

现代信息检索实验报告

| | | |
|----|--------------------|--------|
| 一、 | 实验内容..... | - 2 - |
| 二、 | 设计方案..... | - 2 - |
| 三、 | 系统实现..... | - 2 - |
| 1. | 爬虫..... | - 2 - |
| 2. | 建立索引..... | - 3 - |
| | 1) analysis..... | - 4 - |
| | 2) document | - 5 - |
| | 3) index | - 6 - |
| | 4) store | - 9 - |
| | 5) util | - 9 - |
| 3. | 检索..... | - 10 - |
| | 1) search | - 10 - |
| | 2) similarity..... | - 12 - |
| | 3) cluster | - 13 - |
| 4. | 前端..... | - 16 - |
| 四、 | 运行测试..... | - 18 - |
| 五、 | 创新点 | - 21 - |
| 六、 | 实验分工..... | - 21 - |
| 七、 | 总结..... | - 21 - |

一、 实验内容

新闻搜索：定向采集 3-4 个体育新闻网站，实现这些网站信息的抽取、索引和检索。网页数目不少于 10 万条。能按相关度、时间、热度(需要自己定义)等属性进行排序。

二、 设计方案

使用 python 编写爬虫，从腾讯体育新闻栏目中爬取 10 万个新闻网页，格式化处理后保存到本地；使用 java 完成索引的建立和查询，为提高效率，采用 Hashmap 和索引分块技术，支持按照文档数目或者索引大小进行索引分块；使用 PHP 处理前后端交互，用户输入查询词后，通过调用 java 检索程序获取检索结果，为提供精确的分类结果对结果进行了聚类，然后显示到网页中。

三、 系统实现

1. 爬虫

爬虫的基本思想就是先爬再取，先就是把 URL 存储下来并依此为起点逐步扩散开去，抓取所有符合条件的网页 URL 存储起来继续爬取。我这里用到了广度优先的方法，从一个网页开始，把自带连接加入队列，然后对队列中的其它元素执行相同操作，其中带有查重机制，最后得到的是不同网页的闭包。

爬：Spider 是自己编写的类，用来从一个域（或域组）中抓取信息。他们定义了用于下载的 URL 列表、跟踪链接的方案、解析网页内容的方式，以此来提取 items。要建立一个 Spider，必须用 scrapy.spider.BaseSpider 创建一个子类，并确定三个强制的属性：

name：爬虫的识别名称，必须是唯一的，在不同的爬虫中必须定义不同的名字。

start_urls：爬取的 URL 列表。爬虫从这里开始抓取数据，所以，第一次下载的数据将会从这些 urls 开始。其他子 URL 将会从这些起始 URL 中继承性生成。

parse()：解析的方法，调用的时候传入从每一个 URL 传回的 Response 对象作为唯一参数，负责解析并匹配抓取的数据(解析为 item)，跟踪更多的 URL。

取：在 Scrapy 里，使用一种叫做 XPath selectors 的机制，它基于 XPath 表达式。我要取出的是四项，URL，title，content，commit，举个例子：

```
for title in sel.xpath('//div[@class = "head clearfix"]/div[@class = "title"]/h1/text()').extract() :
    item['title'] += title
```

取出 title 根据网页模式不同有几种方法，这是其中一种，这里的 xpath()为返回一系列 selectors，每一个 select 表示一个 xpath 参数表达式选择的节点。

前几项没有问题，主要是 commit 存在一个机制，当网页完全读入后网页才会调取 api 将数据填上，那么我们需要从 body 中找到 cmtid，通过另一个链接找到 cmt 的数目，再将其返回函数：

```

cmtIdPattern = re.compile(r'cmt_id\s*=\s*(\d+)')
cmtIdSearch = cmtIdPattern.search(jsStr)
cmtId = cmtIdSearch.group(1)

cmtIdUrl = 'http://coral.qq.com/article/' + cmtId + '/commentnum?callback=_cbSum&source=1&t=' + str(131072)

commentJson = urllib2.urlopen(cmtIdUrl).read()
cmtNumPattern = re.compile(r'"commentnum":(\d+)')
cmtNumSearch = cmtNumPattern.search(commentJson)

```

存：我们用最常用的 json 导出。为满足后续实验要求，一个 document 为一行，都按标准格式存储，如下图所示：

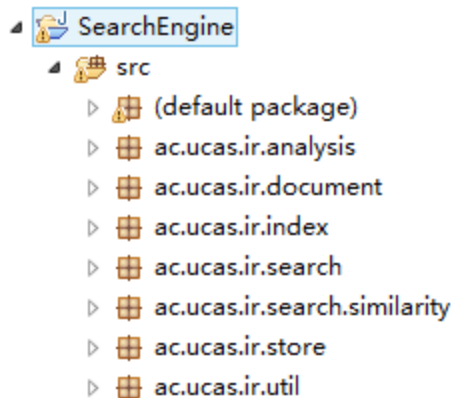
```

{"url": "http://sports.qq.com/a/20141201/044662.htm", "content": "12月1日讯 二双，赢球，紫金军团今日主场通过加时以129-122击败，飞侠王切砍下31分11篮板12助攻，勇"},
{"url": "http://sports.qq.com/a/20141201/044763.htm", "content": "12月1日讯 据《洛杉矶时报》报道，湖人队当家球星表示，他越是愤怒就越能激发他打出好的表现。不过"},
{"url": "http://sports.qq.com/a/20141201/021905.htm", "content": "作者：李淼() 在一连串的失利之后，好胜心极强的不得不面对批评声，在众多讨伐之下，科比开口谈起了"},
{"url": "http://sports.qq.com/a/20140826/054736.htm?tu_type=1", "content": "", "comment": "339", "title": "体坛折翼情侣 朱八臂腿鲁能悍将家暴成性(图)"},
{"url": "http://sports.qq.com/a/20140826/017504.htm?tu_type=1", "content": "", "comment": "1791", "title": "高潜：刘翔观战青奥会跨栏 “千爹”皮带惹眼"},
{"url": "http://sports.qq.com/a/20141201/016948.htm", "content": "12月1日讯 Rant Sports消息，专家布伦丹·帕特尔对球队在首月的表现进行点评，并给出了“D”的评分，在"},
{"url": "http://sports.qq.com/a/20141201/020479.htm", "content": "关键时刻出手36次，仅命中14球。但就算神奇不再，也不会改变。倒下也要像英雄作为一名自小就心仪的转"},
{"url": "http://sports.qq.com/a/20141201/021342.htm", "content": "12月1日讯 在中，每个赛季都会有一些纪录诞生，2014-15赛季，又会有哪些重大纪录发生？我们不妨来"},
{"url": "http://sports.qq.com/a/20141201/021276.htm", "content": "作者：李淼() 本赛季的单场得失分差是8.9分，位列全联盟倒数第三。感觉上遭遇的溃败不少，但仔细看

```

2. 建立索引

建立索引的包结构如下图所示：



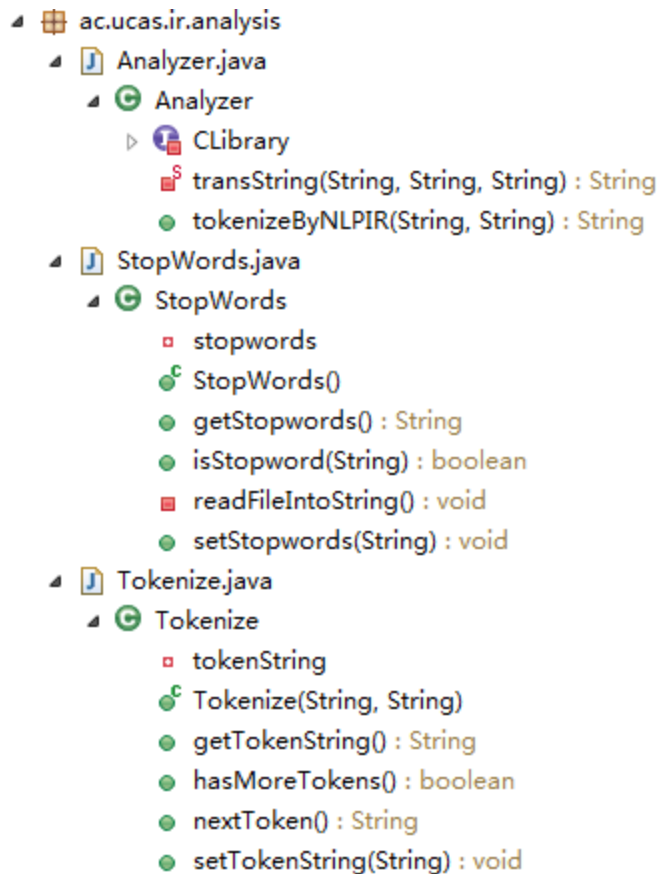
其中 analysis 包主要是进行文档预处理，例如去除停用词，将文档词条化；Document 包定义了两个类 Field 和 Document，其中 Field 是文档的一个域，例如 URL，标题，正文内容，评论数等；Document 类则由文档 ID 和多个 Field 域组成；index 包主要是索引相关的操作，例如倒排索引结点，倒排索引记录，整个倒排索引，以及索引的读写和合并；search 包主要包括 TFIDF 相似度计算，IndexSearcher，Query，排序类和检索结果输出；similarity 包主要是一些相似度计算方法，例如 TFIDF，余弦相似度，BM25 模型等；clusters 实现了层次凝聚式聚类 HAC 和非差别的标签提取，标签是来自于与簇中心最相近的标题。显示 store 包主要是文件的输入输出，例如网页的读入，将索引保存到文件中或者从文件中读入索引。

