

新浪微博实时索引系统*

郭洋^{1,†}

付乔宾^{1,‡}

卢亮^{1,§}

¹中国科学院计算技术研究所, 北京 100080

摘要 本文是中国科学院研究生院网络挖掘技术课程的课程设计报告, 介绍了本组新浪微博实时索引系统的设计和实现, 并给出了系统的测试结果和详细使用说明。

关键词 微博; lucene; 实时索引; 实时搜索

Sina Micro-blog Real Time Indexing System

Yang Guo¹

Qiao-Bin Fu¹

Liang Lu¹

¹Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P. R. China, 100190

Abstract This is a curriculum design report for web mining technology class of Graduate School, Chinese Academy of Sciences. In this report, we describe the design and implementation of the Sina microblog real-time indexing system, and present test results and detailed instructions for use of the system.

Keywords microblog; Weibo; real time index; real time search

1 综述

目前, 微博成为了越来越火的应用。微博信息不同于一般信息, 实时性很强。因而, 微博的实时搜索成为了新的应用需求, 各大搜索引擎和微博网站纷纷提供了自己的微博实时搜索功能。所谓微博的实时搜索, 是指在微博数据中实时查找与用户提交的关键词相匹配的文档的过程, 一般包括微博抓取、索引建立, 搜索结果排序, 以及搜索界面几个部分。典型结构如图1。

微博实时搜索系统的核心是实时索引的建立。Lucene是一个基于Java的全文索引工具包, 可以为微博实时搜索系统提供索引和搜索功能。但Lucene不是一个完整的全文索引应用, 而是是一个基于Java的全文索引引擎工具包, 它可以方便的嵌入到各种应用中实现针

*本文是中国科学院研究生院网络挖掘技术课程设计的报告。

[†]通讯作者, 学号: 201128013229023。电子邮件: guoyang@ict.ac.cn

[‡]学号: 201128013229017。

[§]学号: 2011e8013261173。

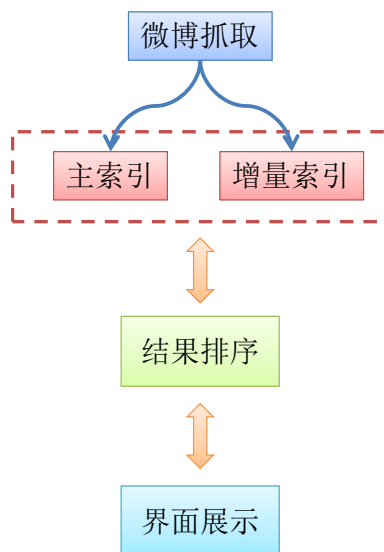


图 1: 微博实时搜索系统

对应用的全文索引/检索功能。Lucene可以对任何可以转化为文本的数据做索引和搜索。由于微博实时搜索的对象是文本数据，在索引建立时涉及到的数据也都是文本，所以可以充分利用Lucene提供的API，而把精力集中在应用逻辑的实现上。

我们小组基于lucene提供的相关工具集，设计和实现了一个高实时性的微博索引系统，可以针对最新的微博信息迅速建立反向索引，使得一篇微博一旦发布就能在尽量短的时间内被搜索到。

2 实时搜索介绍

基于lucene的中文实时索引系统要解决的两个关键问题是：(1)中文分词系统设计；(2)如何实现实时性，下面分别讨论这两个问题。

2.1 分词系统

中文与西方语言最大的区别就在于语句的词汇之间没有明显的分词界限，但是计算机自然语言处理是按词汇进行的，因此中文分词的准确性直接影响中文检索的效果。在中文分词领域，已经有了诸多的开源解决方案，因而我们并不需要自己设计实现一个分词系统，只需要从现有的系统中选择一个符合自己要求的即可。目前最新版本的lucene（lucene 3.5）自身提供的标准分词器（StandardAnalyzer）已经具备了中文分词的功能，但是只支持一元分词，不能够满足大多数应用的需要。除了lucene自带的标准分词器外，常见的第三方分词器有Paoding、mmseg4j、CJKAnalyzer、ChineseAnalyzer、imdict、IK_CAnalyzer等，其中CJKAnalyzer是lucene的contribution中附带的，为中日韩语言分词而设计。这些第三方分词器绝大多数都支持多元分词。

针对索引而言，一元分词和多元分词的主要优缺点对比如表1所示。

表 1: 一元分词和多元分词的主要优缺点

	优点	缺点
一元分词	索引文件不用更新	索引冗余大，查找效率低
多元分词	索引文件小，搜索效率高，准确度高	分词结果受词典影响较大，如果分词词典频繁发生变动，那么索引必须重建

经过对比，我们选择Paoding作为我们的分词模块，Paoding中文分词具有极效率高和高扩展性，并且引入隐喻，支持不限制个数的用户自定义词库。Paoding分词采用细粒度全切分，对于不在词典中的词进行二元分词。Paoding分词（多元分词）和StandardAnalyzer分词（一元分词）的结果对比如表2所示。

表 2: Paoding分词和StandardAnalyzer分词的结果对比

StandardAnalyzer	2008年8月8日晚，举世瞩目的北京第二十九届奥林匹克运动会开幕式在国家体育场隆重举行。	2008/年/8/月/8/日/晚/举/世/瞩/目/的/北/京/第/二/十/九/届/奥/林/匹/克/运/动/会/开/幕/式/在/国/家/体/育/场/隆/重/举/行/
PaodingAnalyzer	2008年8月8日晚，举世瞩目的北京第二十九届奥林匹克运动会开幕式在国家体育场隆重举行。	2008/年/8/月/8/日/晚/举世/瞩目/举世瞩目/目的/北京/二/第 二/十/二 十/第 二十/九/十 九/二 十九/九 届/奥林/奥林匹克/运动/运动会/奥林匹克运动会/开幕/开幕式/国家/体育/体育场/隆重/举行/隆重举行/

