Solr全文检索服务

* 什么是solr：
* Solr的概述：

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

* Solr和lucene的区别：

1. Lucene是一个开放源代码的全文检索引擎工具

包，lucene提供了完整的查询引擎和索引引擎。

1. 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可以非常快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

* Solr的安装配置
* Solr的文件目录：

  
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：该目录包含build过程中产生的war和jar文件，以及相关的依赖文件。

Doc：solr的api文档

Example：solr工程的例子目录

Example/solr：该目录是一个包含了默认配置信息的solr的core目录

Example/multicore：该目录包含了在solr的multicore中设置的多个core目录

Example/webapps：该目录包含一个solr.war，改war可作为solr的运行实例工程。

Licenses：solr相关的一些许可信息

* 运行环境：

Solr需要运行在一个servlet容器中，solr4.12.3需要jdk使用1.7以上，solr默认提供jetty，本教程使用tomcat作为servlet容器

* Solr和tomcat整合
* 创建一个solrhome目录：将solr安装包里的example/solr中的目录拷贝到solrhome目录中：

如图：

说明：

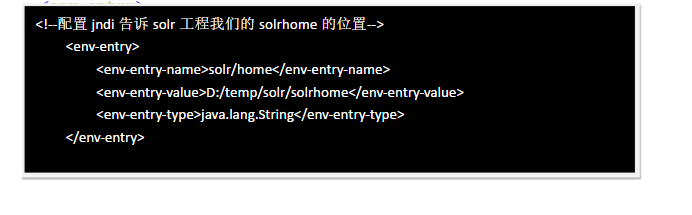
Collection1：叫做一个solr运行实例solrcore，solrcore名称不固定，一个solr运行实例对外单独提供索引和搜索接口。

Solrhome中可以创建多个solr运行实例solrcore

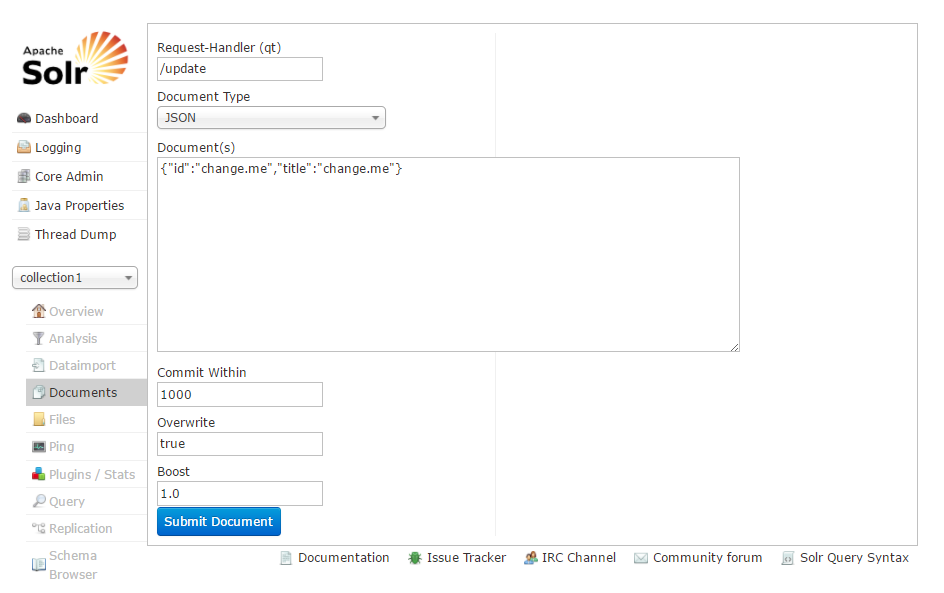
一个solr的运行实例对应一个索引目录。

* 将solr安装包dist/solr-4.10.3.war 拷贝到tomcat的webapp目录下改名为solr.war.
* 将solr.war解压，并删除
* 将solr安装包中的example/lib/ext目录下所有jar包到tomcat的webapp/solr/WEB-INF/lib目录下
* 修改tomcat目录下webapp/solr/WEB-INF/web.xml文件，

