体育新闻检索系统

项目报告

小组成员：郭清沛：201428015029038

胡仁林：201428015029040

张宝婷：201428015029030

谢　敏：201428015029057

完成时间：二〇一四年十二月

目 录

1 系统概述

本大作业实现了一个新闻搜索系统，通过编写网络爬虫爬取了新浪体育新闻中的 10 万体育新闻，对网页内容进行预处理后，利用“结巴”分词系统对爬取的网页内容进行中文分词，并对所有词项构建倒排索引表。用户输入查询词进行查询时，可以将相关网页通过相关性排序进行呈现，给出与查询相关的文字内容摘要，同时，由于互联网用户普遍存在的惰性，本新闻搜索系统还能将查询自动补齐，对相似新闻进行聚类，提供鼠标预览功能，以及相关搜索推荐功能，以满足用户的各种需求。

本新闻搜索系统的前端由网页进行呈现，后端功能用 Python 进行实现，所有功能模块除中文分词采用了“结巴”分词系统以外，均由自己编写，未采用开源框架进行实现。

2 设计方案

一个搜索引擎大体包括如图 2-1 所示的一些功能模块，新闻搜索系统也不例外。

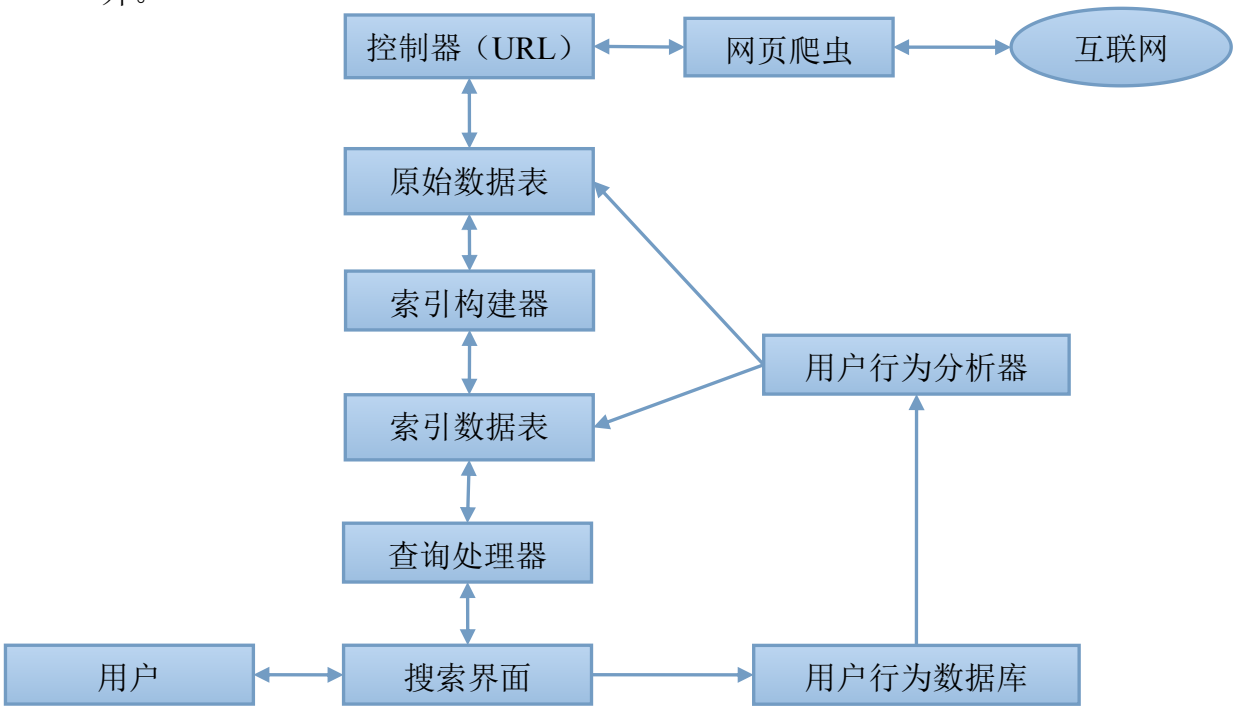


图 2-1 搜索引擎功能模块

本新闻搜索系统将从新浪体育中爬取的体育新闻预处理后存储在原始数据表中，通过索引构建器根据爬取的新闻构建其索引，并将索引存储在数据库中，用户进行查询时，查询处理器根据用户的查询进行相关处理并提交给交互界面，这样一个基本的新闻搜索系统就实现好了。但为了增加用户体验，还需增加一些功能模块对查询结果进行处理，以及对用户行为进行分析。包括对查询结果的排序，预览，摘要显示，相关新闻的聚类，以及基于用户行为的查询自动补齐和相关检索推荐。

