

**Geeking搜索引擎项目报告**

团队名称：GeeK

团队成员：

陈新荃 2014E8018461048

高妍 201428013229077

林裕杰 201428017729020

肖卡飞

2014年12月14日

目录

[目录 2](#_Toc406232280)

[**一．项目总体介绍** 3](#_Toc406232281)

[**二．设计方案** 4](#_Toc406232282)

[1. 网页爬虫 5](#_Toc406232283)

[1.1 爬取策略 5](#_Toc406232284)

[1.2 具体实现 5](#_Toc406232285)

[2. 索引构建 6](#_Toc406232286)

[2.1 数据结构 6](#_Toc406232287)

[2.2 构建方法 8](#_Toc406232288)

[2.3 网页过滤 9](#_Toc406232289)

[2.4 文本分词 10](#_Toc406232290)

[3. 检索策略 11](#_Toc406232291)

[3.1 检索流程 12](#_Toc406232292)

[3.2 结果排序 12](#_Toc406232293)

[3.3 结果聚类 14](#_Toc406232294)

[4. 前端处理 17](#_Toc406232295)

[4.1 页面元素 17](#_Toc406232296)

[4.2 自动补齐 18](#_Toc406232297)

[4.3 搜索词推荐 19](#_Toc406232298)

[4.4 摘要快照及高亮 21](#_Toc406232299)

[**三．测试与评估** 22](#_Toc406232300)

[1. 测试环境 22](#_Toc406232301)

[2. 测试结果 23](#_Toc406232302)

[2.1 功能测试 23](#_Toc406232303)

[2.2 性能测试 24](#_Toc406232304)

[**四．创新点** 25](#_Toc406232305)

[1. 相似度计算 25](#_Toc406232306)

[2. 中间字自动补全 26](#_Toc406232307)

[3. 网页聚类算法 26](#_Toc406232308)

[**五．总结** 26](#_Toc406232309)

[1. 项目总结 26](#_Toc406232310)

[2. 个人总结 27](#_Toc406232311)

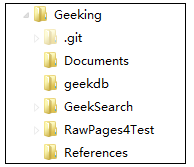
# **一．项目总体介绍**

Geeking，是一款体育新闻搜索引擎，由GeeK团队开发。Geeking能够爬取各大门户网站的体育新闻（目前支持搜狐、腾讯、网易、MSN），并建立成倒排索引，实现体育新闻搜索。其功能包括：相关度排序、相似新闻聚类、搜索词推荐、网页摘要、关键词高亮、网页快照等。

Geeking项目由Java语言编写，总代码量约为3750行（包括java、jsp文件以及空行）。该项目的实现除了团队编写的核心内容之外，还采用了htmlparser（网页过滤）、ansj\_seg（ict.中文分词）等第三方工具辅助开发。

团队使用GitHub作为协同开发工具，目前项目版本为0.2，总共发生了超过200次代码提交。项目地址为：<https://github.com/UCAS-GeeK/Geeking>。

