SOLR全文检索服务

# 什么是全文检索

全文检索：将非结构化数据中的一部分信息提取出来，重新组织使其变得有一定结构，然后对此有一定结构的数据进行检索，从而达到检索速度相对较快的目的，这部分从非结构化数据中提取出来的然后重新组织的信息，我们称之为索引。

例如：字典，字典里的拼音和部首就是索引，对每个字的解释是非结构化，如果没有索引就必须顺序检索（即一个一个字的找）。

这种先键索引，然后根据索引进行搜索的过程就是全文检索 。

# 为什么要用全文检索

结构化数据：指具有固定格式或有限长度的数据，如数据库，元数据等。

非结构化数据：指不定长或无固定格式的数据，如邮件，word文档等。

传统的数据库查询（通过关键字查询效率低下）而且需要做大量的工作，全文检索则可以对非结构化的数据进行很好实现还价案子查询，具有很高的效率。

# 企业站内搜索技术选型

在一些大型的门户网站或者电子商务网站都需要站内搜索功能，使用传统的数据库查询方式无法满足一些高级的搜索功能 ，比如：搜索速度要快、搜索结果按相关度排序、搜索内容格式不固定等。

* 1. 单独使用Lucene

单独使用Lucene实现站内搜索需要开发的工作量较大，主要表现在:索引维护，索引性能优化，搜索性能优化。

* 1. 使用百度或Google第三方接口

通过第三方接口实现站内搜索，这样和第三方引擎系统紧密，不方便扩展。

* 1. 使用Solr实现

基于solr实现站内搜索扩展性较好并且减少程序员的工作量，因为Solr提供了较为完备的搜索引擎解决方案，因此在文户，论坛等系统中常用此方案。

# Lucene简介

Lucene是简单而功能强大的基于Java的搜索库（工具包）。它可以用于任何应用程序来搜索功能。 Lucene是开源项目。它是可扩展的，高性能的库用于索引和搜索几乎任何类型的文本。 Lucene库提供了所需的任何搜索应用程序的核心业务。索引和搜索。

# Lucene实现全文检索的流程

* 1. 索引和搜索流程图

创建索引

查询索引

1.获取文档（io）

1.用户查询接口

索引库

原始文档

返回结果

2.创建查询

2.构建文件对象

4.渲染结果

查询索引

3.分析文档

3.执行查询

4.创建索引

* 1. 创建索引理论

5.2.1 获取原始文档

互联网上的页面（爬虫或蜘蛛,也称网络机器人），数据库中的数据，磁盘上的文件。

5.2.2 创建文档对象

文档（Document）

Filed（域）

Name：file\_path（文件路径）

Value：e://

Filed（域）

Name：file\_name（文件名称）

Value：SpringMVC.TXT

Filed（域）

其他：

不同的文档有 不同的域，同一个文档可以有相同的域

5.2.3 分析文档

将原始内容创建为包含域（Field）的文档（Document），需要对域中的内容进行分，析，分析的过程是经过对原始文档提取单词，将字母转化为小写，去掉标点符号，去掉停用词等过程生成最终的词汇单元，可以将词汇单元理解为一个一个的单词。

例如：Lucene is a Java full-text search engine

分析得到 lucene java full search engine

每个单词叫做一个term，不同域中拆分出来的相同的单词是不同的term，term包含两部分内容，一部分是文档的域名，一部分是单词的内容。Term（key（文件名称）， value（Spring）） term k 域 （文件内容） value（spring）。

5.2.4创建索引

* 1. 创建索引

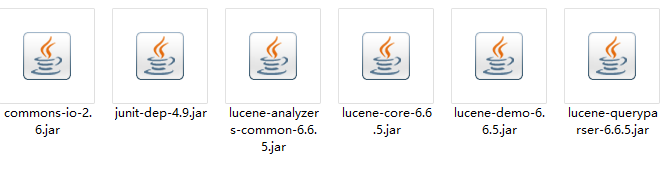
5.3.1 配置开发环境

下载Lucene：官方网站：<http://lucene.apache.org/>

版本：lucene6.6.5。

Jdk：要求1.7版本以上。

5.3.2使用的jar包

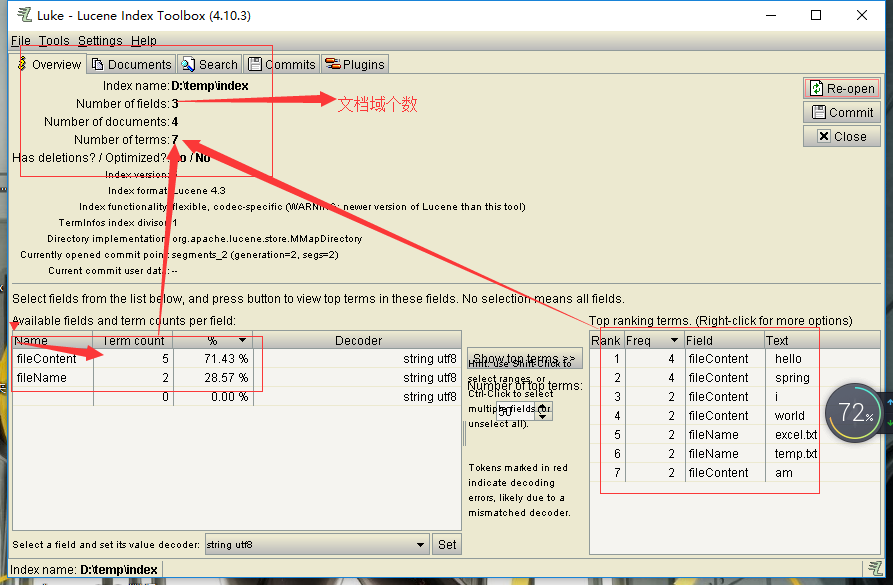


5.3.3 创建索引的过程

|  |
| --- |
| package com.nanhua.study712;  import java.io.File;  import java.nio.file.Paths;  import org.apache.lucene.analysis.Analyzer;  import org.apache.lucene.analysis.standard.StandardAnalyzer;  import org.apache.lucene.document.Document;  import org.apache.lucene.document.Field;  import org.apache.lucene.document.TextField;  import org.apache.lucene.document.Field.Store;  import org.apache.lucene.document.StoredField;  import org.apache.lucene.index.IndexWriter;  import org.apache.lucene.index.IndexWriterConfig;  import org.apache.lucene.store.Directory;  import org.apache.lucene.store.FSDirectory;  import org.junit.Test;  /\*\*  \* lucene创建索引  \*  \* @author 汪进银  \*  \*/  public class TestLucene {  @Test  public void CreateIndex() throws Exception {  // 新建索引库  File path = new File("D://temp//index");  Directory directory = FSDirectory.open(Paths.get("D://temp//index"));  // 新建分析器对象  Analyzer analyzer = new StandardAnalyzer();  // 新建配置对象  IndexWriterConfig config = new IndexWriterConfig(analyzer);  // 创建一个IndexWriter对象（参数一个索引库，一个配置）  IndexWriter indexWriter = new IndexWriter(directory, config);  // 创建一个文档  File f = new File("D:\\index");  File[] list = f.listFiles();  for (File file : list) {  // 创建一个文档对象  Document document = new Document();  // 文件名称  String file\_name = file.getName();  Field fileNameField = new TextField("fileName", file\_name, Store.YES);  // 文件路径 StoredField重载方法支持多种类型  String file\_path = file.getPath();  Field filePathField = new StoredField("filePath", file\_path);  // 文件内容  // String file\_content = FileUtils.readFileToString(file);  // Field fileContentField = new TextField("fileContent", file\_content,  // Store.YES);  // 添加到document  document.add(fileNameField);  document.add(filePathField);  // document.add(fileContentField);  // 将创建的文档写进索引库  indexWriter.addDocument(document);  }  indexWriter.close();  } |

