关于 Lucene 索引工具的性能优化研究

张春燕,刘发升
(江西理工大学 信息工程学院,江西 赣州 341000)

摘 要:随着计算机的发展,为了能更好地搜索到所需的内容,全文检索引擎已经变得越来越重要。Lucene 作为当前最流行的开源 Java 索引工具包,索引性能的提升是非常关键也是值得研究的,同时索引过程快慢是衡量一个搜索引擎的重要指标。在 Java 的基础上,通过更改 Lucene 提供的内置参数来适应不同计算机的硬件达到性能调节的作用。其次,更是提出了一种修改源代码的方法,主要是对 Hits 进行改进,提出了一种在结果非常多的情况下加速查询的方法,通过对索引过程和 Hit 结果集进行两方面的设置和改进达到性能提高的双重效果。仿真实验结果表明该改进方法不仅提高了索引工具性能的优化,同时减少了负载。通过这两方面的改进,可以使 Lucene 开发的搜索引擎达到更高的效率。

关键词:搜索引擎;Lucene;索引;Hits
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2011)05-0121-03

Lucene Indexing Tools Research Based on Optimization of Performance

ZHANG Chun-yan, LIU Fa-sheng
(College of Info. Eng., Jiangxi University of Science and Technology, Ganzhou 341000, China)

Abstract: With the development of the computer, in order to better search to the desired content, full-text search engines have become increasingly important. Lucene as the most popular open source Java toolkit index, the index is critical to enhance performance is worthy of study, while the speed of the indexing process is an important measure of a search engine indexes. Based on Java, Lucene provided by changing the built-in parameters to adapt to different computer hardware to the role of performance tuning. Secondly, proposed a method of modifying the source code, mainly for Hits to improve, a lot of cases in the results to accelerate the query method, the results of the indexing process and Hit both sets of settings and improvement to the dual effect of improving performance. The simulation results show that the improved method not only improves the optimization performance index tool, while reducing the load. These two improvements, can make Lucene search engine developed to achieve higher efficiency.

Key words: search engine; Lucene; index; Hits

0 引 言

说到底搜索引擎的原理是很简单的,在 J2se 中就知道如何在一篇 String 类型组成的文本进行正则表达式的匹配工作,或者使用 String 类型的 indexOf 方法去寻找要搜索的内容。这个方法对于一两篇文章来说是可行的。因为就算去搜索 10 篇文章也不会使用掉太多的时间。可是现在,如果有 50 亿篇文章去搜索的话,那么这种方式就变得极为不可行了。可能每查找一次都要用掉几十个小时。于是采用另外一种方法,那就是使用索引。

搜索引擎貌似很复杂,实际上核心就是这个索引。索引的结构很简单,类似一张数据库表[1]。基本结构就是:第一个字段是 keyword,第二个字段是在出现了 keyword 的文章的标题。这样只要搜索了 keyword,它就能迅速地定位到出现了 keyword 的文章有哪几篇。这就是索引的最基本的结构。其实搜索引擎就是在搜索索引中的内容[2]。不论做的是 web 搜索引擎还是文件搜索引擎,只要有了索引,问题就解决掉一大半了,接下来就是利用解析器把索引解析,报告哪篇文章出现过你搜索的 keyword,就可以实现一个最基本的搜索系统[3,4]。

目前开源的搜索工具包就是 Lucene,有了它就可以很轻松地建立索引库。这也是为什么值得研究 Lucene 的主要原因。有了 Lucene 最关键的还是数据源的问题。笔者最开始研究 Lucene 的时候就是使用本地机器上的文本文件作为数据源的。其实数据源是什

收稿日期:2010-11-03;**修回日期**:2011-01-23
基金项目:江西省科技攻关项目(赣财教[2005]132 号);江西省教育科技计划项目(GJJ08283)
作者简介:张春燕(1987-),女,江西赣州人,硕士研究生,研究方向为数据挖掘与数据库;刘发升,教授,博士,研究方向为数据挖掘与数据库。

么对于建立一个初级的搜索引擎来说根本就不重要。不管是 word 文档,pdf,txt,html 最终就是使用一个分词器把它们统一建立成一个索引库。有索引库就可以对索引库进行解析,看看具体哪个词在哪些文件中出现过。然后以一个结果的形式展现出来[5~7]。

1 修改参数提高 IndexWriter 索引性能

每台机器的硬件性能不同,所以要充分发挥这一点,利用 Lucene 提供的参数设置来调节整个索引的过程。其实索引过程的最大局限就是在要索引的文件过大的时候,因为对文件的频繁读写产生了性能上的瓶颈。Lucene 在这方面早为我们想到了。

Lucene 提供的三个内存参数如下[8~12]:

