# 索引库优化

## 什么事索引库

索引库是lucene的重要的存储结构，它包括两部分：原始记录表、词汇表

原始记录表：存放的是原始记录信息，lucene为存入的内容分配一个唯一的编号。

词汇表：存放的是经过分词器拆分出来的词汇和该词汇在原始记录表中的编号。

## 为什么要将索引库进行优化

在默认情况下，向索引库中添加一个Document对象时，索引库自动会添加一个扩展名叫\*.cfs的二进制压缩文件，这样会使得\*.cfs文件增多，同时索引库的容量也不断增加，影响索引库大小。

## 索引库优化方案

1.3.1合并cfs文件,合并后的cfs文件是二进制压缩字符，能解决是的文件大小和数量的问题

indexWriter.addDocument(document);

indexWriter.optimize();

indexWriter.close();

1.3.2使用RAMDirectory，类似于内存索引库，能解决是的读取索引库文件的速度问题, 它能以空换时，提高速度快，但不能持久保存，因此启动时加载硬盘中的索引库到内存中的索引库，退出时将内存中的索引库保存到硬盘中的索引库，且内容不能重复。

Article article = **new** Article(1,"培训","传智是一家Java培训机构");

Document document = LuceneUtil.*javabean2document*(article);

Directory fsDirectory = FSDirectory.*open*(**new** File("E:/indexDBDBDBDBDBDBDBDB"));

Directory ramDirectory = **new** RAMDirectory(fsDirectory);

IndexWriter fsIndexWriter = **new** IndexWriter(fsDirectory,LuceneUtil.*getAnalyzer*(),**true**,LuceneUtil.*getMaxFieldLength*());

IndexWriter ramIndexWriter = **new** IndexWriter(ramDirectory,LuceneUtil.*getAnalyzer*(),LuceneUtil.*getMaxFieldLength*());

ramIndexWriter.addDocument(document);

ramIndexWriter.close();

fsIndexWriter.addIndexesNoOptimize(ramDirectory);

fsIndexWriter.close();

# 分词器

## 什么事分词器

采用一种算法，将中英文本中的字符拆分开来，形成词汇，以待用户输入关健字后搜索

## 分词器工作流程

1. 按分词器拆分出词汇
2. 取出停用词和禁用词
3. 如果有英文，把英文字母转为小写，即搜索部分大小写

**private** **static** **void** testAnalyzer(Analyzer analyzer, String text) **throws** Exception {

System.*out*.println("当前使用的分词器：" + analyzer.getClass());

TokenStream tokenStream = analyzer.tokenStream("content",**new** StringReader(text));

tokenStream.addAttribute(TermAttribute.**class**);

**while** (tokenStream.incrementToken()) {

TermAttribute termAttribute = tokenStream.getAttribute(TermAttribute.**class**);

System.*out*.println(termAttribute.term());

}

}

## 使用第三方IKAnalyzer分词器--------中文首选

需求：过滤掉上面例子中的“说”，“的”，“呀”，且将“传智播客”看成一个整体 关健字

步一：导入IKAnalyzer分词器核心jar包，IKAnalyzer3.2.0Stable.jar

步二：将IKAnalyzer.cfg.xml和stopword.dic和xxx.dic文件复制到MyEclipse的src目录下，

再进行配置，在配置时，首行需要一个空行.

# 搜索结果高亮

# 搜索结果摘要

如果搜索结果内容太多，我们只想显示前几个字符， 必须与高亮一起使用

String keywords = "培训";

List<Article> articleList = **new** ArrayList<Article>();

QueryParser queryParser = **new** QueryParser(LuceneUtil.*getVersion*(),"content",LuceneUtil.*getAnalyzer*());

Query query = queryParser.parse(keywords);

IndexSearcher indexSearcher = **new** IndexSearcher(LuceneUtil.*getDirectory*());

TopDocs topDocs = indexSearcher.search(query,1000000);

Formatter formatter = **new** SimpleHTMLFormatter("<font color='red'>","</font>");

Scorer scorer = **new** QueryScorer(query);

Highlighter highlighter = **new** Highlighter(formatter,scorer);

Fragmenter fragmenter = **new** SimpleFragmenter(4);

highlighter.setTextFragmenter(fragmenter);

**for**(**int** i=0;i<topDocs.scoreDocs.length;i++){

ScoreDoc scoreDoc = topDocs.scoreDocs[i];

**int** no = scoreDoc.doc;

Document document = indexSearcher.doc(no);

String highlighterContent = highlighter.getBestFragment(LuceneUtil.*getAnalyzer*(),"content",document.get("content"));

document.getField("content").setValue(highlighterContent);

Article article = (Article) LuceneUtil.*document2javabean*(document,Article.**class**);

articleList.add(article);

}

**for**(Article article : articleList){

System.*out*.println(article);

}

}

# 搜索结果排序

搜索结果是按某个或某些字段高低排序来显示的结果

## 影响网站排名的先后的有多种

head/meta/

网页的标签整洁

网页执行速度

采用div+css

## lucene的显示结果次序与相关度得分有关

ScoreDoc.score;

默认情况下，Lucene是按相关度得分排序的，得分高排在前，得分低排在后

如果相关度得分相同，按插入索引库的先后次序排序

## Lucene中的设置相关度得分

IndexWriter indexWriter = **new** IndexWriter(LuceneUtil.*getDirectory*(),LuceneUtil.*getAnalyzer*(),LuceneUtil.*getMaxFieldLength*());

document.setBoost(20F);

indexWriter.addDocument(document);

indexWriter.close();

## Lucene中按单个字段排序

Sort sort = new Sort(new SortField("id",SortField.INT,true));

TopDocs topDocs = indexSearcher.search(query,null,1000000,sort);

## Lucene中按多个字段排序

Sort sort = **new** Sort(**new** SortField("count",SortField.*INT*,**true**),**new** SortField("id",SortField.*INT*,**true**));

TopDocs topDocs = indexSearcher.search(query,**null**,1000000,sort);

在多字段排序中，只有第一个字段排序结果相同时，第二个字段排序才有作用提倡用数值型排序

# 条件搜索

6．1什么是条件搜索

用关健字与指定的单列或多例进行匹配的搜索

6．2单字段条件搜索

|  |
| --- |
| QueryParser queryParser = new QueryParser(LuceneUtil.getVersion(),"content",LuceneUtil.getAnalyzer()); |

6．3多字段条件搜索，项目中提倡多字段搜索

QueryParser queryParser = new MultiFieldQueryParser(LuceneUtil.getVersion(),new String[]{"content","title"},LuceneUtil.getAnalyzer());