5.3.4 利用luke查看索引库（下载相应版本的luke）

Luke gitHub下载地址（https://github.com/DmitryKey/luke/releases）



* 1. 查询索引

5.4.1 用户查询接口



Lucene不提供制作用户搜索界面的用户，需要根据自己的需求开发搜索界面

5.4.2创建查询

用户输入查询关键字执行搜索之前需要先构建一个查询对象，查询对象指定查询的field文档域，查询关键字等，查询对象会生成基本的查询语法。

例如：filedName：lucene 表示要搜索的是field域的内容为Lucene的文档。

5.2.3执行查询





5.2.4 创建查询索引过程

|  |
| --- |
| @Test  public void testSearch() throws Exception {  // 1.创建一个Directory对象，指向索引库  Directory directory = FSDirectory.open(Paths.get("D://temp//index"));  // 2、创建IndexReader 需要指定Directory对象  DirectoryReader reader = DirectoryReader.open(directory);  // 3、根据IndexWriter创建IndexSearcher需要指定reader  IndexSearcher indexSearcher = new IndexSearcher(reader);  // 4.创建一个termQuery对象，指定查询的域和查询的关键字  Query query = new TermQuery(new Term("fileContent", "spring"));  // 5.执行查询 TopDocs返回有两个文档  TopDocs topDocs = indexSearcher.search(query, 2);  // 6.返回查询结果，遍历查询结果  ScoreDoc[] scoreDoc = topDocs.scoreDocs;  for (ScoreDoc score : scoreDoc) {  int i = score.doc;  Document document = indexSearcher.doc(i);  String name = document.get("fileName ");  String path = document.get("filePath");  String content = document.get("fileContent");  System.out.println("文件名:" + name);  System.out.println("文件路径:" + path);  System.out.println("文件内容:" + content);  }  // 7.关闭reader  reader.close();  }  } |

* 1. 中文分词

5.5.1 常见的中文分词器

自带的中文分词器

A．StandarAnalyzer 单字分词，一个字一个字地进行分词 比如“我爱中国” “我”，“爱”，“中”，“国”。

B．CJKAnalyzer 二词分法 如：“我是中国人” “我是” “是中” “中国”

C. SmartChinessesAnalyzer 对中文支持较好，但扩展性查。

第三方的的中文分词器IK-Analyzer

下载地址：http://code.google.com/p/ik-analyzer/

5.5.2导入架包

<!--中文分词器 -->

<dependency>

<groupId>org.apache.lucene</groupId>

<artifactId>lucene-analyzers-smartcn</artifactId>

<version>4.10.3</version>

</dependency>

5.5.3实现（Demo）

|  |
| --- |
| package com.nanhua.study713;  import java.io.File;  import java.io.StringReader;  import org.apache.lucene.analysis.TokenStream;  import org.apache.lucene.analysis.cn.smart.SmartChineseAnalyzer;  import org.apache.lucene.document.Document;  import org.apache.lucene.document.Field;  import org.apache.lucene.document.IntField;  import org.apache.lucene.document.StringField;  import org.apache.lucene.document.TextField;  import org.apache.lucene.index.DirectoryReader;  import org.apache.lucene.index.IndexReader;  import org.apache.lucene.index.IndexWriter;  import org.apache.lucene.index.IndexWriterConfig;  import org.apache.lucene.queryparser.classic.QueryParser;  import org.apache.lucene.search.IndexSearcher;  import org.apache.lucene.search.Query;  import org.apache.lucene.search.ScoreDoc;  import org.apache.lucene.search.TopDocs;  import org.apache.lucene.search.highlight.Fragmenter;  import org.apache.lucene.search.highlight.Highlighter;  import org.apache.lucene.search.highlight.QueryScorer;  import org.apache.lucene.search.highlight.SimpleHTMLFormatter;  import org.apache.lucene.search.highlight.SimpleSpanFragmenter;  import org.apache.lucene.store.Directory;  import org.apache.lucene.store.FSDirectory;  import org.apache.lucene.util.Version;  import org.junit.Test;  public class TestLucene {  @Test  public void getWriter() throws Exception {  Integer ids[] = { 1, 2, 3 };  String citys[] = { "青岛", "南京", "上海" };  String descs[] = { "青岛是一个美丽的城市。",  "南京是一个有文化的城市。南京是一个文化的城市南京，简称宁，是江苏省会，地处中国东部地区，长江下游，濒江近海。全市下辖11个区，总面积6597平方公里，2013年建成区面积752.83平方公里，常住人口818.78万，其中城镇人口659.1万人。[1-4] “江南佳丽地，金陵帝王州”，南京拥有着6000多年文明史、近2600年建城史和近500年的建都史，是中国四大古都之一，有“六朝古都”、“十朝都会”之称，是中华文明的重要发祥地，历史上曾数次庇佑华夏之正朔，长期是中国南方的政治、经济、文化中心，拥有厚重的文化底蕴和丰富的历史遗存。[5-7] 南京是国家重要的科教中心，自古以来就是一座崇文重教的城市，有“天下文枢”、“东南第一学”的美誉。截至2013年，南京有高等院校75所，其中211高校8所，仅次于北京上海；国家重点实验室25所、国家重点学科169个、两院院士83人，均居中国第三。[8-10] 。",  "上海是一个繁华的城市。" };  // Analyzer analyzer=new StandardAnalyzer(); // 标准分词器  SmartChineseAnalyzer analyzer = new SmartChineseAnalyzer();  IndexWriterConfig iwc = new IndexWriterConfig(Version.LATEST, analyzer);  Directory dir = FSDirectory.open(new File("D://index2"));  IndexWriter writer = new IndexWriter(dir, iwc);  for (int i = 0; i < ids.length; i++) {  Document doc = new Document();  doc.add(new IntField("id", ids[i], Field.Store.YES));  doc.add(new StringField("city", citys[i], Field.Store.YES));  doc.add(new TextField("desc", descs[i], Field.Store.YES));  writer.addDocument(doc); // 添加文档  }  writer.close();  }    @Test  public void search() throws Exception {    Directory dir = FSDirectory.open(new File("D://index2"));  IndexReader reader = DirectoryReader.open(dir);  IndexSearcher is = new IndexSearcher(reader);  // 中文分词器  SmartChineseAnalyzer analyzer = new SmartChineseAnalyzer();  QueryParser parser = new QueryParser("desc", analyzer);  Query query = parser.parse("青岛");  long start = System.currentTimeMillis();  TopDocs hits = is.search(query, 10);  long end = System.currentTimeMillis();  System.out.println("匹配 " + "南京文明" + " ，总共花费" + (end - start) + "毫秒" + "查询到" + hits.totalHits + "个记录");  QueryScorer scorer = new QueryScorer(query);// 获得  Fragmenter fragmenter = new SimpleSpanFragmenter(scorer);  SimpleHTMLFormatter simpleHTMLFormatter = new SimpleHTMLFormatter("<b><font color='red'>", "</font></b>");  Highlighter highlighter = new Highlighter(simpleHTMLFormatter, scorer);  highlighter.setTextFragmenter(fragmenter);    for (ScoreDoc scoreDoc : hits.scoreDocs) {  Document doc = is.doc(scoreDoc.doc);  System.out.println(doc.get("city"));  System.out.println(doc.get("desc"));  String desc = doc.get("desc");  if (desc != null) {  TokenStream tokenStream = analyzer.tokenStream("desc", new StringReader(desc));  System.out.println(highlighter.getBestFragment(tokenStream, desc));  }  }  reader.close();  }  } |

* 1. 索引的管理（增删查改）

5.6.1增加

|  |
| --- |
| @Test  public void CreateIndex() throws Exception {    // 新建索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 新建分析器对象  Analyzer analyzer = new StandardAnalyzer();  // 新建配置对象  IndexWriterConfig config = new IndexWriterConfig(Version.LATEST,analyzer);  // 创建一个IndexWriter对象（参数一个索引库，一个配置）  IndexWriter indexWriter = new IndexWriter(directory, config);  // 创建一个文档  File f = new File("D:\\index");  File[] list = f.listFiles();  for (File file : list) {  // 创建一个文档对象  Document document = new Document();  // 文件名称  String file\_name = file.getName();  Field fileNameField = new TextField("fileName", file\_name, Store.YES);  // 文件路径 StoredField重载方法支持多种类型  String file\_path = file.getPath();  Field filePathField = new StoredField("filePath", file\_path);  // 文件内容  String file\_content = FileUtils.readFileToString(file);  Field fileContentField = new TextField("fileContent", file\_content, Store.YES);  // 添加到document  document.add(fileNameField);  document.add(filePathField);  document.add(fileContentField);  // 将创建的文档写进索引库  indexWriter.addDocument(document);  }  indexWriter.close();  } |