为实现以上所述功能，本部分将详细描述本新闻搜索系统的设计方案，包括前端、后端设计方案和交互接口，以及底层数据库的设计方案。

2.1 前端设计方案

本系统的前端是由网页进行显示的，其搜索界面设计如图 2-2 所示：

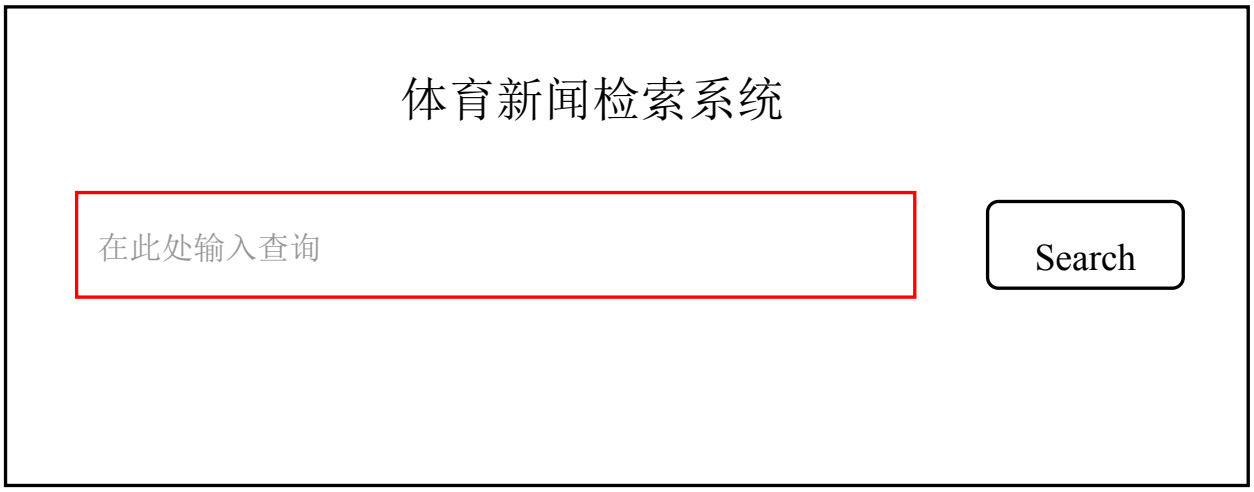


图 2-2 搜索界面

搜索结果展示设计如图 2-3 所示，每一条搜索结果都会给出一个蓝色的链接以进入该网页，显示网页内容中跟查询词相关部分的摘要，以及用绿色部分显示链接的 URL。同时将鼠标移到某一条搜索结果上时还能预览该结果的内容。在网页的右边会给用户推荐一些相关的搜索，以便用户查询。

图 2-3 搜索结果展示

前端不仅仅只是一个展示的平台，同时它也包含一些功能的实现，包含有与后端的通信功能，这部分内容将在 2.3 节进行讲述，查询自动补齐功能和预览功能，下面就对这两部分功能进行讲解。

1） 查询自动补齐

设置查询自动补齐功能可以方便用户查询，同时潜在地影响用户输入，当用户点击自动补齐的查询进行查询时，则搜索页面显示出来的是补齐后的查询所查找的内容。

为实现这一功能，在前端搭建一个单独的数据库以存放用户查询的历史记录，根据数据库中的历史记录，当用户输入某一查询的前几个字时，实时地从前端数据库中搜索与用户正在搜索的查询前缀匹配的历史查询记录，并在查询框下方显示出来。

2） 预览

预览功能即用户将鼠标置于某一条查询记录上时，捕获这一动作，然后在页面上返回后端发来的查询结果中的全文内容。

2.2 后端设计方案

根据系统需求，本系统后端的核心功能包括网络爬虫、索引构建、查询处理、以及和客户端的通信，下面将详细介绍这几部分的设计方案。

2.2.1 网络爬虫

网络爬虫是一个自动程序，可自动在互联网上搜索信息，查看网页内容，从中找到相关信息，然后从该网页的所有链接出发，继续寻找相关的信息。网络爬虫在抓取网页时一般有两种算法：广度优先和深度优先。广度优先是指网络爬虫会先爬取起始网页中链接的所有网页，然后再选择其中的一个链接网页，继续爬取在此网页中链接的所有网页；深度优先是指网络爬虫会从起始页开始，一个链接一个链接跟踪下去，处理完这条线路之后再转入下一个起始页，继续跟踪链接。

本系统的网络爬虫将采用广度优先算法，因为这个方法可以让网络爬虫并行处理，提高爬取速度。也就是说，我们将从新浪体育新闻主页出发，将该页面下的所有链接网页都爬取完后，再选择其中某一个网页链接继续爬取此网页中链接的所有相关网页。

由于通过网络爬虫爬取的数据都是原始网页数据，带有许多网页标签，如<p>等标签，因此需要对爬取的网页进行预处理，将标签剔除，另外爬虫还需将原始数据切分成几个块如标题、关键字等，以便之后进行查询处理。

2.2.2 索引构建

索引构建是一个信息检索系统的关键部分，索引构建的成功与否关系到查询结果的质量好坏。索引构建模块就是要构造词项的倒排记录表，然而构建倒排记录表的最基本的需求就是进行分词，只有先分好词才能进行索引的构建。

中文分词是一个巨大的工作量，不仅跟计算机学科相关，还关系到汉语言学的研究，因此我们采用了已有的中文分词系统——“结巴”中文分词。它支持三种分词模式：精确模式、全模式和搜索引擎模式，其搜索引擎模式完全符合本系统的需求，它能够试图将句子最精确地切开，并对长词进行再次切分，以提高系统的召回率。

