

基于 Lucene 的全文搜索排序算法的研究与改进

阮曙芬

(中国地质大学 江城学院, 湖北 武汉 430020)

摘要: 在文本检索过程中, 排序算法一定程度上影响到搜索引擎的质量。论文首先分析了 Lucene 组织结构, 包括建立索引, 检索索引文件以及结果集排序的工作过程和原理, 着重剖析了 Lucene 基于向量模型的排序算法, 并在原有排序算法基础上, 采用基于关键词加权方式改进了全文检索的排序结果。实验结果证明, 改进后的排序算法提高了系统的结果精确度, 满足了项目的实际需求。

关键词: Lucene; 向量空间模型; TF-IDF; 排序算法

中图分类号: TP311.13

文献标识码: A

文章编号: 2095-414X(2013)06-0084-04

1 Lucene 搜索技术

Lucene^[1]是 Apache 软件基金会 Jakarta 项目组的一个成员项目, 它是一个高性能、可伸缩的信息检索库, 但仅是一个用 Java 语言编写的全文信息检索的开源工具包 Lucene 具有开放源码、索引速度快、支持动态增删索引、内存占用小等优点, 可以方便嵌入到各种实际应用中, 实现全文信息检索功能。

Lucene 作为一个优秀的全文搜索框架, 同时也是一个面向对象设计(OOP)的成功实例"首先检索库的功能模块原子化程度和内聚力较高, 在重新定义实现用户功能时, 只需修改特定功能模块, 而无需修改其他功能模块;其次, 它定义了一个与平台无关的索引文件格式;第三, 系统的核心功能和重要组成部分设计定义为抽象类, 用户可以通过继承等方式重新定义, 从而实现自己的功能目标, 最后, 通过提供灵活的 API 接口, 可以让新用户很快上手。

Lucene 搜索过程(如图1中虚线部分所示)首先用户输入查询语句, 经过词法分词器和语言分词器分析处理得到一系列词(Term), 然后通过词法分析得到一棵查询树, 通过索引存储将索引文件读入内存, 利用查询树搜索索引, 得到每个词的文档链表, 对文档链表进行操作, 得到结果文档集, 按查询的相关性将搜索结果文档集进行排序, 最后将排序后的查询结果返还给用户。

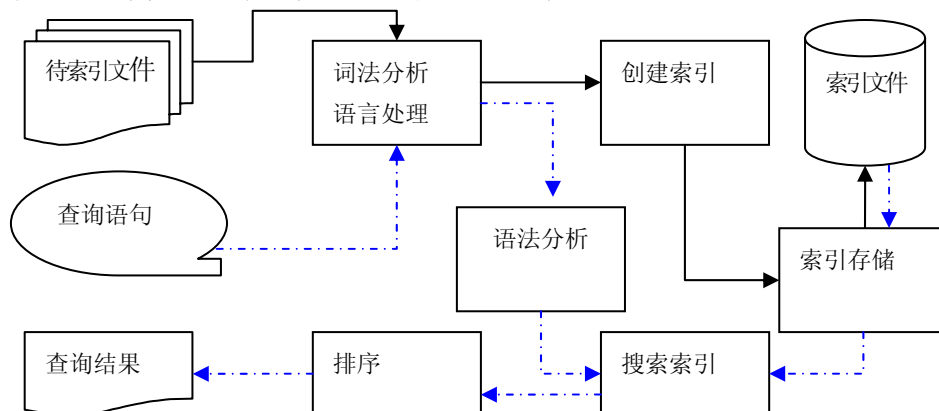


图1 检索与索引过程

2 Lucene 包组织结构

Lucene 将所有源代码分为7个模块, 各个模块的功能如表1所示:

表 1 Lucene 各个模块的功能

包名	功能描述
org.apache.Lucene.analysis	语言分析器，主要用于的切词，支持中文主要是扩展此类
org.apache.Lucene.document	索引存储时的文档结构管理，类似于关系型数据库的表结构
org.apache.Lucene.index	索引管理，包括索引建立、删除等
org.apache.Lucene.queryParser	查询分析器，实现查询关键词间的运算，如与、或、非等
org.apache.Lucene.search	检索管理，根据查询条件，检索得到结果
org.apache.Lucene.store	数据存储管理，主要包括一些底层的 I/O 操作
org.apache.Lucene.util	一些公用类

从面向对象(OOP)的角度来观察，Lucene 采用了一条最基本的程序设计准则即通过引入额外的抽象层来降低系统的耦合性。首先，引入数据存储管理功能包完成所有对文件存储管理操作的封装，然后，引入索引功能包完成索引部分操作的封装，完成对索引核心的抽象；在索引核心的基础上设计定义对外的接口语言分析器功能包和检索功能包，在高度的面向对象程序设计的理论支撑下，使得 Lucene 的整体框架便于理解，易于开发扩展。