5.6.2删除索引

|  |
| --- |
| @Test  public void deleteAll() throws Exception{  // 新建索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 新建分析器对象  Analyzer analyzer = new StandardAnalyzer();  // 新建配置对象  IndexWriterConfig config = new IndexWriterConfig(Version.LATEST,analyzer);  // 创建一个IndexWriter对象（参数一个索引库，一个配置）  IndexWriter indexWriter = new IndexWriter(directory, config);  indexWriter.deleteAll(); //删除所有  indexWriter.close();  }    /\*\*  \* 根据条件删除  \* @throws Exception  \*/  @Test  public void delete() throws Exception{  // 新建索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 新建分析器对象  Analyzer analyzer = new StandardAnalyzer();  // 新建配置对象  IndexWriterConfig config = new IndexWriterConfig(Version.LATEST,analyzer);  // 创建一个IndexWriter对象（参数一个索引库，一个配置）  IndexWriter indexWriter = new IndexWriter(directory, config);  Query query = new TermQuery(new Term("fileContent", "hello"));  indexWriter.deleteDocuments(query); //根据条件删除  indexWriter.close();  } |

5.6.3修改索引

|  |
| --- |
| @Test  public void Update() throws Exception {  // 新建索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 新建分析器对象  Analyzer analyzer = new StandardAnalyzer();  // 新建配置对象  IndexWriterConfig config = new IndexWriterConfig(Version.LATEST, analyzer);  // 创建一个IndexWriter对象（参数一个索引库，一个配置）  IndexWriter indexWriter = new IndexWriter(directory, config);  Document doc = new Document();  doc.add(new TextField("fileN", "测试文件名", Store.YES));  doc.add(new TextField("fileC", "测试文件内容", Store.YES));  indexWriter.updateDocument(new Term("fileContent", "hello"), doc, analyzer);  indexWriter.close();  } |

5.6.4 查询索引（重点）

A.匹配所有

|  |
| --- |
| @Test  public void testMatchAllDocsQuery() throws Exception {  // 1.创建一个Directory对象，指向索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 2、创建IndexReader 需要指定Directory对象  DirectoryReader reader = DirectoryReader.open(directory);  // 3、根据IndexWriter创建IndexSearcher需要指定reader  IndexSearcher indexSearcher = new IndexSearcher(reader);  Query query = new MatchAllDocsQuery();  TopDocs topDocs= indexSearcher.search(query,10);    ScoreDoc[] scoreDoc = topDocs.scoreDocs;  for (ScoreDoc score : scoreDoc) {  int i = score.doc;  Document document = indexSearcher.doc(i);  String name = document.get("fileName");  String path = document.get("filePath");  String content = document.get("fileContent");    System.out.println("文件名:" + name);  System.out.println("文件路径:" + path);  System.out.println("文件内容:" + content);  }  // 7.关闭reader  reader.close();  } |

B.Term查询（在4.3中有）

C.根据数值范围查（实现价格按区间查询）

Query query = NumericRangeQuery.newIntRange("fileSize", 100, 200, true, true);

是否包含最小值

最小值

最大值

域名

是否包含最大值

|  |
| --- |
| @Test  public void testNumbericRangeQuery() throws Exception{  // 1.创建一个Directory对象，指向索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 2、创建IndexReader 需要指定Directory对象  DirectoryReader reader = DirectoryReader.open(directory);  // 3、根据IndexWriter创建IndexSearcher需要指定reader  IndexSearcher indexSearcher = new IndexSearcher(reader);    Query query = NumericRangeQuery.newIntRange("fileSize", 100, 200, true, true);  TopDocs topDocs = indexSearcher.search(query, 10);  ScoreDoc[] scoreDoc = topDocs.scoreDocs;  for (ScoreDoc score : scoreDoc) {  int i = score.doc;  Document document = indexSearcher.doc(i);  String name = document.get("fileName");  String path = document.get("filePath");  String content = document.get("fileContent");  System.out.println("文件名:" + name);  System.out.println("文件路径:" + path);  System.out.println("文件内容:" + content);  }  // 7.关闭reader  reader.close();  } |

D．BooleanQuery组合关系查询

BooleanQuery booleanQuery = new BooleanQuery();

Query query1 = new TermQuery(new Term("fileContent", "hello"));

Query query2 = new TermQuery(new Term("fileContent", "i"));

booleanQuery.add(query1, Occur.MUST);

booleanQuery.add(query2, Occur.MUST);

组合关系代表的意思如下:   
     MUST和MUST表示“与”的关系，即“并集”。   
     MUST和MUST\_NOT前者包含后者不包含。   
     MUST\_NOT和MUST\_NOT没意义   
     SHOULD与MUST表示MUST，SHOULD失去意义；   
     SHOUlD与MUST\_NOT相当于MUST与MUST\_NOT。   
     SHOULD与SHOULD表示“或”的概念。

|  |
| --- |
| @Test  public void testBooleanQuery() throws Exception{  // 1.创建一个Directory对象，指向索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 2、创建IndexReader 需要指定Directory对象  DirectoryReader reader = DirectoryReader.open(directory);  // 3、根据IndexWriter创建IndexSearcher需要指定reader  IndexSearcher indexSearcher = new IndexSearcher(reader);    BooleanQuery booleanQuery = new BooleanQuery();  Query query1 = new TermQuery(new Term("fileContent", "hello"));  Query query2 = new TermQuery(new Term("fileContent", "i"));    booleanQuery.add(query1, Occur.MUST);  booleanQuery.add(query2, Occur.MUST);    TopDocs topDocs = indexSearcher.search(booleanQuery, 10);  ScoreDoc[] scoreDoc = topDocs.scoreDocs;  for (ScoreDoc score : scoreDoc) {  int i = score.doc;  Document document = indexSearcher.doc(i);  String name = document.get("fileName");  String path = document.get("filePath");  String content = document.get("fileContent");  System.out.println("文件名:" + name);  System.out.println("文件路径:" + path);  System.out.println("文件内容:" + content);  }  // 7.关闭reader  reader.close();  } |

E．解析查询

|  |
| --- |
| public void testQueryParse() throws Exception{  // 1.创建一个Directory对象，指向索引库  Directory directory = FSDirectory.open(new File("D://temp//index"));  // 2、创建IndexReader 需要指定Directory对象  DirectoryReader reader = DirectoryReader.open(directory);  // 3、根据IndexWriter创建IndexSearcher需要指定reader  IndexSearcher indexSearcher = new IndexSearcher(reader);  //参数1，默认查询的域  QueryParser queryParse = new QueryParser("fileContent", new StandardAnalyzer() );  //"\*:\*" 域：值  Query query = queryParse.parse("hello");      TopDocs topDocs = indexSearcher.search(query, 10);  ScoreDoc[] scoreDoc = topDocs.scoreDocs;  for (ScoreDoc score : scoreDoc) {  int i = score.doc;  Document document = indexSearcher.doc(i);  String name = document.get("fileName");  String path = document.get("filePath");  String content = document.get("fileContent");  System.out.println("文件名:" + name);  System.out.println("文件路径:" + path);  System.out.println("文件内容:" + content);  }  // 7.关闭reader  reader.close();  } |

# 什么是Solr

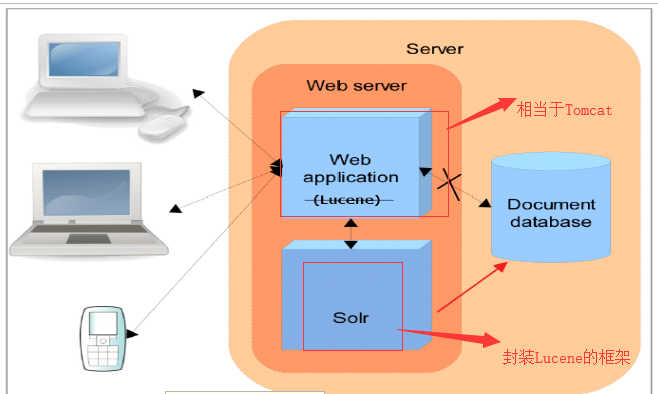
* 1. 什么是Solr

Solr 是Apache下的一个顶级开源项目，采用Java开发，它是基于Lucene的全文搜索服务器。Solr提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展，并对索引、搜索性能进行了优化。Solr可以独立运行，运行在Jetty、Tomcat等这些Servlet容器中，Solr 索引的实现方法很简单，用 POST 方法向 Solr 服务器发送一个描述 Field 及其内容的 XML 文档，Solr根据xml文档添加、删除、更新索引 。Solr 搜索只需要发送 HTTP GET 请求，然后对 Solr 返回Xml、json等格式的查询结果进行解析，组织页面布局。Solr不提供构建UI的功能，Solr提供了一个管理界面，通过管理界面可以查询Solr的配置和运行情况。