1)minMergeDocs--最小合并文档数。

设计这个参数的目的是为了控制内存的索引的文档到多少数量的时候再将它们写入硬盘。所以这个值是越大越好的,因为内存的读取速度比硬盘的读取速度是要高出好几个数量级的,所以这是一个提高性能的很明显的参数。本参数的默认值是 10。

2)maxMergeDocs--最大合并文档数。

本参数设定为索引块中的文档数的最大数量。一个索引块的文件越多,检索的速度就会越快。默认值是 Integer. MAX_VALUE(Int 的最大数值)。

3)mergeFactor--合并因子。

本参数是一个频率,在 Lucene 的一个索引块中可以存放多少文档以及把磁盘上的索引块合并成一个大的索引块的频率。

当索引块中的文档数达到一定数量的时候,索引就必须写到一个新的索引上,而且如果个数也达到一定数量的话,就会把这些个索引合并成一个索引块,参数默认为 10。同理这个值也是越大越好。

程序 1 列出了这三个参数的用法 1。

程序 1:提高索引性能

```
……
    int mergeFactor = 10;
    int minMergeDocs = 10;
    int maxMergeDocs = Integer. MAX_VALUE;
    IndexWriter indexWriter = new IndexWriter(indexDir,luceneAnalyzer,true);
    indexWriter. mergeFactor = mergeFactor;
    indexWriter. minMergeDocs = minMergeDocs;
    indexWriter. maxMergeDocs = maxMergeDocs;
    //Add documents to the index
    for(int i = 0; i < textFiles. length; i++){
    if(textFiles[i]. isFile( ) >> textFiles[i]. getName( ). endsWith(". txt")){
    Reader textReader = new FileReader(textFiles[i]);
    Document document = new Document( );
    document. add(Field. Text("content",textReader));
    document. add(Field. Keyword("path",textFiles[i]. getPath( )));
    indexWriter. addDocument(document);
    }
    }
……
```

通过上面的程序,分析得知,以上的 3 个参数为提高性能提供了非常灵活的配置。以下是这三个参数不同的时候索引的时间发生的不同变化(见表 1)。

表 1 测试结果

合并因子	最小合并文档数	最大合并文档数	文档数量	时间(秒)
10	10	IntegerMAX_VALUE	10,000	423
100	10	IntegerMAX_VALUE	10,000	270
100	100	IntegerMAX_VALUE	10,000	213
100	100	100	10,000	220
1000	1000	IntegerMAX_VALUE	10,000	194

通过表 1 可以清楚的反应,设置合并因子和最小合并文档数是可以明显地提高索引性能的。其中值是给的越大越好,当然也要根据计算机的硬件情况来定。

2 对 Lucene Hits 的改进

下面来介绍第二种方式,本方法需要修改源代码。通常都是使用 Hits 来访问 Search 的结果[13]。Search 的速度确实很快,不过当结果很多的话,例如 1 万个以上通过 Hits 访问所有的结果速度将会非常慢,就是简单地从每个结果中读一个 Field,在笔者的测试机上用了接近 2 分钟。本希望通过 Lucene 查找出符合需求的数据 ID,再通过 ID 去判断数据库中的其他域来决定最终的结果。这样连取 ID 就需要 2 分钟用户是受不了的。

于是把代码导入到 Eclipse 中进行源代码的研究。通过阅读 Hits 的代码后终于找到了解决问题的办法。Lucene 的代码看起来并不是特别专业。比如下面这两个 Hits 的初始化函数。首先里面的 q,s,f 什么的让人看起来就不是太舒服(其他的代码里还用 i,j 做循环变量)。其次这两个函数只有 o 那一个赋值不一样,明显应该只写一个,让另一个变量来调用。最后程序里面直接用了 50 这个常数,犯了编程的大忌[14,15]。代码如下:

```
Hits(Searcher s, Query q, Filter f) throws IOException {
    weight = q. weight(s);
    searcher = s;
    filter = f;
    nDeletions = countDeletions(s);
    getMoreDocs(50); // retrieve 100 initially
```

```
        lengthAtStart = length;
    }

    Hits(Searcher s, Query q, Filter f, Sort o) throws IOException {
        weight = q.weight(s);
        searcher = s;
        filter = f;
        sort = o;
        nDeletions = countDeletions(s);
        getMoreDocs(50); // retrieve 100 initially
        lengthAtStart = length;
    }
```

通过这两个函数,应该看出 Hits 初始化的时候只调入了前 100 个文档。

一般是通过 Document doc(int n)函数来访问的。这个函数里面先判断了有多少数据已经被调入了,如果要访问的数据不在,就去调用 getMoreDocs 函数,getMoreDocs 会取得需要的 2 倍文档进来。

