQuery类

这个主要是用于对表的查询操作，定义了四个方法：

* + 构造方法GenericQuery(String)初始化私有成员queryStr，即设定此次查询所使用的SQL语句；
  + 私有方法Query() 通过 SQL语句queryStr，对表进行查询，并逐行将表录入二维数组info；
  + String[][] fetchInfo() 调用私有方法Query后返回查询结果info。

具体说明见代码注释：

package genericOperation;

import java.sql.*\**;

public class GenericQuery { *// 通用查询类，给定一个SQL查询语句（形如select \* from tableName where ），获得语句的查询结果*

    private String queryStr; *// 需执行的SQL查询语句*

    private String[][] info; *// 查询结果，n条记录，m列字段*

    public GenericQuery(String queryStr) { *// 用SQL查询语句来构造实例*

*this*.queryStr = queryStr;

        info = null; *// info被初始化为null*

    }

    public String[][] fetchInfo() { *// 类的实例通过调用fetchInfo方法即可获得查询结果info*

        Query();

        return info;

    }

    private void Query() { *// 根据SQL查询语句进行查询，并将结果记录在info二维String数组中*

        Connection con;

        Statement sql;

        ResultSet rs; *// 结果集*

        try {

            EstablishConnection.loadDriver();

            con = EstablishConnection.connectDatebase("D:/Temp/Database.accdb"); *// 调用EstablishConnection类中的connectDatabase静态方法建立连接*

            sql = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_READ\_ONLY); *// rs的游标可以上下移动，更容易实现查询操作*

            rs = sql.executeQuery(queryStr); *// 执行该SQL查询语句*

            ResultSetMetaData rsMetaData = rs.getMetaData(); *// 结果集rs调用getMetaData方法返回一个结果集的元数据对象rsMetaData，rsMetaData可以获得结果集的列数*

            rs.last(); *// 将游标移动到结果集尾端，再调用rs.getRow()即可获得结果集的行数*

            int rowCount = rs.getRow(), colCount = rsMetaData.getColumnCount(), id = 0;

            if (rowCount != 0) { *// 如果结果集存在记录，就将查询结果赋值给info，否则info仍为null*

                info = new String[rowCount][colCount];

            }

            rs.beforeFirst(); *// 将rs游标移动到第一条记录之前*

            while (rs.next()) {

                for (int i = 0; i < colCount; i++) {

                    info[id][i] = rs.getString(i + 1);   *// 取出rs指向的记录每一个字段的取值并赋值给info*

                }

                id++;

            }

            con.close();

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

}

### analyticalFunction包中的WordsFreq类和GetResult类

WordsFreq类是一个词与词频二元组类，有两个成员word和freq，代表word出现的频次。它实现了泛型接口Comparable<T>，comparaTo方法中自定义了排序规则：先按频次降序再按拼音序升序。

GetResult 类，定义静态方法 getTopN和getOne分别用于返回 TopN查询结果和单个词语词频查询结果。

具体说明见代码注释：

package analyticalFunction;

import java.text.Collator;

import java.util.*\**;

class WordsFreq implements Comparable<WordsFreq> {

*// 创建该类的目的是实现自定义排序（先按频次降序再按拼音升序），实现了Comparable<T>泛型接口，从而可以使用Collection类的sort方法对列表进行排序*

    String word;

    int freq;

    WordsFreq(){}

    WordsFreq(String word, int freq){

*this*.word = word;

*this*.freq = freq;

    }

    public int compareTo(WordsFreq wf) {

        int res = wf.freq - freq;  *// 先比较词频*

        if (res == 0) {  *// 如果词频一致，则按拼音升序*

            Collator collator = Collator.getInstance(Locale.CHINA);

            res = collator.compare(word, wf.word);

        }

        return res;

    }

}

class GetResult {  *// 该类定义两个静态方法，getTopN用于获取TopN结果，getOne用于获取单个查询的结果*

    static String[][] getTopN(String[][] info, int n, int colIdx, boolean isCooccurrence){

*// isCooccurrence判断本次查询结果是否为共现查询，如果是共现查询，colIdx直接被指定为0（因为info中只存在查询字段的信息）；如果是分布查询，colIdx由TOPNDistribution类的静态成员指定*