3 Lucene 排序的原理分析

在搜索引擎的研究中，一个核心问题就是排序算法的确定，较成熟的方法包括向量空间概率模型和统计语言模型等，Lucene 的评分机制采用向量空间模型，那么，我们把文档或查询看做一系列的词(Term)，每个词都有一个权重(Term Weight)，不同的词根据自己再文档或查询中的权重来影响文档相关性的评分计算。我们将此文档或查询中的所有词的权重看做一个向量：

Document/ Query = {Term1, Term2, ..., Term N}

Document/ Query Vector= {Weight1, Weight2, ..., Weight N}

我们把所有搜索出的文档向量及查询向量放到一个 N 维空间中，每个词是一维，如果两个向量间的夹角越小，相关性就越大，所以计算夹角的余弦值作为相关性的评分，夹角越小，余弦值越大，评分越高，相关性越大^[1]。（如图 2）

Lucene 的实际计算公式如公式（1）^[1]：

$$\text{Score}=\text{coord}(q, d)*\text{queryNorm}(q)*\sum(\text{tf}(\text{tind})*\text{idf}(\text{t})^2*\text{t.getBoost}()*\text{norm}(\text{t}, \text{d})) \tag{1}$$

其中，

- t，词项（term）；也被称为词元，是粒度最小的内容对象。
- tf，词项频数（term frequency）；即词项在文档中出现的频数（次数）。
- df，文档频数（document frequency）；即整个数据集中出现过该指定词项的文档的频数（数量）。idf，逆文档频数（invert document frequency）；即与文档频数成反比的值。
- coord(q, d)：一次搜索可能包含多个搜索词，而一篇文档中也同时可能包含多个搜索词，此项则表示，当一篇文档中包含的搜索词数量越多，此文档得分越高。
- queryNorm(q)：计算每个查询条目的方差和，该值不影响排序，仅使得不同的 query 之间的分数可以比较，每个查询项权重的平分方和(sumOfSquaredWeights)由 Weight 类完成。

假设：查询向量为 $V(q) = \langle w(t1, q), w(t2, q), \dots, w(tn, q) \rangle$

文档向量为 $V(d) = \langle w(t1, d), w(t2, d), \dots, w(tn, d) \rangle$

向量空间维数位 n，为查询语句和文档的并集长度，当某个 Term 不在查询语句中 $w(tq,)$ 为零，当某个 Term 不在文档中是 $w(t, d,)$ 为零，w 代表 Weight，计算公式为：

$$w=\text{tf}*\text{idf}$$
$$V(q)*V(d)=w(t1, q)*w(t1, d)+\cdots w(tn, q)*w(tn, d)$$
$$=\text{tf}((t1, q)*\text{idf}((t1, q)*\text{tf}((t1, d)*\text{idf}((t1, d)+\cdots\text{tf}((tm, q)*\text{idf}((tm, q)*\text{tf}((tm, d)*\text{idf}((tm, d)$$

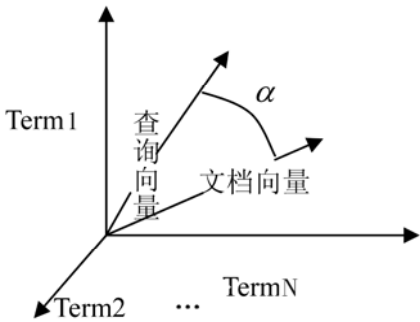


图 2 向量相关性

在查询一般不输入同样的词, 因此假设 $tf(t, q)=1$ 。idf 是指 Term 在多少文档中出现次数, 因此 $idf(t, q)$ 和 $idf(t, d)$ 值相等, 当索引中的文档总数足够大时, 查询语句这篇小文档可以忽略, 因此可以假设 $idf(t, q)=idf(t, d)=idf(t)$, 因此:

$$V(q) \cdot V(d) = idf(t_1) \cdot tf(t_1, d) \cdot idf(t_2) + \dots + idf(t_m) \cdot tf(t_m, d) \cdot idf(t_m) \quad (2)$$

将公式 (2) 带入 (1) 得

$$\cos(q, d) = \frac{\sum tf(t, d) \cdot idf(t)^2}{\sqrt{\sum w(t, q)^2} \sqrt{\sum w(t, d)^2}} \quad (3)$$

$$|V(q)| = \sqrt{\sum w(t, q)^2} = \sqrt{\sum (tf(t, q) \cdot idf(t, q))^2} = \sqrt{\sum (idf(t, q))^2} \quad (4)$$

$|V(d)| = \sqrt{\sum w(t, d)^2}$, 从 Term 的定义来看, Term 是包含域信息的, 就是说每一个 Term 只能在文档中的一个域中出现, 所以:

$$|V(d)| = \sqrt{\sum w(t, d)^2} = \sqrt{\sum 1} = \sqrt{m} \quad (5)$$

将公式 (4), (5) 带入 (3) 得最终相似度计算公式:

$$\cos(q, d) = \frac{1}{\sqrt{\sum idf(t)^2}} * \sum tf(t, d) \cdot idf(t)^2 * \frac{1}{\sqrt{m}}$$