* 1. Solr与Lucene的区别

Lucene是一个开放源代码的全文检索引擎工具包,它不是一个完整的全文检索引擎Lucene提供了完整的查询引擎和索引引擎，目的是为软件工程人员提供一个简单易用的工具包，以方便在系统中实现全文检索的功能，或者以Lucene为基础构建全文检索引擎。

Solr的目标是打造一款企业级的搜索引擎系统，它是一个搜索引擎服务，可以独立运行，通过Solr可以非常迅速的构建企业的搜索引擎，通过solr也可以高效的完成站内搜索功能。



# 安装Solr与环境配置

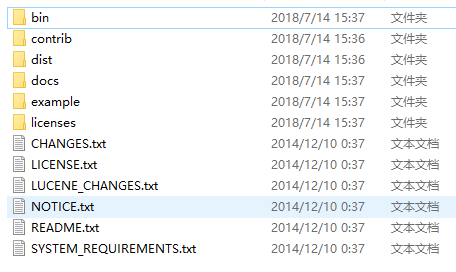
* 1. 下载地址

7.1.1<http://mirror.bit.edu.cn/apache/lucene/solr>（官网下载<http://archive.apache.org/dist/lucene/solr/>（历史版本）

Wondows：solr-7.0.4.zip

Linux：solr-7.0.4.tgz

7.1.2 解压



bin:solr的运行脚本

contrib.：solr的一些贡献软件/插件，用于增强solr的功能

dist:包含builde过程产生的war或jar文件以及相关的依赖文件

docs:solr的api文档

example：solr的例子工程

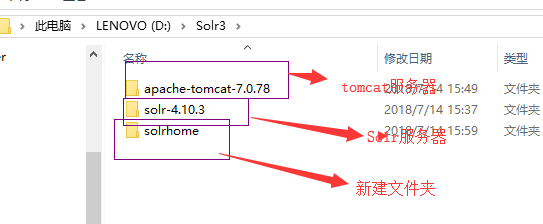
licenese：solr的一些许可信息

* 1. 运行环境

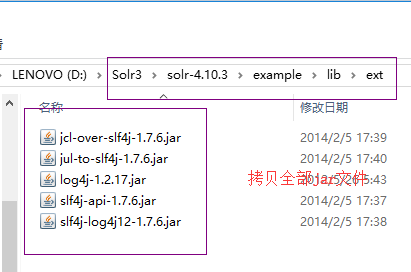
Solr需要运行在一个Servlet容器中，Solr4.10.3要求1.7以上，Solr默认提供Jetty（java写的Servlet容器）。

* 1. Solr与Tomcat整合配置

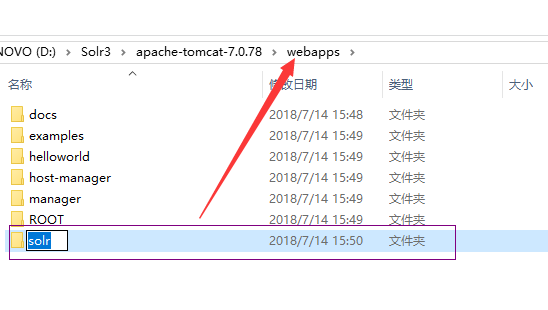
7.3.1准备 在d://创建solr文件夹 将tomcat服务器 和 solr拷贝到该目录下



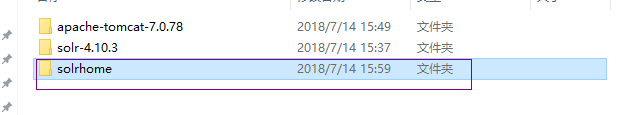
7.3.2 solr-4.10.3\example\lib\ext目录下全部jar包拷贝到tomcat安装或解压文件夹下的lib目录



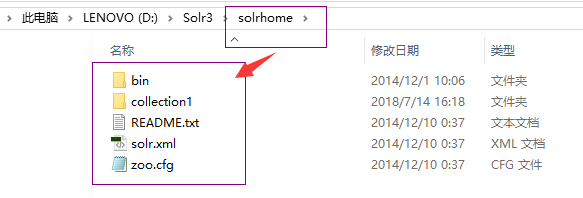
7.3.3拷贝solr-4.40.3\example\webapps目录下的solr.war项目包，到tomcat安装或解压文件夹下的webapps目**录**



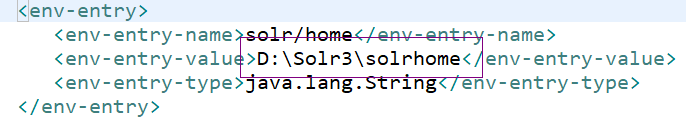
7.3.4**创建solr工作目录（索引文件，配置文件位置目录）**

****

**7.3.5拷solr-4.10.3\example\solr目录下的所有文件和文件夹到D:\Solr3\solrhome工作目录下**

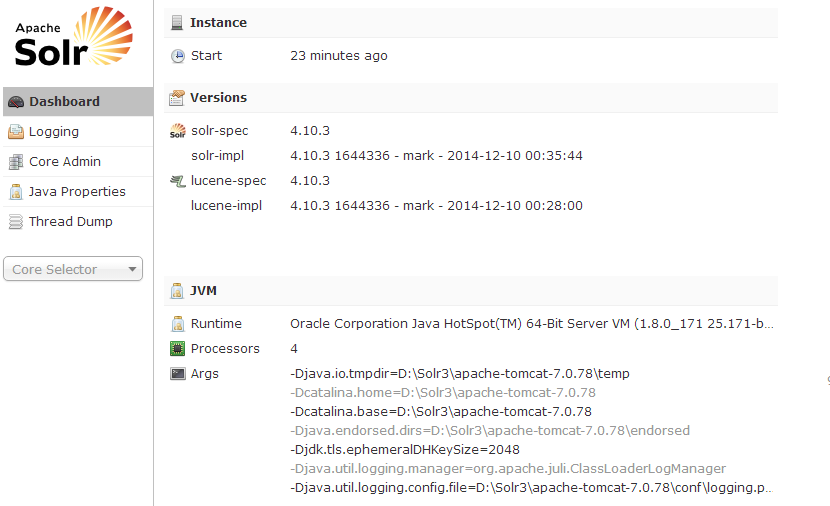
****

**7.3.6 进入tomcat安装或解压文件夹下的webapps\solr\WEB-INF目录，打开web.xml文件添加一下内容**

****

**7.3.7启动tomcat 在浏览器中输入**<http://localhost:8989/solr/#/>

**如成功：**

****

# 配置中文分词器

* 1. 安装中文分词器

使用IKAnalyzer分词器

第一步：将IKAnalyzer2012\_u6.jar添加到solr/WEB-INFO/lib下

第二步：复制IKAnalyzer2012的配置文件和自定义的词典和停用词典到solr的classpath下

第三部：在scheam.xml中添加fieldType，使用中文分析器

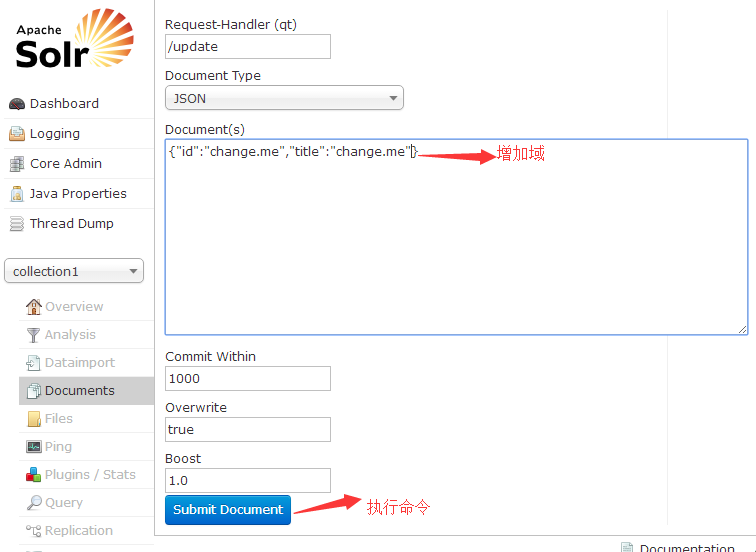
|  |
| --- |
| <fieldType name="text\_ik" class="solr.TextField"  <analyzer class=”org.wltea.analyzer.lucene.IKAnalyzer” />  </fieldType > |