Paoding没有发布针对lucene 3.5的版本，其现有发行包不支持最新的lucene 3.5，需要我们自己从版本库检出一份，从源码编译，并按照相关要求对词典进行配置。

```
svn co http://paoding.googlecode.com/svn/trunk/paoding-analysis/
```

2.2 实时索引原理

目前，实时搜索成为了搜索引擎发展的重要方向，越来越多的开发者开始关注搜索的实时性，微博信息的搜索是最有代表性的实时搜索之一。实时搜索的核心是实时

索引。lucene内在的实现机制使得它在实时搜索方面有一些与生俱来的缺点，如果想要用Lucene实现实时，必须在向索引新添加文档后，立马提交IndexWriter (commit)，并且在搜索的时候重新打开对应的IndexReader，然而当索引在硬盘上的时候，尤其是索引非常大的时候，IndexWriter的commit操作和IndexReader的open操作都是非常耗时的，根本达不到实时性的需要。因而，在构建实时索引时，我们需要内存中的索引RAMDirectory和硬盘上的索引FSDirectory相互配合来解决问题。这也是现有基于lucene的实时搜索系统普遍采用的方法。

实时索引的原理：

(1) 初始化阶段

实时索引的原理是同时利用两个内存索引A、B和一个硬盘索引D。初始化阶段A在内存中创建一个索引区域，索引B为null，索引D打开硬盘上的索引。如图2所示。

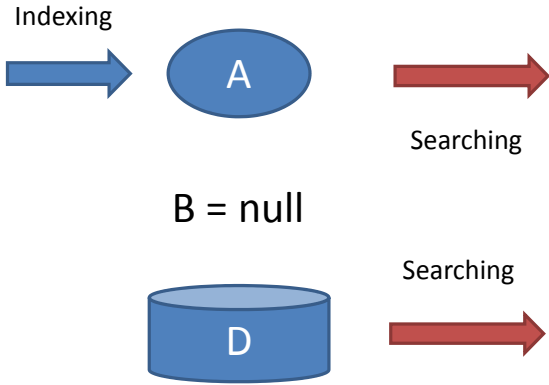


图 2: 初始化阶段

当有新的文档需要被索引时，该文档直接被添加到内存索引A中，然后马上commit，并且重新打开对应的IndexReader，硬盘索引D不变。搜索的时候，索引A和D同时返回结果。由于内存中索引的IndexReader是每添加完文档后立刻更新的，而且速度很快，而硬盘上的索引一旦打开，在下次合并之前会一直使用，从而使得一篇文档一经索引，就能够立刻被搜索的到。

(2) 合并阶段

经过一段时间，索引A中的文档达到一定数量，这个时候需要启动合并过程，将内存中的索引合并到硬盘上。合并是一个相对而言比较耗时的过程。这时候我们创建内存索引X，然后把B指向A，A指向X，在合并过程中新添加的文档全部索引到X中（即A中）。如图3所示。

在合并阶段，如果有索引请求发送过来，则新文档被直接添加到A，然后重新打开A的IndexReader；如果有搜索请求发送过来，则A、B、D对应的三个IndexReader全都返回搜索结果，由于索引A的IndexReader是添加文档后立刻更新的，因而能够保证新添

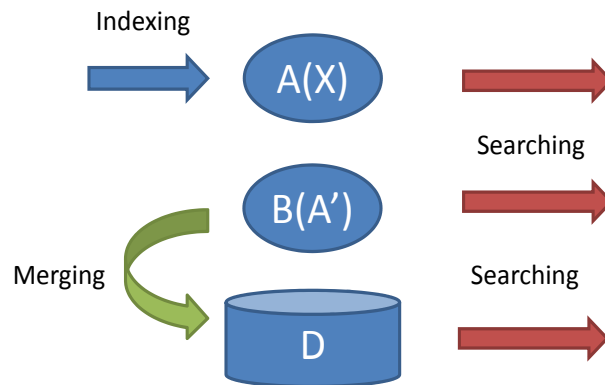


图 3: 合并阶段

加的文档能够马上被搜索到，这个时候虽然索引B已经在同硬盘索引进行合并，然而由于硬盘索引的IndexReader还没有重新打开，因而索引B中的数据不会被重复搜索到。

(3) 合并后阶段

当索引A中的数据已经完全合并到硬盘上之后，则需要重新打开硬盘索引的IndexReader，然而这可能是一个相当耗时的过程，因而在成功打开之前我们一直不释放索引B，并且不更新硬盘的IndexReader，而使用一个临时的IndexReader来打开硬盘索引，重新打开后，更新硬盘索引的IndexReader，删除内存索引B，并将B置为null。

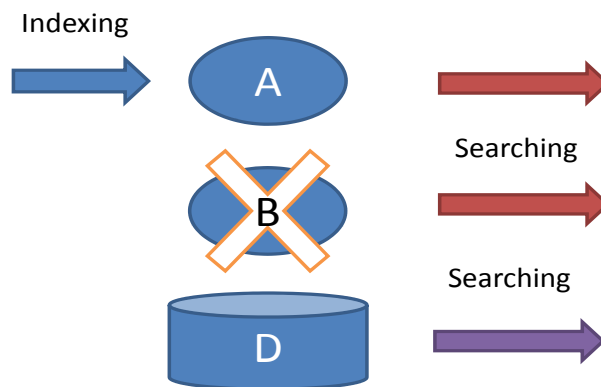


图 4: 合并后阶段

3 RealTimeIndex 系统设计与实现

3.1 总体架构设计

实时索引系统主要由四个模块构成：索引模块、搜索模块、合并模块、日志模块。如图5所示。其中，索引模块负责接受索引请求，为微博创建反向索引（inverted index）；搜

索模块负责创建线程池，接受搜索请求，返回搜索结果；合并模块负责定期将内存中的索引写到硬盘上，从而防止内存占用过大，这个过程中，各个模块之间的同步尤为重要；日志模块负责记录系统的运行状态。