索引构建主要分为 4 个步骤，基本流程如图 2-4 所示：

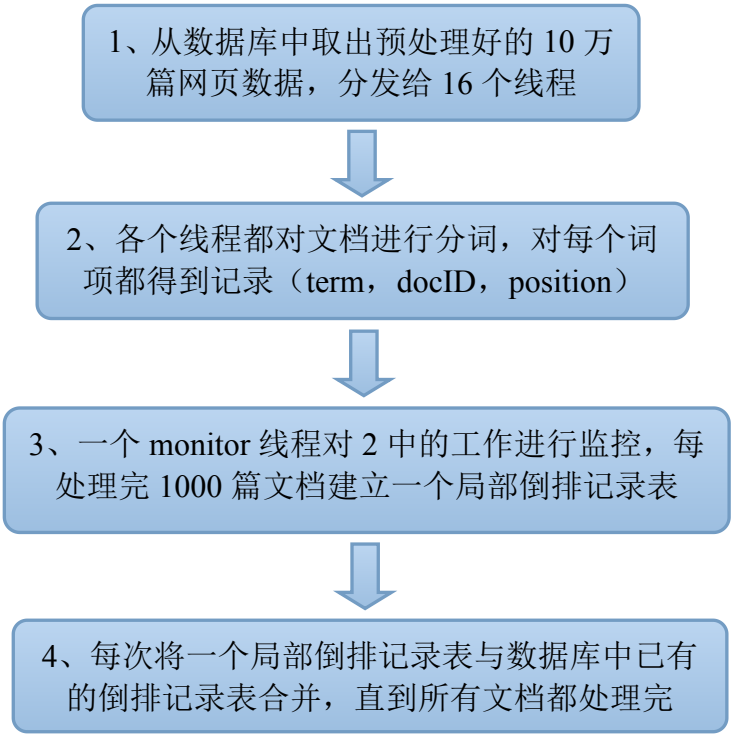


图 2-4 索引构建步骤

图 2-2 中第 1、2 步是构建索引的前提工作，第 1 步由于数据量比较大，因此将任务分成 16 个线程进行处理；第 2 步每个线程都独立对所分得的文档进行分词，分词采用前文所述的“结巴”中文分词系统，每个线程不仅需要进行分词，还需要记录下该词项所在的文档编号以及位置，如果该词项在 REMOVE_LIST（如引号、空格、网页标签等）中，则去除该词，否则将（term，docID，position）信息放入一个缓冲池中，此处引入一个全局的计数变量 Counter，以便记录完成了多少篇文档，每处理完一篇文档 Counter 都加 1；第 3、4 步是索引构建模块的核心部分，其目的就是构建每个词项的包含位置信息的文档索引列表，其中的文档用文档编号来进行标识，位置信息则是该词在该篇文档中出现的位置。由于数据量很大，引入一个 monitor 线程来监控第 2 步中各线程的完成情况。由于引入了 Counter 变量，因此每处理完 1000 篇文档就可以对其建一个局部倒排记录表，每建一个局部倒排记录表都跟数据库中已有的倒排记录表合并一次，最终得到全局倒排记录表，以便查询时使用。

第 3、4 步 monitor 线程构建倒排记录表并加入数据库中的具体流程图如图 2-5 所示：

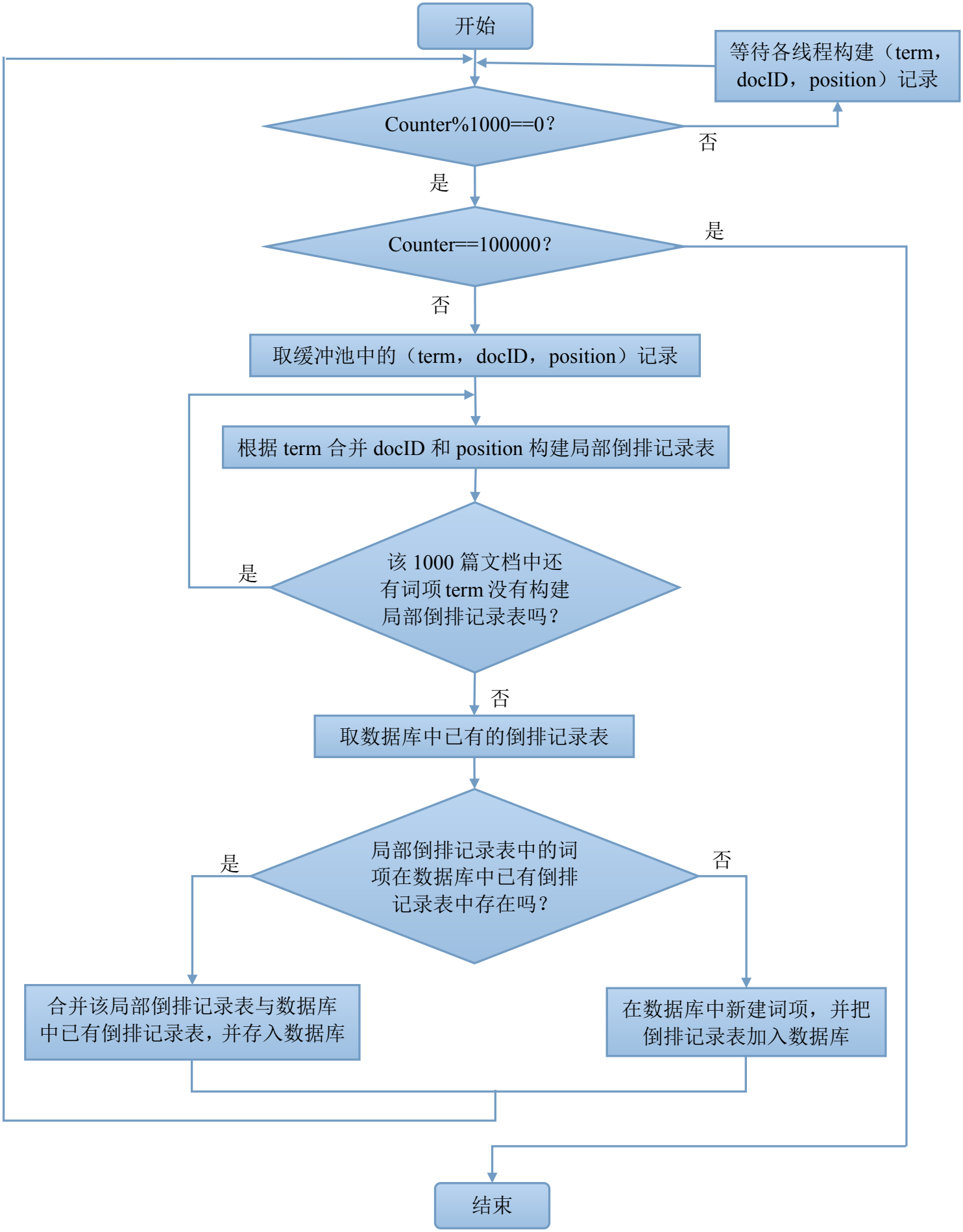
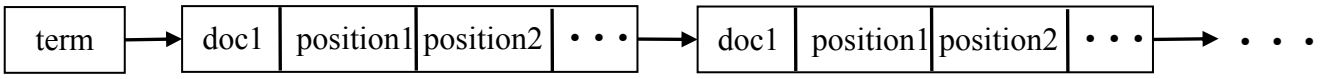


图 2-5 倒排记录表构建流程图

从图 2-5 可看出，一旦有 1000 篇文档已经处理完得到（term，docID，position）记录，monitor 线程就会构建一次局部倒排记录表，并取出数据库中已有的倒排记录表与其合并，合并就是把 docID 与 position 信息都加入到相应的词项 term 的倒排记录表中，倒排记录表如下表所示：