如图所示：



* 启动tomcate 在浏览器中输入localhost8080/solr
* Solr后台管理界面：



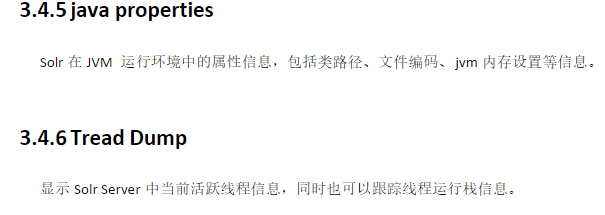
Dashboard：仪表盘，显示了该solr实例开始启动运行的时间，版本，资源系统，jvm等信息

Logging：solr运行的日志信息

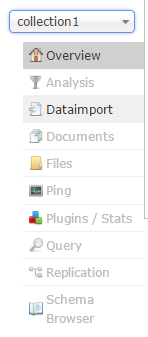
Cloud：Cloud即solrcloud，即solr云（集群），当使用solrclound模式运行时会显示此菜单，如下图是solrClound的管理界面



CoreAdmin：solrcore的管理界面，solrcore是solr的一个独立运行实例单位，它可以对外提供索引和搜索服务，一个solr工程可以运行多个solrcore（solr实例）一个core对应一个索引目录。



* （重要的）core selector
* 界面：

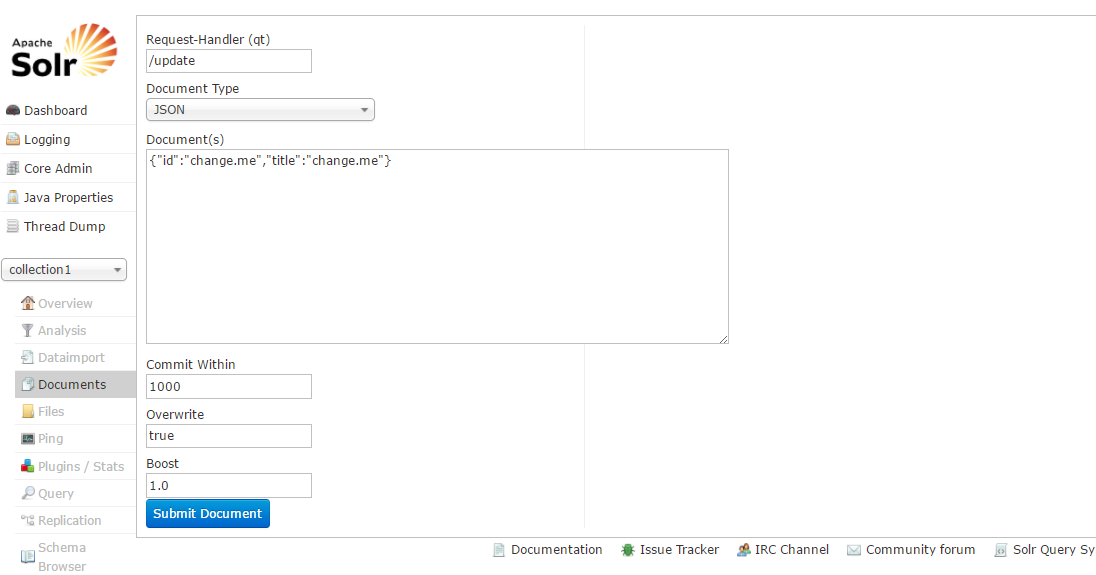
  
Analysis：

界面：

Dataimport：

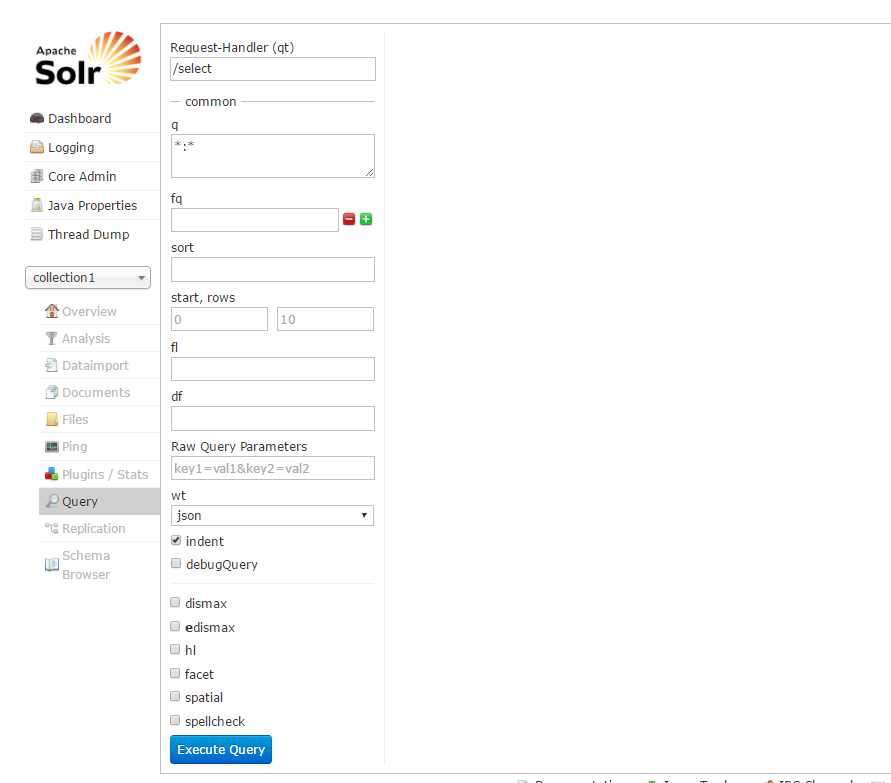
可以定义数据导入处理器，从关系数据库将数据导入到solr索引库中

Document（重点）

通过菜单可以创建索引，更新索引，删除索引等操作，界面如下：

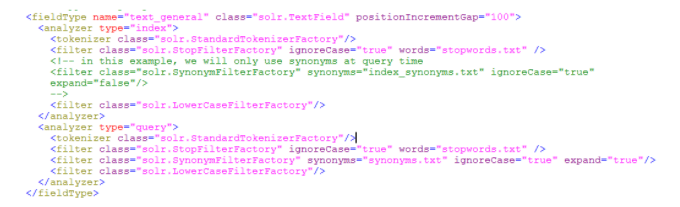
/update 表示更新索引，solr默认根据id（唯一约束）域来更新document的内容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更新。

Query（重点）：

界面：

通过/select执行搜索索引，必须指定“q“查询条件可搜索

* Solr索引：
* FieldType域类型定义：



FiledType子节点包括：name，class，positionIncrementGap等一些参数：

Name：是这个FieldType的名称

Class：是solr提供的包solr.TextField,solr.TextField

允许用户通过分析器来定制索引和查询，分析器包括一个分词器（tokenizer）和多个过滤器（filter）

