

题 目 基于Lucene的企业统一搜索系统的设计与实现

软件学院 院（系）软件工程 专业

学 号71112314

学生姓名 解鸿浩

校内导师 吴含前

企业导师 胡小舟

起止日期 2016.1-2016.5

设计地点 中国银联股份有限公司

基于Lucene的企业统一搜索系统的设计与实现

摘要

随着自助入网业务规模的不断过大，技术开放平台的技术文档数量和功能入口数量变得越来越庞大，平台的用户进入开放平台的网站时，难免需要在查找某个文档、FAQ时耗费时间，对于有明确目的的访客来说，这会在一定程度上降低用户的生产效率。

为了提高用户检索相关文档和技术的效率，进而提高生产开发速度，本文以中国银联自助入网平台为对象，主要任务是给平台构建一个可以供用户检索需要的API和技术文档等内容的搜索系统。主要的工作包括：一，提取并统一化平台现有的支持性数据内容；二，针对统一化之后的数据，借助Lucene工具包进行索引处理；三，将用户输入的关键词和被检索数据全集进行搜索、打分，返回给用户；四，关联其他站点如技术社区，一次搜索多个站点的内容。

论文从项目背景出发，介绍了全文检索的基本过程，Lucene源码结构和逻辑结构，对比Lucene全文检索和其他全文检索的区别，详细设计了该搜索系统的结构，并给出了关键模块实现过程。

关键词：全文检索；Lucene；索引

**Designing and Implementing an Unified Enterprise Searching System Based on Lucene**

**Abstract:**

As the business continues to scale too large,the number of APIs and tech documennts becomes larger,too.When users of platform enter the website, Inevitably need to spend time finding a document, FAQ. For visitors who have a clear purpose, which reduces user productivity to some extent.

In order to improve the efficiency of retrieving related documents and technologies for users,and then improving the efficiency of producing and developing software,this paper take the UnionPay’s Auto-Join Platform as target ojbect,and it’s main task is to build a searching system for retrieving necessary APIs and technical documents and other contents for users.Main process contains as follows: Firstly,extract and unite those data stored on the platform;Secondly,aiming at those data united by step 1,indexing the data by toolkit provided by Lucene;Thirdly,execute searching and scoring between keywords input by users and whole set of retrieved data,then return the result to users;At last,connecting other site,such as technical community,completing the aim of searching multiple sites at once.

This paper starts from the background of project, then firstly introduced the general process of full-text search,Lucene source code structure and logic structure,compared the differences between Lucene full-text search and other full-text search,designed the structure of this searching system in detail,and then provide the implemention process of some key modules.

**Keywords:**Full-text Retrieving;Lucene;Indexing

目录

[第一章 绪论 1](#_Toc451266781)

[1.1 引言 1](#_Toc451266782)

[1.2 全文检索技术发展现状 2](#_Toc451266783)

[1.3 构建统一搜索系统需要考虑的问题以及采用的手段、方法 4](#_Toc451266784)

[1.4 本文的研究目的和主要研究内容 6](#_Toc451266785)

[第二章 应用到的技术及原理 7](#_Toc451266786)

[2.1 Lucene介绍 7](#_Toc451266787)

[2.1.1 Lucene的总体架构： 7](#_Toc451266788)

[2.1.2索引和搜索过程 8](#_Toc451266789)

[2.1.3 Lucene的主要模块介绍 10](#_Toc451266790)

[2.1.4 Lucene的索引 10](#_Toc451266791)

[2.1.6 Lucene计算文档和查询语句的相关性 12](#_Toc451266792)

[2.2中文分词技术（中文分词原理） 15](#_Toc451266793)

[2.2.1 分词的意义 15](#_Toc451266794)

[2.2.2 IKAnalyzer分词原理 15](#_Toc451266795)

[第三章 系统设计 19](#_Toc451266796)

[3.1系统整体架构 19](#_Toc451266797)

[3.2 跨站搜索设计 21](#_Toc451266798)

[3.3 权限控制设计 22](#_Toc451266799)

[3.4 系统接口设计 23](#_Toc451266800)

[3.4.1 搜索模块相关数据模型说明 23](#_Toc451266801)

[3.4.2 索引模块接口 27](#_Toc451266802)

[第四章 系统实现 30](#_Toc451266803)

[4.1 Web模块 30](#_Toc451266804)

[4.1.1 配置文件 30](#_Toc451266805)

[4.1.2 前端页面实现 32](#_Toc451266806)

[4.1.3 控制器实现 35](#_Toc451266807)

[4.2 搜索服务模块 36](#_Toc451266808)

[4.2.1 maven配置文件 36](#_Toc451266809)

[4.2.2 服务实现 38](#_Toc451266810)

[4.3 索引管理模块 42](#_Toc451266811)

[4.3.1 配置文件 42](#_Toc451266812)

[4.3.2 创建索引 43](#_Toc451266813)

[4.3.3 查询索引记录 45](#_Toc451266814)

[4.3.4 索引缓存/内存索引 46](#_Toc451266815)

[4.4 原始数据处理模块 47](#_Toc451266816)

[4.4.1 generate.sh脚本 47](#_Toc451266817)

[4.4.2 createIndex.sh脚本 49](#_Toc451266818)

[第五章 总结和展望 51](#_Toc451266819)

[5.1 总结 51](#_Toc451266820)

[5.2 工作展望 51](#_Toc451266821)

# 第一章 绪论

## 1.1 引言

技术开放平台的主要作用就是提供应用程序接口、技术文档供开发人员使用，平台的存在将技术规范化，大大提高了生产效率。而银联的自助入网平台不仅提供了入网需要的各种必要应用程序接口、技术文档，并且将以前签约等复杂的手续流程省去，商家和机构可以直接跳过签约这一步，先在平台上进行入网测试，然后再决定是否签约。这无疑为更多商家和机构提供了试用银联结算交易技术并进一步接入银联的机会，有助于银联业务的顺利推广。

然而，随着业务规模的不断过大，技术开放平台的技术文档数量和功能入口数量变得越来越庞大，平台的用户进入开放平台的网站时，难免需要在查找某个文档、FAQ时耗费时间，对于有明确目的的访客来说，这会在一定程度上降低用户的生产效率。

解决思路无非两种：一是不断改进网站界面，使功能结构更加清晰明了，但是这种方法随着网站内容的增加效果也会下降，网站维护的成本也会不断上升，因为对应的后端接口也需要同时改造，因此并非长远之计；另一种方法就是提供搜索入口，提供全文检索功能，全文检索是以计算机数据如文本等为处理对象，提供按照文档资料的内容而非外在特征来实现的信息检索手段。针对文档的内容创建一系列索引，能够帮助人们方便、快速查找到信息。Lucene是目前比较流行的全文检索工具包，已经在许多搜索项目中得到了广泛的应用。

本文将介绍Lucene全文检索的基本原理，利用Lucene设计并实现针对企业技术开放平台的统一搜索系统。系统针对官方开放平台、技术论坛，搜索可以跨两站同时进行，尽可能返回更多对用户有用的信息。

## 1.2 全文检索技术发展现状

网络飞速发展，数据存储技术不断完善，储存大量信息已经不再是难题，人们关心的重点是如何在这些信息中快速、准确地找到有用信息。

信息检索的核心技术是全文检索技术，全文检索是以各种各种数据诸如文字、声音、图像等为检索对象，提供按照文字的内容实现的信息检索手段，而非数据的外在构建结构。

网络上的数据分为总体两种类型，结构化数据和非结构化数据。结构化数据是指具有固定格式或有限长度的数据，如数据库，元数据等；而非结构化数据是指不定长或无固定格式的数据，如邮件，word文档等。某些场合，还会提到半结构化数据的概念，如XML，HTML等。根据不同的应用场景，可以按照结构化数据来处理，也可以抽取出纯文本按照非结构化数据处理。非结构话数据又叫全文数据。

根据数据搜索的分类，也可以被分类成结构化数据搜索和非结构化数据搜索。结构化数据搜索顾名思义，如对数据库的搜索，使用SQL语句；非结构化数据搜索，如Linux系统中的grep命令，再入更常见的搜索引擎。在此我们重点讨论非结构化数据搜索，即全文检索。

全文检索有两种主要方法：其一是顺序扫描法(Serial Scanning)：所谓顺序扫描，即要查找包含某字符串的文件，那么从头开始扫描已有文件，每个文档一行一行扫描，如果匹配到这个字符串，那这个文件就是目标文件，如此往复直到遍历全部文件。另外一种是从非结构化数据中提取出来一部分信息，重新组织成具有一定结构的数据，然后对此进行搜索，从而达到加快搜索速度的目的，这个过程叫做全文检索(Full-text Search)。

目前的全文检索技术大致分为两个过程，索引创建(Indexing)和搜索索引(Search)。索引创建就是从数据源中提取信息，创建索引的过程。搜索索引指得到用户的查询请求，搜索创建的索引，然后返回搜索结果的过程。

非结构化数据中所存储的信息是从文件到字符串的映射，即某文件包含某字符串。而搜索的过程是找出包含目标字符串的文件，就是从字符串到文件的映射。于是如果将从字符串到文件的映射保存下来，则会大幅度提高搜索的效率。因为从字符串到文档的映射和文档到字符串映射是两个相反的关系，我们将保存这种映射关系的数据结构称为反向索引。

反向索引其实是包含词典和倒排表的数据结构。词典就是出现在文档中的词语或短语的集合，即一些列字符串；倒排表是每个节点都指向包含节点内容的文档链表。

全文检索的索引创建一般有如下过程：

1 获取要索引的源文档（Document）。在此举两个简单文件的例子：

文件A

Students should be allowed to go out with their friends, but not allowed to drink beer.