第四部：

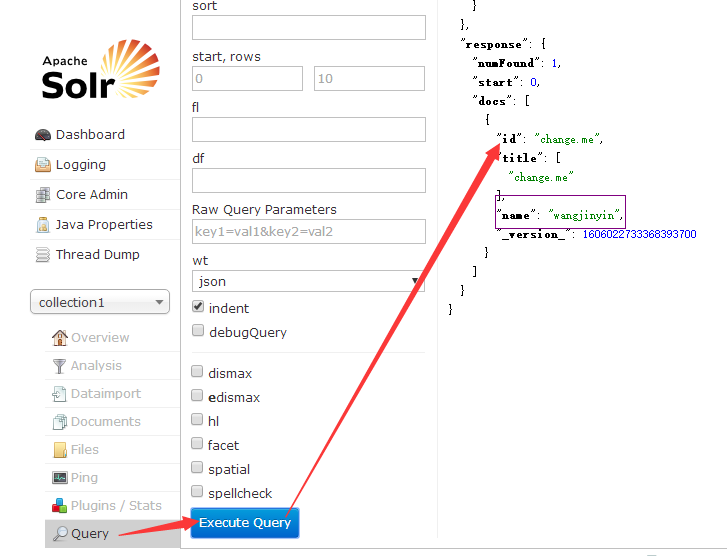
|  |
| --- |
| <field name="title\_ik" type="text\_ik" indexed="true" stored="true"></field>  <field name="content\_ik" type="text\_ik" indexed="true" stored="false" multiValued="true">  </field> |