项目目录结构如下：



Documents：团队开发过程中积累的开发文档，包括例会日志、协同开发教程、设计文档等等。

geekdb：团队开发过程中使用的小型数据库。

GeekSearch：项目工程，开发平台为eclipse。

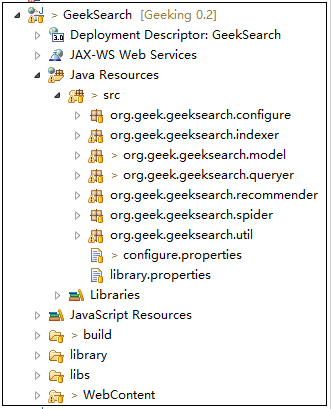
RawPages4Test：测试用的html网页。

References：参考文档，包括代码规范等。

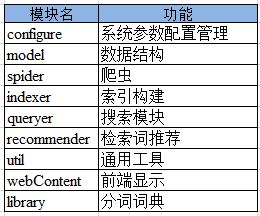
本报告后续内容将根据Geeking项目工程GeekSearch来展开。

# **二．设计方案**

GeekSearch工程主要模块结构如下：



其中，主要的模块所实现的功能如下：



下面详细介绍各个模块的设计与实现。

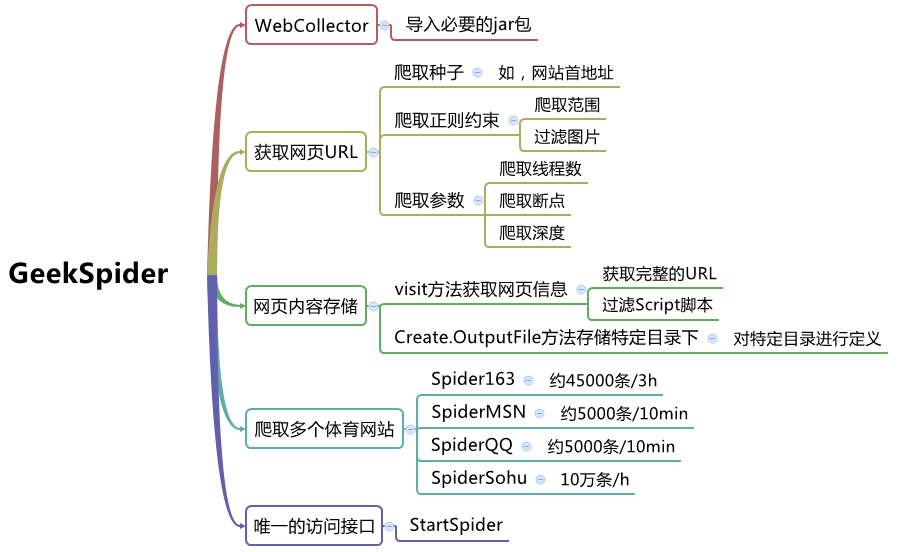
## 1. 网页爬虫

### 1.1 爬取策略

本项目的GeekSpider是基于开源JAVA爬虫WebCollector实现而成。WebCollector是一个无须配置、便于二次开发的JAVA爬虫框架（内核），它提供精简的API，只需少量代码即可实现一个功能强大的爬虫。GeekSpider在此内核上采用基于广度优先算法的多线程网页获取，并通过一定正则约束条件，剔除获取到本地的网页的图片信息，script脚本信息等。并以网页的URL命名保存在本地（其中‘：’用‘$’代替，‘/’用‘#’代替，如：http$##sports.163.com#04#0918#23#10J

MFUK400051CA0.html）。

### 1.2 具体实现



以爬取Sports163为例：

1. 继承WebCollector中的BreadthCrawler父类，派生出Spider163类；
2. 爬取网页信息；
   1. 设定爬取种子入口，一般把爬虫的种子设为网站的首页；
   2. 定义正则约束条件，给定爬取约束范围并且过滤网页中的图片格式信息；
   3. 爬取参数的设定(断点爬取、爬取深度、爬取线程数等）；
   4. 自定义visit函数访问获取的URL页面信息，并通过自定义的CreatHtml类的OutputFile方法把网页信息保存在指定目录下，其中在保存本地的过程中剔除不必要的Script脚本信息，为快照的实现减小文本规模。

按照类似方法，能够获取 MSN，QQ，Sohu等门户网站的体育新闻信息。

## 2. 索引构建

索引构建主要包括网页过滤抽取、相关数据库表建立等。具体数据结构和构建方法将分别在2.1和2.2中介绍，整个索引构建的流程如下：

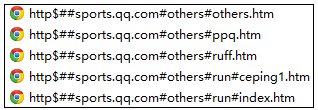
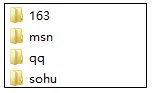
1. 从**原始网页库**读取网页；
2. 抽取关键信息（title，keywords，description等）生成**网页信息表**（PagesIndex）并写进数据库；
3. 提取网页正文；
4. 对正文进行分词并将词项映射到**词项ID**；
5. 根据分词结果生成文档索引（DocsIndex, 即**正向索引**），同时生成**词项ID-词项映射表**（TermsIndex），并写进数据库；
6. 读取文档索引表，合并相同词项，统计TF，DF，词项在文档中位置POS，生成**倒排索引**（InvertedIndex），并写进数据库。
7. 索引构建结束。

### 2.1 数据结构

存储在Mysql数据库中的表格包括：网页信息表、文档索引、词项ID映射表、倒排索引表，而原始网页库存储在文件系统。

**原始网页库（RawPages）**

原始网页库以html文件形式存储，以其url命名，其中，将 “/”替换成 “#”，将“：”替换成“$”，并以分类目录方式存储，如：sina、sohu。



**网页信息表（PagesIndex）**



文档ID：主键

URL：候选键，与文档ID一一对应；

标题：用于结果排序、相似度计算和显示；

摘要：用于结果排序、相似度计算和显示；

日期：网页发布日期，用于结果排序、显示；

类型：门户网站类型，用于快照时根据类型进入原始网页库目录读取文件；

关键字：用于热词推荐、显示。

**词项-词项ID 映射表（TermsIndex）**

|  |  |  |
| --- | --- | --- |
| 属性名 | 词项ID | 词项 |
| 对应名称 | TermID | Term |
| 实例 | 11 | 体育 |
| 类型 | int | varchar |
| 范围 | 0 - 4294967295 | 20B |

将文档正文进行了分词，过滤停用词之后，将词项映射成ID 。

**文档索引表（DocsIndex）**

|  |  |  |
| --- | --- | --- |
| 属性名 | 文档ID | 词项IDs |
| 对应名称 | DocID | TermIDs |
| 实例 | 1 | 11#2#13… |
| 类型 | int | text |
| 范围 | 0 - 4294967295 | 64KB |