文件B

My friend Jerry went to school to see his students but found them drunk which is not allowed.

2 将文档分词（Tokenizer）

分词的工具会将文档分成多个独立的单词，同时将标点符号去除，去除停止词(Stop word)。停止词是几乎每种语言都有的的一类特殊词语，不具有具体意义，比如汉语的“的”、“了”等，这些词语通常不构成搜索关键词，因而这种词会在索引创建时会被自动去掉。英语中挺词(Stop word)如：“the”,“a”，“this”等。对于每一种语言的分词组件(Tokenizer)，都有一个停止词(stop word)词典。文档经过分词(Tokenizer)后，产生一系列词元(Token)。在我们的例子中，便得到以下词元(Token)：{Students，allowed，go，their，friends，allowed，drink，beer，My，friend，Jerry，went，school，see，his，students，found，them，drunk，allowed}

3 语言处理器（Linguistic Processor）处理词元（Token）

语言处理器(linguistic processor)主要将词元(Token)做一些同语言相关的处理。对于英语，语言处理组件一般将所有词汇变为小写，即stemming;把单词转为词根，即lemmatization。这个过程的处理结果称为词（Term）。在此例中，得到的词如下：“student”，“allow”，“go”，“their”，“friend”，“allow”，“drink”，“beer”，“my”，“friend”，“jerry”，“go”，“school”，“see”，“his”，“student”，“find”，“them”，“drink”，“allow”。

4索引组件(Indexer)处理语言处理器得到的词

索引组件利用得到的词创建一个字典，对字典按照首字母顺序进行排序，最后合并相同的词成文文档倒排链表。在此倒排链表中，有几个定义：Document Frequency 即文档频次，表示总共有多少文件包含此词；Frequency 即词频率，表示此文件中包含了几个此词。

然后是搜索过程，搜索主要分为以下过程：

1 用户输入查询语句。

2 对查询语句进行词法分析、语法分析以及语言处理。词法分析主要切分语句，提取单词以及关键字，语法分析的目的是生成语法树，语言处理同索引过程中的语言处理。

3 搜索索引，查找与语法树相符的文档。首先，从反向索引表找到包含关键词的文档链表；其次，对包含关键词的链表进行运算（并、或、差操作）；最终得到我们需要的文档链表。

4 根据查询语句和步骤3中得到文档链表中文档的相关性打分，对链表排序

5 包装并返回查询结果给用户

以上就是全文检索目前比较成熟的解决方案。可以看出，在构建全文检索系统的过程中，比较重要的是索引的建立（其中包括文档的分词）以及搜索时文档相关性计算，因为这两点直接关系到了搜索的效率和准确性，这两点在后文正文中还会作详细介绍。

## 1.3 构建统一搜索系统需要考虑的问题以及采用的手段、方法

项目的总体工作目标：

本项目针对公司的实际业务需求，为提高生产效率，更好地推广公司业务，方便用户使用自助入网平台，将设计并实现一个搜索接口。该模块将嵌入自助入网平台，并且在搜索范围上联接开放平台和技术社区，扩大搜索的范围。同时尝试在实现功能的基础上进行优化，并对将应用向改造分布式系统的方法进行探索和尝试。

通过对目前全文检索技术的评述，在设计针对银联自助入网平台的搜索系统时，需要考虑的主要有一下几点：如何将源数据（文档、网页）处理成适合构建索引的非结构型数据，并方便地完成字符串到文件的映射（倒排）；如何针对不同类型的数据建立索引并存储；如何将两个乃至多个网站的数据索引维护在同一逻辑空间内，便于跨站搜索；如何设计用户的搜索权限，使得系统仅向用户展示其有权限查看的文档；如何优化索引的读取速度以缩短搜索的响应时间。

a 将源数据（文档、网页）处理成适合构建索引的非结构型数据，并方便地完成字符串到文件的映射（倒排）

为解决这个问题，根据网站中数据的特点，采用了Shell脚本处理数据库中的文本数据，将其分类并过滤掉HTML/JS标签，各自导成纯文本文件，文本文件的名称格式为“类型\_文档id\_权限码”，将其存储在相应类型的路径下；采用Java的类库处理富文本文件（word/pdf/excel等），将纯文本提取出来，忽略样式等无关内容，也导成纯文本文件，以相同的命名格式保存到对应路径下。

b 针对不同类型的数据建立索引并存储

使用基于Java实现的Lucene工具包，单独建立一个工程，对上一步处理过的数据源进行在再加工：每个文件有文件名、文件内容两种要素，其中文件名又包括文档类型、文档id、文档权限码三个子要素。在建立索引时，将文件名作为一个域，再对文件内容中的每一行创建域，这样就求保证了文档的内容作为索引的一部分；最后Lucene的类IndexWriter将索引写入磁盘。这个创建索引的工程被打包成jar包，由shell脚本调用执行。

c 将两个乃至多个网站的数据索引维护在同一逻辑空间内，便于跨站搜索

其他内部站点的索引生成方法类似于上文，主站中的shell脚本会定时调用一个独立的程序从其他需要检索的站点的约定路径下载索引文件到本机的约定路径，供搜索系统使用。

d 设计用户的搜索权限，使得系统仅向用户展示其有权限查看的文档

设计了一套权限规则和权限码，在第一步处理中将权限码附加到处理后的文件上，在搜素时获取当前登录用户的权限码，与检索出的文档比对，判断是否具有查看的权限。权限设计将在正文中详细阐述。

e优化索引的读取速度以缩短搜索的响应时间

硬盘因其硬件限制，天然成为了限制读写速度的因素，影响了简历索引的效率，也是性能提高的瓶颈。分析建立索引的过程，可以发现建立索引时，首先生成多个较小的段索引，保存到磁盘上，达到一定的阈值时，索引合并就发生了。但是这个过程完成后，剩余了一些索引文件，可以考虑将剩下的这些段索引再次合并，形成一个索引文件，减少内存和硬盘之间读写交换的频率，这样一来，检索效率就能在一定程度上提高。当网站规模较小，索引文件大小也较小时，也可以在内存中建立一套存储索引的数据结构，直接在内存中维护索引，并定时更新，这样就加快了读取索引的速度。

## 1.4 本文的研究目的和主要研究内容

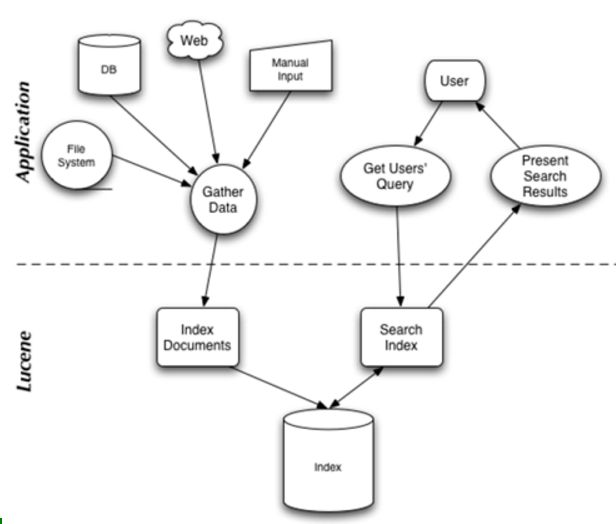
综上所述，企业中对建立高效的站内搜索系统进行了大量的研究和尝试，也有了很多成熟的方案，但基于每个站点的数据特点不同，从构建索引到给用户返回搜索结果都需要由开发者根据具体情况做定制化开发。本文将要设计和实现的系统基于中国银联自助入网平台，依据生产需求和现有网站结构，结合现有的搜索入口以及开源的搜索技术，提出自己的解决方案，实现一个可以在企业平台内运行的搜索系统，系统针对官方开放平台、技术论坛，搜索可以跨两站同时进行，并且具备基本的权限控制功能，尽可能返回更多对用户有用的信息。

# 第二章 应用到的技术及原理

## 2.1 Lucene介绍

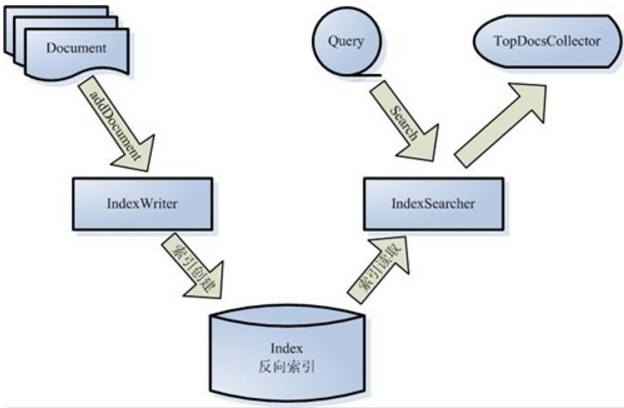
Lucene是一个开源、高效可扩展的全文检索工具库，使用Java实现，不需要额外的配置，且只有纯文本文件才能被索引(Indexing)继而被搜索(Search)，其他格式的文件，则需要开发者自己完成从文件中提取纯文本，或从网络中抓取文件的工作。