但是 getMoreDocs 的代码比较让人疑惑,里面一段代码是这样的:

```
int n = min * 2;      // double # retrieved
TopDocs topDocs = (sort == null) ? searcher.search(weight, filter, n) : searcher.search(weight, filter, n, sort);
```

每次翻倍的时候都要去调 search 重新查找,除非 search 里面有缓存,否则性能指数一定下降啊! 实际上 Hits 最终使用的也是 TopDocs,Searcher 组合来实现输出结果,那不如直接使用下层一点的对象了。原来的代码是:

```
Hits hits = searcher.search(query);
for(int i=0;i<hits.length();i++) {
    Document doc = hits.doc(i);
    szTest.add(doc);
}
```

现在改为:

```
TopDocs topDoc = searcher.search(query.weight(searcher), null, 100000);
/*注意最后一个参数,是 search 返回的结果数量,应该比你最大可能返回的数量大,否则 ScoreDoc 里面就是你设置的数量。*/
ScoreDoc[] scoreDocs = topDoc.scoreDocs;
for(int i=0;i<scoreDocs.length;i++) {
    Document doc = searcher.doc(scoreDocs[i].doc);
    szTest.add(doc);
}
```

结果把 12000 个 ID 加入 ArrayList 用时 0.4 秒,快了几百倍。

只需要 ID 字段,但是返回整个 Doc,其他两个文本 Field 也返回了。因为 Lucene 是倒索引保存信息的,每一个文本 Field 需要重新组合成原始的字符串,这也是要耗时间的。

searcher 的 doc 函数有一个可以限定只取部分域的:

```
Document doc(int n, FieldSelector fieldSelector)
下面定义一个 FieldSelector,只取某一个给定名字的 Field
class SpecialFieldSelector implements FieldSelector {
    protected String m_szFieldName;
    public SpecialFieldSelector(String szFieldName) {
        m_szFieldName = szFieldName;
    }
    public FieldSelectorResult accept(String fieldName) {
        if(fieldName.equalsIgnoreCase(m_szFieldName)) {
            return FieldSelectorResult.LOAD;
        }
        else {
            return FieldSelectorResult.NO_LOAD;
        }
    }
}
```

再修改笔者的测试代码:

```
ScoreDoc[] scoreDocs = topDoc.scoreDocs;
ArrayList<Document> szTest = new ArrayList<Document>();
FieldSelector fieldSelector = new SpecialFieldSelector(FIELD_ID);
for(int i=0;i<scoreDocs.length;i++) {
        Document doc = searcher.doc(scoreDocs[i].doc, fieldSelector);
        szTest.add(doc);
}
```

现在返回 1.2W 个 ID 耗时 0.25 秒。虽然比前面只少了大约 150 毫秒,但是提高了接近 40%,在负载比较大的应用中还是很重要的。

3　结束语

文中对 Hits 进行改进,提出了一种在结果非常多的情况下加速查询的源代码改进方法,以及通过更改 Lucene 的内置参数来得到性能的优化效果。Lucene 的代码在有些方面其实不是很专业更谈不上是权威的,还有很大的改进空间,只要研究人员多研究 Lucene 的代码一定会发现更多可以改进的地方,为开源搜索贡献力量。

参考文献:

[1] 车　东. 在应用中加入全文检索功能:基于 JAVA 的全文索引引擎 LUCENE 简介[EB/OL]. 2002-08. http://www.chedong.com/tech/lucene.html.

(下转第 238 页)