2.2.3 查询处理

查询处理就是处理用户在搜索页面中的查询请求，并根据查询请求返回相关文档的过程。判断文档是否相关是查询处理的一个重要过程，此处我们根据向量空间模型来计算出现查询词的文档得分，并将得分最高的前 100 篇文档返回给用户。查询处理模块可以分为以下 5 个步骤，如图 2-6 所示：

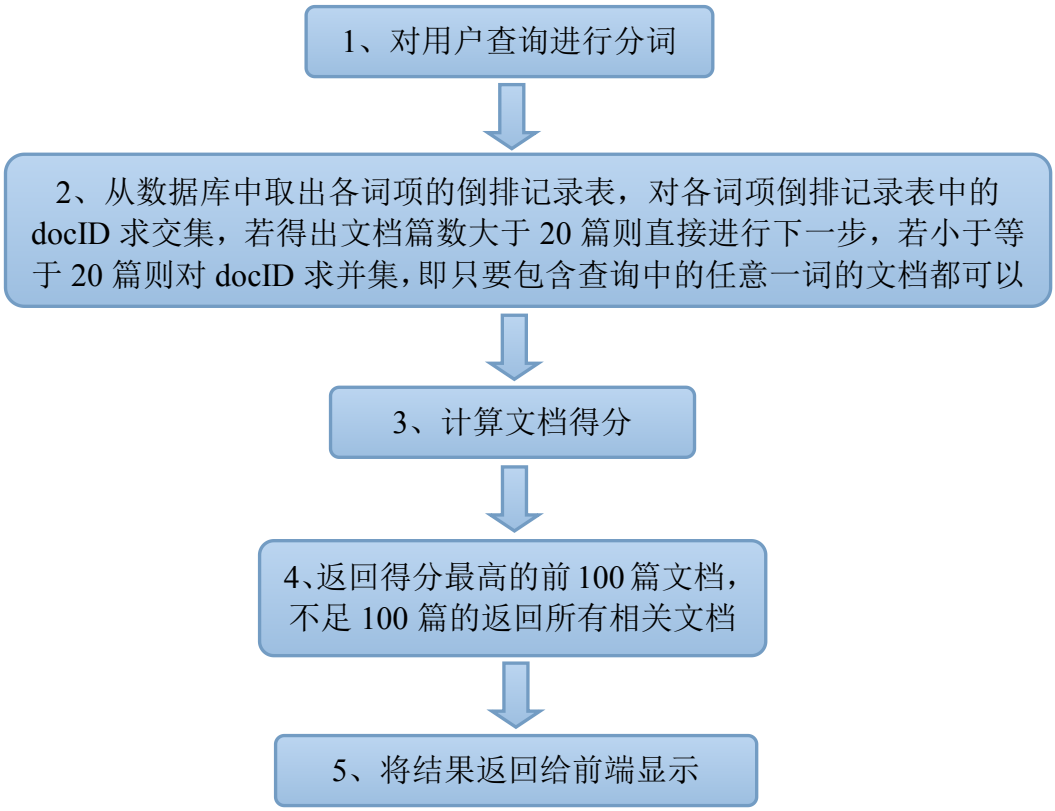


图 2-6 查询处理步骤

图 2-6 中第 1 步的分词同样是采用 jieba 中文分词系统，以便分词结果与文档的分词结果是一致的，这样才能保证文档的召回率；第 2 步根据查询的分词结果，取出各个词项的倒排记录表，由于每个词项后的倒排记录非常多，因此首先将其中的 docID 取交集，若得出的文档数大于 20 篇则直接进行下一步，若文档

数不够，再对查询中关键词(IDF 大)的倒排记录表试图并集，这样可以大大减少求并集所带来的计算量；第 3 步根据向量空间模型计算从第 2 步中得出的 docID 代表的各篇文档的得分，是查询处理中关键的一步；第 4 步根据第 3 步中计算的文档得分，返回得分最高的 100 篇文档，若不够 100 篇则返回所有相关文档；第 5 步将结果返回给前端显示，其中根据本新闻搜索系统的功能设置，不仅需要将相关文档返回给前端显示，还需给前端提供文档的摘要，对相似文档进行聚类以及给出相关检索推荐。

下面就详细介绍文档得分是如何进行计算的，以及将查询结果返回给前端显示前所实现的摘要功能、聚类功能以及相关检索推荐功能。

1）文档得分的计算

由于每个新闻网页包含有评论量、关键词、标题和正文四个部分，因此需要分别对这四个部分进行得分计算，然后再计算一个总得分作为文档的最后得分，根据该最后得分来进行后续功能的设计。四个部分的得分计算方式如下：

1、评论量得分 CommentScore：直接对评论量取对数，其对数值就是评论量的得分

2、关键词得分 keyWordsScore：计算文档关键词与查询词项进行匹配后的余弦得分；

3、标题得分 TitleScore：文档标题与查询词项进行匹配后的余弦得分；

4、正文得分 CosScore：文档正文与查询进行匹配后的余弦得分。

文档的最后得分 Result 通过给这四个得分赋予一定权值得到，其计算方式为：

$$\mathrm{Re}\,sult=\frac{0.15*CommentScore+0.2*keyWordsScore+0.15*TitleScore+0.5*CosScore}{CommentScore+keyWordsScore+TitleScore+CosScore}$$

上述得分除评论量外都要计算余弦得分，即查询与文档的余弦相似度，其计算方式为：对于查询中的每一个词项，首先根据其倒排记录表得出相应词项的 idf，根据查询得出每个查询词项的词频 tf，并计算其对数值作为 tf，通过 tf-idf 计算查询权重 QuerryWeight，然后计算文档权重 DocWeight，此次只需要计算在第 2 步中得出的 docID 中的文档即候选文档集的权重，归一化查询和文档的权重

后，最后计算 DocWeight*QuerryWeight，以便消除文档长度对得分的影响，文档得分就是归一化后的结果，同时对于不在第 2 步产生的 docID 中的文档，其得分结果都为 0。具体计算流程及公式如图 2-7 所示：