Util 包主要是其它包需要用到的一些功能类。

每个包的具体介绍如下：

1) analysis

analysis 包结构如下：



➤ Analyzer

Analyzer 是一个分析器，将输入字符串词条化，其中 CLibrary 是一个私有接口，通过这个接口调用 ICTCLAS 分词工具；transString()是对字符串进行各种编码格式间转换的私有成员方法；tokenizeByNLPIR()是词条化的公有接口，该成员方法通过调用以上两个私有成员方法完成字符串的词条化。

➤ StopWords

停用词类，有一个私有成员变量 stopwords，用来保存停用词；StopWords()是无参构造函数，实例化一个停用词类；以 get 和 set 开头的是成员变量的获取器和设置器，其它类中类似，不再详述；isStopword()用来判断一个词项是否为停用词；readFileIntoString()是一个私有方法，主要完成从文件中读入停用词的工作。

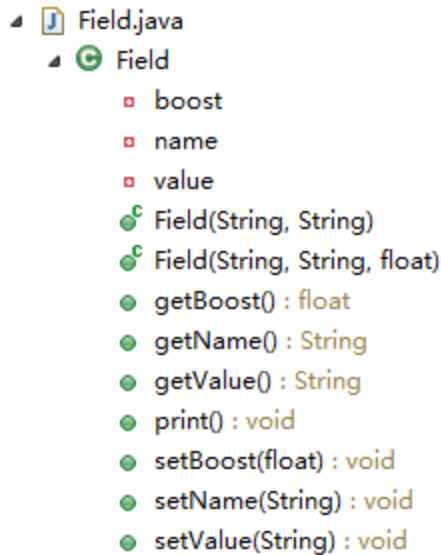
➤ Tokenize

词条化类，通过调用 Analyzer 分析器完成词条化工作。tokenString 是一个私有成员变量，用来保存字符串词条化后的词条；hasMoreTokens()用来判断是否还有未处理的词条，nextToken()用来获取下一个词条。

2) document

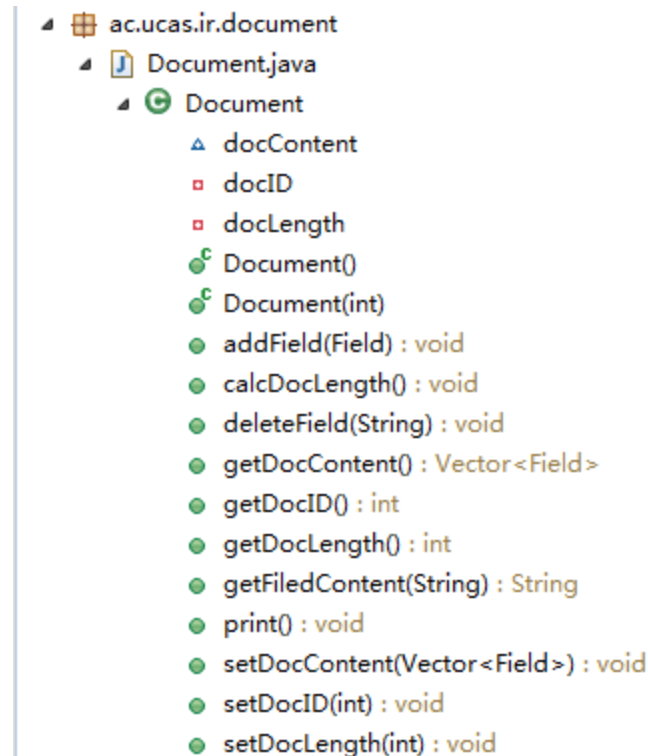
document 包包括 Field 和 Document 两个类，两个类的说明如下：

➤ Field



Field 包括 name, value 和 boost 三个成员变量，分别表示名称，值和权重，权重用来衡量不同域的重要程度，例如标题，正文等；print()是一个输出函数，将域的信息输出到控制台。其它类的 print()函数类似，以后不再赘述。

➤ Document

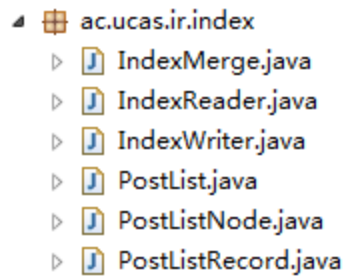


Document 类包括 docContent, docID 和 docLength 三个私有成员变量，其中 docID 是文档标识，每篇文档唯一；docLength 值文档的长度；docContent 是文档包含的域，是 Vector<Field> 类型；addField()和 deleteField()分别用来添加和删除域；calcDocLength()成员方法用来计算文

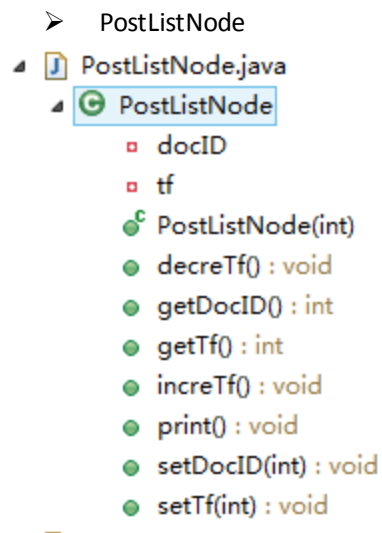
档长度，通过累加各个域的长度实现。

3) index

index 包结构如下：



下面分别介绍各个类的作用及实现：



PostListNode 类是倒排索引的一个结点，有 docID 和 tf 两个私有成员变量，分别表示文档 ID 和词项频度；原来还有词项位置信息列表，但由于时间关系没有实现短语查询，为了提高索引构建和查询的效率，删除了没有充分利用的位置信息。increTf() decreTf() 分别用来增加和减少频度。

```

➤ PostListRecord
└─ PostListRecord.java
   └─ PostListRecord
      ├── df
      ├── postListRecord
      ├── term
      ├── PostListRecord(String)
      ├── addPostListNode(PostListNode) : void
      ├── containDoc(int) : boolean
      ├── delcreTFByDocID(int) : void
      ├── delDocument(int) : void
      ├── getDf() : int
      ├── getDocIDs() : List<Integer>
      ├── getPostListRecord() : List<PostListNode>
      ├── getTerm() : String
      ├── getTfByDocID(int) : int
      ├── IncreTFByDocID(int) : void
      ├── merge(PostListRecord) : void
      ├── print() : void
      ├── setDf(int) : void
      ├── setPostListRecord(List<PostListNode>) : void
      └── setTerm(String) : void