# Solr管理索引库

* 1. 增加域对象



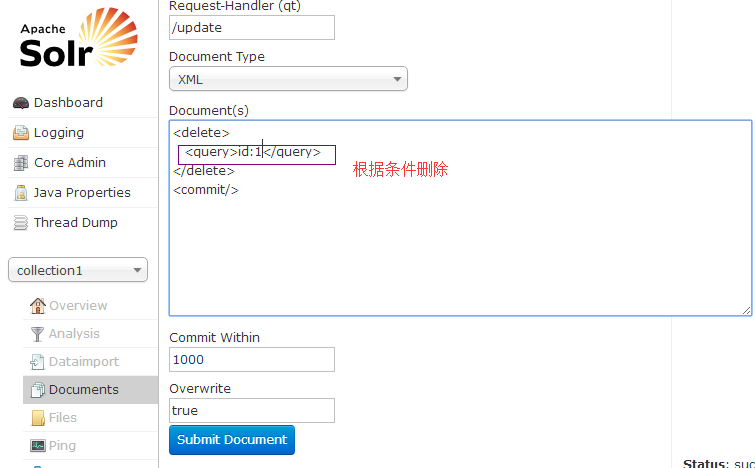
* 1. 查询域

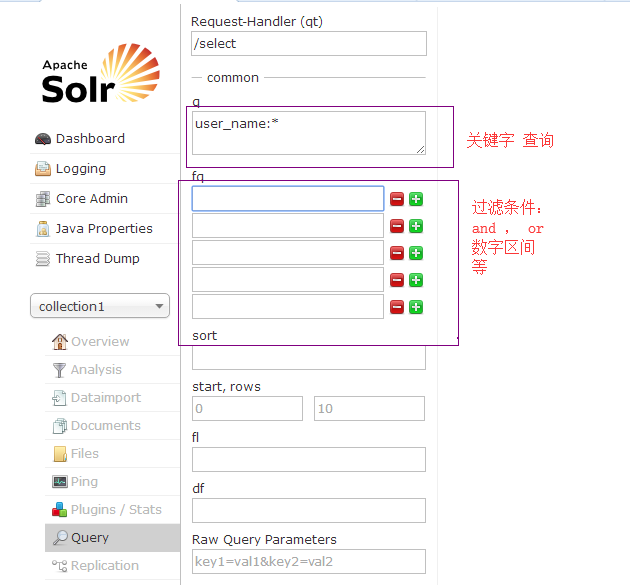


* 1. 删除域

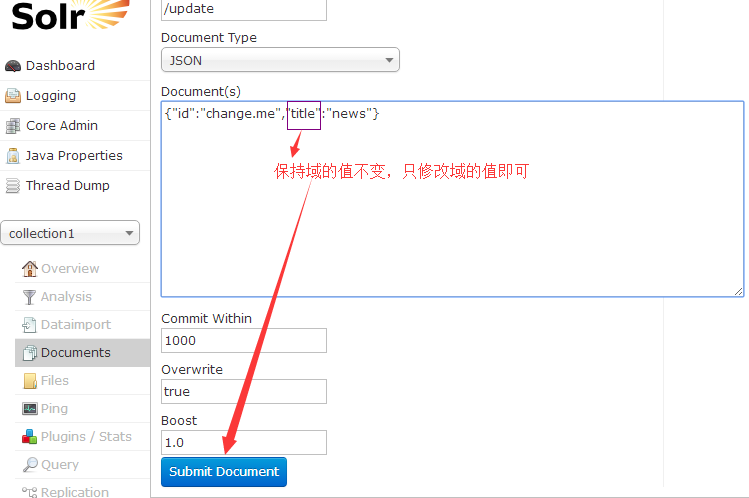


根据条件删除





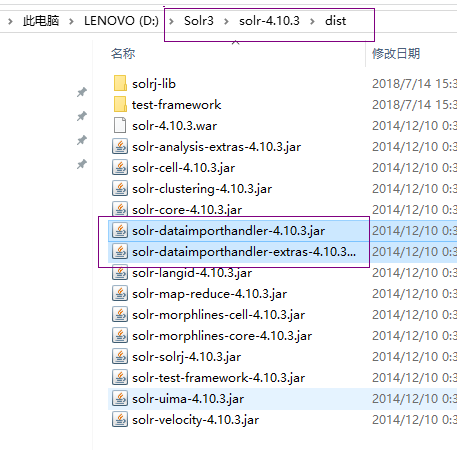
* 1. 修改



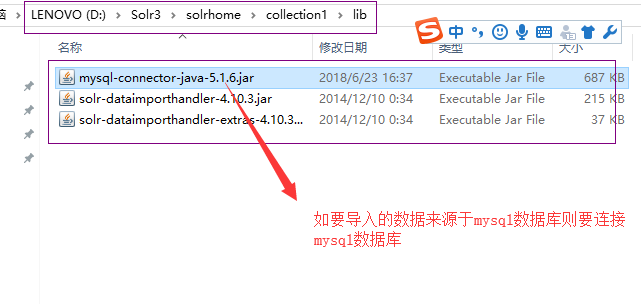
# 批量导入处理器

使用dataImport插件实现批量导入数据

第一步:把dataImport所依赖的jar包添加collection1的lib中



如果collection1的lib中没有lib则手动创建



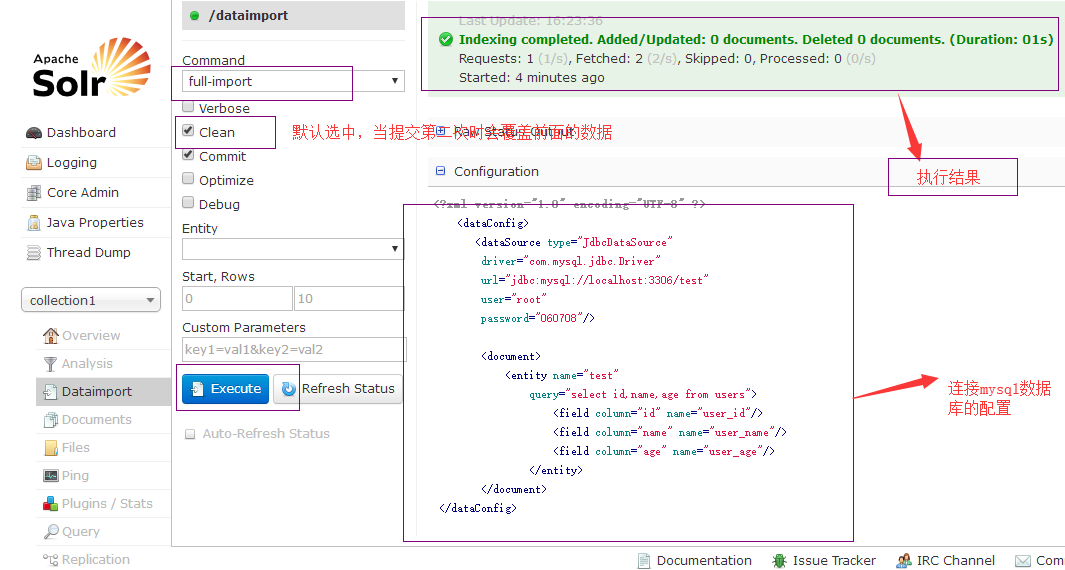
第二步：配置solrconfig.xml文件添加一个requestHandler

|  |
| --- |
| <requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">  <lst name="defaults">  <str name="config">data-config.xml</str>  </lst>  </requestHandler> |

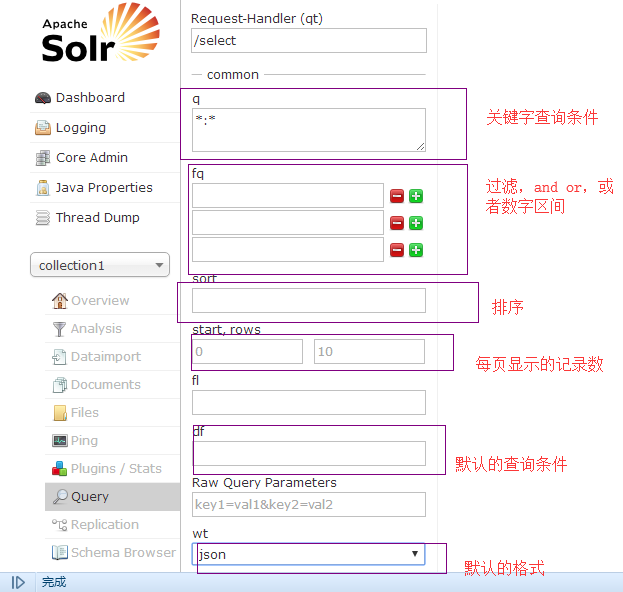
第三步：数据库的配置信息

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <dataConfig>  <dataSource type="JdbcDataSource"  driver="com.mysql.jdbc.Driver"  url="jdbc:mysql://localhost:3306/test"  user="root"  password="060708"/>    <document>  <entity name="test"  query="select id,name,age from users">      <field column="User\_id" name="id"/>      <field column=" User\_name" name="name"/>      <field column=" User\_age " name="age"/>  </entity>  </document>  </dataConfig> |

第四步：重启tomcat服务 执行导入



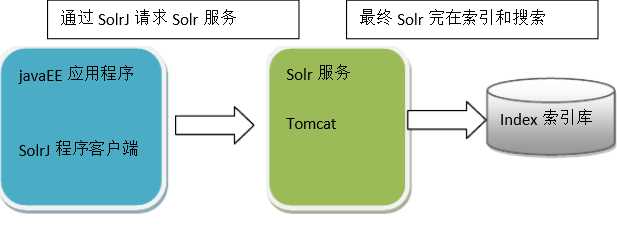
查询界面说明



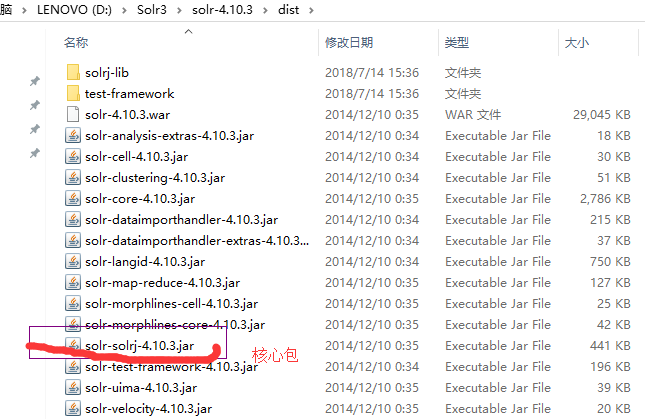
# 使用solrJ管理索引库

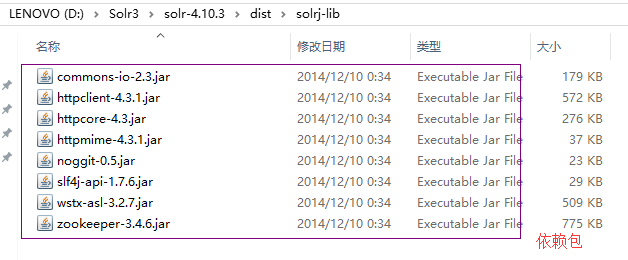
* 1. 什么是solrJ

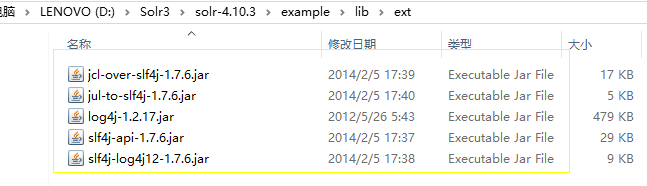
SolrJ是访问solr服务器的客户端，提供索引和搜索的请求方法，Solrj通常嵌入在业务系统中，solr通过solr的API接口操作服务。



* 1. 依赖的jar包



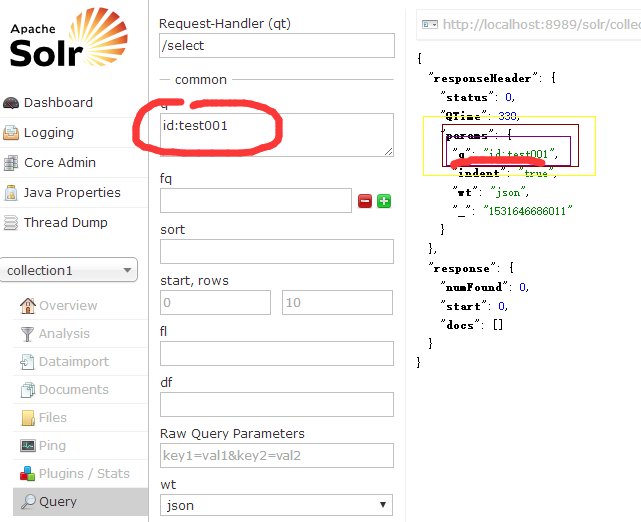




* 1. 增加索引

|  |
| --- |
| package com.nanhua.study715;  import org.apache.solr.client.solrj.SolrServer;  import org.apache.solr.client.solrj.impl.HttpSolrServer;  import org.apache.solr.common.SolrInputDocument;  import org.junit.Test;  /\*\*  \* solrj管理 增加 删除 修改 查询  \*  \* @author 汪进银  \*  \*/  public class solrManage {  @Test  public void testAdd() throws Exception{  // 第一步：把solrJ的jar包添加到工程中。  // 第二步：创建一个SolrServer，使用HttpSolrServer创建对象。 单机版  SolrServer solrServer = new HttpSolrServer("http://localhost:8989/solr");  // 第三步：创建一个文档对象SolrInputDocument对象。  SolrInputDocument document = new SolrInputDocument();  // 第四步：向文档中添加域。必须有id域，域的名称必须在schema.xml中定义。  document.addField("id", "test001"); //必须得有这个域  document.addField("item\_title", "测试商品");  // 第五步：把文档添加到索引库中。  solrServer.add(document);  // 第六步：提交。  solrServer.commit();  }  } |