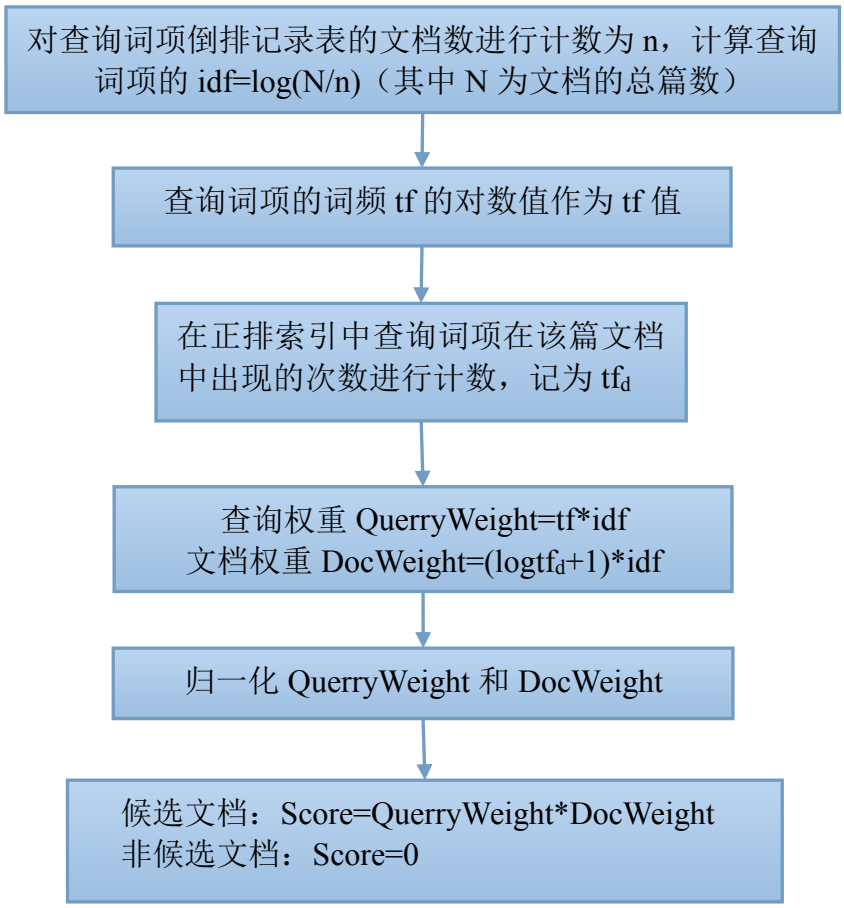


图 2-7 余弦得分的计算

通过余弦得分计算各个部分的得分后，根据最终得分 Result 的计算公式就可以得到该篇文档的最终得分，然后就可以对得分进行排序了，取得分前 100 的文档进行展示。

2）摘要功能

由于在倒排记录表构建时，我们保存了词项在文档中的位置信息，根据此位置信息可以很方便地实现摘要功能。首先需要设置一个摘要窗口大小 WindowSize，根据查询词项在文档中的位置信息，可以得出在某篇相关文档中所有查询词项出现的位置集合，WindowSize 框住的出现查询词个数最多的那个起止位置则是需返回的该片文档的内容。例如一个查询的各词项在一篇相关文档中的位置集合为[4,7,9,20,47,50,66]，而 WindowSize=20，则此时框住查询词项个

数最多的位置区间是[4,24]，有 4 个查询词，因此将文档中第 4 位到第 24 位的词作为摘要返回，从而实现摘要功能。

3）聚类功能

对查询结果中相似网页的聚类采用 K-means 算法实现，其中距离信息的计算用的是向量空间模型来计算余弦相似度。每一篇相关文档都可以写成一个向量 $\vec{d} = (t1, t2, ..., t3)$，其中 t 是查询词项的 tf-idf，任意两篇文档的余弦相似度计算公式为：$\frac{\vec{d}1 \bullet \vec{d}2}{\sqrt{\vec{d}1} \bullet \sqrt{\vec{d}2}}$，由于文档向量非常稀疏，因此可以采用稀疏矩阵的保存方式以节省空间。

利用 K-means 算法进行文档聚类的流程图如图 2-8 所示，在流程中算法的结束条件是文档从一个簇变到另一个簇的文档总和是否大于总文档数的 20%，而不是传统的直到所有簇都不再变动结束，其原因是，相关文档数量可能非常多，若果等到迭代到没有簇发生变化可能需要非常多的迭代次数，计算量也非常大，但是对聚类结果可能没有太多影响，因此在这里修改了 K-means 算法的结束条件。