### 2.1.1 Lucene的总体架构：



图[1]

由上图可知，Lucene可以承担两个过程的工作：索引和搜索，下图是一些关键的组件：

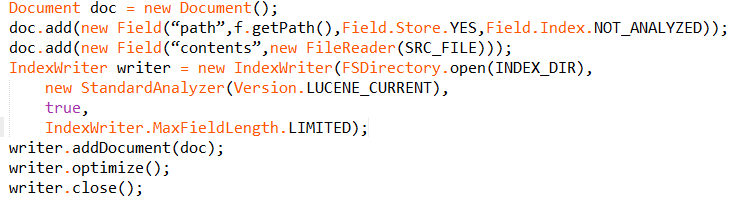


图[2]

被索引的文档用Document对象表示；IndexWriter通过函数addDocument将文档添加到索引中，实现创建索引的过程；当用户有请求时，Query代表用户的查询语句；IndexSearcher通过函数search搜索Lucene Index；IndexSearcher计算term 权重和得分并且将结果返回给用户；返回给用户的文档集合用TopDocsCollector表示.

### **2.1.2 索引和搜索过程**

**对Lucene API的调用实现索引和搜索过程大致如下：**



建立索引这段代码的过程如下：

* 创建IndexWriter的实例，用于将索引文件写入磁盘或内存，它的参数：INDEXDIR是保存索引文件的位置；analyzer的实例的作用是文件分析器词法分析，初步语言处理。
* 创建一个Document的实例，用来表示被索引文档。
* 将不同的Field实例加入到第二步创建的文档doc中。一篇文档包含题目，修改时间，作者，内容等信息。需要用不同的Field表示不同类型的信息，在这个例子中，对文件路径和文件内容这两类信息进行了索引操作。其中要索引的源文件用FileReader的参数SRC\_FILE来表示。
* IndexWriter调用函数addDocument把索引写入到约定的磁盘索引路径中。

用户进行搜索时，大致代码如下

**ndexReader reader = IndexReader.open(FSDirectory.open(new File(index)),true);**

**Searcher searcher = new IndexSearcher(reader);**

**Analyzer analyzer = new StandardAnalyzer(Version.LUCENE\_CURRENT);**

**QueryParser parser = new QueryParser(field,analyzer);**

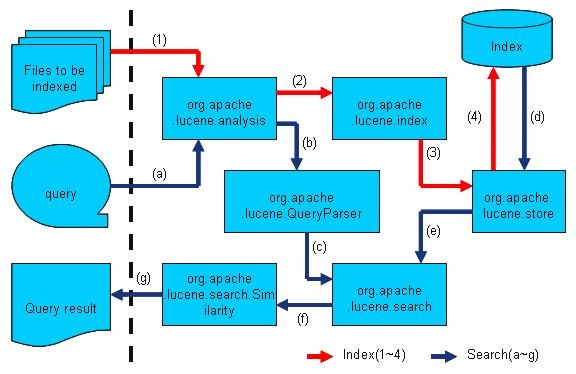
**Query query = parser.parse(qeuryString);**

**TopScoreDocCollector collector = TopScoreDocCollector.create(hitsNum,false);**

**searcher.search(query,collector);**

* IndexReader将保存好的索引从硬盘读到内存中，索引文件被存放在INDEX\_DIR表示的路径中。
* 创建IndexSearcher的实例，即进行搜索操作的对象。
* 创建Analyer，对用户输入的关键词进行lexical analysis和linguistic processing
* 创建QueryParser, 调用parser进行parsing，形成Query syntax tree
* 最后IndexSearcher调用search对Query syntax tree，即Query对象进行搜索，最终得到的TopScoreDocCollector就是搜索结果集合

### 2.1.3 Lucene的主要模块介绍



图[3]

* analysis模块：进行lexical analysis和inguistic processing，产生词元。
* index模块：创建索引
* store模块：读写索引，索引在内存和磁盘间的交换
* QueryParser：parsing
* search模块：利用索引进行搜索
* similarity模块：实现根据相关性进行scoring（打分）

### 2.1.4 Lucene的索引

Lucene创建索引的过程实质上是把倒排表转化成文件形式的过程。Lucene的索引可以被分成以下的几个层次：

1 索引(Index)：每个索引是一个目录，一个Lucene索引同一路径下面的许多文件共同构成。

2 段(Segment)：每个索引由多个彼此之间独立的段组成，新的段在添加新文档之后生成，也可以合并多个不同的段。metadata文件是segments.gen和segments\_5，段的某些属性等信息保存在这两个文件中。

3 文档(Document)：索引的建立的基本单位，每个的段中存储了不同的多篇文档。文档刚被添加时，用一个新生成的段单独存储，然后，段开始合并后，将段中的文档也合并到这个合并后的段中。

4 域(Field)：文档包含比如标题，时间，正文，作者等不同类型的信息，一般需要分别索引后保存到不同的域。

5 词(Term)：词是建立索引过程中的最小单位。

Lucene的索引结构包括正向信息和反向信息。

* 正向信息指：保存了从索引到词所有层次的包含关系，即索引(Index) –段(segment) –文档(Document) –域(Field) –词(Term)，这个指数包含这些段，每个段包含这些文件，每个包含这些域，每个域包含这些单词;因为它是一个层次结构，层次结构的每个级别保存信息和元信息的一个新的水平，该属性信息，如介绍中国地理书，中国地理应先介绍概述，并包含了许多中国各省，每个省介绍了我省的基本情况和城市包含了一些，每个城市部分介绍了城市的基本情况，并包含许多县，每个县的具体情况，每个县的具体描述。
* 反向信息指：保存了词典到倒排表的映射：词(Term) –文档(Document)

比较一下Lucene和数据库：

|  |  |
| --- | --- |
| Lucene | 数据库 |
| 索引数据源：doc(field1,field2,…) doc(field1,field2,…)  经过Lucene indexer的处理，变为Lucene Index.用户搜索时，Lucene searcher利用之前生成的Index，查找最佳匹配的文档结果输出：Hits(doc(field1,field2) doc(field1...)) | 索引数据源：record(field1,field2...) record(field1..)  数据库执行insert操作，将数据插入表中，同时生成对应字段的索引；对数据库进行查询时，数据库引擎执行select操作，在索引的基础上进行查询结果输出：results(record(field1,field2..) record(field1...)) |
| Document：一个需要进行索引的“单元” 一个Document由多个字段组成 | Record：记录，包含多个字段 |
| Field：字段 | Field：字段 |
| Hits：查询结果集，由匹配的Document组成 | RecordSet：查询结果集，由多个Record组成 |

表[1]

由于数据库索引不是为全文索引设计的，因此，使用like "%keyword%"方式查询时，数据库索引是起不到加快查询速度作用的，导致在查询时，搜索的过程又退化成了类似于从头到尾的遍历了，所以这对于含有模糊查询的数据库服务来说，like对性能的危害是灾难性的。更糟糕的情况，如果是需要对多个关键词进行模糊匹配：like"%keyword1%" and like "%keyword2%" 其效率也就可想而知了。

所以建立一个高效检索系统的关键是建立一个类似于科技索引一样的反向索引机制，将数据源（比如多篇文章）排序顺序存储的同时，有另外一个排好序的关键词列表，用于存储关键词-->文章映射关系，利用这样的映射关系索引：[关键词-->出现关键词的文章编号，出现次数（甚至包括位置：起始偏移量，结束偏移量），出现频率]，检索过程就是把模糊查询变成多个可以利用索引的精确查询的逻辑组合的过程。从而大大提高了多关键词查询的效率，所以，全文检索问题归结到最后是一个排序问题。

### 2.1.6 Lucene计算文档和查询语句的相关性

得到了想要的文档后，对于查询结果应该按照与查询语句的相关性进行排序，越相关者越靠前。不妨把查询语句看做一篇小的文档，对文档与文档之间的相关性(relevance)进行打分(scoring)，分数高的相关性好，就应该排在前面。

**首先，一个文档有很多词(Term)组成，**如come, go, that等. **其次对于文档之间的关系，不同的Term权重不同**，比如对于本篇文档，search, Lucene, full-text的权重就较高，this, a , what的权重就低一些。所以如果两篇文档都包含search, Lucene，full-text，这两篇文档的相关性好一些，然而就算一篇文档包the,a,are，另一篇文档不包含the,a,are，也不能影响两篇文档的相关性。因此分析文档之间的关系，先找出哪些单词，记录了最高的权重权之间的关系。然后确定这些词之间的关系。找到合适的词语（期限）过程来计算文档的重要性，称权重（weight）

话语权的计算权重（term weight）有两个参数，第一个是字（term），第二个是一个文档（document）。正确的字重表示这个词本文档中的重要性，更重要的词有更大的权重，因而在文档之间的相关性的计算中发挥更大的作用。分析单词之间（term），以获得与使用向量空间模型算法（vector space model）的工艺文件之间的关系。