        HashMap<String, Integer> freqMap = new HashMap<>();

        for (String[] item: info) {

            String[] words = item[colIdx].split(";");

            int len = words.length;

            if (isCooccurrence) {  *// 共现查询的处理方式*

                for (int i = 0; i < len - 1; i ++) {

                    for (int j = i + 1; j < len; j ++) {

                        String word1 = words[i].trim();  *// 去除words[i]首尾的无效空格*

                        String word2 = words[j].trim();  *// 去除words[j]首尾的无效空格*

                        String word12 = (word1.compareTo(word2) < 0) ? word1 + ";" + word2 : word2 + ";" + word1;

                        freqMap.put(word12, freqMap.getOrDefault(word12, 0) + 1);

                    }

                }

            }

            else {  *// 分布查询的处理方式*

                for(int i = 0; i < len; i ++) {

                    String word = words[i];

                    if (! word.equals("")) {

                        freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);

                    }

                }

            }

        }

        List<WordsFreq> freqList = new LinkedList<>();

        for (String key: freqMap.keySet()) {  *// 将freqMap的元素转移到freqList中，从而可以进行自定义的排序*

            WordsFreq wordsFreq = new WordsFreq(key, freqMap.get(key));

            freqList.add(wordsFreq);

        }

        Collections.sort(freqList);  *// 按词频降序，拼音升序排序*

        Iterator<WordsFreq> iter = freqList.iterator();

        int cnt = 0;

        String[][] result = new String[n][2];

        while(iter.hasNext()) {  *// 用迭代器遍历freqList，将结果赋值给result*

            if (cnt < n) {

                WordsFreq wordFreq = iter.next();

                result[cnt][0] = wordFreq.word;

                result[cnt][1] = String.valueOf(wordFreq.freq);

                cnt ++;

            }

            else {

                break;

            }

        }

        return result;

    }

    static String[][] getOne(String[][] info, String inquiryWord1, String inquiryWord2){  *// 根据传入的两个共现词来查询对应的共现频次，具体做法与getN一致*

        HashMap<String, Integer> freqMap = new HashMap<>();

        for (String[] item: info) {

            String[] words = item[0].split(";");

            int len = words.length;

            for (int i = 0; i < len - 1; i ++) {

                for (int j = i + 1; j < len; j ++) {

                    String word1 = words[i].trim();

                    String word2 = words[j].trim();

                    String word12 = (word1.compareTo(word2) < 0) ? word1 + ";" + word2 : word2 + ";" + word1;  *// 词语共现与先后次序无关，需要进行处理*

                    freqMap.put(word12, freqMap.getOrDefault(word12, 0) + 1);

                }

            }

        }

        String[][] result = new String[1][2];

        String inquiryWord12 = inquiryWord1 + ";" + inquiryWord2;

        result[0][0] =  inquiryWord12;

        result[0][1] = String.valueOf(freqMap.getOrDefault(inquiryWord12, null));

        return result;

    }

}

### statisticalFunction包中的WordsFreq类和GetResult类

这两个类作用与analyticalFunction包中的两个类一致，且实现上十分相似。

具体说明见代码注释：

package statisticalFunction;

import java.text.Collator;

import java.util.Locale;

import java.util.*\**;

class WordsFreq implements Comparable<WordsFreq> {  *// 同分析功能中的WordsFreq类，用于实现自定义的排序功能*

    String word;

    int freq;

    WordsFreq(){}

    WordsFreq(String word, int freq){

*this*.word = word;

*this*.freq = freq;

    }

    public int compareTo(WordsFreq wf) {

        int res = wf.freq - freq;  *// 首先按频次排序*

        if (res == 0) {

            Collator collator = Collator.getInstance(Locale.CHINA);  *// 如果频次一致，就按汉语拼音升序排列*

            res = collator.compare(word, wf.word);

        }

        return res;

    }

}

class GetResult {

    static String[][] getTopN(String[][] info, int n){  *// 相比分析功能中的TopN查询，由于这里创建了数据表，因此程序会简洁一些*

        List<WordsFreq> freqList = new LinkedList<>();

        for (String[] item: info) {  *// item[0]是一篇论文的关键词，item[1]是它对应的词频*

            WordsFreq wordsFreq = new WordsFreq(item[0], Integer.parseInt(item[1]));

            freqList.add(wordsFreq);

        }

        Collections.sort(freqList);

        Iterator<WordsFreq> iter = freqList.iterator();

        int cnt = 0;

        String[][] result = new String[n][2];  *// 将结果存入result后返回*

        while(iter.hasNext()) {

            if (cnt < n) {

                WordsFreq wordFreq = iter.next();

                result[cnt][0] = wordFreq.word;

                result[cnt][1] = String.valueOf(wordFreq.freq);

                cnt ++;

            }

            else {

                break;