```

PostListRecord 类代表倒排链表的一个记录，有 term, df 和 postListRecord 三个私有成员变量，分别表示词项，文档频率和倒排节点链表。addPostListNode() 和 delDocument() 分别用来添加和删除倒排节点；decreTFByDocID() 和 increTFByDocID() 分别用来减少和增加词项在某一文档的频率；containsDoc() 用来判断该词项是否在某个文档中出现；getDocIDs() 用来获取包含该词项的文档 ID 列表；getTfByDocID() 用来获取词项在某个文档中的频率；merge() 用来将该倒排记录和另一倒排记录合并。

```

➤ PostList
└─ PostList.java
   └─ PostList
      └─ index
         ● PostList()
         ● addPostListRecord(PostListRecord) : void
         ● clear() : void
         ● containTerm(String) : boolean
         ● decreTermFrequencyByDocID(String, int) : void
         ● delDocument(int) : void
         ● delPostListRecord(String) : void
         ● getDfByTerm(String) : int
         ● getDocIDListByTerm(String) : List<Integer>
         ● getIndex() : HashMap<String, PostListRecord>
         ● getTermRecord(String) : PostListRecord
         ● getTerms() : Set<String>
         ● getTermsByDocID(int) : List<String>
         ● getTermSize() : int
         ● getTermTfByDocID(String, int) : int
         ● increTermFrequencyByDocID(String, int) : void
         ● print() : void

```

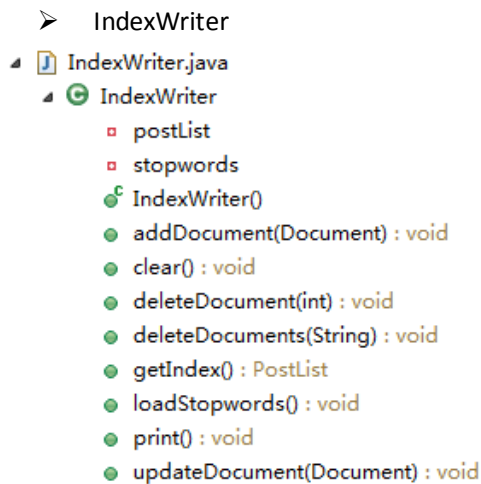
PostList 类代表一个完整的倒排链表，只有 index 一个成员变量，它的类型是 HashMap<String, PostListRecord>，是词项到倒排记录的 hash 映射。addPostListRecord() 和 delPostListRecord() 用来添加或者删除一个倒排记录；increTermFrequencyByDocID() decreTermFrequencyByDocID() 用来增加或者减少一个词项在某个文档中的频度；containTerm() 用来判断索引是否包括某个词项；delDocument() 用来从索引中删除某个文档；getDfByTerm() 用来获取某个词项的文档频率；getDocIDListByTerm() 用来返回包括该词项的文档 ID 列表；getIndex() 用来获取整个索引；getTermRecord() 用来获取某个词项的倒排记录；getTerms() 用来返回所有特征词项；getTermsByDocID() 用来返回某个文档的特征词项；getTermTFByDocID() 用来返回某个词项在某个文档中的出现频率；clear() 用来清空该倒排链表。

```

➤ IndexMerge IndexReader
└─ IndexMerge.java
   └─ IndexMerge
      ● indexMerge(PostList, PostList) : PostList
└─ IndexReader.java
   └─ IndexReader
      ● loadIndexFromDirectory(String) : PostList

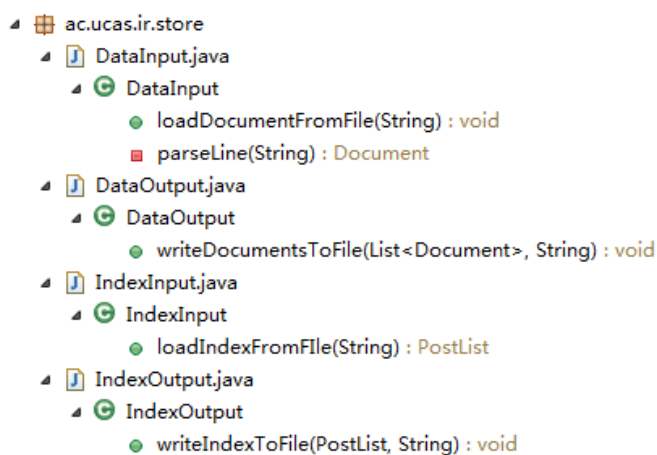
```

IndexMerge 类实现两个倒排链表的合并，IndexReader 类实现从某个文件夹中读入所有的索引文件，合并出一个完整的倒排链表。



`IndexWriter` 是构建索引的主要外部接口，包括倒排链表和停用词两个私有成员变量；`addDocument()` `deleteDocument()` `updateDocument()` 分别用来向索引中添加，删除和更新一个文档；`deleteDocuments()` 用来删除包括某个特征此项的所有文档；`loadStopwords()` 用来加载停用词。

4) store



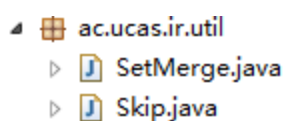
`store` 包中主要定义了一些和 IO 相关的类。

`DataInput` 类通过调用 `IndexOutput` 类实现读入文档，建立索引并将索引保存到文件中；`loadDocumentFromFile()` 通过从文件中读入文档建立索引，`parseLine()` 通过解析文件中读入的行实例化一个 `Document` 类。

`DataOutput` 类用来输出文档到文件。

`IndexInput` 类用来从文件中读入索引。



5) util



`util` 包包括 `SetMerge` 和 `Skip` 两个类，其中 `SetMerge` 类用来完成两个集合的合并操作；`Skip` 用来处理某些无用字符，例如空格，`tab` 和换行。

3. 检索

1) search

- ▶  Dataout.java
- ▶  Doctermstfidf.java
- ▶  IndexSearcher.java
- ▶  Query.java
- ▶  Sort.java

Java 类功能：

Dataout 将 search 的搜索结果传给前端接口。

Doctermstfidf 记录 docid 和文档与 query 的相似度。

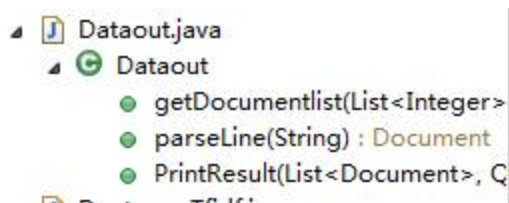
IndexSearcher 为 search 的主体，里面实现了 query 检索。

Query 主要实现对前端给出的 query 进行分析。

Sort 为对搜索结果实现排序。

以下是各个类的详细介绍：

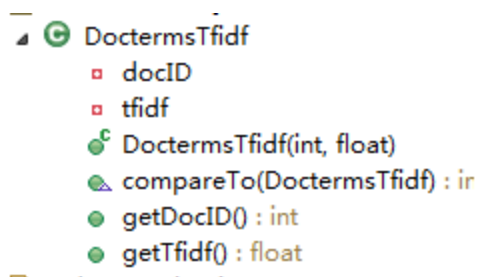
➤ Dataout



getDocumentlist 实现了根据 docidlist 返回 documentslist 的功能，方便将结果返回前台。

parseLine 实现了对返回结果的格式化。PrintResult 将结果反馈给前台。

➤ Doctermstfidf



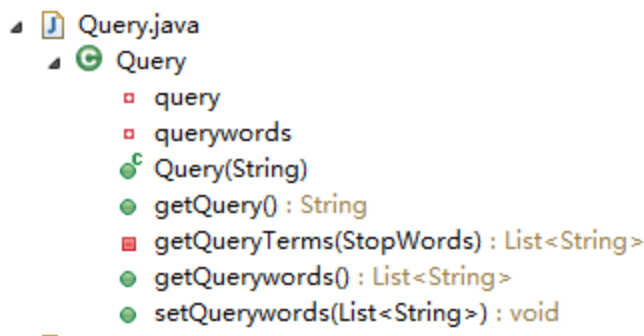
这个类有两个成员 docID 和 tfidf，分别是文档 id 和与 query 的相似度权重。compareTo 实现了 implements Comparable<Doctermstfidf> 的接口，方便之后的排序。其他的是构造函数和获取成员函数。

➤ IndexSearcher



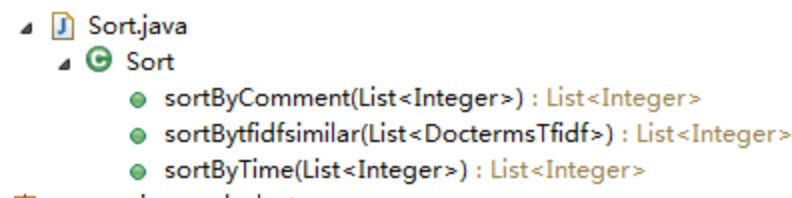
CaluAlldocIDlist 实现了计算所有 docidlist 的中 id 相似度权重, 并和 id 封装成了 DoctermsTfidf。getdocIDlistbyquery 实现了输入 query 和索引输出 docidlist。getDocIDListByTerm 实现输入一个词项和索引输出该词项所在的所有文档。getQueryEvaluateOfDocument 实现输入一个 query, 文档 id, index 和文档总数, 输出该 id 文档和 query 的相似度权重。getTopkDocuments 实现输入 docidlist 和 K 输出需要的 docidlist, 就是对排序后的结果进行进一步的筛选。Search 为计算出所有的 docid 并排序。UnionDocIDList 为对不同的 term 搜出的 docidlist 进行 union, 采用并的方式。