positionIncrementGap:可选属性，定义在同一个文档中此类型数据的空白间隔，避免短语匹配错误，此值相当于Lucene的短语查询设置slop值，根据经验设置为100

索引分析器：使用solr.standardTokenizerFactory 标准分词器，solr.StopFilterFactory停用分词过滤器，solr.LowerCaseFilterFactory小写过滤器。

搜索分析器中：使用solr.standardTokenizerFactory 标准分词器，solr.StopFilterFactory停用分词过滤器，solr.LowerCaseFilterFactory小写过滤器。这里还用到了

solr.SynomymFilterFactory同义词过滤器。

* Field定义：

在field结点内定义具体的Field，field定义包括name，type（为之前定义过的各种filedType），indexed（是否被索引），stored（是否被存储），multiValued（是否存储多个值）等属性。



* Uniquekey

Solr中默认定义唯一主键key为id域，如下：

<uniqueKey>id</uniqueKey>

Solr在删除，更新索引时使用id域进行判断，也可以自定义唯一主键注意在创建索引时必须指定唯一约束。

* copyField复制域

copyField复制域，可以将多个field复制到一个Field中

以便进行统一的检索：比如输入关键字搜索title标题内容content

定义title，content，text的域：

<copyField source="title" dest="text"/>

<copyField source="content" dest="text"/>

根据关键字只搜素text域中的内容就相当于搜索title和content，将title和content复制到text中。

* dynamicField（动态字段）

动态字段就是不用指定具体的名称，只要定义字段名称的规则。

如：

<dynamicField name="\*\_i" type="int" indexed="true" stored="true"/>

<dynamicField name="\*\_is" type="int" indexed="true" stored="true" multiValued="true"/>