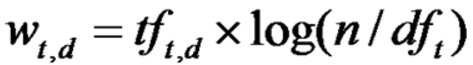
向量空间模型算法以及Lucene的打分机制

**1 计算权重(Term weight)的过程。**

有两个因素会影响一个词在一篇文档中的权重：

* Term Frequency (tf)：即此词在此文档中出现的频数。tf 越大，则这个词的重要性越高。
* Document Frequency (df)：这个词被多少文档包含。df 越大，则这个词的重要性越低。

次数的字出现在文档中，越多的重要文件的字描述，诸如“搜索”的词出现在本文档中多次，本文档中描述主要发言的这一问题。然而，一个英语文档中的次数这似乎更重要的是不要解释这一术语，它是由所述第二因子来调整，第二个因素表明，有包含单词更多的文件，这表明这个词太普通足够这些文件中，并因此降低重要性区分。



n指所有文档的总数

**2 判断Term之间的关系从而得到文档相关性的过程，也即向量空间模型的算法(VSM)**

把文档看作一系列词，每一个词有一个权重值，不同的词在文档中的权重影响了文档相关性的打分计算。我们把此文档中所有词(term)的权重(term weight) 看作一个向量。

Document = {t1, t2, …… ,tN}

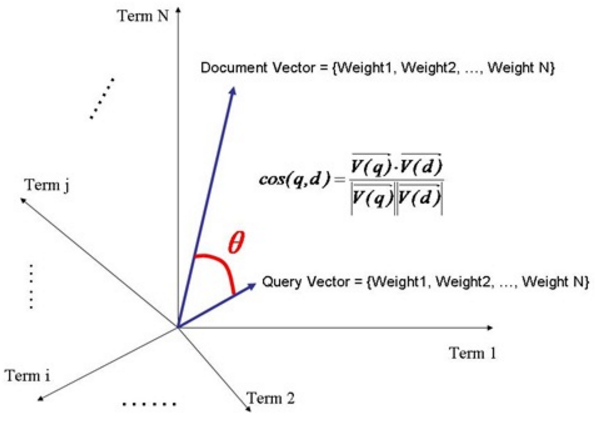
Term Weight = {weight1, weight2, …… ,weight N}

同样我们把查询语句看作一个简单的文档，也用向量来表示。

Query = {t1, t2, …… ,tN }

Query Weight = {weight1, weight2, …… , weight N}

假设在一个N维空间中，我们把所有的文档向量及查询向量放到里面，每个词(term)是一维，如图：



图[4]

通常的设定是相关性和两个向量之间的夹角呈负相关。而计算夹角大小的直接方法就是计算两个向量的余弦值，因此，我们计算角度作为相关得分的余弦，较小的角，更大的余弦值，分数越高，越大的相关性。

一般情况下，查询语句是比较短的，它包含的词也少，因此查询向量相应的维数少，而文档很长，相应的包含词很多，导致向量维数大。在这里，既然要放到相同的向量空间，维数必然是相同的，当维数不同时，取并集，不包含某个词，则权重为0.

打分公式如下：

例如，查询语句有10个term，共有三篇文档搜索出来。其中各自的权重如下表格：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | t1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | t10 |
| d1 |  |  | .477 |  | .477 | .176 |  |  |  | .176 |
| d2 |  | .176 |  | .477 |  |  |  |  | .954 |  |
| d3 |  | .176 |  |  |  | .176 |  |  |  | .176 |
| query |  |  |  |  |  | .176 |  |  | .477 |  |

表[2]

经过计算，三篇文档同查询语句的相关性打分：

Score(qeury,d1)=0.08

Score(qeury,d2)=0.825

Score(qeury,d3)=0.327

于是文档二相关性最高，先返回，其次是文档一，最后是文档三, 到此为止，我们可以找到我们最想要的文档了。

## 2.2中文分词技术（中文分词原理）

### 2.2.1 分词的意义

词是能够独立活动的最小的、有实际意义的语言要素，英文单词和单词之间是以空格作为天然分割符的，而汉语是以字作为基本语言单位，词语和词语之间没有明显的间隔符号，因此，中文词语分析（中文分词）是中文信息处理的基础。

### 2.2.2 IKAnalyzer分词原理

目前，中文的分词方法有很多种，经过综合考量，本项目中选用了IKAnalyzer作为中文分词的组件，IKAnalyzer是一个开源的、基于Lucene API实现的分词器，下面将简单介绍。

Lucened的分词器IKAnalyzer

IKAnalyzer是一个开源的，基于java语言开发的轻量级的中文分词语言包，它是以Lucene为应用主体，结合中国那句成分分析算法的字字典和语法。从3.0版本，IK发展作为公共字组成，独立的Lucene Java项目为导向，同时提供的Lucene实现的默认的优化。 IKAnalyzer从字典单词实现了一个简单的词歧义消除算法痕IK字独自想模拟得出的语义词。

IKAnalyzer实现了以词典分词为基础的正反向全切分算法和最大匹配分词算法，是LuceneAnalyzer接口的实现。

正反向切分算法

该算法适用于企业知识检索和搜索习惯之于互联网用户，用户可以使用这句话涵盖中国词搜索，如用"人民"搜索含"人民币"的文章，这是大部分用户的搜索思维。

完全分割要求输入序列分割，分割各方接受的形式和一些仅由几可接受的形式或分之一，一些细分忽略了分割的其他可能的形式，因此设立了分词方法的基础上，哪一部分不管模糊的正确的政策，他们很可能会错过正确的分割，导致字错误或失败。建立在基于分割方法整体分割，因为整个分割取得分割的所有可能的形式，这样就避免了在省略的形式可能分裂从根本上克服了缺陷部分割方法。

全切分分割算法可以得到在其句子的覆盖范围和字覆盖范围的形式的所有可能的分裂是100％，但整个单词分割没有被广泛使用在文本处理，其原因如下：

1) 全切分分割算法只是为了得到正确的字的前提下，因为整个分割无歧义的检测，最终的分割结果是正确的，完整的自主模糊处理方法的依赖，如果评价是错误的，会导致错误的结果。

2) 结句长度分割分割的数量随一方面是指数级增长增加将导致填充存储数据库庞大的垃圾;另一方面，当长句一定长度后，由于过度的分割形式，从而导致严重下降字效率。

最大匹配分词算法

这种方法将按照一定的策略和字符串进行分析，如果一个字符串在词典中，配型成功找到了“足够大的”机器词典条目匹配。标识一个字，根据扫描方向的不同分为正向匹配和逆向匹配。基本思想是：假定在词典中单词的最长字具有第i个汉字字符，则用字符的当前字符串作为匹配字段我发现词典前要处理的文档。我如果在字典中这样的单词，匹配成功，作为一个匹配字段是分割词的。如果你不能找到这样的字典中的单词我，匹配失败，本场将匹配的最后一个字被删除，剩下的字符串匹配流程再造......如此进行，直到找到一个匹配，即分割剩余长度的一个字或零串。这样就完成了比赛，那么直到文档已扫描我删除单词串匹配处理。

分词效果：

例子：中华人民共和国香港特别行政区   
 0 - 7 = 中华人民共和国  
 7 - 14 = 香港特别行政区

 例子：据路透社报道，印度尼西亚社会事务部一官员星期二(29日)表示，日惹市附近当地时间27日晨5时53分发生的里氏6.2级地震已经造成至少5427人死亡?，20000余人受伤，近20万人无家可归。  
   
 1 - 4 = 路透社  
 4 - 6 = 报道  
 7 - 12 = 印度尼西亚  
 12 - 14 = 社会  
 14 - 16 = 事务  
 18 - 20 = 官员  
 20 - 23 = 星期二  
 24 - 27 = 29日  
 28 - 30 = 表示  
 31 - 33 = 日惹  
 34 - 36 = 附近  
 36 - 40 = 当地时间  
 40 - 43 = 27日  
 44 - 46 = 5时  
 46 - 49 = 53分  
 48 - 50 = 分发  
 49 - 51 = 发生  
 50 - 52 = 生的  
 52 - 54 = 里氏  
 54 - 58 = 6.2级  
 58 - 60 = 地震  
 60 - 62 = 已经  
 62 - 64 = 造成  
 64 - 66 = 至少  
 66 - 71 = 5427人  
 71 - 73 = 死亡  
 75 - 80 = 20000

最大的优势是基于自动分割的词汇没有维修费用，简单，缺点是效率低指数，但对于中小型应用的基础上，2元切分的分割是足够的。切片索引2元基于尺寸和源文件差不多，但一般只有30％-40％的原始文件的是英语，索引文件不同，

|  |  |  |
| --- | --- | --- |
|  | 自动切分 | 词表切分 |
| 实现 | 实现简单 | 实现复杂 |
| 查询 | 增加了查询分析的复杂性， | 适合更复杂的查询语法规则的实现 |
| 存储效率 | 索引文件大 | 更高的索引效率，更小的文件大小 |
| 维护成本 | 无成本维护 | 高维护成本：需要分别维护中日韩等语言，词频统计等工作也需要包含其中 |
| 适用领域 | 嵌入式系统：有限的操作环境资源  分布式系统：无词汇同步问题  多语言环境：没有词汇维修费用 | 专业搜索引擎的高存储效率和查询需求 |