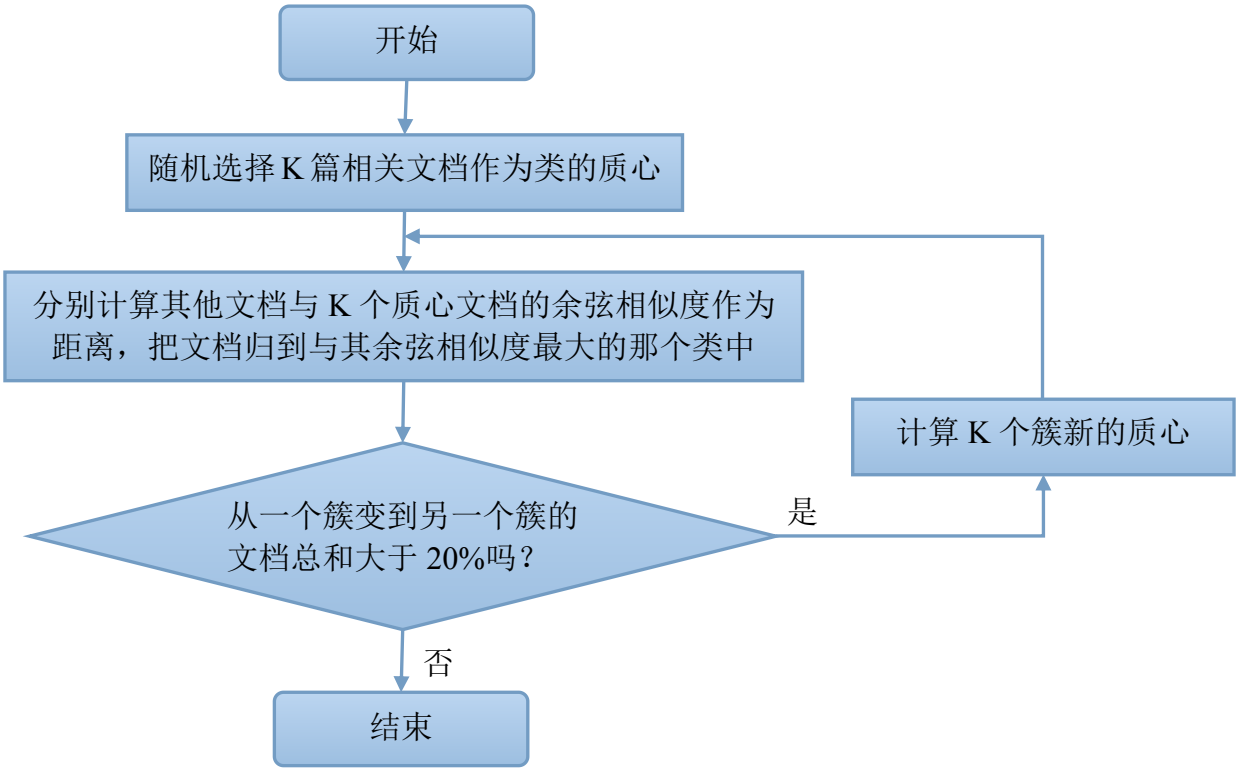


图 2-8 K-means 算法流程图

4）相关检索功能

相关检索功能是基于聚类的结果实现的，将查询结果聚类后，从中选取 n 篇与质心最靠近的文档，提取其与查询相关关键字，得出相关检索的内容。

2.3 前后端通信

本节主要讲述前端和后端是如何进行通信的，只有前后端能够正常通信，才能实现与用户的交互。本节内容主要包含前后端的通信方法以及通讯数据格式的设计。

2.3.1 通信方法

前后端通信的流程如图 2-9 所示：

图 2-9 前后端通信流程

其中后端利用 Python tornado 模块开一个服务器以监听前端发来的用户请求，前端通过获取后端返回的请求数据，然后展现给用户。

2.3.2 通信数据格式

前端与后端的通信是通过发送和接收 Json 格式的数据来实现的，其数据格式将分别说明如下：

1）前端发送数据的格式

前端只需将用户输入的查询发生给后端，因此只需要采用 String 类型的字符串，进行 URLEncoder 重新编码后，通过 http 发送 Get 请求到后端服务器即可。

2）后端发送数据的格式

通过解析前端发来的请求，得到用户所输入的查询内容，然后通过后端的查询处理模块得到查询结果，最后将查询结果写成 Json 格式的数据返回给前端，其中后端返回给前端的数据格式如下：

```
{
"list":[
{
Comment: " ",
ShortTitle: "      ",
TiTle: "        ",
URL: " ",
Media: " ",
Author: "",
Content: "        "
AbstractString: "      "
KeyWords: "      "
},
{
Comment: " ",
ShortTitle: "      ",
TiTle: "        ",
URL: " ",
Media: " ",
Author: "",
Content: "        "
AbstractString: "      "
KeyWords: "      "
},
]
"RelatedDocs":[
{
"docid":
   "url" : "    "
   "s_title" : "    "
},
{
"docid":
   "url" : "    "
   "s_title" : "    "
},
]
}
```

整个 Json 数据用{ }包围起来，其中包含两个字段：list 和 RelatedDcos，list 包含返回结果的具体内容；RelatedDcos 返回相关查询推荐的文档。list 中返回的每一条查询结果都用一个{ }包围，{ }内是根据用户查询搜索出来的需要返回给前端显示的内容，其中引号" "内的内容为每个字段的值，各字段的含义分别为：

Comment 评论条数、ShortTitle 短标题、TiTle 网页原标题、URL 网页链接、Media 该条新闻的媒体来源、Author 本新闻的作者、Content 新闻内容、AbstractString 根据查询处理得出的摘要内容、KeyWords 关键字。

2.4 数据库设计

由于前端数据库比较简单，只需存储用户的历史查询记录，因此在这部分只讲述后端数据库的设计部分，这部分数据库设计是整个系统的核心。

根据系统需求，需要有 4 对映射关系：词项-词项编号、文档-文档编号、文档编号-词项编号、词项编号-文档编号。后台数据库需要设置以下 4 个表格：

1、listcrawler 表：保存网络爬虫爬取的源数据；

2、textcrawler 表：保存爬虫处理后的数据；

3、inverseindex 表：保存词项的倒排记录表；

4、docindex 表：保存文档的索引，以便返回查询结果。

下面就分别描述这几个表的具体设计，主要是其字段的设置。

2.4.1 listcrawler 表

列名	解释
ListcrawlerType	爬取的数据的媒体来源
ListCrawlerkey	所爬数据的标题
TextCrawlerUrl	所爬数据的 URL

2.4.2 textcrawler 表

由于爬虫需要将原始数据切分成好几个块以便之后查询处理结束生成 Json 数据返回给前端显示，因此需要一个表格来保存处理后的数据，其格式如下：

列名	解释
Id	唯一标识每一个新闻页面
URL	每篇新闻的链接
Title	新闻标题

Short_Title	提取的网页短标题
keyWords	新闻关键字
media	新闻来源
author	新闻作者
comment	新闻的评论数量
content	新闻的正文

2.4.3 inverseindex 表

列名	解释
Id	唯一标识每条记录
Item	词项
inverseDocs	倒排记录表，其中包含了词项所出现的所有新闻的编号和其位置信息
Df	词项所出现的新闻的篇数

2.4.4 docindex 表

为了便于实现聚类、相关搜索推荐需建立一个表格用来存储文档与词项的关系，其格式如下：

列名	解释
DocId	新闻的编号，其外键是 textcrawler 表中的 Id
KeyWordsId	关键词中词项的编号，与词项映射
ShortTitleId	短标题中词项的编号，与词项映射
TitleId	标题中词项的编号，与词项映射
itermsId	词项编号，与 terms 一一对应，外键是 inverseindex 表中的 Id
iterms	每篇新闻中所出现的所有词项