<dynamicField name="\*\_s" type="string" indexed="true" stored="true" />

<dynamicField name="\*\_ss" type="string" indexed="true" stored="true" multiValued="true"/>

* 配置IK中文分词器的步骤：
* 将IK中文分词器的包复制粘贴到tomcat中webapp/solr/lib中
* 将Ik中文分词器中的ext.dic stopword.dic IKAnalyzer.cfg.xml放在tomcat中webapp/solr/classes中，如果没有该文件夹，就自己创建。
* 在F:\solr\solrhome\collection1\conf\下的schema.xml中配置：

<!-- IKAnalyzer-->

<fieldType name="text\_ik" class="solr.TextField">

<analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>

</fieldType>

<!--IKAnalyzer Field-->

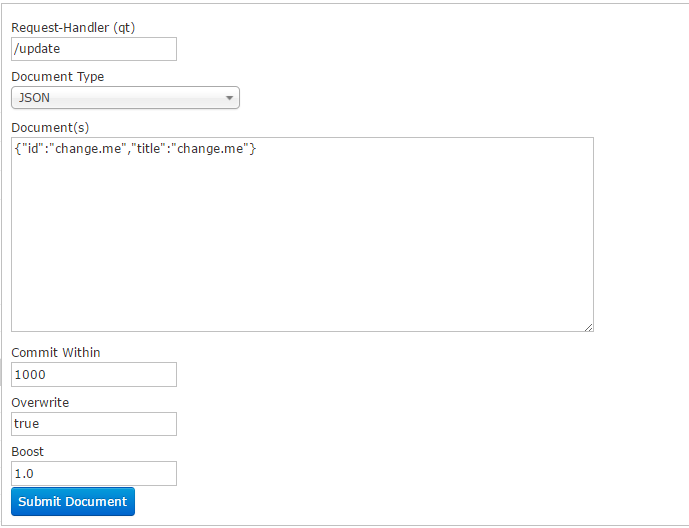
<field name="title\_ik" type="text\_ik" indexed="true" stored="true"/>

<field name="conent\_ik" type="text\_ik" indexed="true" stored="false" multiValued="true"/>

* 重启tomcat
* Solr后台:
  + - 索引维护;

使用/update进行索引维护，进入solr管理界面solrcore

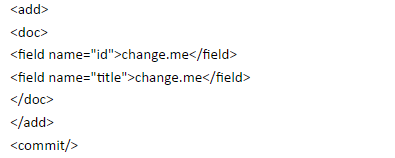
下的Document下:



Overwrite=“true“：solr在作索引的时候，如果文档已经存在，就用xml中的文档进行替换》

Commit Within: solr在做索引的时候，每个1秒，做一次文档提交

为了方便测试也可以在Document中立即提交，在</add>后面添加<commit/>如下：



* + - 添加/更新索引

Solr默认根据id（唯一约束）域来更新Document的内

容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更

新。

注意 ：唯一标识Field必须有，这里使用solr默认的id

* + - 删除索引
    - Dataimport-handler

配置步骤：

1. 在solrhome/collection1下创建一个lib文件夹：



2 在collection1/conf下修改solrconfig.xml，添加requestHandler；

<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">

　 <lst name="defaults">

　 <str name="config">data-config.xml</str>

　 </lst>

　</requestHandler>

3 编辑data-config.xml文件，存放在solrCore的conf目录：

<?xml version="1.0" encoding="UTF-8" ?>

<dataConfig>

<dataSource name="jdbcDataSource"

type="JdbcDataSource"

driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/test"

user="root"

password="1234"/>

<document>

<entity name="information" dataSource="jdbcDataSource" query="select stu\_id,name,sex,address from student" >

<field column="stu\_id" name="id"></field>

<field column="name" name="information\_name"></field>

<field column="sex" name="information\_sex"></field>

<field column="address" name="information\_address"></field>

</entity>

</document>

</dataConfig>

4 在schema.xml中配置field的信息

<field name="information\_name" type="string" indexed="true" stored="true" />

<field name="information\_sex" type="string" indexed="true" stored="true" />

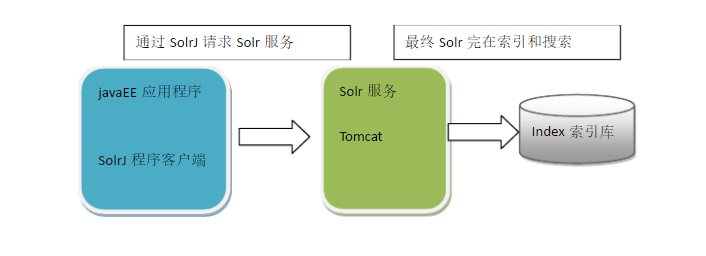
<field name="information\_address" type="text\_ik" indexed="true" stored="true" />