➤ Query



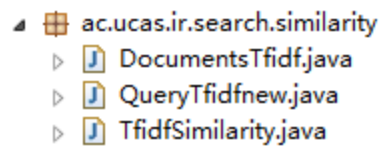
getQueryTerms 根据 stopWords 和成员 query 输出对 query 的分析结果。其他的为构造函数或设置获取函数。

➤ Sort



主要实现了相似度的排序, 热度和 time 封装好后, 排序交给前端来处理。

2) similarity



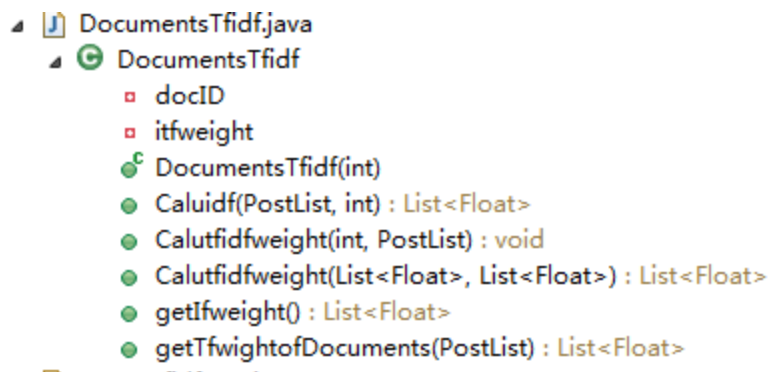
DocumentsTfidf 实现了文档的 tfidf 的空间向量计算为之后的聚类提供基础。

QueryTfidfnew 实现了 query 和文档的模型相似度权重计算。

TfidfSimilarity 实现了两篇文档的相似度计算。

以下是各个类的详细介绍：

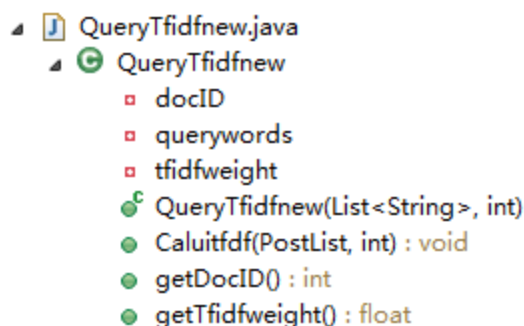
➤ DocumentsTfidf



成员：docid 和 itfweight（实际为 itfidfweight）

Caluidf 输入索引和文档总数计算出 idf；Calutfidfweight 实现了输入索引和文档总数计算出 tfidf；而 Calutfidfweight(List<Float>,List<Float>)实现了 tf vector 和 idf vector 的输入给出 tfidfweight vector 的输出；gettfWeight 为获取函数；getTfwightofDocuments 实现了文档的 tf 计算。

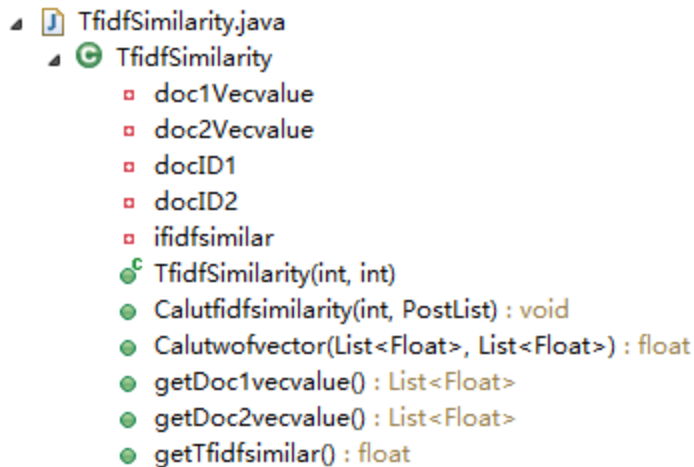
➤ QueryTfidfnew



成员：docid, querywords, tfidfweight

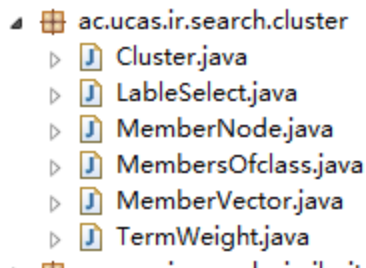
Caluitfdf 输入索引和文档总数计算 tfidfweight，其他为类获取和设置函数。

➤ TfidfSimilarity



成员：doc1 的 tfidf 空间向量，doc2 的 tfidf 空间向量，docID1 文档 1id，docID2 文档 2id，ifidfsimilar 为两篇文档的相似度。
Calutfidfsimilar 输入索引和文档总数，计算两篇文档的相似度；caluTwofvector 实现两个空间向量的向量积。

3) cluster



类的功能：

Cluster 采用 HAC 实现对检索结果的聚类。

LabSelect 采用非差别式簇标签生成方法进行标签抽取。

MemberNode 为簇中的成员信息。

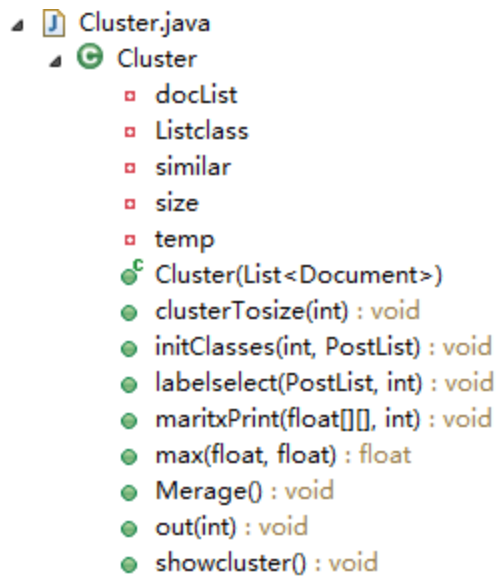
MembersOfclass 为记录簇的信息，和簇合并的操作。

MemberVector 为记录 MemberNode 的 tfidf 的空间向量。

TermWeight 记录标签提取时的 term 和权重。

类的详述：

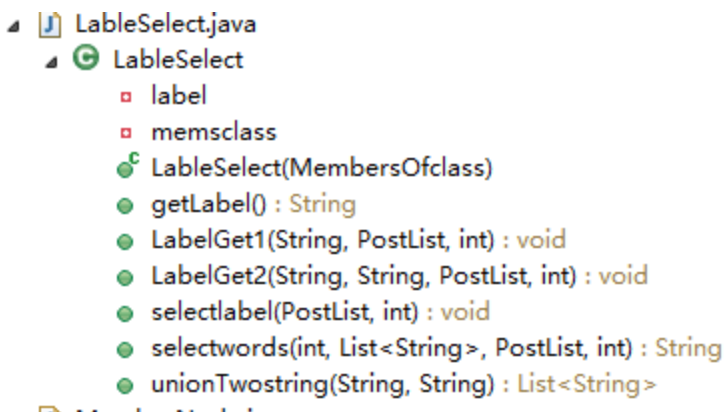
➤ Cluster



成员: doclist 为 documents 的 list, Listclass 为簇 list, similar[][] 为记录簇的距离信息, temp[][] 为簇合并时的辅助矩阵, size 为簇的总数。

ClusterTosize 实现聚成几簇; initClasses 为簇的初始化构造; Labelselect 为对所有簇进行标签提取; max 为实现大小比较; Merage 实现一次簇合并, 采用的是 HAC 中找出相似度最大的两簇合并, 同时更新簇和 similar[][]; out 实现簇结果整合; showcluster 实现将聚类结果返回给前端。