表[3]

# 第三章 系统设计

## 3.1系统整体架构

经过研究论证，将系统架构设计如下：

图[5]

如图[2]所示，整个系统可大致分为两部分：一是一个完整的Web应用（或服务/接口）；另外一部分是独立于Web应用之外的数据处理服务。将这些应用细分，可以分为四个模块：

1. 视图层：

这一层的主要功能是接收用户输入的检索条件，传递给代理服务器，Apache将搜索请求分发给对对应的Java服务器，进行进一步的处理；后端得到搜索结果后，将结果返回给视图层，视图层将结果分页显示，并着重显示特殊搜索结果，以方便用户快速找到需要下载的控件、文档等；此外，系统还提供了热词功能，最近一段时间被用户频繁搜索的关键字会显示在搜索框下面。热词的实现将在系统实现部分详细讲述。

1. 控制器及服务层：

这个模块的作用类似于MVC模式中的Controller和Model，将数据包装，选择对应的服务进行处理，同时接受返回的结果，再包装成适合前端页面处理的格式并返回给用户界面。这一层使用了SpringMVC作为基础框架，因为这些都是比较常见的，在此不再赘述。

1. 搜索条件解析（parsing）以及搜索过程：

用户的搜索条件是基于自然语言的，系统需要将这些语句切分成适合系统检索用的多个词语。这一部分会用到中文的分词工具，目前有很多优秀的第三方中文分词包，可以拿来使用，配合Lucene的parser包，来完成关键词提取和运算工作，在此系统中使用了IKAnalyzer这个分词工具，第二章原理中已经介绍过。在得到一系列搜索关键词之后，利用Lucene的search包可以根据查询条件，检索得到结果。

1. 数据源处理并建立索引：

对于一个检索系统来说，索引的重要程度与搜索不相上下。索引的合理性直接关系到系统的检索效率和检索结果。

我们的数据源主要包括静态的HTML页面、数据库中的记录以及用户和管理员上传到平台的文档等内容，这带来的主要问题就是数据格式的不统一：相对于简单的文本，这些数据可以被看作富文本，因为不仅包括文字信息，还有一些描述样式的文字；在数据库记录中，甚至包含了二进制数据以及很多主键ID数字，这些对于搜索是没有实际意义的，因此在实际处理时需要考虑去掉。

在本系统的初步设计中，我们使用脚本（Shell或Python）程序去处理这些数据源。对于静态HTML页面，脚本首先查找指定目录下的所有HTMl文件然后处理；对于数据库中的记录，脚本将执行数据库命令将数据dump出来，然后根据dump的格式进行处理；对于文档，如word、pdf，使用了java的一些类库提取文本数据，后面也将会作出详细说明。

所有这些初步处理的目的就是统一数据格式，这些数据源最终将被处理成html标签的形式，为索引过程作准备。并且这些脚本是定时运行的，目的是为了在平台数据有变化时及时更新索引源。

## 3.2 跨站搜索设计

此外，系统要实现横跨两个站点的搜索，因此这两个站点的索引要在逻辑空间上统一起来。下图是两站索引之间的关系



图[6]

如图[3]所示，系统的搜索入口位于自助入网平台上，用户的搜索请求会分发给后端的搜索服务处理。搜索的对象——自助入网平台和技术开放社区——包含的数据在此前由脚本程序处理过，变为统一的文本文件，在经过java程序处理后，产生了各自的索引文件。搜索服务要使用这些索引，还需要一个传输程序将两站的索引统一到一个空间下。

在进行搜索时，用户可以选择搜索的范围（默认是自助入网平台），接到请求以后，搜索服务会根据用户的搜索范围确定索引的路径（由配置文件事先约定好），然后再进行相应的后续处理过程。

## 3.3 权限控制设计

平台上有一些文档和FAQ并不是针对所有用户开放的，因此需要一套权限规则来控制对用户展示的搜索结果。

在平台已有的设计上，加入但对针对搜索操作的权限控制，属于数据级别的权限控制，粒度控制在文档级别。平台的用户的所有信息都由单独的SSO系统管理，包括用户名、密码、所属机构等，权限管理理所当然要在SSO系统中管理。

下图是用户、角色、权限之间的关系：



图[7]

图中，圆形表示各个单独的用户，也即在SSO中存储的独立用户实体；椭圆形表示根据业务内容和保密级别创建的角色，每个用户在同一时间内能被分配一个角色，但一个角色可以被分配给多个用户，这也符合常识；正方形表示的是权限的概念，权限可以分配给各个文档或者被检索的网页，每个角色有多个权限，而不同角色的权限必然不完全相同，但是不同的角色之间可以有某个或多个共同的权限。

用户由用户id表示,用户与用户id一一对应；角色可以同角色代码表示，分配各用户，保存到用户表中，同时有一张单独的角色表，维护角色-权限关系；权限用权限码表示，分配给文档、网页等。由此可见，用户有哪些权限可以由用户->角色->权限这条途径得出，再与被检索内容的权限比对，可以判定用户是否能查看这些内容。

用户权限和数据权限判定的详细方法将在系统实现部分阐述。

## 3.4 系统接口设计

### 3.4.1 搜索模块相关数据模型说明

#### 3.4.1.1 SearchRslt

SearchRslt表示搜索服务返回的结果，它包括如下字段:

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **描述** |
| type | String | **文档的类型。**包括article/api/faq/filedown/doc/docx |
| id | Integer | **主键id。**每个文档在数据库中都有一个id |
| title | String | **文档标题。**每个文档存储和显示的标题 |
| content | String | **文档内容。**文档的具体内容 |
| sysCd | String | **系统名称。**针对每个站点有一个系统标识 |

表[4]

#### 3.4.1.2 SearchController

SearchController是检索功能的控制类，它接受从前端提交过来的关键字并做后继操作，它的主要方法searchList包含以下参数：

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **描述** |
| model | Model | **org.springframework.ui.model** |
| request | HttpServletRequest | **HTTP请求类** |
| response | HttpServletrResponse | **HTTP响应类** |
| keyword | String | **关键词参数 包含在request中** |

表[5]

类中还包含有其他类提供的服务

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **描述** |
| fileDownService | FileDownService | **检索下载类文档的服务类,通过spring的AutoWired方式注入** |

表[6]

#### 3.4.1.3 KeySearchService

KeySearchService是全文检索功能的服务类，它包含以下属性：

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **描述** |
| logger | Logger | **日志类，提供打印日志功能** |
| cvsMap | Map<String,String> | **保存从磁盘上读取到的cvs数据，格式为id-title** |
| ajIndexPath | String | **自助入网平台索引存储路径** |
| upbbsIndexPath | String | **银联社区索引存储路径** |
| ajLstPath | String | 自助入网平台文档标题cvs文件路径 |
| upbbsLstPath | String | 银联社区文档标题cvs文件路径 |
| wordPath | String | Word/PDF文件存储路径 |
| filedownPath | String | 文件下载的单独索引路径 |
| ajIndexDir | Directory | aj索引的内存引用，用于加快检索速度 |
| upbbsIndexDir | Directory | 社区索引的内存引用，用于加快检索速度 |
| fdIndexDir | Directory | 文件下载索引的内存引用，用于加快检索速度 |

表[7]

KeySearchService包含的方法：

**public static ArrayList<String[]> readCsv(String lstPath) throws Exception;**

参数：

String lstPath 表示cvs的路径

返回值：

ArrayList<String[]> 字符串数组构成的列表

作用：

读取参数路径指向的cvs文件

**public static void loadCsvFile() throws Exception;**

作用：

定时任务执行，每天加载一次csv文件内容

**public static String getTitlebyID(String sysCd,int idKey,String typeName) throws Exception;**

参数：

String sysCd 站点标识

Int idKey 文档id

typeName 文档类型

返回值：

String

作用：

通过文档id获取文档具体标题

**public static List<SearchRslt> search(String word) throws Exception**

参数:

String word 检索关键词

返回值：

List<SearchRslt> 查询结果类构成的列表

作用：

搜索的具体方法，返回结果

**public static List<FileDown> searchFileDown(String keyword) throws Exception**

参数：

String keyword 关键词

返回值：FileDown类构成的列表

作用：单独检索文件下载信息

**public static List<SearchRslt>searchByIndex(String keyword,String indexPath,String sysCd) throws Exception;**

参数：

String keyword 关键词

String indexPath 索引路径

String sysCd 站点标识

返回值：查询结果类构成的列表

作用：由search/searchFileDown调用，进行具体检索

**public static String getWordNameByPrefix(String prefix)**

参数：

String prefix 文件前缀名

返回值：富文本文件名

作用：获取富文本文件的文件名

**public static String GetValueByKey(String fieldPath,String key)**

参数：

String fieldPath

String key

返回值：key对应的value值

作用：根据fieldPath对应的key取得相应value

**public static List<String> getHotWords() throws Exception;**

参数：无

返回值：当前热词的列表

作用：获取目前站点中词频最高的词语

### 3.4.2 索引模块接口

#### 3.4.2.1 GenIndex

GenIndex是一个类，包含的方法可以将处理过的数据文件处理成Lucene可检索的索引文件，并作为定时任务由shell脚本每天执行一次，包含的属性如下：

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **描述** |
| indexTmpPath | String | **索引临时存储目录** |
| indexPath | String | **索引目录** |
| filedownTmpPath | String | **可下载文件索引临时目录** |
| filedownIndexPath | String | **可下载文件索引目录** |
| count | int | **被索引文件计数** |