以上文档相似度评分相关的代码都位于 org.apache.Lucene.search.similarities 包下, TFIDFSimilarity 类中, 该类主要提供了以下的功能^[3]:

```
public abstract float coord(int overlap, int maxOverlap)
public abstract float queryNorm(float sumOfSquaredWeights)
public abstract float tf(float freq) 用于为计算词项频数和和相关平滑提供接口。
public abstract float idf(long docFreq, long numDocs) 计算文档频率提供接口。
public abstract void computeNorm(FieldInvertState state, Norm norm)
```

4 搜索结果排序的改进

Lucene 在显示检索结果时, 评分的基本特点是所查询的词在一个文档中位置并不重要; 若一个文档中所含该查询词的次数越多, 则其得分越高; 一个命中的文档中, 如果除了该查询词之外, 其它词越多, 则其得分越少, 但是这种排序精确度不高, 不能充分体现文档或者网页的重要性。具体来说 Lucene 默认采用向量模型的排序算法得到文档得分, 分数越高、其排名越靠前^[4], 而忽略了标题、关键字等激励因子远大于正文的激励因子。

本文提出修改文档标题和摘要域的权重, 通过使用 Setboost() 方法^[5]改变 field 的权重, 例如, 如果 term 出现在标题或摘要中获取激励因子乘以与 tf 成递增关系的倍数 X, 提高得分, 修改后的打分公式如下, field_flag 为命中 bool 变量 0 或者 1。

$$Score = coord(q, d) * queryNorm(q) * \sum (tf(tind) * idf(t)^2 * t.getBoost() * (field_flag * X + 1) norm(t, d))$$

其次, 在文档打分相同条件下, 查询用户往往希望看到最近的文档, 本文提出在原有的 score 基础上加上时间因子。修改后的打分公式为: $L_Score(d) = score * 90\% + d_time * 10\%$, 其中 $d_time = (\text{当前时间} - \text{文档时间域}) / \text{模}$ 。

5 测试环境及结果

实验测试以 Lucene3.6 为开发工具包, 开发工具为 eclipse7.0, 开发平台为 jdk1.6, 测试文档分别为 doc1、doc2、doc3、doc4, 每个文本文档包含 title、abstract、tex、date 四个域, 实验有两组结果, 分别为第一组实验四个文档 title 和 abstract 不包含索引关键字; 第二组修改文档 1、3、4 在标题和摘要中随机添加索引关键词。对两组实验进行索引查询后的排序得分如图 3。

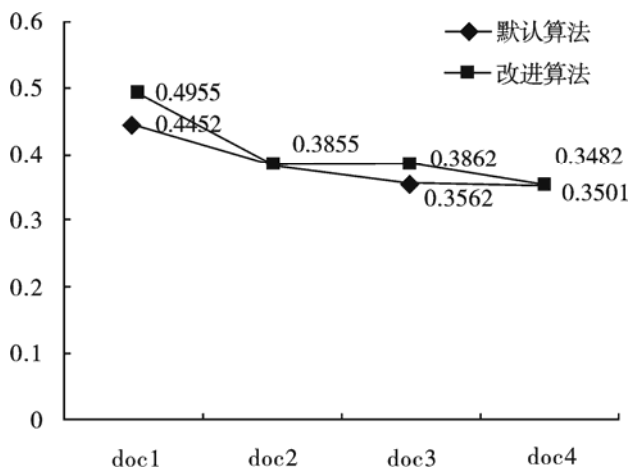


图3 实验结果对比

通过实验结果分析，关键词的位置激励因子起到了排序作用，实验排序结果达到预期效果，但是在时间因素影响排序方面还可以改进为以月或日或小时为比较单位。另外根据应用需求，系统还使用 Filter 对搜索结果进行过滤，可以获得更小范围内更精确的结果。

参考文献：

- [1] 罗刚. 解密搜索引擎技术实战[M]. 北京：电子工业出版社，2011.
- [2] 李刚，宋伟，邱哲. Ajax+Lucene 构建搜索引擎[M]. 北京：人民邮电出版社，2006.
- [3] 高凯，郭立伟等. 网络信息检索技术及搜索引擎系统开发[J]. 北京：科学出版社，2010.
- [4] 管建和，甘剑锋. 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, (2).
- [5] Lucene 官方网站[EB/OL]. <http://Lucene.apache.org/>. 2013-09-01.

Research and Improvement of Full-text Retrieval Sorting Algorithm Based on Lucene

RUAN Shu-fen

(Jiangcheng College, China University of Geosciences, Wuhan Hubei 430020, China)

Abstract: In the process of text retrieval, the sort algorithm affects the quality of search engine to a certain. This paper analyzes the structure of Lucene, to be familiar with the course and theory of creating index files, searching index files, sorting the results and discussing the improved presentation of sorting algorithm of Lucene. On the foundation of the sorting algorithm, the sorting result is improved by weighting major descriptor. Experimental results show that the improved sorting algorithm increase accuracy of the search system, and meet with the real needs of the project.

Key words: Lucene; VSM; TF-IDF; Sorting Algorithm