图 5: 系统模块图

系统类图如图6所示。

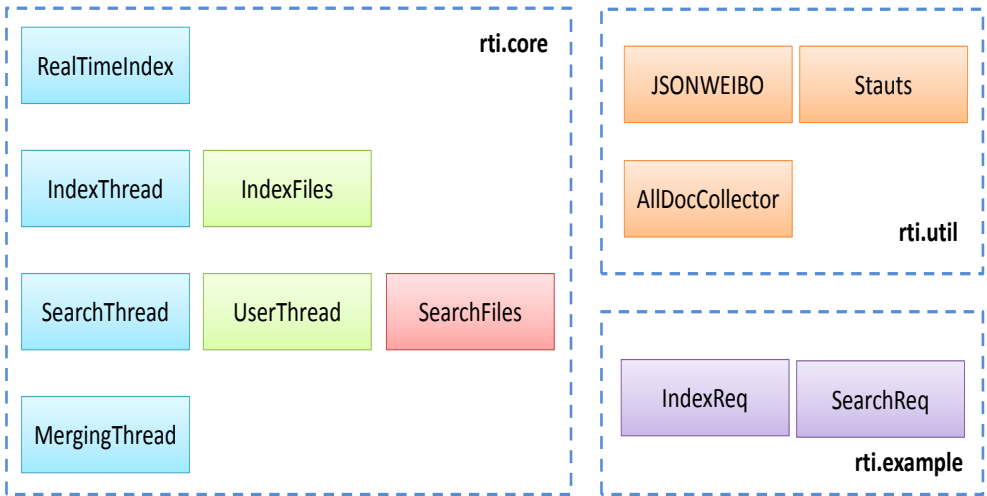


图 6: 系统类图

rti.core是实时索引系统的核心代码包，rti.util是工具类，rti.example是调用本系统接口的示例程序。此外，本系统还依赖于commons-logging，log4j，java-json和paoding-analysis。

系统启动时，首先根据用户参数设置各程序运行参数，然后分别启动索引线程、搜索线程和合并线程。搜索线程启动后，接受搜索请求，并为每个搜索请求从线程池中选择一个空闲线程为用户提供服务。各线程之间的关系如图7所示。

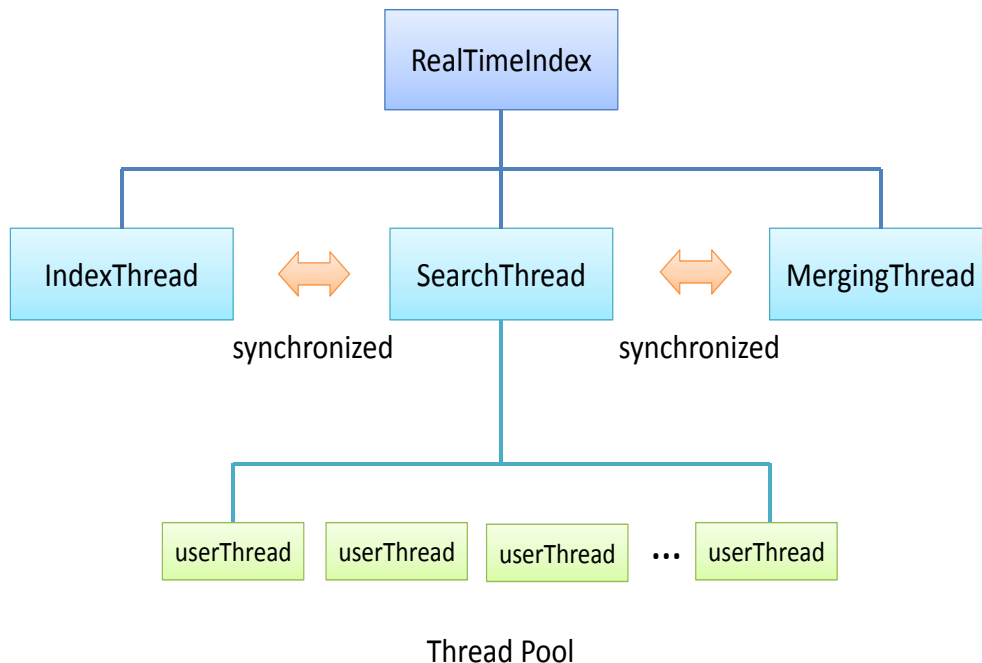


图 7: 线程关系

各线程之间的同步过程如下:

(1) 索引线程每次更新完内存索引后, 需要更新搜索线程中的对应IndexReader。

```

// reopen the searchThread's ramReader
RealTimeIndex.getSearchThread().reopenRamReader(); // synchronized
method
  
```

(2) 合并线程启动条件满足的时候, 启动合并过程, 这时需要创建新的内存索引空间, 并改变索引线程的IndexWriter和搜索线程的IndexReader。

```

// before merge
Directory newRamDir = new RAMDirectory();
// initial the dir
IndexWriter iw = RealTimeIndex.initDir(newRamDir, true);
RealTimeIndex.getSearchThread().changeReader(newRamDir); // synchronized
method
RealTimeIndex.getIndexThread().changeWriter(iw); // synchronized method
  
```

(3) 合并过程完成之后, 需要重新打开搜索线程的硬盘索引IndexReader, 同时更新其内存索引的IndexReader。

```

// after merge
  
```

```
RealTimeIndex.getSearchThread().reopenFsReader(); // synchronized method
    . this operation costs a lot of time
```