            }

        }

        return result;

    }

}

### FrequencyCount类

这个类实现了字段值的频次统计。定义了四个方法：

* 构造方法FrequencyCount(String)，根据要统计的字段名实例化类对象。
* createFreqTables() 将从Paper Information表中得到的原始数据转化为可以插入到词频表中的数据，如果相应词频表不存在，则会创建并插入数据。
* String[][] freqInquiry(String)和String[][] freqInquiry(int)是两个重载方法，一个用来返回单个查询结果一个用来返回TopN查询结果。

package statisticalFunction;

import java.util.*\**;

import genericOperation.*\**;

public class FrequencyCount {  *// 该类实现了统计功能中的频次统计*

    private String colName, formatColName, tableName, formatTableName;  *// colName是频次统计的字段名，tableName是要创建的频次统计表的表名*

    public FrequencyCount(String colName) {

*this*.colName = colName;

        tableName = colName + " Frequency";

        formatTableName = tableName + " Frequency";

        if (tableName.contains(" ") || tableName.contains("/")) {  *// 表名中可能含有空格和正斜杠，需要将其格式化，否则SQL语句解析异常*

            formatTableName = "[" + tableName + "]";

        }

        formatColName = colName;

        if (colName.contains(" ") || colName.contains("/")) {

            formatColName = "[" + colName + "]";

        }

    }

    public void createFreqTables() {  *// 根据成员遍历colName创建频次统计表*

        String[] newInfoNames = {colName, "Frequency"};

        GenericQuery getInfo = new GenericQuery("select " + formatColName + " from [Paper Information]");  *// 新建GenericQuery实例getInfo，用于获取要统计的字段的所有信息*

        String[][] info = getInfo.fetchInfo();

        HashMap<String, Integer>freq = new HashMap<>();  *// 储存统计结果*

        for (String[] item: info) {

            for (String splitItem: item[0].split(";")) {

                freq.put(splitItem, freq.getOrDefault(splitItem, 0) + 1);

            }

        }

        GenericCreateTable table = new GenericCreateTable(tableName, newInfoNames);

        if (table.initialTable()) {  *// table.initialTable()返回结果为true表明表不存在，需要创建，否则无需创建*

            for (String key: freq.keySet()) {  *// 创建词频统计表并插入每一个统计信息*

                if (! key.equals("")) {

                    String[] infoBlock = {key, String.valueOf(freq.get(key))};

                    table.insertToTable(infoBlock);

                }

            }

        }

    }

    public String[][] freqInquiry(String word) {  *// 获取单个词语word的词频*

        word = word.trim();

        GenericQuery getInfo = new GenericQuery("select " + formatColName + ", Frequency from " + formatTableName + " where " + formatColName + " = '" + word + "'");

        String[][] info = getInfo.fetchInfo();

        return info;

    }

    public String[][] freqInquiry(int n) {  *// 获取TOP N词频查询结果*

        GenericQuery getInfo = new GenericQuery("select " + formatColName + ", Frequency from " + formatTableName);

        String[][] info = getInfo.fetchInfo();

        return GetResult.getTopN(info, n);

    }

}

### CooccurrenceCount类

CooccurrenceCount类实现了分析功能中的共现查询。该类主要包含两个重载的静态方法： CooccurrenceInquiry(String, String, String)用于查询给定字段（colName）中两个词（word1和word2）的共现次数；CooccurrenceInquiry(String, int)用于获取共现次数的TopN结果。

package analyticalFunction;

import genericOperation.*\**;

public class CooccurrenceCount {  *// 该类实现了分析功能中的共现查询*

    public static String[][] CooccurrenceInquiry(String colName, String word1, String word2) {  *// 查询word1和word2的共现次数，colName表征word1和word2来源于哪个字段*

        word1 = word1.trim();  *// 去除word1首尾的无效空格*

        word2 = word2.trim();  *// 去除word2首尾的无效空格*

        String formatColName = new String(colName);

        if (colName.contains(" ") || colName.contains("/")) {

            formatColName = new String("[" + colName + "]");

        }

        String queryStr = "select " + formatColName + " from [Paper Information]";

        GenericQuery getInfo = new GenericQuery(queryStr);

        String[][]  info = getInfo.fetchInfo();  *// 从Paper Information中先获得所有论文在colName字段的值，info为n行1列*

        if (word1.compareTo(word2) < 0) {

            return GetResult.getOne(info, word1, word2);  *// 调用GetResult的静态方法getOne获得查询结果*

        }

        else {

            return GetResult.getOne(info, word2, word1);