表[8]

包含的方法：

|  |  |  |
| --- | --- | --- |
| **方法名称** | **返回值类型** | **描述** |
| main | void | **程序主入口** |
| copyFiles | void | **将旧的索引替换为新产生的索引文件** |
| createIndex | void | **生成文件索引** |
| transformWord2Txt | void | **将word文件转换成txt纯文本** |
| transformPDF2Txt | void | **将pdf文件转换成txt纯文本** |
| indexDoc | void | **建立索引的核心方法** |
| deleteFile | boolean | **删除单个文件** |
| deleteDirectory | boolean | **删除目录以及目录下的文件** |

表[9]

方法详细说明：

**public static void main(String []args) throws Exception;**

参数：String []args

返回值：void

说明：程序调用的入口函数，由shell脚本启动，程序打包成jar包的形式

**public static void copyFiles(String srcPath,String desPath)**

参数：srcPath 源文件路径 desPath 目标文件路径

返回值：void

说明：拷贝临时目录中的索引文件到索引目录中

**public static void createIndex() throws Exception**

参数：

返回值：void

说明：生成索引的主要业务方法，将生成索引的步骤串联起来

**static void transformWord2Txt(File file,String txtPath)**

参数：file文件对象 txtPath存放结果txt的路径

返回值：void

说明：将word文件转换成txt纯文本

**static void transformPDF2Txt(File file, String txtPath)**

参数：file文件对象 txtPath存放结果txt的路径

返回值：void

说明：将PDF文件转换成txt纯文本

**static void indexDoc(IndexWriter writer, File file) throws Exception**

参数：writer 索引写对象 file被索引的文件或目录

返回值：void

说明：建立索引的核心方法

**public static boolean deleteFile(String sPath)**

参数：sPath 被删除文件的文件名

返回值：是否删除成功

说明：删除单个文件

**public static boolean deleteDirectory(String sPath)**

参数：sPath 被删除目录的路径

返回值：目录删除成功返回true,否则返回false

说明：删除目录（文件夹）以及目录下的文件

# 第四章 系统实现

本章将针对系统的主要模块的实现作出说明，由于篇幅限制，在此只挑选这些模块中关键的部分进行论述。

## 4.1 Web模块

Web模块的作用是提供用户搜索的入口，这个模块接受用户输入的参数，经apache处理后，分发给相应的控制器处理，控制器会调用一些服务，完成处理过程；后端处理完搜索得到排序好的结果后，返回给前端，前端将结果展示给用户。

### 配置文件

所需配置文件均需放置在项目的classpath路径下，不同配置文件的具体配置方法如下所述:

1. **applicatioinContext.xml**

*applicationContext.xml*为配置springMVC的配置文件，约定了前端velocity模板与后端的映射关系。以下是springMVC配置文件的部分主要内容。

|  |
| --- |
| <bean id=”*velocityConfigurer*” class=”*org.springframework.web.servlet.view.velocity.VelocityConfigurer*”>  <property name=*”resourceLoaderPath”*>  <value>views</value>  </property>  <property name=*'velocityProperties'*>  <props>  <prop key=”f*ile.resource.loader.cache*”>true</prop>  <prop key=”*file.resource.loader.modificationCheckInterval*”>2</prop>  <prop key=”*input.encoding*”>UTF-8</prop>  <prop key=”*output.encoding*”>UTF-8</prop>  <prop key=”c*ontentType*”>text/html;charset=UTF-8</prop>  </props>  </property>  </bean> |

在这个配置文件中，需要配置一个bean,这个bean负责加载velocity的配置类，将velocity注入到Spring的启动中。

这个id为*velocityConfigurer* bean的resourceLoaderPath属性表示模板文件的存放路径，将显示页面的velocity模板放到这个目录下面，velocity引擎将会按照命名规则自动加载并渲染对应的模板，输出给浏览器显示。

velocityProperties属性表示velocity模板的配置参数，是一个map的数据结构，包括是否设置缓存（f*ile.resource.loader.cache*）、自动检查文件更新（*file.resource.loader.modificationCheckInterval*）、输入参数字符编码（*input.encoding*）、输出参数字符编码（*output.encoding*）以及http的content-type（c*ontentType* text/html;charset=UTF-8）。

1. **applicationContext-quartz.xml**

该配置文件是用来配置定时任务框架Quartz的，定时任务包括读取CSV文件，更新内存索引等，下面是一个配置文件部分内容。

<bean id=”*CsvFileLoadScheduleJob*” class=”*com.unionpay.autojoin.web.search.KeySearchService*”></bean>

<bean id=”*CsvFileLoadScheduleMethod*” class=”*org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean*”>

<property name=*”targetObject”*>

<ref bean=”*CsvFileLoadScheduleJob*”/>

</property>

<property name=*”targetMethod”*>

<value>loadCsvFile</value>

</property>

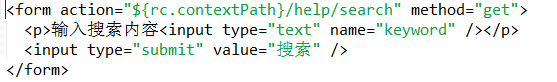
</bean>

在这个配置文件中的bean负责定时任务。*CsvFileLoadScheduleJob*指定了 名为*com.unionpay.autojoin.web.search.KeySearchService*的类执行定时任务。*CsvFileLoadScheduleMethod*指定了名为*org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean*的工厂类执行上一个bean设置的定时任务，属性targetObject为目标任务，用ref关联已经定义好的bean,属性targetMethod指定执行定时任务的方法。

### 前端页面实现

前端页面主要由velocity模板（嵌入HTML代码中）、必要的javascript和CSS组成。这里挑选了一些重要的部分。

搜索入口：



其中$(rc.contextPath)是利用velocity模板给所有页面设置的全局变量，代表统一的访问路径（在后端赋值）。

搜索结果页面：

特殊下载展示

<div class="fdlist">

#foreach($fd in $fdlist)

<div class="fd\_class">

<img src="$fd.getImgPath()">

<a href="$fd.getDownloadPath()">$fd.getName()</a>

<img src="${rc.contextPath}/img/download.png">

<a href="$fd.getDownloadPath()">下载</a>

<span>软件授权：$fd.getAuth()</span>

<span>更新时间：$fd.getUpdateTime()</span>

<span>软件版本：$fd.getVersion()</span>

<span>所属分类：$catagory.getName()</span>

</div>

#end

</div>

其他普通搜索结果：

<div class="searchList">

#foreach($item in $searchRslt)

<div class="searchItem">

#if($item.getType() == "file")

<a href="{$rc.contextPath}/download/file/$item.getFileName()">$item.getTitle</a>

#elif ($item.getType() == "common")

<a href="{$rc.contextPath}/help/$item.getId()">$item.getTitle</a>

#else

无标题

#end

<span class="summary">$item.getSummary()</span>

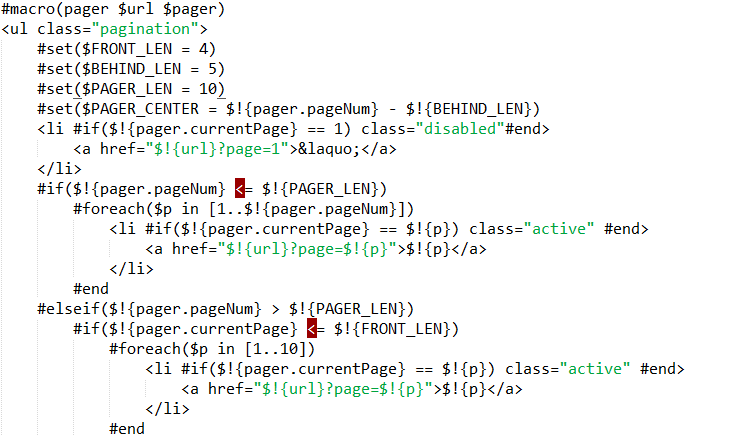
<span class="source">来源：$item.getSource()</span>

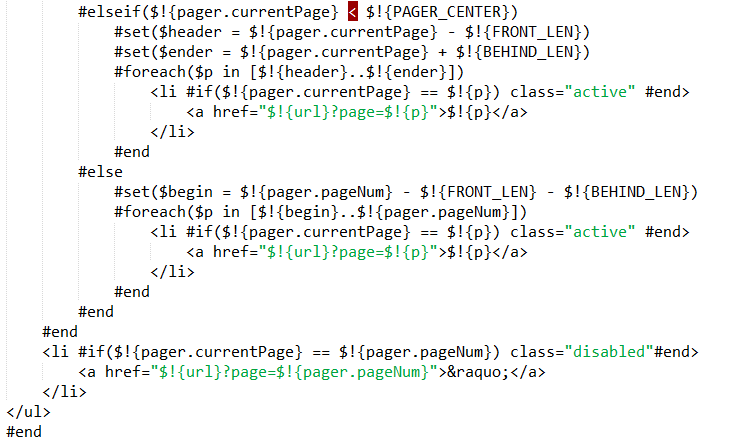
</div>

#end

</div>

页面分页：





### 控制器实现

控制器负责接收前端的请求并交给服务层处理，将处理够的结果整理包装返回给浏览器。主要的控制器是SearchController

SearchController:

public class SearchController{

// 一些属性和服务

@RequestMapping(value="search")

public String search(Model model,

HttpServletRequest request,

HttpServletrResponse response,

String keyword){

List<SearchRslt> result = new ArrayList<SearchRslt>();

result = keySearchService.search(keyword);