多个线程对同一资源的读写通过锁机制来保证。

系统的运行状态随时间变化如图8所示：

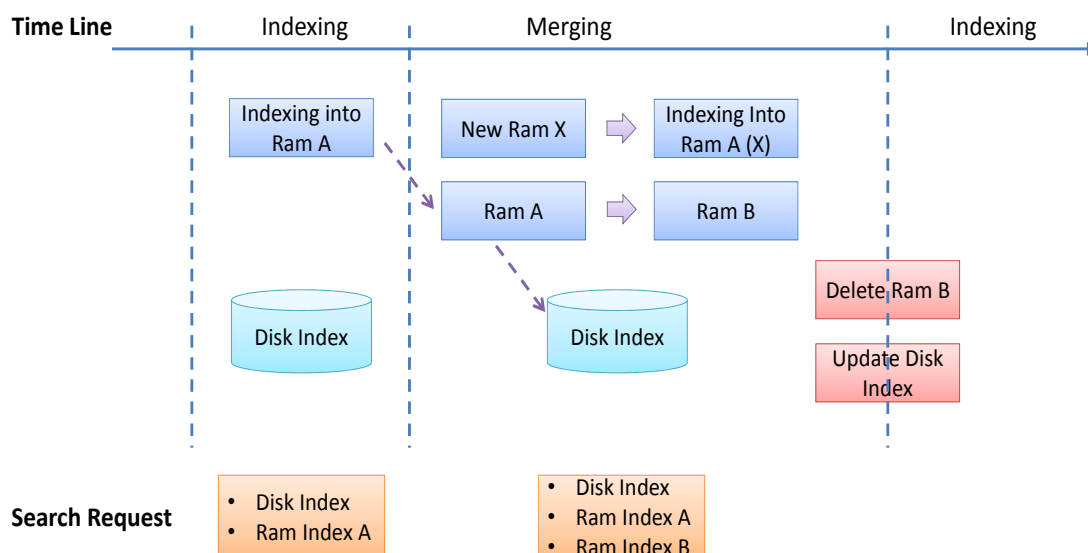


图 8: 系统的运行状态

3.2 索引部分设计和实现

3.2.1 Lucene索引机制

图9展示了Lucene的一般索引机制架构，Lucene使用各种解析器对各种不同类型的文档进行解析。比如对于HTML文档，HTML解析器会做一些预处理的工作，比如过滤文档中的HTML标签等等。HTML解析器的输出的是文本内容，接着Lucene的分词器(Analyzer)从文本内容中提取出索引项以及相关信息，比如索引项的出现频率。接着Lucene的分词器把这些信息写到索引文件中。

其中，Lucene使用的是倒排索引，它是索引部分最核心的内容。倒排索引是一种以索引项为中心来组织文档的方式，每个索引项指向一个文档序列，这个序列中的文档都包含该索引项。利用倒排索引轻松的找到那些文档包含了特定的索引项。

3.2.2 索引建立部分设计

索引建立的步骤如下：

1. 数据抓取端利用socket通信通知索引监听端元数据路径
2. 将指定的JSON格式的微博数据文件解析为JSON对象数组

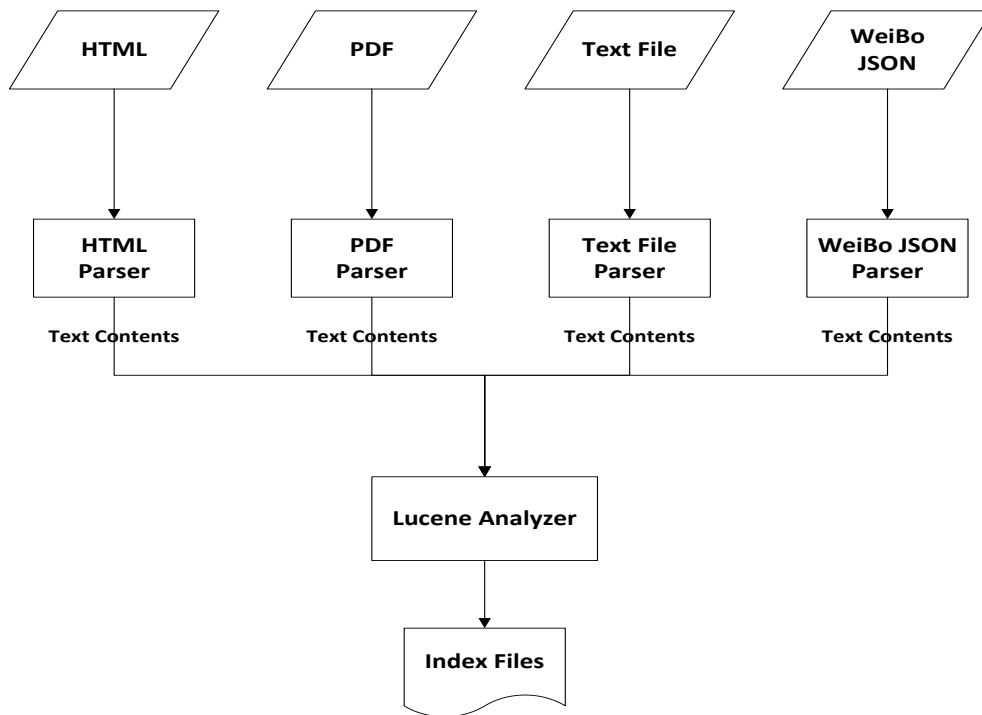


图 9: lucene索引机制

3. 提取JSON对象中的各个域，并构造包含多个字段Field对象的Document对象
4. 提交Document对象，并进行索引建立

图10展示索引建立的具体流程。

3.2.3 索引建立关键部分实现

(1)JSON格式数据提取

此部分依赖于JSON in Java (<http://www.json.org/java/>) 开源代码包，核心代码是public static JSONArray toJSONArray(String weibo)函数，它将指定的json文件转化成json对象数组。

部分代码如下：

```

...
public static String title = "statuses";
BufferedReader br = new BufferedReader(new InputStreamReader(new
    FileInputStream(weibo), RealTimeIndex.CHARSET));
String line = null;

StringBuffer strBuff = new StringBuffer();
  