49-68.
[2] Kilov H, Ross J. Information modeling: an object - oriented approach[M]. Englewood Cliffs, N. J.: Prentice-Hall, 1994.
[3] Mealy G H. Another Look at Data[C]//Proceedings of the Fall Joint Computer Conference. Washington, DC: Thompson Books; London: Academic Press, 1967:525-534.
[4] Guizzardi G. Ontological Foundations for Structural Conceptual Models[M]. The Netherlands: Universal Press, 2005.
[5] Wand Y, Weber R. On the Ontological Expressiveness of Information Systems Analysis and Design Grammers[J]. Journal of Information Systems, 1993, 3:217-237.
[6] Wand Y, Weber R. Mario Bunge's ontology as a formal foundation for information systems concepts[C]//Studies on Mario Bunge's Treatise. Amsterdam: Rodopi, 1990: 123 - 149.
[7] Wand Y, Weber R. On the deep structure of information systems[J]. Information Systems Journal, 1995, 5(3): 203 - 223.
[8] Weber R. Ontological Foundations of Information Systems [M]. Melbourne: Coopers & Lybrand, 1997.
[9] Milton S K, Kazmierczak E, Keen C. Data modeling languages: An ontological study[C]//Proceedings of the 9th European Conference on Information Systems. [s. l.]: [s. n.], 2001:304-315.
[10] Guarino N. Formal Ontology and Information Systems[C]//Proceedings of the International Conference on Formal Ontology and Information Systems (FOIS). Trento, Italy: IOS Press, 1998:3-15.
[11] Niles I, Pease A. Towards a Standard Upper Ontology[C]//Proceedings of FOIS. Ogunquit, Maine, USA: [s. n.], 2001:17-19.
[12] Colomb R M. Ontology and the SemantieWeb[M]. Netherlands: 150 Press, 2007:87-90.
[13] Carasik R P, Johnson S M, Patterson D A, et al. Towards a Domain Description Grammar: An Application of Linguistic Semantics[C]//Proceedings of the Fourth International Workshop on Computer-Aided Software Engineering. Irvine, CA: [s. n.], 1990:410-419.
[14] 王杏林,曹晓东. 概念建模[M]. 北京:国防工业出版社, 2007.
[15] 石　莲,孙吉贵. 描述逻辑综述[J]. 计算机科学,2006,33(1):194-225.
[16] 邓志鸿,唐世渭,张　铭,等. Ontology 研究综述[J]. 北京大学学报,2002,38(5):730-738.
[17] Doninifm, Lenzerinim, Nardid, et al. Reasoning in description logics[C]//Studies in Logic, Language and Information. [s. l.]: CLSI Publications, 1996:193 - 238.
[18] Cali A, Calvanese D, De Giacomo G, et al. Reasoning on UML class diagrams in description logics[C]//Proc. of IJCAR Workshop on Precise Modeling and Deduction for Object-oriented Software Development (PMD 2001). [s. l.]: [s. n.], 2001.
[19] 王　军,王继军,甘　丹,等. 基于描述逻辑的概念图推理[J]. 计算机科学,2008,35(8):176-179.
[20] 许卓明,董逸生,陆　阳. 从 ER 模式到 OWL DL 本体的语义保持的翻译[J]. 计算机学报,2006,29(10):1786-1796.
[21] Jarrar M. Towards Automated Reasoning on ORM Schemes -Mapping ORM into the DLRidf Description Logic[C]//Proc. of the 26th Int. Conf. on Conceptual Modeling (ER2007). [s. l.]: Springer, 2007:181-197.

(上接第 123 页)

[2] 李晓明,刘建国. 搜索引擎技术及趋势[EB/OL]. 2003-04. http:// www. Se-express. com/se/se07. hml.
[3] 蔡建超,郭一平,王　亮. 基于 LUCENE. Net 校园网搜索引擎的设计与实现[J]. 计算机技术与发展,2006,16(11):74-80.
[4] 李　尉,霍　涛,惠勇侠. WWW 的信息检索技术研究[J]. 济南大学学报(自然科学版),2001,15(3):284-286.
[5] 赵　汀,孟祥武. 基于 LUCENE API 的中文全文数据库的设计与实现[J]. 计算机工程与应用,2003(20):179-183.
[6] 张校乾,金玉玲,侯玉波. 一种基于 lucene 检索系统的一种全文数据库的设计与实现[J]. 现代图书情报技术,2005(2):40-44.
[7] 彭洪汇,林作铨. Internet 上的搜索引擎和元搜索引擎[J]. 计算机科学,2002, 29(9):1-12.
[8] Cospodnetic O, Hatcher E. Lucene in Action[M]. [s. l.]: Manning Publications Co, 2005:15-18.
[9] Owens S J. Lucene Tutorial [EB/OL]. 2009-11. http// dark. Sleep/m/LUCENE/.
[10] 曹元大,贺海军,涂哲明. 中文 web 文档全文检索系统的设计与实现[J]. 北京理工大学学报,2002 ,22(1):68-71.
[11] 颜维龙,盖　杰,武港山,等. 面向网络的全文检索中索引文件的组织[J]. 计算机应用研究,2002, 11(2):124-126.
[12] 李　刚,宋　伟,邱　哲. 征服 Ajax 和 Lucene 构建搜索引擎[M]. 北京:人民邮电出版社,2006.
[13] Du Lin, Zhang Yibo, Sun Yufang. The design and implementation of Web—based Chinese text retrieval system SEARCH 2000 [J]. Journal of Chinese Information Processing, 2000, 14(6):14-20.
[14] Rogers W, Candela G, Harman D. Space and time improvements for indexing in information retrieval[C]//The Fourth Annual Symposium on Document Analysis and Information Retrieval(SDAIR'93). LasVegas: [s. n.], 1995.
[15] Sahon G, Wong A. On the specification of term value in automatic indexing[J]. Journal of Documentation, 1973, 29(4):351-372.