TermIDs属性的值是TermIDs的一个序列，用#分开。

**倒排索引表（InvertedIndex）**

|  |  |  |
| --- | --- | --- |
| 属性名 | 词项ID | 文档IDs |
| 对应名称 | TermID | DocumentIDs |
| 实例 | 1 | 2|1:10:[1, 13]#2:10:[1, 13]# |
| 类型 | int | text |
| 范围 | 0 - 4294967295 | 64KB |

文档IDs格式：一个 termID 对应的一条 documentIDs 在数据库中的存储格式（其中位置信息取词项在文档索引中的index号）:

DF|docID1: tf-idf:[pos1,pos2...]#docID2: tf-idf:[pos1,pos2…]…

将每篇文档对每个词项的tf-idf权重预先计算并存储，可以在检索时加快计算文档得分速度。

### 2.2 构建方法

工具：Mysql， Navicat for MySQL

语言：SQL

生成表格：PagesIndex（网页信息表）、DocsIndex（文档索引表）、TermsInde（词项ID-词项映射表）、InvertedIndex（倒排索引表）

**创建PagesIndex（网页信息表）**

CREATE TABLE PagesIndex

(

DocID INTEGER primary key,

Url varchar(300),

Title varchar(300),

Description text,

Date varchar(30),

Type varchar(30),

Keywords varchar(300)

)

**创建DocsIndex（文档索引表）**

CREATE TABLE DocsIndex

(

DocID INTEGER primary key,

TermIDs text

)

**创建TermsIndex（词项ID-词项映射表）**

CREATE TABLE TermsIndex

(

TermID INTEGER primary key,

Term varchar(20)

)

**创建InvertedIndex（倒排索引表）**

CREATE TABLE InvertedIndex

(

TermID INTEGER primary key,

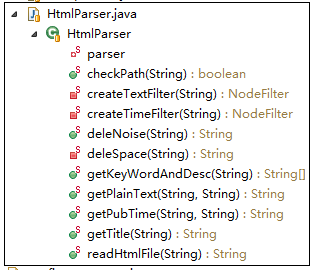
DocumentIDs text

)

### 2.3 网页过滤

本项目的网页过滤部分采用开源第三方工具htmlparser（http://htmlparser.sou

rceforge.net/），并结合了正则表达式过滤等方法，加以封装，以实现项目需求。封装后的代码结构如下：



该类主要实现了对网页信息的抽取，包括title、description、keywords、public time以及网页正文等。目前可实现对搜狐、网易、腾讯、MSN四大门户网站的新闻网页过滤。其中四大门户网站网页中的title、description和keywords内容均可以通过相同的网页标签定位并获取。但是对于public time，四大网站的标签均不同，并且对于同一个网站，其不同年份的public time标签也都不同，因此此处并未采用htmlparser来抽取，而是使用了正则表达式。

### 2.4 文本分词

项目中的分词模块使用了Ansj（https://github.com/NLPchina/ansj\_seg）第三方开源中文分词工具。Ansj作者孙健，其实现是基于中科院的 ictclas 中文分词算法，比其他常用的开源分词工具（如mmseg4j）的分词准确率更高。

**分词方法**

Ansj主要分词方法有：基本分词（BaseAnalysis）、精准分词（ToAnalysis）、nlp分词（NlpAnalysis）、面向索引的分词（IndexAnalysis）。四种方法的对比如下表：



考虑到基本分词功能单一，因此不采用该方法。而对于面向索引分词，特点是充分考虑了歧义句的因素，比如对“主副食品”分词，能够产生[主副食品/n, 主副食, 副食, 副食品, 食品]五个分词结果，鉴于其分词结果会派生出很多新词，会极大增加倒排索引的规模，因此不在本项目总采用。

NLP分词功能强大，能够支持新词发现功能。但是执行时间相对较长，并且非常消耗内存。而精准分词在易用性、稳定性、准确性、以及分词效率上都取得了一个不错的平衡，也是作者亲自推荐的方法（https://github.com/NLPchina/an

sj\_seg/issues/156）。

我们经过粗略测试，在同样的软硬件环境下，NLP方法初始化时间约为100秒，而精准分词方法只需10秒左右来初始化。而在内存占用方面，NLP方法相比于精准分词方法占用了极大部分内存，而且其分词时间也更久。