结果：



* 1. 删除

|  |
| --- |
| @Test  public void deleteDocumentById() throws Exception {  // 第一步：创建一个SolrServer对象。  SolrServer solrServer = new HttpSolrServer("http://localhost:8989/solr");  // 第二步：调用SolrServer对象的根据id删除的方法。  solrServer.deleteById("1");  // 第三步：提交。  solrServer.commit();  }  @Test  public void deleteDocumentByQuery() throws Exception {  SolrServer solrServer = new HttpSolrServer("http://localhost:8989/solr");  //根据查询删除  solrServer.deleteByQuery("title:change.me");  solrServer.commit();  } |

* 1. 更新

与添加一致只要id相同就是更新

* 1. 查询 （几乎包含所有的查询）

|  |
| --- |
| @Test  public void queryDocumentWithHighLighting() throws Exception {  // 第一步：创建一个SolrServer对象  SolrServer solrServer = new HttpSolrServer("http://localhost:8989/solr");  // 第二步：创建一个SolrQuery对象。  SolrQuery query = new SolrQuery();  // 第三步：向SolrQuery中添加查询条件、过滤条件。。。  query.setQuery("id:test001");  // 指定默认搜索域  query.set("q", "\*:\*");  //过滤条件  query.set("fq", "\*:\*");  //设置排序  query.addSort("price", ORDER.desc);  //分页  query.setStart(0);  query.setRows(5);  //设置默认域  query.set("df","");  //设置显示的域  query.set("fl","");    // 需要高亮时，可开启高亮显示  query.setHighlight(true);  // 高亮显示的域  query.addHighlightField("item\_title");  //高亮前缀  query.setHighlightSimplePre("<em>");  //高亮后缀  query.setHighlightSimplePost("</em>");  // 第四步：执行查询。得到一个Response对象。  QueryResponse response = solrServer.query(query);  // 第五步：取查询结果。  SolrDocumentList solrDocumentList = response.getResults();  System.out.println("查询结果的总记录数：" + solrDocumentList.getNumFound());  // 第六步：遍历结果并打印。  for (SolrDocument solrDocument : solrDocumentList) {  System.out.println(solrDocument.get("id"));  // 取高亮显示  Map<String, Map<String, List<String>>> highlighting = response.getHighlighting();  List<String> list = highlighting.get(solrDocument.get("id")).get("item\_title");  String itemTitle = null;  if (list != null && list.size() > 0) {  itemTitle = list.get(0);  } else {  itemTitle = (String) solrDocument.get("item\_title");  }  System.out.println(itemTitle);  System.out.println(solrDocument.get("item\_price"));  }  } |

# SolrCloud详解及环境配置

* 1. 什么是Solrcloud

SolrCloud(solr 云)是Solr提供的分布式搜索方案，当你需要大规模，容错，分布式索引和检索能力时使用SolrCloud。当一个系统的索引数据量少的时候是不需要使用SolrCloud的，当索引量很大，搜索请求并发很高，这时需要使用SolrCloud来满足这些需求。

SolrCloud是基于Solr和Zookeeper的分布式搜索方案，它的主要思想是使用Zookeeper作为集群的配置信息中心。

它有几个特色功能：

1）集中式的配置信息

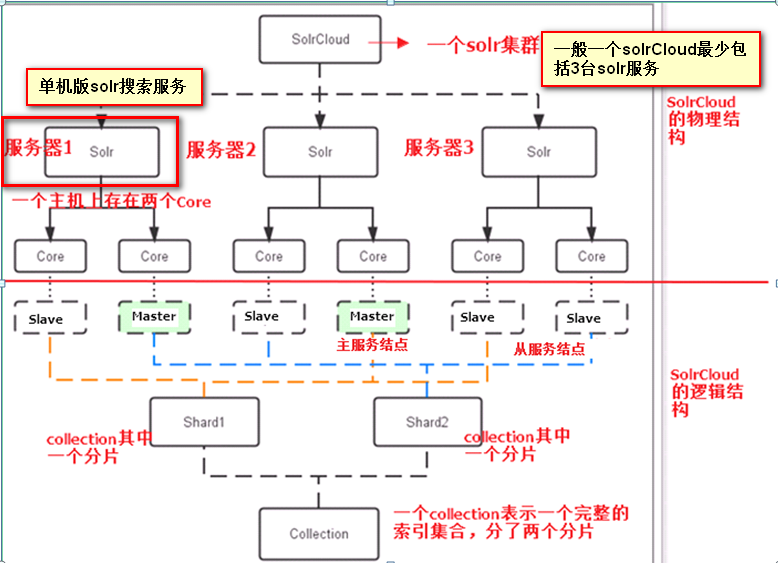
2）自动容错

3）近实时搜索

4）查询时自动负载均衡

* 1. SolrCloud的作用

solrcloud为了降低单机的处理压力，需要由多台服务器共同来完成索引和搜索任务。实现的思路是将索引数据进行Shard分片，每个分片由多台服务器共同完成，当一个索引或搜索请求过来时会分别从不同的Shard的服务器中操作索引。solrcloud是基于solr和zookeeper部署，zookeeper是一个集群管理软件，solrcloud需要由多台solr服务器组成，然后由zookeeper来进行协调管理。



12.3 solrCloud环境搭建

12.3.1 搭建zookeeper集群步骤

需要三台zookeeper、分别是zk1、zk2、zk3，对应的端口分别为2281、2282、2283。

# SolrCell

* 1. SolrCell介绍

用户的共同需求是能够摄取二进制和/或结构化文档，例如Office，Word，PDF和其他专有格式。该阿帕奇提卡项目提供了对包装许多不同的文件格式分析器，如PDFBox的，POI等的框架。Solr的ExtractingRequestHandler使用Tika允许用户将二进制文件上传到Solr并让Solr从中提取文本然后将其编入索引。虽然这个处理程序是一个很好的概念证明，但实际上并不建议用于生产。

索引现有数据的两种常用方法是数据导入处理程序（DIH）和Tika（Solr Cell）/ ExtractingRequestHandler。这些可用于索引数据库或结构化文档（例如Word文档，或PDF或......）中的数据。这些是很好的工具，可以快速启动和运行，我看到生产网站可以很好地使用这些工具中的一个或两个。

在开始之前，有一些有助于理解的概念。

* 1. 基本的概念

Tika将自动尝试确定输入文档类型（word，pdf等）并适当地提取内容。如果需要，可以使用stream.type参数为Tika显式指定MIME类型

Tika通过生成一个XHTML流来完成所有工作，并将其提供给SAX ContentHandler。（SAX是简单的XML的应用程序接口相当于dom但其不保存在内存中 ContentHandler这是一个大多数 SAX 应用程序实施的主要接口：如果需要通知应用程序基本解析事件，则它将实现此接口并且使用 setContentHandler 方法，向 SAX 解析器注册一个实例。解析器将使用该实例报告与基本文档相关的事件，如启动和终止元素与字符数据。）

然后Solr对Tika的SAX事件作出反应并创建要索引的字段。

Tika根据DublinCore等规范生成元数据信息，如标题，主题和作者。有关支持的文件类型，请访问http://tika.apache.org/site。

所有提取的文本都被添加到“内容”字段中

我们可以将Tika的元数据字段映射到Solr字段。我们可以推动这些领域

我们还可以传入字段值的文字。

我们可以将一个XPath表达式应用于Tika XHTML以限制生成的内容。

* 1. 输入参数

13.3.1. fmap。<source\_field> = <target\_field> - 将一个字段名称映射（移动）到另一个字段名称。示例：fmap.content = text将导致通常由Tika生成的内容字段移动到“text”字段。

13.3.2. boost。<fieldname> = <float> - 提升指定的字段。

13.3.3.literal。<fieldname> = <value> - 创建具有指定值的字段。如果Field是多值的，可以是多值的。

13.3.4.uprefix = <prefix> - 为具有给定前缀的模式中未定义的所有字段添加前缀。当与动态字段定义结合使用时，这非常有用。示例：uprefix = ignored\_将有效地忽略Tika生成的所有未知字段，因为示例架构包含<dynamicField name =“ignored\_ \*”type =“ignored”/>