        }

    }

    public static String[][] CooccurrenceInquiry(String colName, int n) {  *// 实现了CooccurrenceInquiry方法的重载，用于获得TOP N的共现结果*

        String formatColName = new String(colName);

        if (colName.contains(" ") || colName.contains("/")) {

            formatColName = new String("[" + colName + "]");

        }

        String queryStr = "select " + formatColName + " from [Paper Information]";

        GenericQuery getInfo = new GenericQuery(queryStr);

        String[][]  info = getInfo.fetchInfo();

        return GetResult.getTopN(info, n, 0, true);  *// 调用GetResult的静态方法getN获得查询结果*

    }

}

### TopNDistribution类

TopNDistribution类用于实现分析功能中的分布查询。其中定义了四个方法：

* 构造方法TopNDistribution(String, String)，根据要查询分布的词语构造实例，并在方法体中调用init函数，获取该词语的所有分布信息info；
* init方法构造SQL查询语句，创建GenericQuery类实例进行分布查询；
* Boolean infoNotNull方法判断用户所请求的词语是否存在于记录中（如果存在，info不为null）
* String[][] distributionInquiry(String)方法根据用户请求的查询分布的字段名来返回分布查询结果。

具体说明见代码注释：

package analyticalFunction;

import genericOperation.*\**;

public class TopNDistribution {  *// 该类实现了分析功能中的分布查询*

    static int authorAddress = 0, keywords = 0, author = 1, journal = 2;  *// 静态成员将要查询分布的字段名映射为索引*

    private String tableName, formatTableName, formatWordName;  *// 在tableName列筛选含有wordName的论文信息，再进行分布查询*

    private String[][] info;  *// 分布查询结果*

    public TopNDistribution(String tableName, String wordName) {  *//根据tableName和wordName来构造实例，具体要查询什么分布，用户要调用DistributionInquiry方法*

*this*.tableName = tableName;

        formatTableName = tableName;

        if (tableName.contains(" ") || tableName.contains("/")) {  *// 表名中可能含有空格和正斜杠，需要将其格式化，否则SQL语句解析异常*

            formatTableName = "[" + tableName + "]";

        }

        formatWordName = wordName;

        if (wordName.contains(" ") || wordName.contains("/")) {  *// 词语名中也可能含有空格和正斜杠，同样需要进行格式化*

            formatWordName = "[" + wordName + "]";

        }

        init();  *// 初始化，获得tableName列值为wordName的所有分布信息*

    }

    private void init() {

        if (tableName.equals("Keywords")) {  *// 如果tableName是关键词，用户可能查询分布的字段为机构、作者、期刊*

            String queryStr = "select [Author Address], Author, Journal from [Paper Information] where " + formatTableName + " like '%" + formatWordName + ";%'";  *// wordName后要加分号表示整个词语，否则查询到的记也可能以wordName为前缀*

            GenericQuery getInfo = new GenericQuery(queryStr);   *// 使用genericOperation包中的GenericQuery类进行含有该词语的题录信息获取，存储在二维数组info中*

            info = getInfo.fetchInfo();

        }

        else {  *// 如果tableName是机构，用户可能查询分布的字段为关键词、作者、期刊*

            String queryStr = "select Keywords, Author, Journal from [Paper Information] where " + formatTableName + " like '%" + formatWordName + ";%'";

            GenericQuery getInfo = new GenericQuery(queryStr);

            info = getInfo.fetchInfo();

        }

    }

    public boolean infoNotNull() {  *// 判断tableName列值为wordName的记录是否存在，如果不存在返回false，否则返回true*

        if (info == null) {

            return false;

        }

        return true;

    }

    public String[][] distributionInquiry(String colName) { *// 根据tableName和colName，调用GetResult的静态方法getTopN获得分布查询结果*

        if (tableName.equals("Keywords")) {

            switch(colName) {

                case "Author":

                    return GetResult.getTopN(info, 10, author, false);

                case "Journal":

                    return GetResult.getTopN(info, 10, journal, false);

                default:  *// default其实代表 case "机构"*

                    return GetResult.getTopN(info, 10, authorAddress, false);

            }

        }

        else {

            switch(colName) {

            case "Author":

                return GetResult.getTopN(info, 10, author, false);

            case "Journal":

                return GetResult.getTopN(info, 10, journal, false);

            default:  *// default其实代表 case "关键词"*

                return GetResult.getTopN(info, 30, keywords, false);  *// 题目要求关键词要查询TOP 30*

            }

        }

    }

}