注意：<field column="pid" name="id"/>必须有一个id域，这里使用Solr默认的id域，域值是从关系数据库查询的pid列值。

5 重启tomcat服务器。

* SolrJ
  + 什么是solrJ

Solrj是访问solr服务的java客户端，提供索引和搜索的请求方法，solrj通常在嵌入在业务系统中，通过solrJ的api接口操作Solr服务，如下图



* + 创建索引：

见代码：

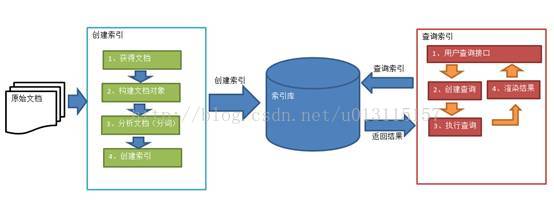
1. **public** **class** SolrTest {
2. /\*\*
3. \* @Title : TestAdd
4. \* @功能描述: TODO 索引添加
5. \* @开发者：陈强
6. \* @参数： @throws Exception
7. \* @返回类型：void
8. \* @throws ：
9. \*/
10. @Test
11. **public** **void** TestAdd()throws Exception
12. {
13. String url="http://localhost:8080/solr/";
14. SolrServer solrServer=**new** HttpSolrServer(url);
15. SolrInputDocument doc=**new** SolrInputDocument();
16. doc.addField("name", "value");
17. doc.addField("id", "haha");
18. solrServer.add(doc);
19. solrServer.commit();