13.3.5.defaultField = <字段名称> - 如果未指定uprefix且无法确定Field，则将使用默认字段。

13.3.6.extractOnly = true | false - 默认为false。如果为true，则从Tika返回提取的内容，而不对文档编制索引。这实际上包括提取的XHTML作为响应中的字符串。手动查看时，使用XML以外的响应格式来帮助查看嵌入的XHTML标记可能很有用。请参阅TikaExtractOnlyExampleOutput。

13.3.7.resource.name = <文件名> - 文件的可选名称。Tika可以用它作为检测哑剧类型的提示。

capture = <Tika XHTML NAME> - 分别使用名称捕获XHTML元素以添加到Solr文档。这对于将XHTML的块抓取到单独的字段中非常有用。例如，它可以用于获取段落（<p>）并将它们索引到一个单独的字段中。请注意，内容仍会被捕获到整个“内容”字段中。

captureAttr = true | false - 将Tika XHTML元素的索引属性分成单独的字段，以元素命名。例如，从HTML中提取时，Tika可以将<a>标签中的href属性作为名为“a”的字段返回。请参阅以下示例。

xpath = <XPath expression> - 提取时，仅返回满足XPath表达式的Tika XHTML内容。有关Tika XHTML格式的详细信息，请参见http://tika.apache.org/1.2/parser.html。另请参见TikaExtractOnlyExampleOutput。

lowernames = true | false - 将所有字段名称映射为带有下划线的小写。例如，Content-Type将映射到content\_type。

literalsOverride = true | false - <！> Solr4.0如果为true，则文字字段值将覆盖具有相同字段名称的其他值，例如元数据和内容。如果为false，则文字字段值将附加到来自Tika的任何提取数据，并且结果字段需要是多值的。默认值：true

resource.password = <密码> - <！> Solr4.0受密码保护的PDF或OOXML文件的可选密码。文件格式支持取决于Tika。

passwordsFile = <文件名> - <！> Solr4.0包含密码映射的文件名模式的文件的可选名称。请参阅下面的“加密文件”一章

如果extractOnly为true，则为其他输入参数：

extractFormat = xml | text - 默认为xml。控制提取内容的序列化格式。xml格式实际上是XHTML，就像将-x命令传递给tika命令行应用程序一样，而text就像-t命令一样。

# ZooKeeper的功能及其工作原理

* 1. Zookeeper是什么

Zookeeper是一个分布式，开源的分布式应用程序协调服务，是Google的Chubby一个开源的实现，它是集群的管理者，监视集群中各个节点的状态，根据节点提交的反馈进行下一步合理的操作，最终，将简单易用的接口和性能高效、功能稳定的系统提供给用户

# Tika介绍

摘要：本文主要针对自然语言处理（NLP）过程中，重要基础部分抽取文本内容的预处理。首先我们要意识到预处理的重要性。在大数据的背景下，越来越多的非结构化半结构化文本。如何从海量文本中抽取我们需要的有价值的知识显得尤为重要。另外文本格式常常不一，诸如：pdf，word，excl，xml，ppt，txt等常见文件类型你或许经过一番周折还是有办法处理的。倘若遇到database，html，邮件，RTF,图像，语音等文件，你是否素手无策了。基于此本文总结Apache Tika内容抽取工具，其强大之处在于可以处理各种文件，另外节约您更多的时间用来做重要的事情。本文第一节采用核心概念讲解第二节知识扩展补充。第三节典型DOME配有源代码第四节参考核心文件和Tika工具的JAR包共享

* 1. Tika 介绍

15.1.1 概念

Tika是一个内容分析工具，自带全面的parser工具类，能解析基本所有常见格式的文件，得到文件的metadata，content等内容，返回格式化信息。总的来说可以作为一个通用的解析工具。特别对于搜索引擎的数据抓去和处理步骤有重要意义。Tika是Apache的Lucene项目下面的子项目，在lucene的应用中可以使用tika获取大批量文档中的内容来建立索引，非常方便，也很容易使用。Apache Tika toolkit可以自动检测各种文档(如word,ppt,xml,csv,ppt等)的类型并抽取文档的元数据和文本内容。Tika集成了现有的文档解析库，并提供统一的接口，使针对不同类型的文档进行解析变得更简单。Tika针对搜索引擎索引、内容分析、转化等非常有用。

15.1.2 Tika架构

应用程序员可以很容易地在他们的应用程序集成Tika。Tika提供了一个命令行界面和图形用户界面，使它比较人性化。在本章中，我们将讨论构成Tika架构的四个重要模块。下图显示了Tika的四个模块的体系结构：

语言检测机制

每当一个文本文件被传递到Tika，它将检测在其中的语言。它接受没有语言的注释文件和通过检测该语言添加在该文件的元数据信息。支持语言识别，Tika 有一类叫做语言标识符在包org.apache.tika.language及语言识别资料库里面包含了语言检测从给定文本的算法。Tika 内部使用N-gram算法语言检测。

MIME检测机制

Tika可以根据MIME标准检测文档类型。Tika默认MIME类型检测是使用org.apache.tika.mime.mimeTypes。它使用org.apache.tika.detect.Detector 接口大部分内容类型检测。内部Tika使用多种技术，如文件匹配替换，内容类型提示，魔术字节，字符编码，以及其他一些技术。

解析器接口

org.apache.tika.parser 解析器接口是Tika解析文档的主要接口。该接口从提取文档中的文本和元数据，并总结了其对外部用户愿意写解析器插件。采用不同的具体解析器类，具体为各个文档类型，Tika 支持大量的文件格式。这些格式的具体类不同的文件格式提供支持，无论是通过直接实现逻辑分析器或使用外部解析器库。

Tika Facade 类

使用的Tika facade类是从Java调用Tika的最简单和直接的方式，而且也沿用了外观的设计模式。可以在 Tika API的org.apache.tika包Tika 找到外观facade类。通过实现基本用例，Tika作为facade的代理。它抽象了的Tika库的底层复杂性，例如MIME检测机制，解析器接口和语言检测机制，并提供给用户一个简单的接口来使用。

15.1.3 tika的特点

•统一解析器接口：Tika封装在一个单一的解析器接口的第三方解析器库。由于这个特征，用户逸出从选择合适的解析器库的负担，并使用它，根据所遇到的文件类型。

•低内存占用：Tika因此消耗更少的内存资源也很容易嵌入Java应用程序。也可以用Tika平台像移动那样PDA资源少，运行该应用程序。

•快速处理：从应用连结内容检测和提取可以预期的。

•灵活元数据：Tika理解所有这些都用来描述文件的元数据模型。

•解析器集成：Tika可以使用可在单一应用程序中每个文件类型的各种解析器库。

•MIME类型检测： Tika可以检测并从所有包括在MIME标准的媒体类型中提取内容。

•语言检测： Tika包括语言识别功能，因此可以在一个多语种网站基于语言类型的文档中使用。

15.1.4Tika的功能

Tika支持多种功能：

•文档类型检测

•内容提取

•元数据提取

•语言检测

# 16.Mysql数据库索引和lucence数据库的区别：

1）Lucene的Term index和Term Dictionary其实对应的就是MySQL的B+Tree的功能，为关键字key提供索引。Lucene的inverted index可以比MySQL的b-tree检索更快。

2）Term index在内存中是以FST（finite state transducers）的形式保存的，其特点是非常节省内存。所以Lucene搜索一个关键字key的速度是非常快的，而MySQL的B+Tree需要读磁盘比较。

3）Term dictionary在磁盘上是以分block的方式保存的，一个block内部利用公共前缀压缩，比如都是Ab开头的单词就可以把Ab省去。这样Term dictionary可以比B-tree更节约磁盘空间。

4）Lucene对不同的数据类型采用了不同的索引方式，上面分析是针对field为字符串的，比如针对int，有TrieIntField类型，针对经纬度，就可以用GeoHash编码。

5）在 Mysql中给两个字段独立建立的索引无法联合起来使用，必须对联合查询的场景建立复合索引，而Lucene可以任何AND或者OR组合使用索引进行检索。