# solr基础知识

## 1. Solr 是什么？

Solr它是一种开放源码的、基于 Lucene Java 的搜索服务器，易于加入到 Web 应用程序中。Solr 提供了层面搜索(就是统计)、命中醒目显示并且支持多种输出格式（包括XML/XSLT 和JSON等格式）。它易于安装和配置，而且附带了一个基于HTTP 的管理界面。可以使用 Solr 的表现优异的基本搜索功能，也可以对它进行扩展从而满足企业的需要。Solr的特性包括：

* 高级的全文搜索功能
* 专为高通量的网络流量进行的优化
* 基于开放接口（XML和HTTP）的标准
* 综合的HTML管理界面
* 可伸缩性－能够有效地复制到另外一个Solr搜索服务器
* 使用XML配置达到灵活性和适配性
* 可扩展的插件体系

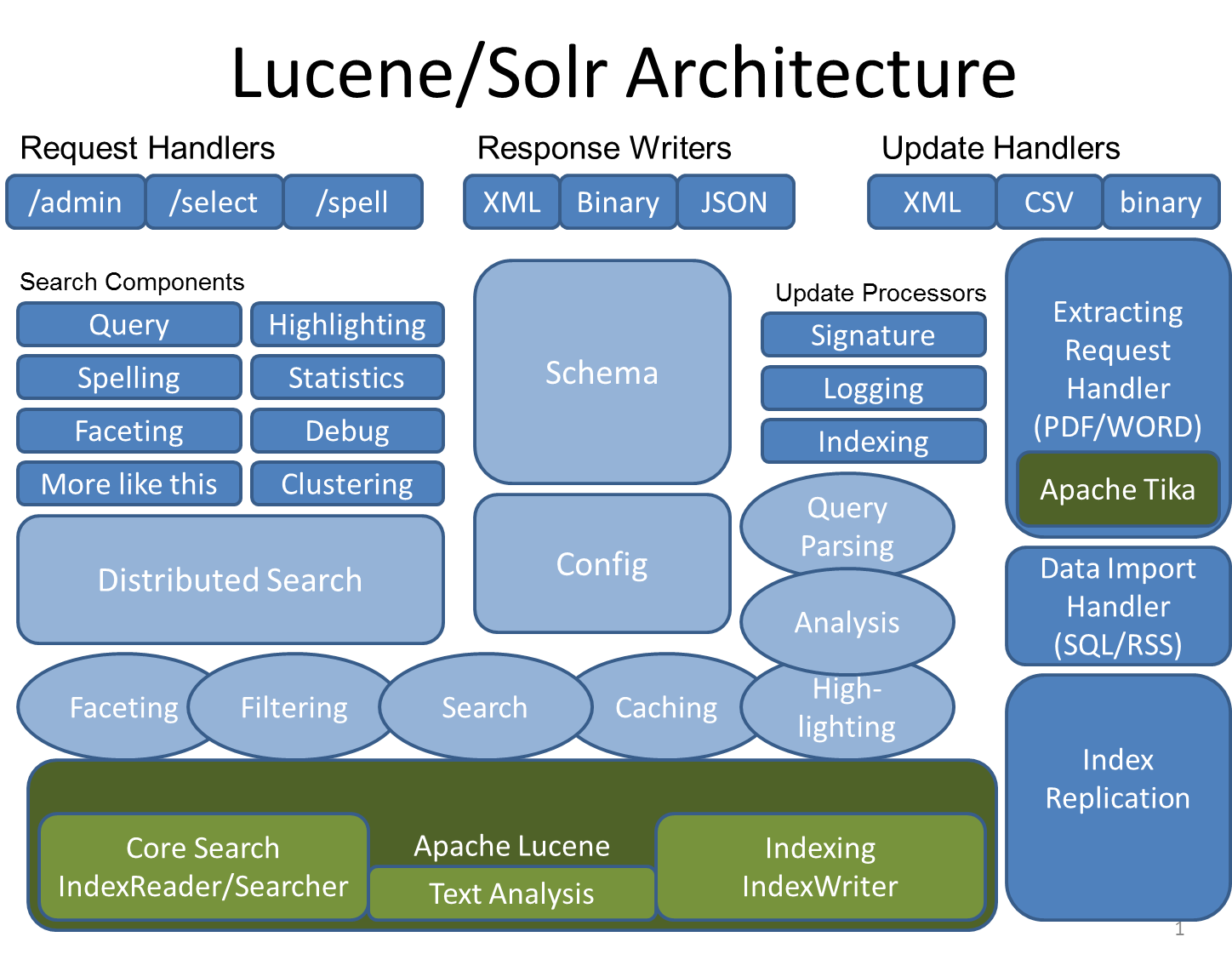
## 2. Lucene 是什么？

Lucene是一个基于Java的全文信息检索工具包，它不是一个完整的搜索应用程序，而是为你的应用程序提供索引和搜索功能。Lucene 目前是 Apache Jakarta(雅加达) 家族中的一个开源项目。也是目前最为流行的基于Java开源全文检索工具包。目前已经有很多应用程序的搜索功能是基于 Lucene ，比如Eclipse 帮助系统的搜索功能。Lucene能够为文本类型的数据建立索引，所以你只要把你要索引的数据格式转化的文本格式，Lucene 就能对你的文档进行索引和搜索。

## 3. Solr vs Lucene

Solr与Lucene 并不是竞争对立关系，恰恰相反Solr 依存于Lucene，因为Solr底层的核心技术是使用Lucene 来实现的，Solr和Lucene的本质区别有以下三点：搜索服务器，企业级和管理。Lucene本质上是搜索库，不是独立的应用程序，而Solr是。Lucene专注于搜索底层的建设，而Solr专注于企业应用。Lucene不负责支撑搜索服务所必须的管理，而Solr负责。所以说，一句话概括 Solr: Solr是Lucene面向企业搜索应用的扩展。

Solr与Lucene架构图:



Solr使用Lucene并且扩展了它！

* 一个真正的拥有动态字段(Dynamic Field)和唯一键(Unique Key)的数据模式(Data Schema)
* 对Lucene查询语言的强大扩展！
* 支持对结果进行动态的分组和过滤
* 高级的，可配置的文本分析
* 高度可配置和可扩展的缓存机制
* 性能优化
* 支持通过XML进行外部配置
* 拥有一个管理界面
* 可监控的日志
* 支持高速增量式更新(Fast incremental Updates)和快照发布(Snapshot Distribution)

## 4. RHEL下部署Solr

1)下载solr4.7.0.zip

2) 解压solr-4.7.0.zip

|  |