综合上述考虑，本项目中对于索引构建过程中对文本的分词采用精准分词方法，为了保持一致性，对于搜索语句的分词也采用该方法。

**词典**

Ansj默认分词词典有386211个词项，共6MB左右。即便如此，该词典也只涵盖了首字母从A-G的常用词项，不过在实际体验中对分词结果的影响不大。

停用词词典需要自行添加，本项目添加的停用词有1534个，词典大小共10.2KB。

## 3. 检索策略

Geeking搜索引擎的检索过程从服务器端初始化开始，具体流程如下：

1. 启动服务器，初始化搜索服务，包括初始化配置、数据库，加载词典、热词、和部分倒排索引表到内存；
2. 用户输入查询词；
3. 根据热词和历史搜索记录对查询词自动补全；
4. 对查询词分词；
5. 根据分词结果从词项映射表获取词项ID；
6. 根据词项ID从倒排索引表中获得各个词项ID对应的倒排文档集合；
7. 如果查找不到相关文档，则从数据库中读取该词项ID对应的倒排文档集合；
8. 若仍无相关文档，则执行相似单词查找，将推荐词返回给前端。
9. 将文档集合按照其长度升序排列做合并操作；
10. 根据tf-idf权重计算文档集合中每篇文档得分，并排序；
11. 对排序后的相关文档集聚类，将聚类后的最终结果返回给前端；
12. 前端显示搜索结果。

### 3.1 检索流程

用户检索的过程从服务器初始化之后开始，主要的步骤和流程如下所示：

**服务器启动**



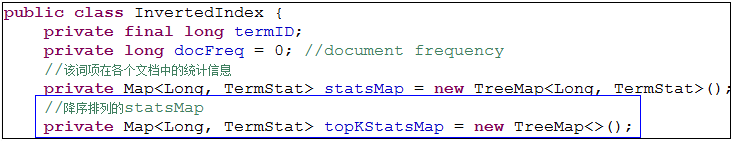
**检索流程**



### 3.2 结果排序

**胜者表**

由于倒排索引中已经存储了每篇文档的tf-idf信息，因此，在搜索引起首次启动，加载倒排索引到内存的时候，程序会对每个词项的倒排索引按照和tf-idf值由高到底排序，并且截取出TopK（在configuration.properties里可配置，默认为50）篇文档。只有这TopK篇文档才可以参与最后的相似度运算。



**合并算法**

在对每个搜索词项的相关文档集进行合并之前，程序会先按照相关文档的规模从小到大进行排序，然后再按照该顺序进行相关文档合并，从而减少不必要的合并计算。

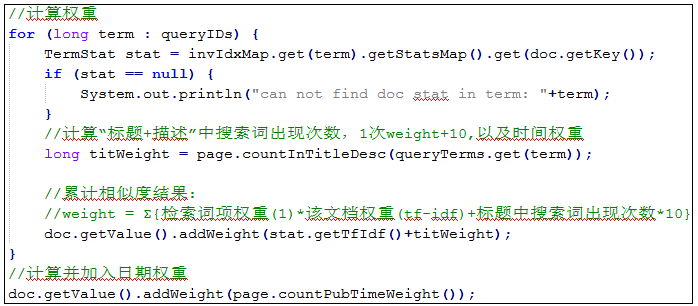


**相似度计算**

本项目权重计算基于**tf-idf**，并且充分考虑了体育新闻的**时效性**，计算了时间权重。此外，考虑到如果搜索词出现在某篇文档的**标题或者摘要**中，那么该文档的相关度应该更高。因此我们采取的相关度权重算法为：

其中，括号中前加号前半部分是nnn.ntn算法，而后半部分则是考虑到搜索词在“标题+摘要”中的权重。如果搜索词在标题和摘要中出现了次，那么其权重将会增加。而具体是增加10分的权重或者是更多，有待日后不断调试，以获得最优方案。最后的求和的部分是时效性权重，计算公式为：

其中，是指epoch时间，即是从Epoch（1970年1月1日00:00:00 UTC）开始到现在所经过的秒数，而是指从epoch到网页发布那天所经历过的秒数。公式的意思是，如果网页发布时间距离用户搜索当天越近，那么权重分数将会更高，反之更低。



代码实现如上图，整个权重分数的算法充分考虑了tf、idf、标题、发布时间这几个因素，在实际使用当中有很不错的效果。

### 3.3 结果聚类

**3.3.1对百度新闻搜索结果聚类的调研**

**调研过程**

在百度上搜索“北京冬奥会”的新闻，显示前3条新闻结果如下图：



在每条新闻右下角，矩形红色框住的地方“\*条相同新闻”就是百度搜索引擎对该新闻与之同类的其他新闻的聚类。点击第1条新闻的“35条相同新闻”进去，对这些同类新闻进行分析，查找其聚类依据，以及排序规律。将这些新闻的信息做成如下表格。序号0表示该显示新闻，序号1-10表示排序后隐藏的相同新闻。