```

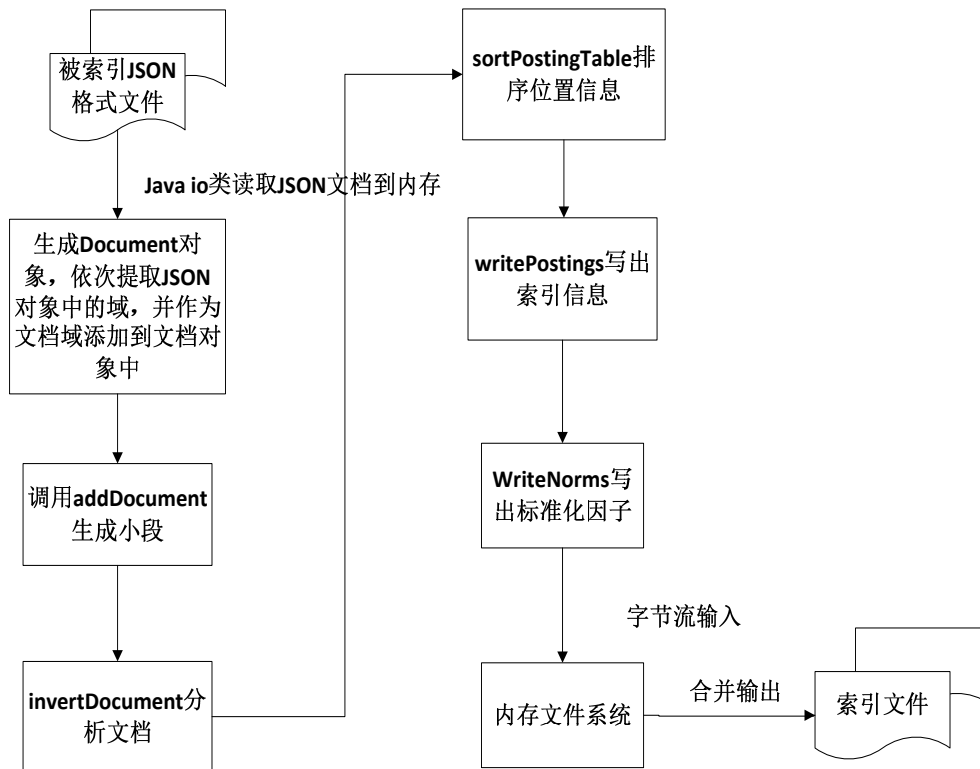


图 10: 索引建立流程

```

while((line = br.readLine())!=null)
{
    strBuff.append(line);
}

JSONObject jsonObj = new JSONObject(strBuff.toString());
array = jsonObj.getJSONArray(title);
...

```

此函数通过使用Java IO类将最新产生的JSON格式的微博数据读取到内存，并将其按照JSON格式构造JSON对象数组。其中我们使用www.json.org上公布的org.json工具进行json格式解析。

(2)索引建立部分

此部分的核心代码是public void index(String docsPath) 函数，该函数对指定的json文件建立索引。我们首先将微博的创建日期转化为秒数，以方便后面进行日期范围搜索。代码如下：

```
...
```

```

DateFormat fullFormat = new SimpleDateFormat("E MMM dd HH:mm:ss Z yyyy",
    Locale.ENGLISH);
Date date = fullFormat.parse(jj.getString("created_at"));
doc.add(newNumericField(JSONWEIBO.dateLabel, NumericUtils.
    PRECISION_STEP_DEFAULT, Field.Store.YES, true).setLongValue(date.
    getTime()));
...

```

同时我们设置微博的text字段可以进行索引，代码如下：

```

...
labFiled = new Field(JSONWEIBO.strLabel[j], jj.getString(JSONWEIBO.
    strLabel[j]), Field.Store.YES, Field.Index.NOT_ANALYZED_NO_NORMS);
labFiled.setIndexOptions(IndexOptions.DOCS_ONLY);
...

```

构造完毕文档对象，我们将其添加到索引中：

```

ramWriter.addDocument(doc);

```

3.3 搜索部分设计和实现

搜索部分的核心实现可分为几步：构造查询对象、创建文档集合对象，创建查询过滤器，从多个索引中搜索结果，对搜索结果进行合并，将搜索结果转化为JSON格式的结果并返回。

(1) 构造查询对象

首先用分词器PaodingAnalyzer和索引构造查询分析器QueryParser对象：

```

QueryParser parser = new QueryParser(Version.LUCENE_35, "text", new
    PaodingAnalyzer());

```

然后，用查询分析器分析查询关键词来构造查询对象：

```

Query query = parser.parse(keywords);

```

(2) 创建文档集合对象

文档集合对象用来存放查询的结果文档集合，初始创建时空。由于我们三个IndexSearcher，所以需要创建三个文档集合对象。我们编写了一个AllDocCollector类来创建文档集合对象，它创建的对象可以存放IndexSearcher对象搜索到的所有文档。另外，文档集合对象中还存有各文档的在搜索过程中计算出来的评分。

(3) 创建查询过滤器

由于我们需要查询在指定时间范围内索引的文档，所以我们需要给IndexSearcher的search方法指定一个查询过滤器。这里我们用到了Lucene提供的数字范围过滤器NumericRangeFilter类：

```
Filter filter = NumericRangeFilter.newLongRange("created_at", start, end, true, true);
```

(4) 从多个索引中搜索结果

利用之前构造的查询对象、文档集合对象和查询过滤器在三个IndexSearcher对象ramSearcher、mergingSearcher和fsSearcher分别查询与关键词相关的文档，结果放在三个文档集合对象中。

(5) 对搜索结果进行合并

由于我们只从搜索结果中取前backnum个文档条目，所以我们需要将查询得到的三个文档集合对象中的条目进行“合并”，然后取评分最高的前backnum个文档条目。实际上，由于全部文档条目可能过多，而各个文档集合对象中的条目分别是按评分有序排列的，所以并不需要真的合并三个文档集合对象中的所有文档，排序之后取前backnum条，而是逐条比较各文档集合对象评分最高的条目并取出，只要取backnum就可以停止比较了。

(6) 将搜索结果转化为JSON格式的结果并返回

我们查询的结果是一系列的Lucene自含的Document对象，需要转成项目商定的JSON格式。这里我们用到了用于解析json的java包org.json。典型的做法如下：

```
JSONObject jsonObj = new JSONObject();  
jsonObj.put("text", doc.get("text"));
```

其中，doc是Document对象，jsonObj是JSONObject对象。最后，把JSONObject对象转成String并插入到结果列表中。当所有条目都处理完成之后，返回结果列表。