24. }
25. /\*\*
26. \* @Title : TestDeleteIndex
27. \* @功能描述: TODO 索引删除
28. \* @开发者：陈强
29. \* @参数： @throws SolrServerException
30. \* @参数： @throws IOException
31. \* @返回类型：void
32. \* @throws ：
33. \*/
34. @Test
35. **public** **void** TestDeleteIndex() throws SolrServerException, IOException
36. {
38. String url="http://localhost:8080/solr/";
39. SolrServer solrServer=**new** HttpSolrServer(url);
40. //UpdateResponse response = solrServer.deleteById("haha");
42. solrServer.deleteByQuery("id:haha");
43. solrServer.commit();
44. }
45. /\*\*
46. \* @Title : Test1
47. \* @功能描述: TODO 搜索索引
48. \* @开发者：陈强
49. \* @参数： @throws Exception
50. \* @返回类型：void
51. \* @throws ：
52. \*/
53. @Test
54. **public** **void** Test1() throws Exception
55. {
56. String url="http://localhost:8080/solr/";
57. SolrServer solrServer=**new** HttpSolrServer(url);
58. //使用SolrQuery
59. SolrQuery query=**new** SolrQuery();
60. //两种查询方法 “q“ ：query
61. // 1.query.set("q","id:\*");
62. query.setQuery("id:\*");
63. QueryResponse response = solrServer.query(query);
64. SolrDocumentList results = response.getResults();
65. System.out.println(results.size());
66. **for** (SolrDocument solrDocument : results) {
67. System.out.println(solrDocument.getFieldValue("information\_name"));
68. System.out.println(solrDocument.getFieldValue("information\_sex"));
69. System.out.println(solrDocument.getFieldValue("information\_address"));
70. System.out.println(solrDocument.getFieldValue("id"));
72. }
73. }
74. /\*\*
75. \* @Title : test2
76. \* @功能描述: TODO 组合查询，过滤查询，排序，指定域查询
77. \* @开发者：陈强
78. \* @参数： @throws Exception
79. \* @返回类型：void
80. \* @throws ：
81. \*/
82. @Test
83. **public** **void** test2() throws Exception
84. {
86. String url="http://localhost:8080/solr/";
87. SolrServer solrServer=**new** HttpSolrServer(url);
88. //使用SolrQuery
89. SolrQuery query=**new** SolrQuery();
90. query.setQuery("id:\*");
91. //设置过滤条件：”fq“： filter query
92. query.setFilterQueries("information\_sex:boy");
93. //排序
94. query.addSort("id", ORDER.desc);
95. // 只查指定的域
96. query.set("fl","id,information\_name");
97. QueryResponse response = solrServer.query(query);
98. SolrDocumentList results = response.getResults();
99. System.out.println(results.size());
100. **for** (SolrDocument solrDocument : results) {
101. System.out.println(solrDocument.getFieldValue("information\_name"));
102. System.out.println(solrDocument.getFieldValue("information\_sex"));
103. System.out.println(solrDocument.getFieldValue("information\_address"));
104. System.out.println(solrDocument.getFieldValue("id"));
106. }
107. }
108. /\*\*
109. \* @Title : TestSearch
110. \* @功能描述: TODO 高亮，分页
111. \* @开发者：陈强
112. \* @参数： @throws Exception
113. \* @返回类型：void
114. \* @throws ：
115. \*/
116. @Test
117. **public** **void** TestSearch()throws Exception
118. {
119. String url="http://localhost:8080/solr/";
120. SolrServer solrServer=**new** HttpSolrServer(url);
121. SolrQuery query=**new** SolrQuery();
122. //query.add("q", "id:\*");
123. query.setQuery("id:\*");
124. //fq filter query
125. //query.set("fq", "id:haha");
126. //  query.setFilterQueries("id:haha");
127. //query.addSort("id",ORDER.desc);
128. query.setStart(0);
129. query.setRows(3);
130. //
131. query.setHighlight(**true**);
132. query.addHighlightField("information\_name");
133. query.setHighlightSimplePre("<span style='color:red'>");
134. query.setHighlightSimplePost("</span>");
135. QueryResponse response = solrServer.query(query);
136. SolrDocumentList results = response.getResults();
137. Map<String, Map<String, List<String>>> highlighting = response.getHighlighting();
138. System.out.println(results.getNumFound());
139. **for** (SolrDocument solrDocument : results) {
140. System.out.println(solrDocument.getFieldValue("information\_name"));
141. System.out.println(solrDocument.getFieldValue("information\_sex"));
142. System.out.println(solrDocument.getFieldValue("information\_address"));
143. System.out.println(solrDocument.getFieldValue("id"));
144. Map<String, List<String>> map = highlighting.get(solrDocument.getFieldValue("id"));
145. List<String> list = map.get("information\_name");
146. System.out.println(list.get(0));
147. }
148. /\*QueryResponse response = solrServer.query(query);
149. SolrDocumentList results = response.getResults();
150. System.out.println(results.size());
151. for (SolrDocument solrDocument : results) {
152. System.out.println(solrDocument.getFieldValue("information\_name"));
153. System.out.println(solrDocument.getFieldValue("information\_sex"));
154. System.out.println(solrDocument.getFieldValue("information\_address"));
155. System.out.println(solrDocument.getFieldValue("id"));
157. }\*/
158. }
160. }

Lucene的讲解

* 概念：
  + Luncene是apache下的一个开放源代码的全文搜索引擎工具包。提供了完整的查询引擎和索引引擎，部分文本分析引擎。Lucene的目的是为开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能。
* Lucene实现全文检索的流程：
  + 实现的流程图：



1、绿色表示索引过程，对要搜索的原始内容进行索引构建一个索引库，索引过程包括：

确定原始内容即要搜索的内容à采集文档à创建文档à分析文档à索引文档

2、红色表示搜索过程，从索引库中搜索内容，搜索过程包括：

用户通过搜索界面à创建查询à执行搜索，从索引库搜索à渲染搜索结果

* Lucene创建索引的过程：
  + 原始文档：是指要索引和搜索的内容。原始文档：包括互联网上的网页，数据库中的数据，磁盘上的文件等。
  + 获取文档（信息采集）：从互联网上，数据库，文件系统中等获取需要搜索的原始信息，这个过程就是信息采集，信息采集的目的就是为了对原始内容进行索引。
  + 构建文档对象：