表1 百度新闻搜索结果相同新闻聚类分析



**调研结论**

1. 同类新闻之间标题很类似，不同类新闻间标题差异很大；
2. 聚类依据是标题；

聚类过程与keywords、description无关，如上表序号11其keywords不包含与搜索词项相关的关键字，如表中description有的和标题相同，有的和正文第一段内容相同。只有标题是大同小异的，均包含和序号0（即首条显示新闻）极其类似的内容。

1. 同类新闻聚类后的相对顺序同其未聚类前的相对顺序。

同类新闻内的排序依据包括：门户、发布时间、关键字。如10、11的时间靠前，但是因为不含idf高的关键词“冬奥会”，故排在较后。这些排序依据和搜索引擎在首页显示不同栏目新闻的排序依据一致。

**3.3.2聚类算法**

**相似度定义及计算方法**

根据调研结论(2)，相似度定义为标题的相似性。考虑到我们建立索引时候已经将标题分词，这里我们采用最长公共子序列衡量，子序列每个元素是一个词项，该计算方法与k-gram相比可取得同样的准确率却有更高的效率。

**聚类过程**

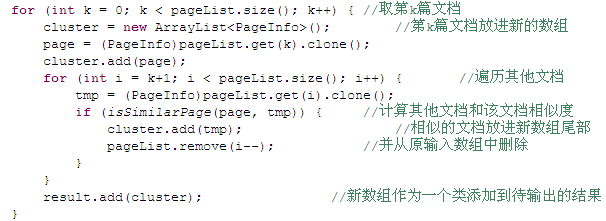
根据调研结论，由于同类新闻之间标题很类似，不同类新闻间标题差异很大，故可以通过调节合适的阈值，将一个类聚好后，再去聚下一个类。

输入： pageList，包含排好序的n篇文档；

输出： result，包含m个类cluster；

1. 按序遍历pageList取其第k篇网页pageList[k]，构造空数组claster，将pageList[k]放入claster，并从pageList中删除pageList[k]；
2. 按序遍历pageList取其第i篇网页pageList[i]，计算pageList[k]和pageList[i]的标题相似度；
3. 若相似度满足阈值要求，则将pageList[i]从放到claster尾部，并从pageList删除pageList[i]；
4. 步(2)若未遍历完pageList则回到步(2)，若遍历完，则将claster作为一个元素放到result中；
5. 步(1)若未遍历pageList完则回到步(1)，若遍历完则结束。

代码如下：

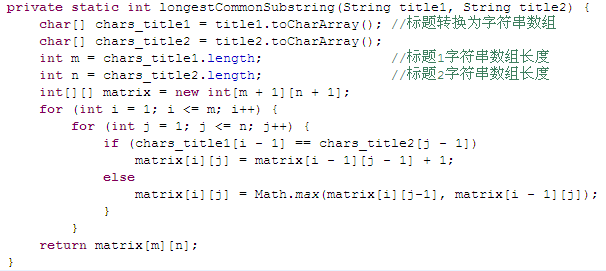


计算相似度（即最长公共子序列）过程：

输入：title1，title2（2篇文档标题）；

输出：matrix[m][n]（最长公共子序列长度）；

1. 将标题title1分词，放进字符串数组chars\_title1，m为chars\_title1长度；
2. 将标题title1分词，放进字符串数组chars\_title1，n为chars\_title2长度；
3. 用动态规划方法求解矩阵矩阵matrix[1...m][1...n]的值。



**时间复杂度**

时间复杂度为O(n2)，n为文档篇数。

由于每次计算文档相似性时采用标题的最长公共子序列衡量，时间复杂度的系数较小。假设标题的平均长度为m，分词后的平均词项个数为m/2，每次求最长公共子序列的时间为(m/2)\*(m/2) = m2/4。又由于算法中每次聚类后的文档不参与其他新类的计算，实际时间复杂度没有n2，时间复杂度和文档能聚成的类别数目有关，若结果聚成c个类时，时间复杂度为O(c\*n)。最好的情况是只聚成1个类时，时间复杂度仅为n\*m2/4。

## 4. 前端处理

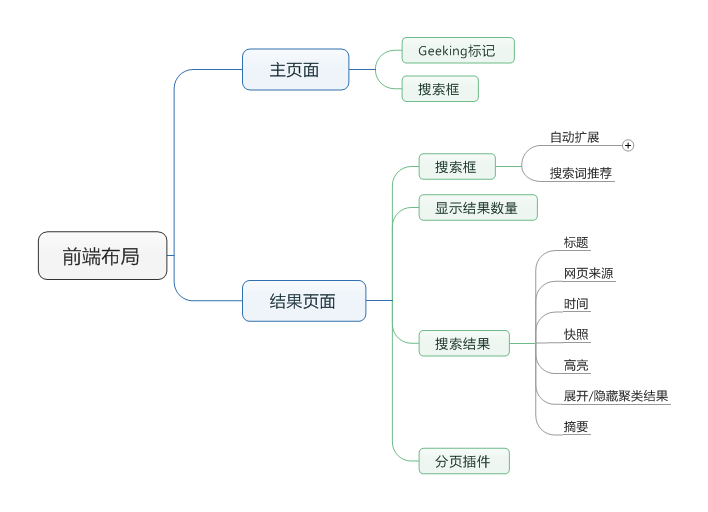
前端总体框架采用Ajax+Jquery+JSP的富客户端方式实现。使用Javascript绑定和处理所有数据，操作Document Object Model进行动态显示及交互。