➤ LableSelect



成员: label 为提取的标签, memclass 为簇。

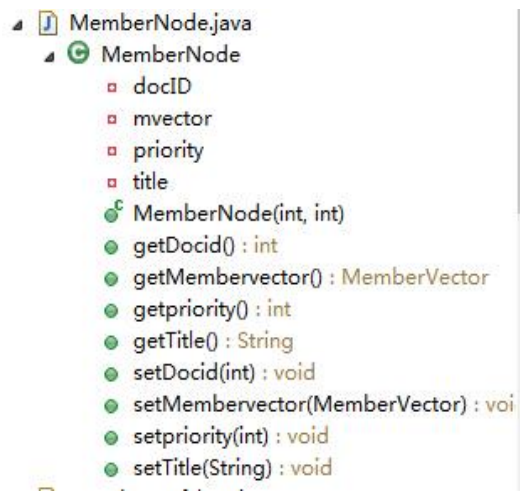
标签提取的算法:

采用的是非差别式簇标签生成方法, 找出簇中与簇中心最相近的两个文档, 使用他们的标题, 当然如果一个簇只有一个文档, 那只返回一个文档的标题, 对于这两种情况有两种获取的标签的方法: 对于一个文档的, 对标题切词并计算所有切词的 tfidf 值, 找出最大的 3 个切词当成标签; 对于有两个文档的, 先将两篇文档的标题取共同的切词, 如果切词个数大于 3, 计算 tfidf 排序找出最大的 3 个当成标签, 如果小于 4 个又不为空, 直接返回; 如果为空的话, 取与簇中最相似的那篇, 对它的标题采用一个文档的标签提取策略。

LabelGet1 为一个文档时的标签提取; labelGet2 为有两个文档时的标签提取; selectlabel 为计算出簇的标签。Selectwords 为提取几个 tfidf 最大权重分词。UnionTwoString 合并两个文

档的标题的函数。

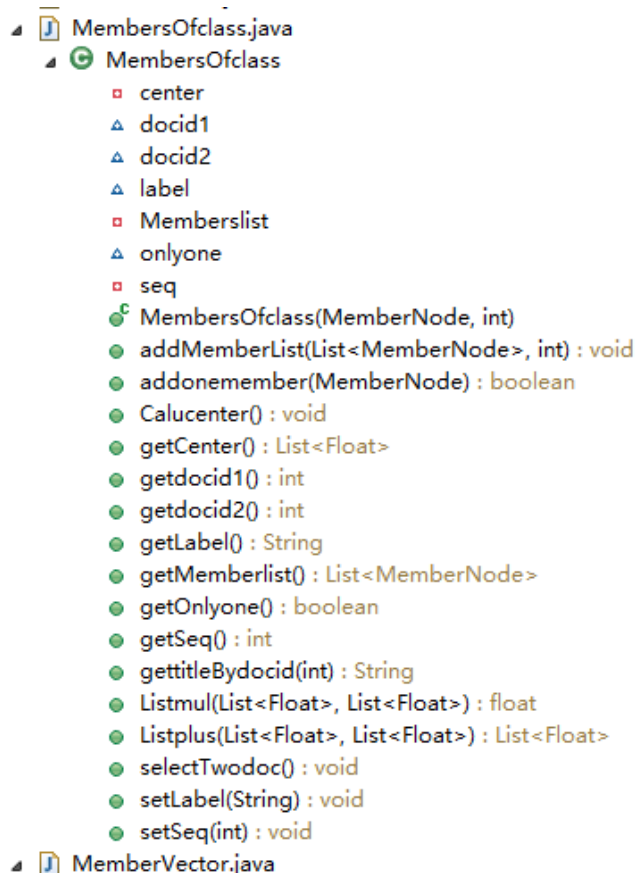
➤ MemberNode



成员 docid 为文档 id，mvector 为文档的空间向量，priority 为文档的优先级，方便在簇也能区分文档的优先级。Title 为文档的标题。

函数都是 java 类中常见的构造函数，获取和设置函数。

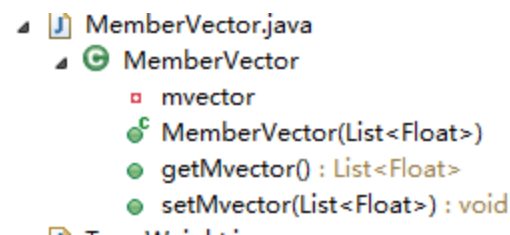
➤ MembersOfClass



成员：center 为簇的中心向量，docid1 为与 center 最相近的文档 id，docid2 为与 center 次最相近的文档 id，label 为簇的标签，onlyone 为簇中是否只有一个文档的标记，seq 为该簇在 similar[][] 的下标。

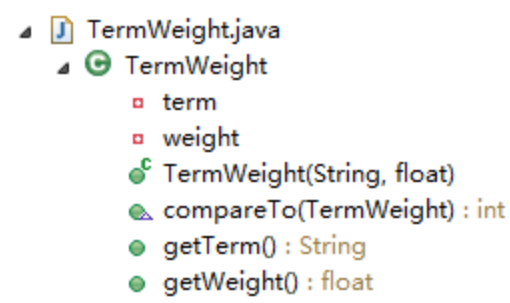
addMemberList 实现簇的合并；addmember 实现该簇增加一个成员，用于簇初始化；calucenter 计算簇的中心向量；getTitleBydocid 实现根据 docid 给出标题；Listmul 计算两向量的向量积；listplus 计算两向量相加；selectTwodoc 找出与 center 最相近的两个文档。其他的为常见函数。