3 系统实现

根据设计方案，前端基于主流 JavaWeb 框架进行编写，后端主要用 Python 语言进行编写，源程序将在附件中展示，现分模块描述各个程序的类、函数功能和成员变量。

3.1 前端实现

由于前端也是一个单独的 Web 系统，因此也有其前台和后台，前端的前、后台实现如下：

1、前台

前台是两个与用户交互的文本页面，主搜索界面是 index.jsp 和 mainpage.jsp，用户进行查询后的结果显示页面是 results.jsp。

2、后台

后台定义了 4 个类，分别是 TestHttp、Retresults、Otherquery、QueryAction，先分别介绍各个类的作用和其中的方法如下：

①类名：TestHttp

作用：与服务器建立连接并进行数据传输

方法名：public static String sendGet(String url,String param)

参数说明：url:服务器 url 地址

param：经过 URL 转码后的用户查询语句

②类名：Retresults

作用：用来封装从服务器传来的 JSON 数据

域值：url,title,keyWords,media,comment,content,snippet

③类名：Otherquery

作用：用来封装从服务器传来的其它相关搜索

域值：otherurl,otherquery

④类名：QueryAction

作用：用来响应用户查询并控制页面跳转

方法名：getQuerystat()

该方法实现的主要功能：调用 TestHttp.sendGet 方法获取服务器数据，进行二次数据封装后返回给前台。

控制页面跳转的 struts2 注解：@Action(value = "getQuerystat", results = { @Result(name = "results", location = "/search/results.jsp"), @Result(name = "search", location = "/search/search.jsp") })，利用该注解可以实现当用户没有输入查询词时返回主页面 search.jsp;否则返回结果页面 results.jsp。

3.2 后端实现

3.2.1 数据库操作接口类

该类负责执行数据库连接、建表，以及所有与数据库相关的操作,是系统与数据库的唯一接口，所有与数据库相关的操作必须从此继承。本系统中我们使用了 MySQL 数据库。

主要成员方法：

__init__(self)：负责整个系统与数据库的连接以及表的创建。

__getConnection(self)：获取当前与数据库的连接(如果连接已存在，则对此连接进行复用)。

__getCursor(self)：获取执行数据库操作的游标。

insertItem(self,valuesList =[])：数据库插入数据的统一接口,执行数据的批量插入。

clearTable(self,tableName = "")：清空指定的表。

3.2.2 爬虫实现的主要类及实现函数

1）主要类 crawler 类是爬虫的基类，所有爬虫从此基类继承

主要方法:

__init__(self,**args)：初始化一个爬虫，获取爬虫执行的数据，和爬虫的数据导入接口。

crawl(self)：每个爬虫爬取数据的具体执行方法。

importData(valuesList=[])：批量执行数据导入。

getData(tableName)：获取爬虫数据库的数据。

2）主要实现函数

load_html(url)：加载 HTML 文档，并以文档结构树的形式返回,以便后面通过 Xpath 对文档结构树进行遍历。

getJsonByApi(league_type,length =0)：直接从网页的 API 中获取相应的 Json 数据。

3.2.3 索引构建主要的函数

主函数：多线程并行对文档进行拆分和索引构建。

sliceDoc(doc)：对一篇文档进行切词。

HandleDoc(task)：将文档组成的列表中的每一篇文档拆分成(docId,term,position)对。

CreateInvertedIndex(SliceList)：根据每篇文档的拆分成的(docId,term,position)对列表构建带有位置信息的倒排索引。

3.2.4 查询处理的主要函数

查询处理中得到文档得分后还需要做摘要聚类等功能，其函数设计如下：

1）查询处理和文档得分函数

ProcessQuerry(Querry)：查询处理的主函数，对查询进行分词后取相应的倒排记录表，为减少相关文档相似度的计算量；并先对其取交，获取包含所有查询词的文档，若文档数不够，再对查询中关键词(IDF 大)的倒排记录表试图取并集，并对并集所有文档进行相似度计算和聚类后返回排名靠前的文档。

CalCosineSimilarity(QuerryVector,DocVector)：计算查询向量和文档向量的余弦相似度。

getSearchResult(Candidates,QuerryList)：根据余弦相似度对所有文档进行评分

DocumentScore(CommentScore,keyWordsScore,TitleScore,CosScore)：文档评分函数，它包含了好几种评分方式，分别有：

CommentScore　-根据评论量取对数后的得分；

keyWordsScore - 文档关键词与查询进行匹配后的余弦得分；

TitleScore - 文档标题与查询进行匹配后的余弦得分；

CosScore - 文档正文与查询进行匹配后的余弦得分。

2）实现摘要功能的函数

getWindowPos(Posting,WindowSize=AbstractWindowSize)：根据查询内容获取相关文档摘要动态生成的窗口。

generateReturnAbstractAndContent(FinalResult)：根据查询内容动态生成摘要，获取前端页面所需的内容并返回。

3）实现聚类功能的函数

cluster(DocId)：计算 DocId 中所有文档的的 term 的 tf_idf,用于之后的构建向量空间模型，计算文档与中心点的相似度。

kmeans(dataSet, tfidf, idf, k)：利用前 k 篇文档初始化 k 个中心点,利用文档与中心点的相似度进行聚类，如果文档属于哪个簇相对上次有变化的文档数小于 20%，则聚类结束，对于每个簇中的文档按照与中心点的相似度从大到小排序，返回每个簇中的前 2 篇文档。