AJAX具有无刷新更新技术、异步与服务器通信、前端与后端负载平衡和界面与应用分离等优点，是一个强大灵活的webservice工具。

除了基本的显示搜索结果页面外，实现功能包括**无刷新分页、自动补齐、搜索词推荐、结果聚类、网页快照**等功能。功能完善，能带给用户良好的体验。

### 4.1 页面元素

**前端布局：**



**主页面**



**搜索结果页面：**



### 4.2 自动补齐

**前端**

当用户输入查询词的时候，会在输入框的下方面动态显示一个下拉框。 同时，Jquery会监听用户在搜索框的每次输入，通过Ajax请求，从服务端获取查询结果，然后结果以JSON的数据类型返回，动态的显示在下拉框中。用户可以选择下拉框中的任意一个词，按回车键即可完成搜索。

**后端**

Server端使用简单的jsp，在初始化的时候使用关键词词库构建了一颗哈希树，接收到查询请求后，开始查询，将查询结果按照热度进行排序后，以JSON的数据格式返回。最多返回5个单词。

**页面展示**



### 4.3 搜索词推荐

**前端**

当发现某次搜索结果为空的时候，认为用户可能输入有误，会查找热词词典，返回最相似的最多三个单词，显示在搜索结果页面上，用户可以点击这些词进行搜索。分3情况显示：

1. 有搜索结果时，不推荐热词；
2. 无搜索结果时，并且能查找到相似热词，推荐热词；
3. 无搜索结果时，并且不能查找到相似热词时，不推荐。

**后台**

为了将用户的输入和字典里的词相比较，找到相似度最高的单词，采用**N-gram**算法和**LevensteinDistance编辑距离**。

1. 首先来说一个N-Gram的概念， N-Gram是指将一个单词划分成若干等长的字串，每一个成为一个gram，n就是用来控制每个gram的长度的；
2. 对于一个单词我们往往会给不同的N做多次切割，这样便于做搜索建议和拼写检查；
3. 构建词典；
4. 对于每个词典中的单词，根据两个值minN~maxN（minN, maxN是根据单词长度由专门的函数产生的）进行分割，介于两个值之间的N都算作一种分割，伪码如下：

for ng = minN to maxN

for i=0 to wordLen-ng +1

gram = word.substr(i,i+ng);

add this gram as a field to the current lucene document

end for

end for

1. 针对每个N，开头结尾的gram也加入索引域。

举例来说：对于three这个词，minN = 2, maxN =3;那么针对three这个词的对应document对象包含的Field有如下：

gram2: th hr re ee

start2: th

end2: ee

gram3: thr hre ree

start3: thr

end3:ree

通过如上的方式将词典里的每个单词都进行了索引。

1. 拼写检查；
2. 根据输入word构建查询条件；

也是通过minN, maxN来讲输入进行gram化，每个对应的分割都构成一个查询Term，同时针对每种分割也有对应的起始和结束查询，过程和构建词典的过程一样。这些条件之间是或的关系。这个用BooleanQuery可以实现。

1. 开始从词典构建的索引中使用上述条件进行查询。
2. 从查询结果中过滤掉自身（如果查询结果包含自身的话）；
3. 计算查询结果中的每个单词和输入word的编辑距离，过滤掉部分距离过小的单词；
4. 放入以编辑距离为衡量指标的优先队列中，队列长度人为确定，具体根据要查询的建议单词的多少确定；
5. 当队列长度满之后，返回结果，注意此处的结果要是以编辑距离增序的结果（使用优先队列的函数可以达到此要求）。
6. 编辑距离解释。

编辑距离是衡量两个字符串相似度的值，介于0和1之间，0表示两个字符串完全相同，1表示两个字符串具有最大的不相似程度。常用的编辑距离有：LevensteinDistance和JaroWinklerDistance. 这里使用Levenshtein算法中，两个字符串之间的距离定 义为将一个字符串转换为另一字符串所需的最少编辑次数，允许的编辑操作有插入、删除、单个字符的替换。该算法由Vladimir Levenshtein在1965年提出，并以作者名来命名。