➤ MemberVector



成员：mvector 为空间向量
主要是用来记录簇中文档的空间向量。

➤ TermWeight

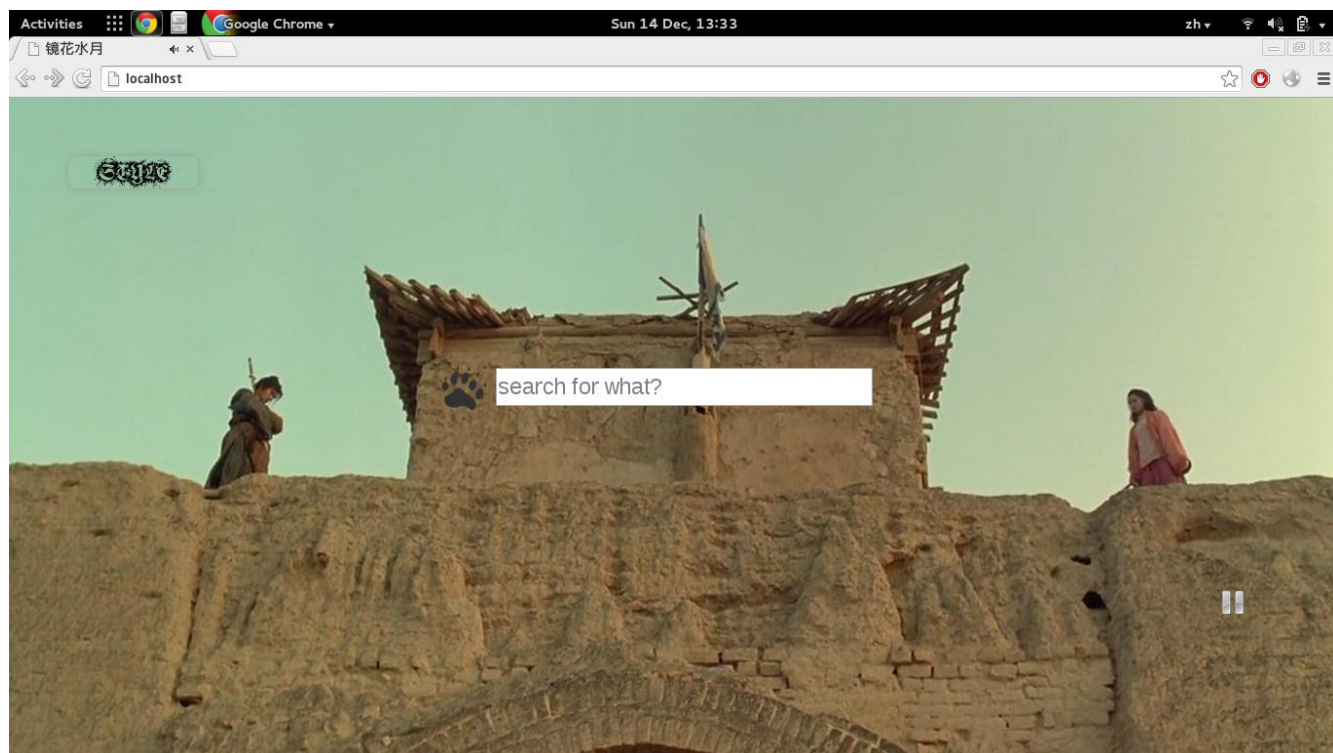


成员：term 为分词，weight 为 term 的权重。
CompareTO 为实现了该类的比较接口，有利于之后的排序。

4. 前端

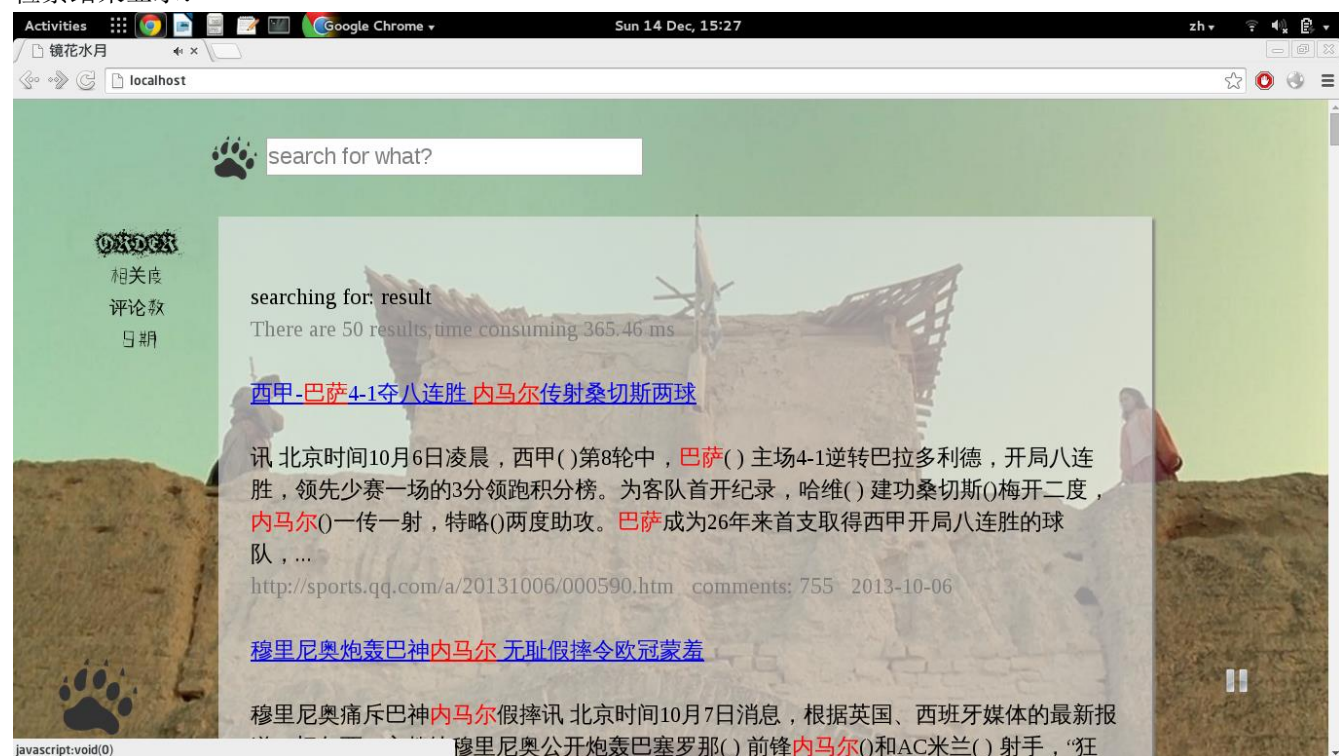
采用 LAMP 架构，本次实验没有用到数据库，所以即 linux+apache+php，网页部分即 html+javascript+css

检索首页：

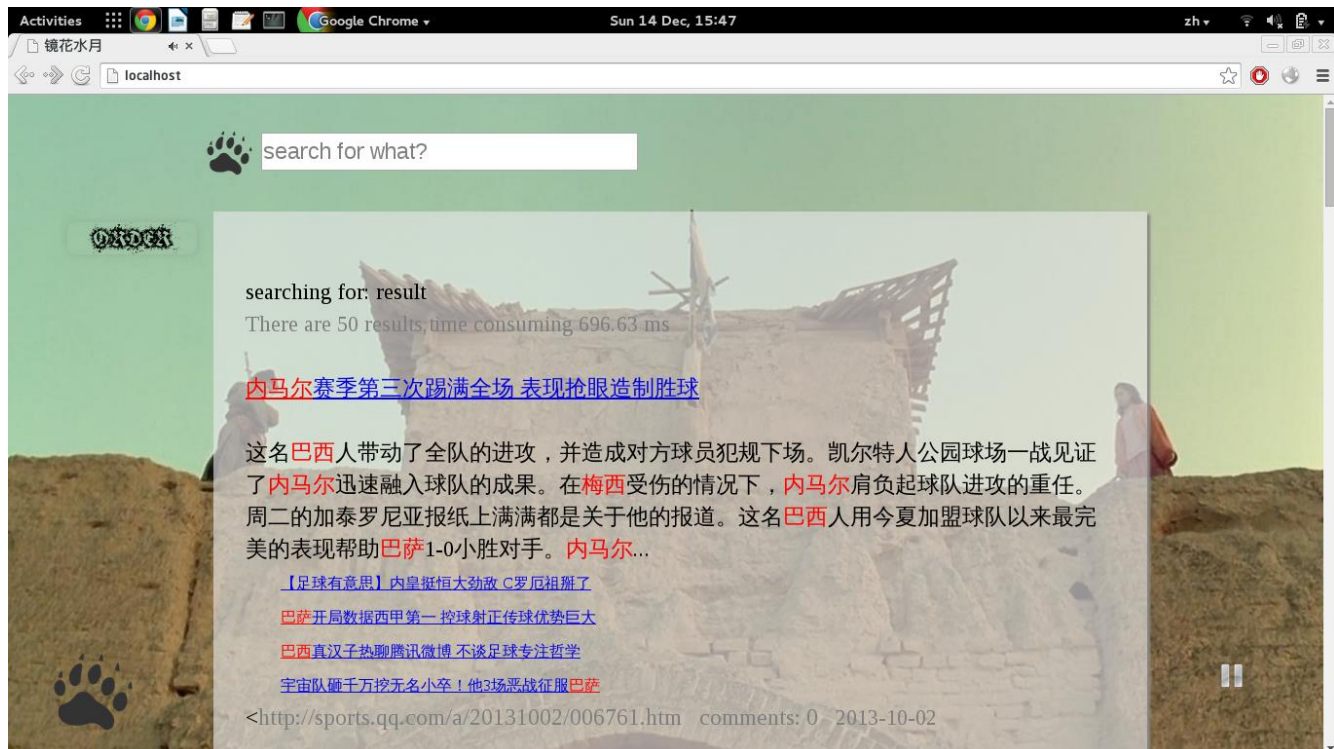


其中用到了各种 html 和 js 元素。有音乐和各种主题供选择。

检索结果显示：



聚类结果显示:

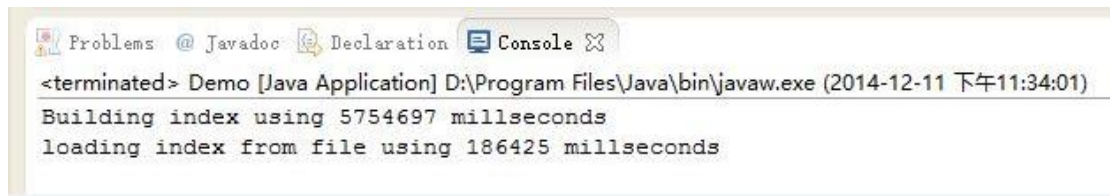


界面上提供了二次搜索, 和按实验要求的按相关度, 评论数, 日期排序。同时把切词后的关键字高亮显示, 这里高亮用的是 php 的替换: `str_replace($result[$y+2], ''.$result[$y+2].', $doc[$order[$x]]);` 即把关键字换成 `关键字`。