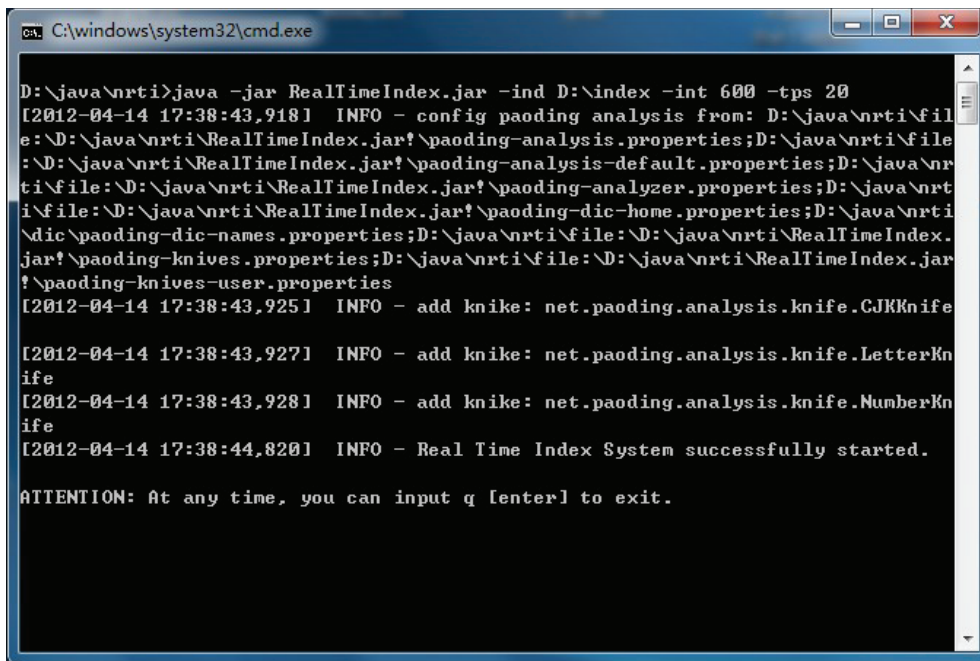
4 系统测试和评估

4.1 系统索引和搜索功能测试

首先，运行实时索引建立程序，命令参数为-ind D:\index -int 600 -tps 20 -u。运行结果如图11所示：

此时程序处于监听状态，接着运行数据抓取测试用例，通告RealTimeIndex 要处理的微博数据的位置，并对其建立索引。部分测试代码如下：

```
...  
Socket socket = new Socket("localhost", 9081);  
OutputStream os = socket.getOutputStream();  
PrintWriter pw = new PrintWriter(os);
```



```
C:\windows\system32\cmd.exe

D:\java\nrti>java -jar RealTimeIndex.jar -ind D:\index -int 600 -tps 20
[2012-04-14 17:38:43,918] INFO - config paoding analysis from: D:\java\nrti\file:
D:\java\nrti\RealTimeIndex.jar!\paoding-analysis.properties;D:\java\nrti\file:
D:\java\nrti\RealTimeIndex.jar!\paoding-analysis-default.properties;D:\java\nrti\file:
D:\java\nrti\RealTimeIndex.jar!\paoding-analyzer.properties;D:\java\nrti\file:
D:\java\nrti\RealTimeIndex.jar!\paoding-dic-home.properties;D:\java\nrti\dic\paoding-dic-names.properties;D:\java\nrti\file:
D:\java\nrti\RealTimeIndex.jar!\paoding-knives.properties;D:\java\nrti\file:
D:\java\nrti\RealTimeIndex.jar!\paoding-knives-user.properties
[2012-04-14 17:38:43,925] INFO - add knife: net.paoding.analysis.knife.CJKKnife
[2012-04-14 17:38:43,927] INFO - add knife: net.paoding.analysis.knife.LetterKnife
[2012-04-14 17:38:43,928] INFO - add knife: net.paoding.analysis.knife.NumberKnife
[2012-04-14 17:38:44,820] INFO - Real Time Index System successfully started.

ATTENTION: At any time, you can input q [enter] to exit.
```

图 11: 程序启动界面

```
String req = "1&D:\\test.json&" + start + "&" + end + "&" + interval + "\r\n";
pw.write(req);
...
```