| --- |
| [root@SnsRedhat2Yangpeng soft]# unzip solr-4.7.0.zip |

3) 将solr-4.7.0/dist/solr-4.7.0.war拷贝到tomcat的webapps下，并改名solr.war

|  |
| --- |
| [root@SnsRedhat2Yangpeng dist]# cp solr-4.7.0.war /root/sp1/soft/tomcat/tomcat/webapps/solr.war |

4) 运行tomcat，solr.war自会解压

|  |
| --- |
| [root@SnsRedhat2Yangpeng bin]# sh startup.sh |

5) 拷贝solr-4.7.0/example/lib/ext/下所有jar包至webapps/solr/WEB-INF/lib/

|  |
| --- |
| [root@SnsRedhat2Yangpeng soft]# cp solr-4.7.0/example/lib/ext/\* tomcat/tomcat/webapps/solr/WEB-INF/lib/ |

6) 在tomcat/tomcat/webapps/solr/WEB-INF下创建classes文件夹

|  |
| --- |
| [root@SnsRedhat2Yangpeng WEB-INF]# mkdir -p classes |

7) 将solr-4.7.0/example/resources/log4j.properties

拷贝到webapps/solr/WEB-INF/classes下

|  |
| --- |
| [root@SnsRedhat2Yangpeng WEB-INF]# cp ../../../../../solr-4.7.0/example/resources/log4j.properties classes/ |

8) 创建solrHome文件夹，这里我建在了和tomcat同级目录,

这时全路径是：/root/sp1/soft/solrHome

|  |
| --- |
| [root@SnsRedhat2Yangpeng soft]# mkdir -p solrHome |

9) 修改webapps/solr/WEB-INF下的web.xml，打开<env-entry>的注释，改为如下内容：

|  |
| --- |
| <env-entry>  <env-entry-name>solr/home</env-entry-name>  <env-entry-value>/root/sp1/soft/solrHome</env-entry-value>  <env-entry-type>java.lang.String</env-entry-type>  </env-entry> |