获取原始内容的目的是为了索引，在索引前需要将原始内容创建成文档（Document），文档中包括一个一个的域（Field），域中存储内容。

这里我们可以将磁盘上的一个文件当成一个document，Document中包括一些Field（file\_name文件名称、file\_path文件路径、file\_size文件大小、file\_content文件内容），如下图：

Document （ 文 档 ） 
Field （ 域 ） 
Name: file_name （ 文 件 名 称 ） 
Value ： springmvc.txt 
Field （ 域 ） 
' “ ： （ 文 袢 内 容 
其 它 Field. 。 。 
Field （ 域 ） 
Name: file_path （ 文 件 路 径 ） 
Value ： e:/.../XXXXXX.txt 
Name: 巳 一 5 （ 文 件 大 小 ） 
Value ： 346543 

注意：每个Document可以有多个Field，不同的Document可以有不同的Field，同一个Document可以有相同的Field（域名和域值都相同）

每个文档都有自己的编号，就是文档的id

* 分析文档：

将原始内容创建为包含域（Field）的文档（document），需要再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、将字母转为小写、去除标点符号、去除停用词等过程生成最终的语汇单元，可以将语汇单元理解为一个一个的单词。

比如下边的文档经过分析如下：

原文档内容：

Lucene is a Java full-text search engine. Lucene is not a complete

application, but rather a code library and API that can easily be used

to add search capabilities to applications.

分析后得到的语汇单元：

lucene、java、full、search、engine。。。。

每个单词叫做一个Term，不同的域中拆分出来的相同的单词是不同的term。term中包含两部分一部分是文档的域名，另一部分是单词的内容。

例如：文件名中包含apache和文件内容中包含的apache是不同的term。

不恰当的比喻Term和map（key，value）相似

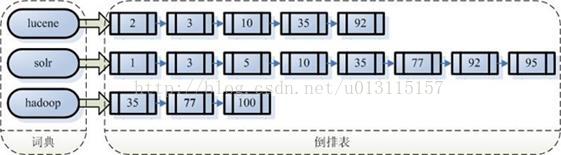
* 索引文档：

对所有文档分析得出的语汇单元进行索引，索引的目的是为了搜索，最终要实现只搜索被索引的语汇单元从而找到Document（文档）。

注意：创建索引是对语汇单元索引，通过词语找文档，这种索引的结构叫倒排索引结构。

传统方法是根据文件找到该文件的内容，在文件内容中匹配搜索关键字，这种方法是顺序扫描方法，数据量大、搜索慢。

倒排索引结构是根据内容（词语）找文档，如下图：



**倒排索引结构也叫反向索引结构，包括索引和文档两部分，索引即词汇表，它的规模较小，而文档集合较大。**

* 创建索引的demo：
  + 指定索引库存放的位置。Directory对象 （有FSDirectory创建）
  + 需要使用到IndexWriter对象来创建索引
    - 参数directory 索引库存放的位置
    - indexWriterConfig 有new IndexWriterConfig(Version.LAST,analyzer)
  + 指定创建索引时使用的分析器Analyzer对象
  + 获取原始文档
  + 创建文档对象Document， 接着创建域 Field
  + 将文档对象使用indexWriter对象写入索引库。

1. @Test
2. **public** **void** createIndex() throws IOException
3. {
4. //指定索引库存放的位置
5. //FSDirectory 全名File System Directory
6. Directory directory=FSDirectory.open(**new** File("E:\\Lucene&solr\\index"));
7. //创建索引时使用的分词器
8. //StandardAnalyzer 官方推荐的标准分词器
9. Analyzer analyzer=**new** StandardAnalyzer();
10. IndexWriterConfig config=**new** IndexWriterConfig(Version.LATEST, analyzer);
11. //需要使用到IndexWriter对象来创建索引
12. IndexWriter indexWriter=**new** IndexWriter(directory, config);
13. //获取文档
14. File file=**new** File("E:\\Lucene&solr\\demo");
15. File[] listFiles = file.listFiles();
16. **for** (File file2 : listFiles) {
17. //创建文档对象
18. Document document=**new** Document();
19. String fileName=file2.getName();
20. //创建域
21. Field filed1=**new** TextField("fileName", fileName, Store.YES);
22. **long** fileSize = FileUtils.sizeOf(file2);
23. Field filed2=**new** LongField("fileSize", fileSize,Store.YES);
24. String filePath=file2.getPath();
25. Field filed3=**new** TextField("filePath", filePath, Store.YES);
26. String fileContent=FileUtils.readFileToString(file2);
27. Field filed4=**new** TextField("fileContent", fileContent, Store.YES);
28. document.add(filed1);
29. document.add(filed2);
30. document.add(filed3);
31. document.add(filed4);
32. //将文档对象使用indexWriter对象写入索引库
33. indexWriter.addDocument(document);
34. }
35. indexWriter.close();
36. }