**效果展示：**



### 4.4 摘要快照及高亮

**前端**

摘要目前是静态摘要，将新闻的导语作为摘要显示出来。并且查找到关键词，进行高亮显示，主要使用jquery动态添加标签，实现了高亮效果。

为了防止摘要过长，自动截取一部分显示。向后台发起Ajax请求，返回分词时候的query列表，使用jquery插件，将title和正文中的query都高亮显示。

快照则是在“Geeking快照”中加入一个超链接，指向缓存的网页所在的地址。

**后端**

将query的结果分词后，接受客户端的Ajax请求，将分词后的列表返回给前端。

**效果展示**



# **三．测试与评估**

本项目测试主要分为两个部分，功能测试和性能测试。其中功能测试包括了黑盒、白盒及回归测试，涵盖了项目开发过程中主要模块的功能测试以及bug修复统计。性能测试包括了索引构建、服务启动和检索过程中分词、排序、聚类等每一步的时间开销。

## 1. 测试环境

**硬件环境：**

CPU：Intel Core2 Duo 2.2GHz

Memory：3.0GB DDR2

硬盘：机械硬盘

**软件环境：**

操作系统：Windows 8.1 Enterprise

服务器：Apache Tomcat 7.0.55

编译器：Eclipse 4.3 Kepller

数据库：Mysql 5.1

JDK：1.8.0

**语料集：**

来源：网易体育，腾讯体育，搜狐体育，MSN Sports

网页数目：



## 2. 测试结果

### 2.1 功能测试

此处功能测试并不基于上述测试环境，而是整个系统开发过程中主要功能基本实现之后，进行功能测试时所遇到的bug和修复统计，具体如下表：

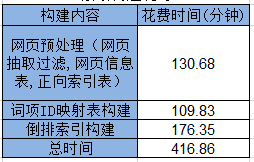
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目名称** | | Geeking搜索引擎 | **测试方法** | 黑盒,白盒,回归 | |
| **模块开发者** | | 高妍、林裕杰 | **程序版本号** | 0.2 | |
| **功能/模块** | **序号** | **功能点** | **测试用例数** | **发现bug数量** | **修复Bug数量** |
| **Web UI与交互模块** | 1 | 自动补齐功能 | 20 | 10 | 10 |
| 2 | 检索词推荐功能 | 20 | 15 | 15 |
| 3 | 检索结果聚类 | 20 | 12 | 12 |
| 4 | 整体布局调整 | 20 | 10 | 9 |
| 5 | 关键词高亮 | 25 | 10 | 10 |
| 6 | 无刷新分页 | 25 | 5 | 5 |
| 7 | 快照功能 | 25 | 2 | 2 |
| 8 | 聚类结果展开/隐藏 | 25 | 12 | 12 |
| 9 |  |  |  |  |
| **SpellCheck模块** | 1 | N-gram词典构建 | 25 | 12 | 12 |
| 2 | LevensteinDistance编辑距离。 | 25 | 2 | 2 |
| 3 | 关键词推荐处理逻辑 | 25 | 12 | 12 |
| **自测执行者** | | 高妍 | **日期** | 2014/12/9 | |
| **开发负责人** | | 陈新荃 | **日期** | 2014/12/9 | |

### 2.2 性能测试

基于上述测试环境，我们利用大型语料集，主要进行了索引构建、用户检索等性能测试。

**索引构建：**

索引构建所用时间：



构建完成后的数据库信息：



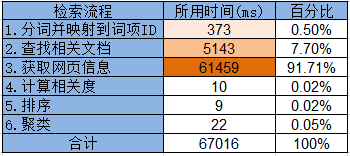
**分析：**

索引构建采用单机、单线程构建，在大型语料集下，性能较低。除此之外，由于构建索引所用计算机是五年前出厂的笔记本电脑，配置较低，也是影响索引构建速度的因素。

**检索服务：**

在小规模的语料集下，词项ID映射表和倒排索引均全部载入内存，因此检索速度基本在1秒之内，对此我们不再进行测试分析。以下测试基于上述大型语料集。

以用户输入查询词“曲棍球”为例，返回50篇相关文档。整个检索过程所耗费的时间如下：



**分析：**

可以看到，执行一次检索总共需要花费67秒的时间，这是令用户无法接受的。

性能低的原因在于词项ID映射表和倒排索引表规模较大，无法一次性载入内存，而程序的检索策略是如果在内存中找不到对应的词项ID或者倒排索引，就必须从数据库中查找，这涉及到了数据库的查找操作，同时伴随着磁盘(机械盘)的读写，而**数据库查询操作是影响性能的最主要因素**。

而整个检索过程花费时间最多的步骤在于前面三步。其中，第1步主要涉及到词项ID映射表的查询和读取；第2步主要涉及到倒排索引表的查询和读取；第3步主要涉及到网页信息表的查询和读取。值得一提的是，在查询词只有一个的条件下，第1步和第2步只需要对数据库执行1次查询读取操作。然而对于第3步，因为相关文档有50篇，因此需要对数据库的网页信息表执行50次的查询读取操作。