3.2.4 web 服务端

主要类：

MainHandler(tornado.web.RequestHandler)：监听 Http 请求。

主要方法：

get(self)：提取客户端发送的请求，并将前端查询进行处理查询后将数据以 json 形式返回给前端。

4 测试结果

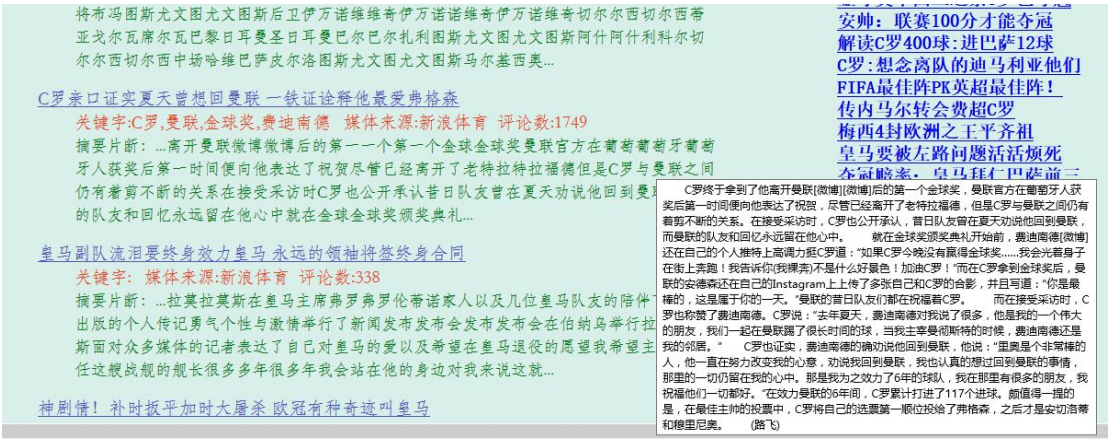
1、后端开启服务器，前端连接服务器后，访问 localhost，显示如下搜索主界面：

2、在搜索框中输入查询，能够自动进行查询补齐提示，如下图所示，输入“C 罗”得到以下提示：

3、点击“go now”按钮或者按回车键，跳转到搜索页面得到搜索结果如下：

搜索结果页面的主体部分每条新闻的标题、关键字、媒体来源和评论数，同时展示了由滑动窗口动态生成的摘要片段。

4、将鼠标放在搜索结果中的某条记录上，得到以下文本的预览如下：



5 总结

本次大作业我们实现了一个新闻检索系统，整个过程包含了网页信息的爬取、索引和查询，查询能够自动补齐，其结果能够进行排序、生产摘要、预览，同同时能对相似新闻进行聚类，给出相关搜索推荐。通过大作业，我们掌握了检索系统的实现流程，加深了对课程知识的理解，并且进一步提高了自己的编程能力，受益匪浅。

5.1 系统特点

由于本新闻检索系统的服务器端除分词外均由自己编码实现，没有采用框架，因此特点比较多，现分别就前、后端特点叙述如下。

5.1.1 前端特点

1、采用主流的 JavaWeb 框架

本系统的客户端是基于 Web 实现的，因此我们采用了主流的 JavaWeb 框架：Struts2+Spring+Hibernate+EasyUI，并且使用了零配置注解开发，减少很多代码量。

2、实现预览功能

实现了查询结果的预览功能，能够让用户直接在查询结果页面中预览相关网页的正文内容。

5.1.2 后端特点

1、多线程并发

由于实验的数据量巨大，单线程处理起来非常慢，而采用分布式在一台电脑上模拟搭建几个节点进行处理其实跟在一台电脑上多线程处理没有太大区别，搭建分布式系统确很麻烦，因此我们选择采用多线程并发来处理数据。在本系统中有两处采用了多线程并发的方式。

网络爬虫采用多线程处理爬取的数据：为了便于查询，从网页爬取数据后还需对数据进行处理，写到表 textcrawler 中，表的内容请查看 2.4 节，该表将爬取的原网页内容切分成了好几个块进行保存，有标题、作者、评论、正文等内容，由于爬取了 10 万网页，因此需要对这 10 万网页进行上述切分，若用单线程速度非常慢，采用多线程并发的模式可以大大提升操作速度。

多线程并发建立索引：在索引构建模块中，我们采用了多线程来生成（term，docID，position）对，由于需要对 10 万网页都进行上述处理，数据量非常大，用多个线程并发处理可以大大提升处理速度。

2、自己实现

整个新闻检索系统，我们除分词部分采用的是“结巴”中文分词系统外，所有功能实现都是自己编写，没有采用已有的搜索引擎搭建框架如 Nutch 等。通过自己实现，对搜索引擎内部实现机制有了较深入的理解。

3、采用主副索引实现构建索引

由于数据量巨大，词项的倒排记录表非常长，在进行索引构建的时候，每处理完 1000 篇文档就构建一次索引跟数据库中已有的主索引进行合并，具体机制见 2.2.2 节，由于读写数据库比较慢，索引合并非常耗时，构建好所有索引比较慢，但是采用这种主索引的方式进行索引构建的可扩展性非常强，不管有多少索引都能够不受内存限制而处理完成，并且这种方式还可以移植到其他大数据环境下。

4、使用滑动窗口动态生成摘要

在为查询结果生成摘要时，采用了一个滑动窗口来获取摘要位置，以便用户看到的摘要内容是包含查询词项最多的文字部分，这种动态生成摘要的方式能够有效避免网页本身的静态摘要与查询匹配度不高的情况。

5.2 难点及解决方法

1、前后端数据交互困难

解决方法：前端采用 Struts2 的模型驱动向后端发送用户数据；后端向前端返回查询结果时使用语句 ServletActionContext.getRequest().setAttribute("reslut", li);

2、前端采用的框架不太稳定

前端采用的框架不太稳定，会出现异常错误，修改代码后需要重启项目才能正常运行，操作繁琐。

3、数据无法一次性导入内存

由于所处理的数据有 10 万网页，数据量很大，在对每个网页进行分词构建索引时无法将所有数据一次性导入内存进行操作。

解决方法：用一个缓冲池保存中间分词结果，当缓冲池中数据达到一定量时便开始构建索引，边构建索引，边分词。

4、数据处理时间长

由于数据量很大，数据的处理时间非常长。

解决方法：利用多线程并发的方式来进行处理，加快处理时间。

5、数据库读写瓶颈

在构建索引时，采用了用数据库保存主索引的方式，然而在合并索引的时候就需要将索引读取和写入数据库，然而数据库读写操作非常耗时，使得索引的合并执行较慢。

解决方法：目前还没有找到合适的解决方法，只能等待。