List<FileDownRslt> fdResult = new ArrayList<FileDownRslt>();

// 一些其他操作

model.add("searchRslt",reslut);

model.add("fdlist",fdResult);

}

}

## 搜索服务模块

这个模块提供搜索的服务，web模块的控制器将数据指派给服务接口，springMVC将这些服务作为bean注入到上下文中，可以由controller调用。

### maven配置文件

这个模块用到了Lucene的工具包，因此需要在maven的pom文件中配置jar包，在IDE中会自动下载，打包时也会自动打入war包。

自助入网平台的搜索功能依赖Lucene实现，目前jar包均已作为maven项目的形式发布到开发中心的Maven仓库内，可以通过如下配置引用, 如:

|  |
| --- |
| <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-core</artifactId>  <version>4.2.7</version>  </dependency>  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-analyzers-common</artifactId>  <version>4.2.7</version>  </dependency>  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-queryparser</artifactId>  <version>4.2.7</version>  </dependency>  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-highlighter</artifactId>  <version>4.2.7</version>  </dependency>  <dependency>  <groupId>org.wltea</groupId>  <artifactId>IKAnalyzer</artifactId>  <version></version>  <scope>system</scope>  </dependency>  <dependency>  <groupId>org.apache.lucene</groupId>  <artifactId>lucene-misc</artifactId>  <version>4.2.7</version>  </dependency> |

使用maven仓库中的最新的客户端版本。

IKAnalyzer.xml配置文件

这个配置文件的作用是为IKAnalyzer对象配置属性，IKAnalyzer是一个第三方的分词器，允许用户配置自己的扩展字典，也可以配置自己的扩展停止词字典，合理配置字典可以提高分词的准确率。

<properties>

<comment>IK Analyzer 扩展配置</comment>

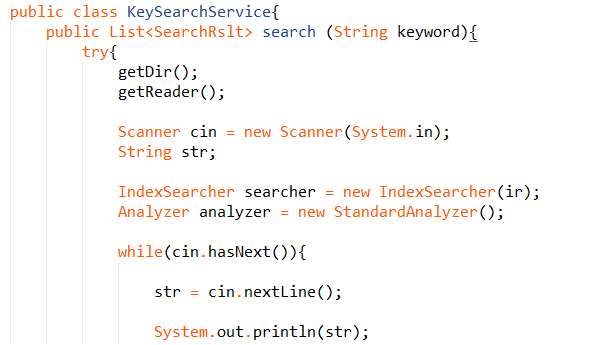
<entry key=”*ext\_dict*” >ext.dic</entry>

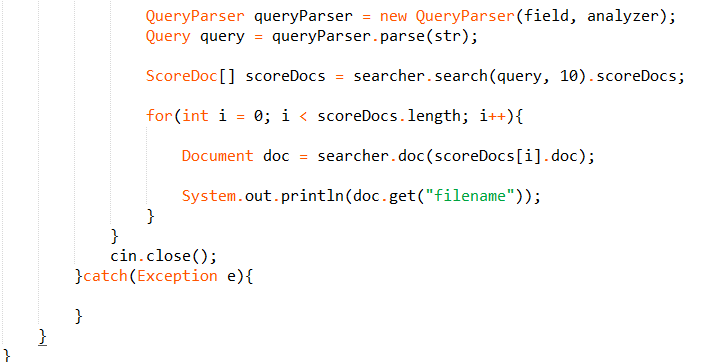
<entry key=”*ext\_stopwords*” >stopword.dic</entry>

</properties>

### 服务实现

KeySearchService





indexReader = IndexReader.open(FSDirectory.open(indexFile));

// 创建一个排序对象，其中SortField构造方法中，第一个是排序的字段，第二个是指定字段的类型，第三个是是否升序排列，true：升序，false：降序。

Sort sort = new Sort(new SortField[] {new SortField("title", SortField.STRING, false),new SortField("content", SortField.STRING, false) });

// 创建搜索类

indexSearcher = new IndexSearcher(indexReader);

// 下面是创建QueryParser 查询解析器

// QueryParser支持单个字段的查询，但是MultiFieldQueryParser可以支持多个字段查询

QueryParser queryParser = new MultiFieldQueryParser(Version.LUCENE\_36, new String[] { "title", "content" },analyzer);

// 利用queryParser解析传递过来检索关键字，完成Query对象的封装

Query query = queryParser.parse(keyword);

splitWord(keyword, true); // 显示拆分结果

// 执行检索操作

TopDocs topDocs = indexSearcher.search(query, 5, sort);

System.out.println("一共查到:" + topDocs.totalHits + "记录");

ScoreDoc[] scoreDoc = topDocs.scoreDocs;

//Lucenen也支持高亮功能，正常应该是<font color='red'></font>这里用【】替代，使效果更加明显

SimpleHTMLFormatter simpleHtmlFormatter = new SimpleHTMLFormatter("【", "】");

Highlighter highlighter = new Highlighter(simpleHtmlFormatter,new QueryScorer(query));

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

// 内部编号 ,和数据库表中的唯一标识列一样

int doc = scoreDoc[i].doc;

// 根据文档id找到文档

Document mydoc = indexSearcher.doc(doc);

String id = mydoc.get("id");

String title = mydoc.get("title");

String content = mydoc.get("content");

TokenStream tokenStream = null;

if (title != null && !title.equals("")) {

tokenStream = analyzer.tokenStream("title",new StringReader(title));

title = highlighter.getBestFragment(tokenStream, title);

}

if (content != null && !content.equals("")) {

tokenStream = analyzer.tokenStream("content",new StringReader(content));

// 传递的长度表示检索之后匹配长度，这个会导致返回的内容不全

content = highlighter.getBestFragment(tokenStream, content);

}

//如果使用了高亮显示的操作，查询的字段中没有需要高亮显示的内容 highlighter会返回一个null回来。

articleList.add(new Article(Integer.valueOf(id),title == null ? mydoc.get("title") : title,content == null ? mydoc.get("content") : content));

}

} catch (CorruptIndexException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (InvalidTokenOffsetsException e) {

e.printStackTrace();

} catch (ParseException e) {

e.printStackTrace();

} finally {

if (indexSearcher != null) {

try {

indexSearcher.close();

} catch (IOException e1) {

e1.printStackTrace();

}

}

if (indexReader != null) {

try {

indexReader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

for (Article article : articleList) {

System.out.println(article);

}

long endTime = System.currentTimeMillis();

System.out.println("索引文件生成，时间 " + (endTime - startTime) + "毫秒。");

由于涉及公司业务机密，以上代码为除去敏感数据后的样例。

## 索引管理模块

### 配置文件

ajConfig.properties配置文件

这个配置文件配置了一些与生成数据文件、建立索引相关的内容，样例如下：

|  |
| --- |
| shareDir=/GLNFSAPS/AJNFS  sysCd=aj |

createIndex.jar运行时会读取这个配置文件并进行一些变量的初始化设置，务必保证文件可读，否则会使索引生成失败。

IKAnalyzer.xml配置文件

这个配置文件的作用是为IKAnalyzer对象配置属性，IKAnalyzer是一个第三方的分词器，允许用户配置自己的扩展字典，也可以配置自己的扩展停止词字典，合理配置字典可以提高分词的准确率。

<properties>

<comment>IK Analyzer 扩展配置</comment>

<entry key=”*ext\_dict*” >ext.dic</entry>

<entry key=”*ext\_stopwords*” >stopword.dic</entry>

</properties>

ext\_dict是IKAnalyzer的扩展词典，ext\_stopwords是扩展停止词，这两个文件可以由开发者自己编写，放在classpath的目录下面。

### 创建索引

public void createIndexFile() {

long startTime = System.currentTimeMillis();

Directory directory = null;

IndexWriter indexWriter = null;

try {

// 创建哪个版本的IndexWriterConfig，根据参数可知lucene是向下兼容的

IndexWriterConfig indexWriterConfig = new IndexWriterConfig(Version.LUCENE\_47, analyzer);

// 创建磁盘目录对象

directory = new SimpleFSDirectory(indexFile);

indexWriter = new IndexWriter(directory, indexWriterConfig);

//获取articlelist articlelist即为数据源初步处理时产生的统一格式文本文件列表

getArticleList();

// 为了避免重复插入数据，每次测试前 先删除之前的索引 indexWriter.deleteAll();

// 获取实体对象

for (int i = 0; i < articleList.size(); i++) {

Article article = articleList.get(i);

// indexWriter添加索引

Document doc = new Document();

doc.add(new Field("id", article.getId().toString(),Field.Store.YES, Field.Index.NOT\_ANALYZED));

doc.add(new Field("title", article.getTitle().toString(),Field.Store.YES, Field.Index.ANALYZED));

doc.add(new Field("content", article.getContent().toString(),Field.Store.YES, Field.Index.ANALYZED));