又因为本项目采用的是mysql数据库，在较低的硬件环境下其查询性能非常耗费时间。

因此综合上述原因，在大规模语料集下，目前本项目的检索性能并不理想。

# **四．创新点**

## 1. 相似度计算

除了在结果排序过程中使用到了胜者表以及合并算法进行优化之外，本项目还对相似度分数计算进行了创新，充分考虑了四种因素：tf、idf、标题、发布时间。

如果查询词在标题中出现的次数越多，那么权重分数将会更高；如果网页发布时间距离用户搜索当天越近，那么权重分数将会更高。

目前来看，基于本项目语料集，该相关度评分算法效果非常不错。

## 2. 中间字自动补全

🡪高妍

## 3. 网页聚类算法

新闻聚类中，根据对百度搜索新闻聚类调研得到的结论，我们采用2个标题字符串的最长公共子序列的值衡量2篇网页的相似度，该方法能使聚类时间复杂度O(n2)的常数因子很小，为m2/4（m为单个标题分词后的词项个数）；并且，根据不同类新闻间相似度很小的实际情况，我们通过预设阈值，先将一个类聚好后，再去聚下一个类。与普通的单连接算法相比，该算法在满足同样聚类准确率的前提下，能显著提高效率，若结果聚成c个类时，时间复杂度为O(c\*n)。

# **五．总结**

## 1. 项目总结

Geeking项目开始于2014年9月23日(第一次会议)，并于12月14日如期完成。

本项目并未使用过多第三方工具，核心功能均由项目组成员自主实现。并且在项目执行过程中，成员之间有着良好的沟通与思想碰撞，产生了许多创新的思想，并将其中的一部分在项目中实现出来，比如文档和检索词计算的公式、中间字的自动补全、根据标题聚类的算法等等。

不过，在进行大型语料集的索引构建和检索测试时，本项目表现出的性能并不理想。这主要是由于大型的倒排索引无法载入内存，以及mysql数据库系统的查询性能低导致的。

因此，本项目后续还可以进行如下方面的改进：

1. 考虑利用Hadoop的MapReduce编程思想进行倒排索引的构建；
2. 考虑对倒排索引进行压缩，使其可一次性载入内存；
3. 考虑使用keyValue数据库（比如Redis），提高数据库查询性能；
4. 考虑引入MVC开发框架等；

GeeK项目组团队成员来自两个不同学院，在这几个月的期间里，虽然大家都学业繁忙，但是所有人都积极参与，发挥各自所长，使得项目进展有条不紊，最后如期完成了任务。感谢每个人的努力与付出！

## 2. 个人总结

**陈新荃：**

第一次像模像样的带领一个团队完成一个项目。花了很多时间和精力在这个项目上，但是获益良多。经历过这样一个项目，除了信息检索相关的知识得到巩固和加强之外，在编程能力上也有了提高，代码的健壮性和可读性方面得到了加强。

深深感受到了项目开发过程的复杂，除了要理清思路把握项目整体的方向和进展，还需要考虑成员之间的分工合作。大到制定开发计划，小到考虑某段代码的开销等等，都需要清晰地思路和周全的考虑。这些对我来说都是历练。在此非常感谢团队成员的积极配合，项目能够圆满完成是大家齐心协力的结果，能与大家一起共事我感到很幸运。

**高妍：**

通过这次信息检索大作业，我对搜索引擎有了更深一步的理解，增强了编程能力，对于算法效率、可用性、可扩展性有了进一步的理解。掌握了JSP、Jquery、Ajax、Java等多项技术，深入理解了N-gram算法和LevensteinDistance编辑距离。其中spellcheck模块是通过阅读Lucene源码给了我跟多启发，在此基础上完成的。更重要的是，在团队合作开发的过程中，积累了团队合作经验，学会使用github，特别感谢组长陈新荃和林裕杰、肖卡飞两位队友，我们的队伍里没有大神，大家都是一起努力的菜鸟，在整个开发过程中，面对开发量大，难度高，缺乏经验这些困难，我们相互帮助，相互支持，相互鼓励，都一一克服了。在大家的共同努力下，我们很好的完成了任务。

**林裕杰：**

首先我要感谢老师给予了我们做这个项目的机会，不仅让我对信息检索过程有了更清晰的认识，还让我学习到了调研分析问题的方法。比如网页聚类的问题，根据对百度搜索新闻聚类调研的实际情况分析，得到有助于简化问题的结论，针对具体聚类对象，得到有效的能够提升聚类效率的算法。

其次我要感谢为我们项目作出许多贡献的组长陈新荃，是组长的聪明和才干让我们的团队能够做好这个项目，以及高妍、肖卡飞两位队友，是他们让这个项目变得富有意义，让我在这个项目过程中受益良多，他们都是我的学习榜样。

**肖卡飞：**