* Field域：
  + 域的三个属性：是否分析，是否索引，是否存储
  + 是否分析（tokenized）

是：将Field值分析出语汇单元即作分词处理，将词进行索引

比如：商品的名称，商品简介等，这些内容用户要输入关键字搜索，由于搜索的内容格式大，内容多需要分析后将语汇单元索引

否：不作分词处理

比如：商品id，订单号，身份证号等

* 是否索引

是：将Field分析后的词或整个Field值进行索引，只有索引方可搜索到

否：不索引无法搜索到

* 是否存储：

是：将Field值存储在文档中，存储在文档中的Field才可以从Document中获取

否：不存储Field值，不存储的Field无法通过Document获取

* 常用的域：

Analyzed 
Indexed 
Storedv 
Field 
StringField(FieldName, 
LongField(FieldName, 
Long 
StoredField(FieldName, 
z, 
FieldValue) e 
TextField(FieldName, 
FieldValue, Store.NO)e 
TextField(FieldName, 
reader)e 
Field 
Field 
(iTE# 
store.YES 
store.N0 
Field 
Long 
Field 
store.YES 
store.N0 
Field 
Fielde 
Field 
Reader, 
Unstored 

* 查询索引过程
  + 实现步骤：
    - 创建一个Directory对象，也就是索引库存放的位置
    - 创建一个indexReader对象，需要指定Directory对象
    - 创建一个indexSearcher对象，需指定IndexReader对象
    - 创建一个TermQuery对象，指定查询的域和查询的关键词
    - 执行查询
    - 返回查询结果，遍历查询结果并输出
    - 关闭IndexReader对象
  + IndexSearcher搜索方法
    - indexSearcher.search（query，n）：根据Query搜索，返回评分最高的n条记录。
    - indexSearcher.search（query，filter，n）：根据Query搜索，添加过滤策略，返回评分最高的n条记录。
    - indexSearcher.search（query，n，sort）：根据Query搜索，添加排序策略，返回评分最高的n条记录。
    - indexSearcher.search（query，filter，n，sort）：根据Query搜索，添加过滤策略，添加排序策略，返回评分最高的n条记录。
  + 总结:
    - TopDocs类
      * ScoreDoc[] scoreDocs ---排名靠前的查询
      * Int totalHits----命中的查询总数
    - 文档document对象的id

scoreDoc.doc

* 索引库的维护：
  + 向索引库添加文档 indexWriter.addDocument（document）；
  + 删除文档
    1. 删除全部文档（慎用）indexWriter.deleteAll（）；
    2. 根据查询条件删除：

Query query=new TermQuery（“context”，“java”）；

indexWriter.deleteDocument（query）；

* 更新文档：原理就是先删除后添加：indexWriter.updateDocument（term，document）；
* 查询索引：
  + 使用Query子查询
  + TermQuery 准确查询指定一个查询的域及查询的关键字
  + NumericRangeQuery 范围查询
  + BooleanQuery 组合查询

查询条件之间的逻辑关系

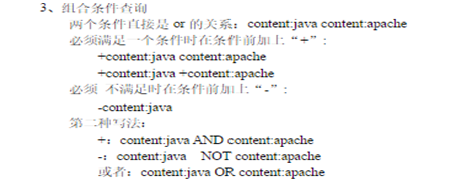
Occur.MUST 相当于and +

Occur.SHOULD 相当于or 空

Occur.MUST NOT 相当于not -

* MatchAllDocsQuery 匹配所有的文档
* 使用QueryParser查询
  + 查询的语法

在queryparser中使用查询语法

1. 指定域指定查询的关键字：FileName：java
2. 范围查询：size：[10 to 200]:范围查询的查询语法在lucene中不支持数值类型的查询。可以查询字符串类型。要查询数值类型的范围还需要使用NumericRangeQuery
3. 

* 