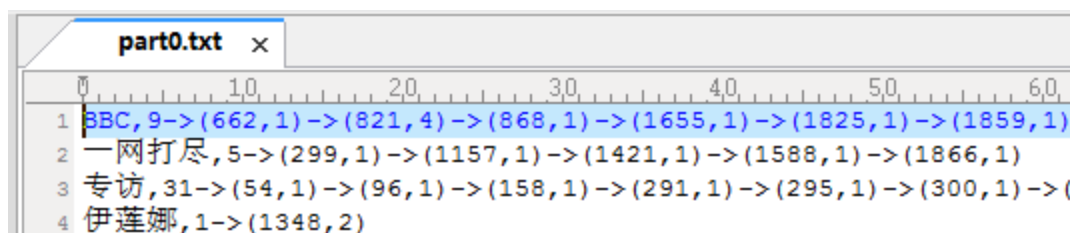
这里跟后端的连接采用较简单的方法: php 的 `exec` 函数执行外部指令, 外部指令的 shell 结果会存储在 `exec` 的返回的一个数组里, 即得到前端的输入。

四、 运行测试

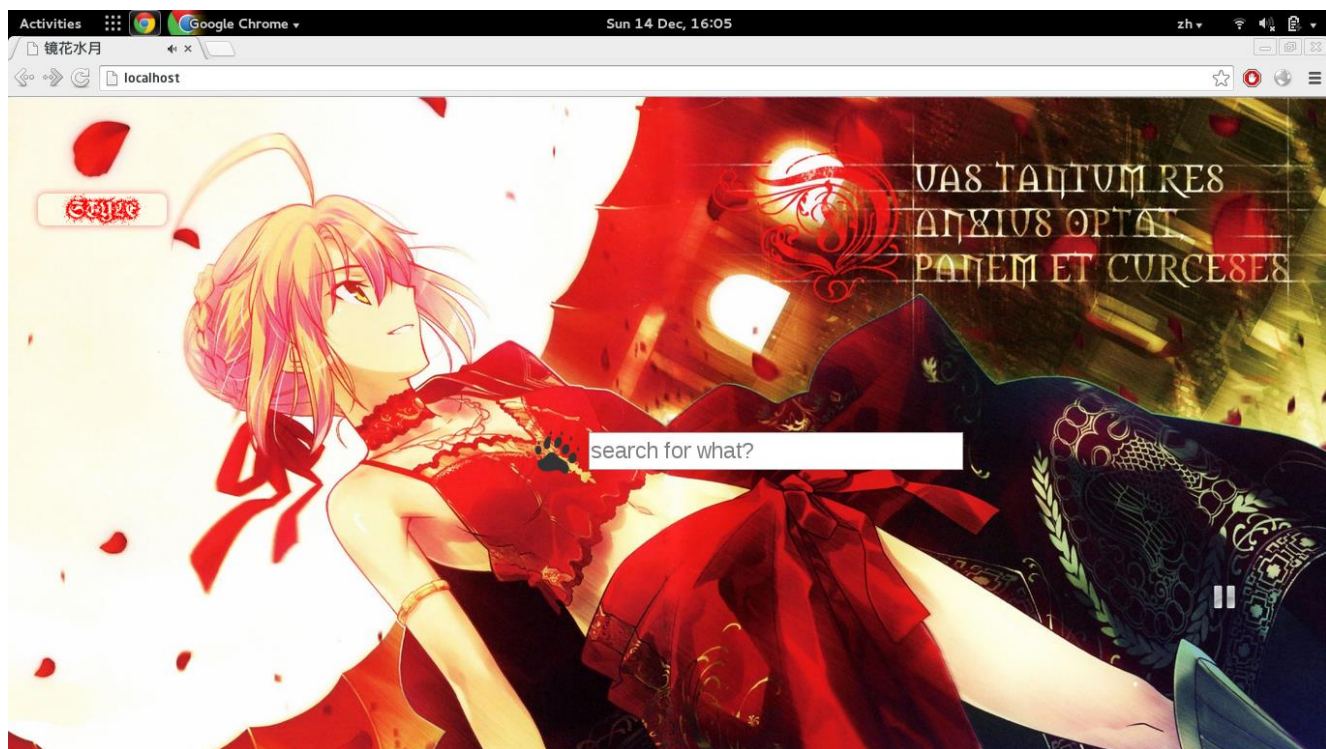
索引: 对 10 万篇文档建立索引并写入文件中保存大概用了 1 小时 40 分钟, 通过调整索引分块的参数可以提升性能, 但是提升不明显。从文件中读入索引到内存大概用了 3 分钟。



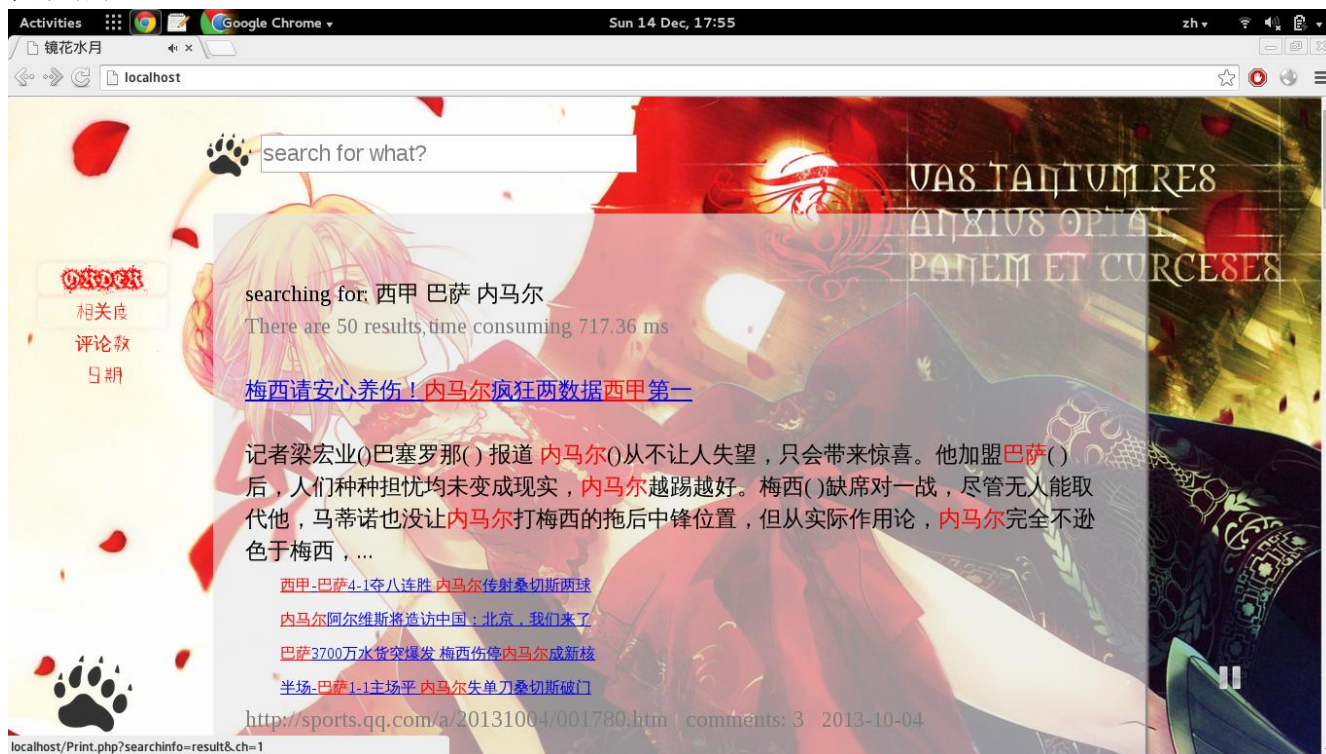
索引截图:



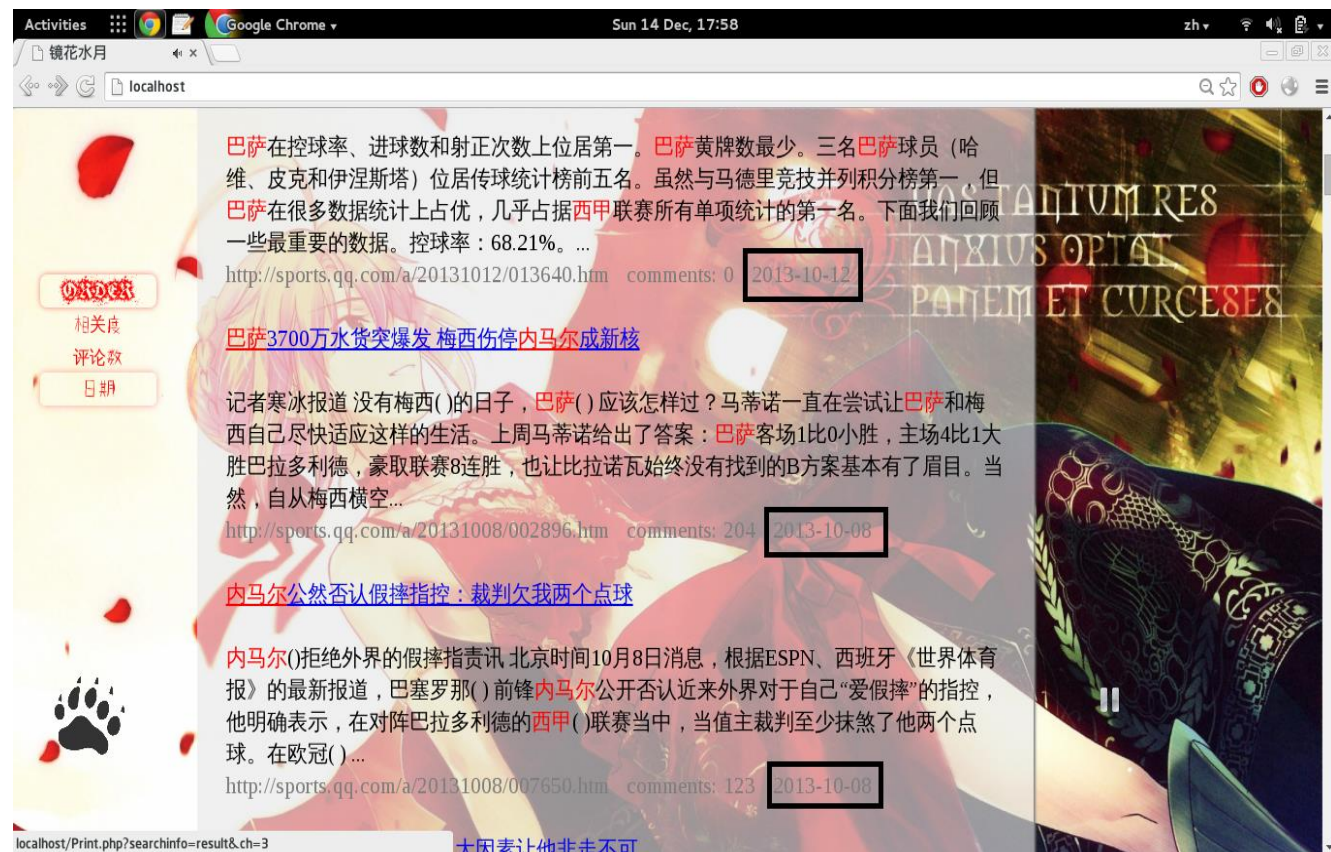
检索页面:



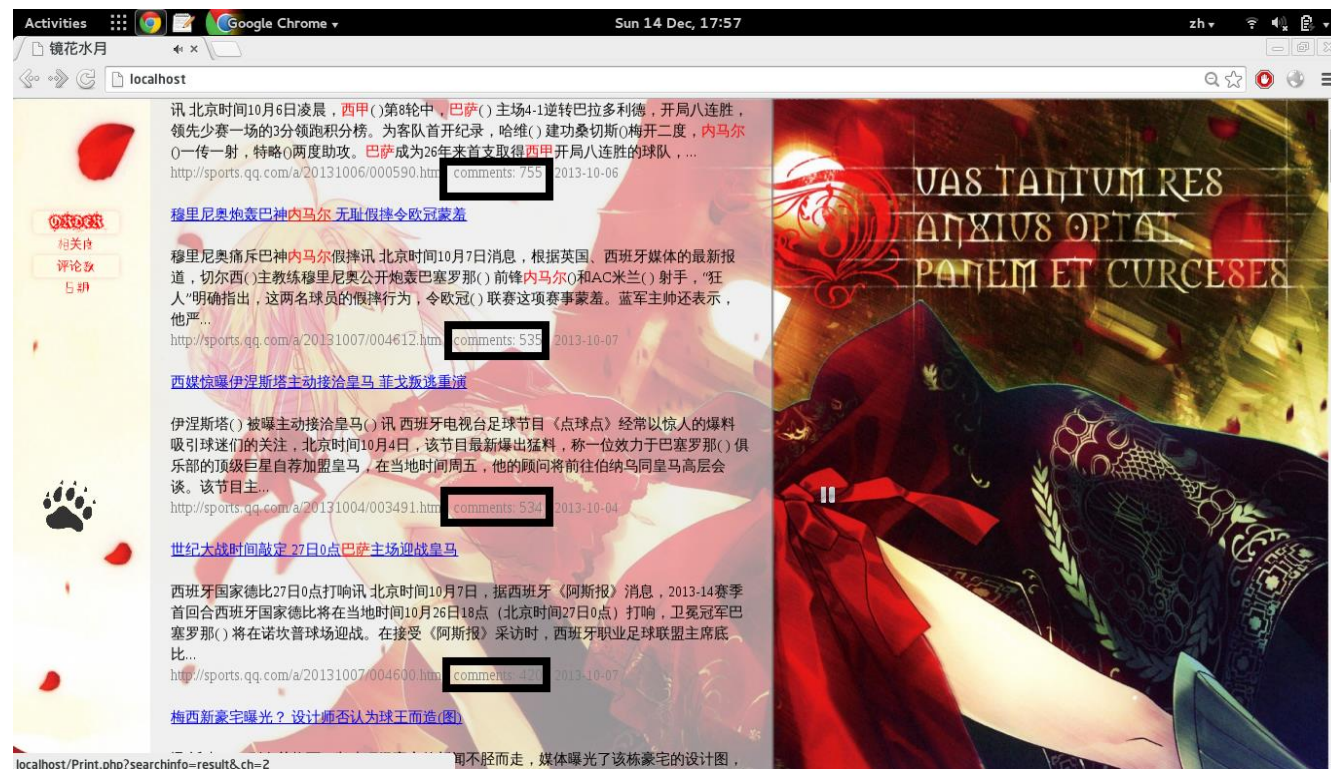
检索结果:



按日期排序:



按热度(评论数)排序:



五、 创新点

这个实验基本上按照书上所讲的流程开发了一个搜索引擎，在算法和技术上并没有比较大的创新。但是我们只是分词部分用了开源软件 ICTCLAS，其余部分，包括爬虫，建立索引，检索和页面展示都是我们自己编写的代码。虽然性能上和成熟的开源软件相比有较大差距，但我们认为通过这个实验，每个人都学到了很多东西：增进了对书本知识的理解，更加深入地明白一个搜索引擎的基本原理和实现，学会了和他人交流协作。

六、 实验分工

余骋远：爬虫，实验报告 3.1，个人总结

罗远浩：建立索引，实验报告其余部分

刘吉： 检索,聚类，实验报告 3.3，个人总结

薛杰瑜：前端，实验报告 3.4，个人总结，PPT

七、 总结

- 余骋远：我主要负责爬虫这一部分，从学习 python 开始到最后完成，说简单也不简单。通过这次实验我学会了 python 语言，正则表达式，httpfox 插件的使用等，对以后的科研工作大有裨益。
- 罗远浩：通过这个实验，我加深了对理论知识的理解，对书上和老师所讲的内容有了更进一步的认识；另一方面，通过编写代码，进一步提高了自己的编程能力，懂得了从多个方面去解决编程时遇到的问题；更重要的是，通过小组合作，提高了我的人际交流，团队协作能力，更加深刻的认识到，面对一个大的项目，团队合作的有效性和必要性。
- 薛杰瑜：借此机会学习了 html javascript css php 等网站架构的相关知识，显示的界面，搜索结果页面比较友好。
- 刘吉：这次的搜索引擎的实验让我成长了很大认识到了自己的很多不足之处，在和队友的合作之间，感觉到了自己很多的不足，但正是这样也提供了我成长的契机。这次实验我负责搜索和聚类，在做搜索的时候需要考虑评分模型，但感觉这些都还需要改进；聚类的话使用了 HAC 效果不错，也很高效，但是标签的抽取效果不甚理想，虽然加入了一些自己的想法，很有带改进。项目的实现全赖队员的努力付出，特别是队长罗远浩的索引给后续工作提供了很大便利。

罗远浩 2014E8013261184 (组长)

余骋远 201428013329025

薛杰瑜 2014E8013261174

刘吉 2014E8015061088

2014-12-14