其中，req指定索引文件信息。

从图12可以看出程序接收到请求，并找到指定JSON文件，其中文件中包含192条微博信息。如图13所示，最后完成索引建立。

此时，索引处于内存中，之后运行搜索测试用例。测试用例采用多线程并发搜索，代码如下：

```
...
String req = "高傲&-1&-1&-1";
String req1 = "早上&-1&-1&200";

multiSearchReq msr = new multiSearchReq(req,1);
multiSearchReq msr1 = new multiSearchReq(req1,2);

Thread search1 = new Thread(msr,"search1");
Thread search2 = new Thread(msr1,"search2");
```

```
C:\windows\system32\cmd.exe

D:\java\nrti>java -jar RealTimeIndex.jar -ind D:\index -int 600 -tps 20
[2012-04-14 17:38:43,918] INFO - config paoding analysis from: D:\java\nrti\file:\D:\java\nrti\RealTimeIndex.jar!\paoding-analysis.properties;D:\java\nrti\file:\D:\java\nrti\RealTimeIndex.jar!\paoding-analysis-default.properties;D:\java\nrti\file:\D:\java\nrti\RealTimeIndex.jar!\paoding-analyzer.properties;D:\java\nrti\file:\D:\java\nrti\RealTimeIndex.jar!\paoding-dic-home.properties;D:\java\nrti\dic\paoding-dic-names.properties;D:\java\nrti\file:\D:\java\nrti\RealTimeIndex.jar!\paoding-knives.properties;D:\java\nrti\file:\D:\java\nrti\RealTimeIndex.jar!\paoding-knives-user.properties
[2012-04-14 17:38:43,925] INFO - add knike: net.paoding.analysis.knife.CJKKnife
[2012-04-14 17:38:43,927] INFO - add knike: net.paoding.analysis.knife.LetterKnife
[2012-04-14 17:38:43,928] INFO - add knike: net.paoding.analysis.knife.NumberKnife
[2012-04-14 17:38:44,820] INFO - Real Time Index System successfully started.

ATTENTION: At any time, you can input q [enter] to exit.

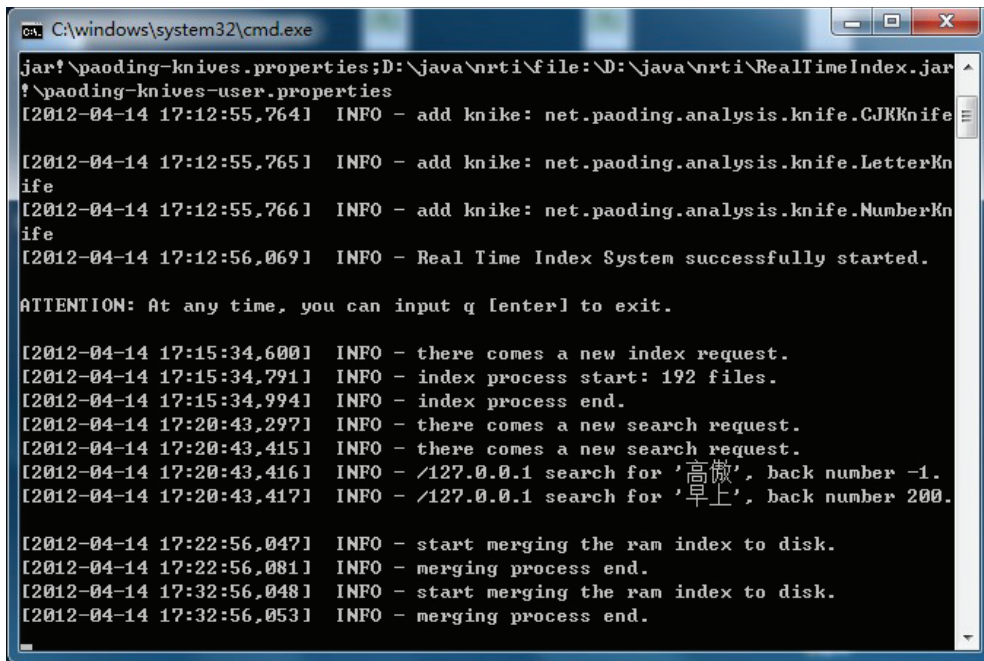
[2012-04-14 17:39:52,338] INFO - there comes a new index request.
[2012-04-14 17:39:52,503] INFO - index process start: 192 files.
[2012-04-14 17:39:52,721] INFO - index process end.
```

图 12: 索引建立界面

本地磁盘 (D:) ▸ index			
共享 ▾ 刻录 新建文件夹			
名称	修改日期	类型	大小
_0.fdt	2012/4/14 19:17	FDT 文件	207 KB
_0.fdx	2012/4/14 19:17	FDX 文件	2 KB
_0.fnm	2012/4/14 19:17	FNМ 文件	1 KB
_0.frq	2012/4/14 19:17	FRQ 文件	12 KB
_0.nrm	2012/4/14 19:17	NRM 文件	1 KB
_0.prx	2012/4/14 19:17	PRX 文件	2 KB
_0.tti	2012/4/14 19:17	TII 文件	2 KB
_0.tis	2012/4/14 19:17	TIS 文件	168 KB
segments.gen	2012/4/14 19:17	GEN 文件	1 KB
segments_2	2012/4/14 19:17	文件	1 KB

图 13: 硬盘上的索引文件

```
search1.start();
search2.start();
...
```



```
jar!\paoding-knives.properties;D:\java\nrti\file:\D:\java\nrti\RealTimeIndex.jar
!\paoding-knives-user.properties
[2012-04-14 17:12:55,764] INFO - add knike: net.paoding.analysis.knife.CJKKknife
[2012-04-14 17:12:55,765] INFO - add knike: net.paoding.analysis.knife.LetterKnife
[2012-04-14 17:12:55,766] INFO - add knike: net.paoding.analysis.knife.NumberKnife
[2012-04-14 17:12:56,069] INFO - Real Time Index System successfully started.

ATTENTION: At any time, you can input q [enter] to exit.

[2012-04-14 17:15:34,600] INFO - there comes a new index request.
[2012-04-14 17:15:34,791] INFO - index process start: 192 files.
[2012-04-14 17:15:34,994] INFO - index process end.
[2012-04-14 17:20:43,297] INFO - there comes a new search request.
[2012-04-14 17:20:43,415] INFO - there comes a new search request.
[2012-04-14 17:20:43,416] INFO - /127.0.0.1 search for '高傲', back number -1.
[2012-04-14 17:20:43,417] INFO - /127.0.0.1 search for '早上', back number 200.

[2012-04-14 17:22:56,047] INFO - start merging the ram index to disk.
[2012-04-14 17:22:56,081] INFO - merging process end.
[2012-04-14 17:32:56,048] INFO - start merging the ram index to disk.
[2012-04-14 17:32:56,053] INFO - merging process end.
```

图 14: 搜索运行情况

从图14和图15看出，两个用户并发送搜索请求，其中两个用户均获得搜索结果。

4.2 系统运行性能参数

本系统没有针对性能进行过多的优化，因而还有很大的提升空间，图16是以上测试的相关性能参数。（这些测试的数据量较小，因而数字受外界因素影响较大。）

5 系统使用说明

系统运行要求：

- 路径名不要含有中文字符
- 本系统需要可以访问到抓取系统的微博存储路径，最简单的方法是与抓取系统运行在一台服务器上
- 被索引的微博数据必须是UTF-8 without BOM 格式