// 添加到索引中去

indexWriter.addDocument(doc);

System.out.println("索引添加成功：第" + (i + 1) + "次！！");

} catch (IOException e) {

e.printStackTrace();

} finally {

if (indexWriter != null) {

try {

indexWriter.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if (directory != null) {

try {

directory.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

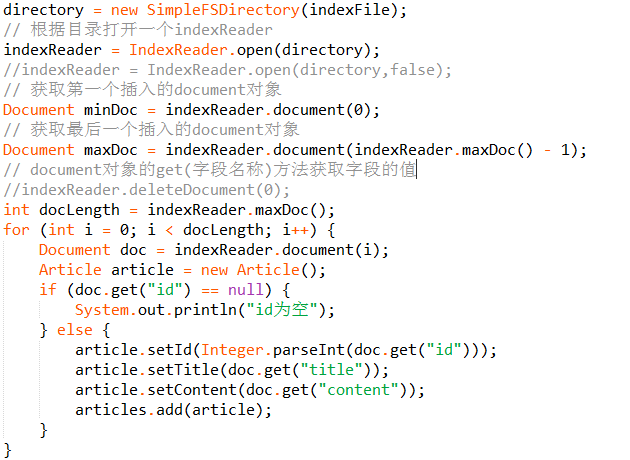
long endTime = System.currentTimeMillis();

System.out.println("创建索引文件成功，总共花费" + (endTime - startTime) + "毫秒。");

}

这是创建索引的主要过程，其中需要一些读取文件的操作，在此不再赘述。创建哪个版本的IndexWriterConfig，根据参数可知lucene是向下兼容的，选择对应的版本就好directory = new SimpleFSDirectory(indexFile) 创建磁盘目录对象，indexWriter = new IndexWriter(directory, indexWriterConfig)创建一个indexWriter; getArticleList()方法获取articlelist， articlelist即为数据源初步处理时产生的统一格式文本文件列表，存放在磁盘上；在遍历文本文件之前，为了避免重复插入数据，每次测试前 先删除之前的索引，indexWriter.deleteAll();每遍历到一个文件，将文件的文件名和文件内容分别用doc.add()方法添加，最后将这个document加入indexWriter的暂存区中；全部文件处理完毕后，indexWriter批量把索引写入磁盘（或内存）。

### 查询索引记录



这个是读取索引方法的一部分，这个方法可以直接读取索引文件，查询索引记录。

### 索引缓存/内存索引

在索引文件较小的时候（几M到几十M），此功能被激活使用，功能就是把生成好的索引文件读到内存中并使用静态变量保存，此后系统可以直接从内存中读取索引，加快搜索速度。

public static Directory dir = null;

File indexFile = new File(indexPath);

Directory fsDir = FSDirectory.open(indexFile);

// 1、启动时，将磁盘中的索引读取到内存中

ramDir = new RAMDirectory(fsDir);

IndexWriterConfig ramConf = new IndexWriterConfig(Version.LUCENE\_35,

analyzer);

// 运行程序时操作内存中的索引

IndexWriter ramIndexWriter = new IndexWriter(ramDir, ramConf);

Document document = File2Document.file2Document(filePath01);

ramIndexWriter.addDocument(document);

ramIndexWriter.close();

// 2、退出时将内存中的索引保存到磁盘中

IndexWriterConfig fsConf = new IndexWriterConfig(Version.LUCENE\_35,

analyzer);

IndexWriter fsIndexWriter = new IndexWriter(fsDir, fsConf);

fsIndexWriter.addIndexes(ramDir);// 把另外几个索引库中的所有索引数据合并到当前的索引库中

fsIndexWriter.close();

## 原始数据处理模块

本模块主要由shell脚本构建，运行在生产服务器上，负责把数据库、下载/上传目录中的原始文件处理成统一的纯文本文件，以便生成索引使用;还有一部分是java程序，负责处理富文本文件，提取纯文本。

### generate.sh脚本

脚本的主要构成如下所示：



大致过程就是执行MySQL命令将每一条记录查出，写入临时文件中，然后使用管道外加awk工具和sed工具对每条记录进行过滤，筛查出权限码，去掉js标签（sed ‘s/<script>.\*<\/script>//g’），去掉HTML标签和符号（sed ‘s/<[^<>]\*//g’ | sed ‘s/&nbsp;//g’），将换行符替换为空（sed ‘s/^M\\n//g’），然后将过滤后的文本重定向写入到正式的文本文件中供索引使用。

### createIndex.sh脚本

这个脚本作用只有一个，执行java命令，执行已经写好的jar包，这个jar包就是createIndex.jar，利用处理过的文本文件生成索引。

#!/bin/bash

APP\_HOME=/home/root1/test

APP\_CLASSPATH=$APP\_HOME/bin

java -Xms50m -Xmx250m createIndex.jar

exit 0

# 第五章 总结和展望

## 5.1 总结

论文分析了银行卡合作组织的自助入网平台在引导商户和机构自助遇到的问题，随着业务的不断扩大，用户越来越难快速准确地找到他们想要的帮助文档，因此需要一个针对技术平台的搜素系统。本文详细介绍了目前已有的通用解决方法，结合银联自助入网平台的网站特点和业务需求，提出了自己的企业内统一搜索系统设计方案。

论文介绍了网络上的数据按结构形式可以被分为总体两种类型，结构化数据和非结构化数据。结构化数据可以用关系型数据库很好的维护，在有些情况下，结构化数据也可以转换成非结构化数据；在需要进行全文检索时，结构化数据就成了阻碍检索效率的因素，因此全文检索要针对的对象就是非结构化数据。

结合业务需求，提出了专门针对自助入网平台和技术社区的搜索系统的解决方案。首先统一结构化数据和非结构化数据，统一处理成纯文本；然后利用Lucene工具包的API处理文档，建立索引；最后在平台中加入搜索功能，利用生成的索引进行检索，将结果返回给用户。

本文还介绍了统一搜索系统的实现，讲解了系统的需求和系统目标，在四层结构（视图-控制-服务-数据处理）的基础上设计诸多功能呢模块，同时介绍了系统的关键模块。

最后，总结了论文所完成的工作，并展望了下一步的工作。

## 5.2 工作展望

当然，企业统一搜索系统还存在着一些不足之处，我将对它进行进一步的完善和改进。

1. 系统的数据处理方案还比较简单粗暴，效率和精确度尚待改进。
2. 前端显示搜索结果的页面还比较简单，只区分了文件下载搜索结果和普通搜索结果，后续的迭代开发中需要作进一步的区分。
3. 热词功能只简单显示了频数最高的关键词，没有做调整，合并类似的词语，这需要更深入的数据挖掘知识，比较有挑战性。
4. 文档完善以及系统的维护工作。
5. 代码结构和设计模式优化。

**致 谢**

值此论文完成之际，谨向所有关心和支持我的人们致以诚挚的谢意！

首先，我要衷心地感谢我的导师吴含前教授。从论文选题、内容和整体结构的确定，到直至最后定稿，吴老师都以极其负责的态度给予悉心指导，为我提出了许多宝贵的意见和建议，使我获益良多。他渊博的学识、严谨的治学态度以及朴实的学术作风时刻激励我不断努力完善自己，对我的悉心关怀和教诲也将鼓舞我在今后的学习和工作上不断努力向上。在此，谨向吴老师致以最诚挚的感谢！

其次，还要感谢在企业实习时团队的成员。没有他们的帮助和共同努力，就没有项目的圆满成功，也就不会有本文的形成。在此，向他们表示衷心的感谢!

**参考文献**

[1] LI Y, DING H. Research and Application of Full Text Search Based on Lucene [J][J]. Computer Technology and Development, 2010, 2: 1-5.

[2] McCandless M, Hatcher E, Gospodnetic O. Lucene in Action: Covers Apache Lucene 3.0[M]. Manning Publications Co., 2010.

[3] Haizhou L, Baosheng Y. Chinese word segmentation[J]. Language, 1998, 212: 217.

[4] 李炳练. 基于 Lucene 的中文分词全文搜索引擎设计与实现[J]. 电脑知识与技术: 学术交流, 2015 (5): 236-237.

[5] 李胜东, 吕学强, 孙军, 等. Lucene 全文索引效率的改进[J]. 山东大学学报 (理学版), 2015, 50(07): 76-79.

[6] 彭波. 搜索引擎的混合索引技术[J]. 计算机工程与应用, 2004, 40(22): 16-19.

[7] 李明宙, 罗艳, 王宗义. Lucene 全文检索引擎的结构机制与应用方式[J]. 广西科学院学报, 2010 (4): 433-435.

[8] 冯斌. 基于 Lucene 小型搜索引擎的研究与实现 [D][D]. 武汉: 武汉理工大学, 2008.

[9] 管建和, 甘剑峰. 基于 Lucene 全文检索引擎的应用研究与实现[J]. 计算机工程与设计, 2007, 28(2): 489-491.

[10] 韩璐. 在 UNIX 系统下用 shell 编程实现对文件的操作[J]. 中国科技信息, 2006 (13): 131-132.