使用方法：

```
java rti.core.RealTimeIndex -index INDEX_PATH [-interval MERGE_INTERVAL
(s)] [-indexport INDEX_PORT][-searchport SEARCH_PORT] [-update]
```



```

23 [2012-04-14 17:15:34,994] DEBUG indexThread rti.core.SearchThread - SearchThread reopening the ramReader.
24 [2012-04-14 17:20:43,297] INFO searchThread rti.core.SearchThread - there comes a new search request.
25 [2012-04-14 17:20:43,415] DEBUG searchThread rti.core.SearchThread - new UserThread start successfully.
26 [2012-04-14 17:20:43,415] INFO searchThread rti.core.SearchThread - there comes a new search request.
27 [2012-04-14 17:20:43,415] DEBUG pool-1-thread-1 rti.core.UserThread - new request: 高傲-16-16-1
28 [2012-04-14 17:20:43,416] DEBUG searchThread rti.core.SearchThread - new UserThread start successfully.
29 [2012-04-14 17:20:43,416] INFO pool-1-thread-1 rti.core.UserThread - /127.0.0.1 search for '高傲', back number -1.
30 [2012-04-14 17:20:43,417] DEBUG pool-1-thread-1 rti.core.SearchFiles - search process started.
31 [2012-04-14 17:20:43,416] DEBUG pool-1-thread-2 rti.core.UserThread - new request: 早上6-16-16200
32 [2012-04-14 17:20:43,417] INFO pool-1-thread-2 rti.core.UserThread - /127.0.0.1 search for '早上', back number 200.
33 [2012-04-14 17:20:43,418] DEBUG pool-1-thread-2 rti.core.SearchFiles - search process started.
34 [2012-04-14 17:20:43,516] DEBUG pool-1-thread-1 rti.core.SearchFiles - search process end.
35 [2012-04-14 17:20:43,516] DEBUG pool-1-thread-1 rti.core.UserThread - get 1 search hit(s).
36 [2012-04-14 17:20:43,516] DEBUG pool-1-thread-2 rti.core.SearchFiles - search process end.
37 [2012-04-14 17:20:43,517] DEBUG pool-1-thread-2 rti.core.UserThread - get 2 search hit(s).
38 [2012-04-14 17:20:43,517] DEBUG pool-1-thread-1 rti.core.UserThread - send the search results to client successfully.
39 [2012-04-14 17:20:43,517] DEBUG pool-1-thread-2 rti.core.UserThread - send the search results to client successfully.
40

```

图 15: 程序运行日志

测试组	微博数目	索引建立时间 (ms)	搜索关键词数 #1	搜索关键词数 #2	搜索结果个数 #1	搜索结果个数 #2	搜索时间 #1	搜索时间 #2
1	195	765	1	1	4	4	78	82
2	+184	250	1	2	10	1	31	20
3	+180	172	2	2	8	25	47	51
4	+196	109	2	2	8	33	54	65
5	+180	176	1	2	1	10	23	45

图 16: 系统测试结果

或者:

```

java rti.core.RealTimeIndex -ind INDEX_PATH [-int MERGE_INTERVAL (s)] [-ip INDEX_PORT][-sp SEARCH_PORT] [-u]

```

程序启动后，任意时刻输入q [enter] 退出程序。

参数说明:

1. -index/-ind : 索引文件的存储路径，该参数为必须参数
2. -interval/-int : 将内存中的索引合并到硬盘上的时间周期，单位秒，默认1800
3. -indexport/-ip : 索引请求监听端口，默认9081
4. -searchport/-sp : 搜索请求监听端口，默认9091
5. -update/-u : 在现有索引基础上更新，如果没有该选项则删除索引路径下的所有索引，然后重新创建

附加参数:

1. -tpoolsize/-tps : 搜索请求线程池大小, 默认100

示例:

```
java -jar RealTimeIndex.jar -ind E:\backslash$test$\backslash$index -
    int 600
java -jar RealTimeIndex.jar -ind E:\backslash$test$\backslash$index -
    int 300 -u -tps 200
```

接口说明:

(1)索引请求信息的格式为: 1&微博文件存储路径&抓取开始时间&抓取结束时间&抓取时间间隔(单位: 秒)\r\n。采用本地编码即可。数据类型分别为: int&String&long&long&long\r\n。

示例: “1&E:\\weibo\\1334132368160_statuses.json&123456789&123458976&300\r\n”

索引请求不会返回任何信息。

(2)搜索请求信息的格式为: 关键字&开始时间&结束时间&返回结果数目\r\n。采用UTF-8编码。数据类型分别为: String&long&long&int\r\n。如果开始时间、结束时间、返回数目不需要指定的话, 使用-1代替。

示例: “空闲&123456789&145678923&-1\r\n”

搜索请求返回为UTF8编码的json格式字符串, 里面包含符合条件的结果。

WebAPI:

我们的程序原生支持的是socket接口, 但是可以通过一个简单的代理, 将socket转成web方式调用, 该方法在测试的时候使用比较方便。

例如, 一个简单的PHP代理如下:

```
<?php
$srv_port = 9091;

// Create the socket and connect
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
socket_connect($socket, 'localhost', $srv_port);

$request = $_REQUEST["keywords"];
//echo($request."<br>");

// Write request to index server
if(!socket_write($socket, $request."<br>"))
{
```

```

        echo("<p>Write failed</p>");
    }

    // Read response from the server
    while($buffer = socket_read($socket, 1024, PHP_NORMAL_READ))
    {
        if($buffer == "")
        {
            echo("<p>NO DATA</p>");
            break;
        }
        else
        {
            // Do something with the data in the buffer
            echo($buffer);
        }
    }
}
?>

```

简单的测试页面如下；

```

<form method="GET" action="request.php">
    <table>
        <tr>
            <td colspan="3">
                <input name="keywords" id="keywords" type="text" size="40">
                <input id="doSearch" type="submit" value="search">
            </td>
        </tr>
    </table>
</form>

```

6 总结

在本课程设计中，我们小组设计并实现了一个高实时性的新浪微博索引系统，提供了socket和webAPI两种接口，该系统在测试中表现良好，通过与抓取程序的配合能够保证一篇微博一旦抓取到，就立马能够被用户实时检索到。