[注意]env-entry-value的值为solrHome的路径。

10) 把solr-4.7.0/ multicore下的所有文件拷贝进solrHome

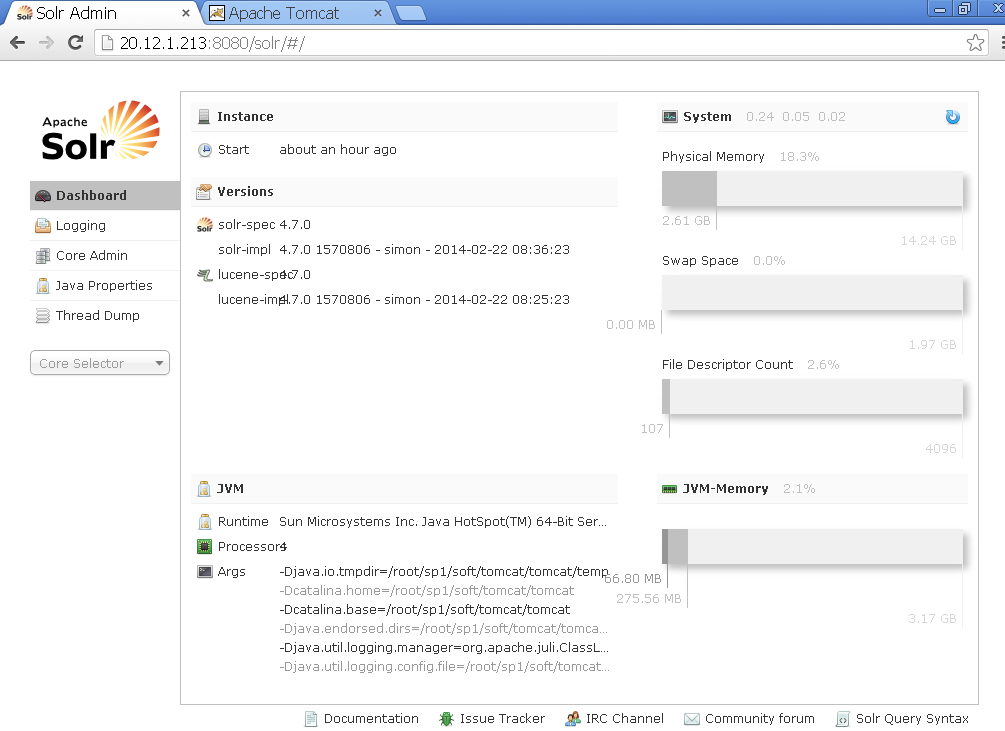
|  |
| --- |
| [root@SnsRedhat2Yangpeng soft]# cp -r solr-4.7.0/example/multicore/\* solrHome/ |

11) 配置完毕，可以启动tomcat进行测试了。

|  |
| --- |
| [root@SnsRedhat2Yangpeng soft]# sh tomcat/tomcat/bin/startup.sh |

请求URL：<http://20.12.1.213:8080/solr>

出现如下界面：

ok，环境部署成功。

## 5. Solr基础

因为 Solr 包装并扩展了Lucene，所以它们使用很多相同的术语。更重要的是，Solr 创建的索引与 Lucene 搜索引擎库完全兼容。通过对 Solr 进行适当的配置，某些情况下可能需要进行编码，Solr 可以阅读和使用构建到其他 Lucene 应用程序中的索引。在 Solr 和 Lucene 中，使用一个或多个 Document 来构建索引。Document 包括一个或多个 Field。Field 包括名称、内容以及告诉 Solr 如何处理内容的元数据。

例如，Field 可以包含字符串、数字、布尔值或者日期，也可以包含你想添加的任何类型，只需用在solr的配置文件中进行相应的配置即可。Field 可以使用大量的选项来描述，这些选项告诉 Solr 在索引和搜索期间如何处理内容。

现在，查看一下表中列出的重要属性的子集：

|  |  |
| --- | --- |
| 属性名称 | 描述 |
| Indexed | Indexed Field 可以进行搜索和排序。你还可以在 indexed Field 上运行 Solr 分析过程，此过程可修改内容以改进或更改结果。 |
| Stored | stored Field 内容保存在索引中。这对于检索和醒目显示内容很有用，但对于实际搜索则不是必需的。例如，很多应用程序存储指向内容位置的指针而不是存储实际的文件内容。 |

### 5.1 配置文件solr.xml

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

<solr persistent="false">

<cores adminPath="/admin/cores" host="${host:}"

hostPort="${jetty.port:8983}" hostContext="${hostContext:solr}">

<core name="core0" instanceDir="core0" />

<core name="core1" instanceDir="core1" />

</cores>

</solr>

这里没有配置defaultCoreName，默认为collection1。其中<cores>的adminPath属性指明通过浏览器查看主目录多个核的属性的相对地址。<core>是关于每个核的名字和目录的指示，例如，这里有两个核，core0和core1，它们各自的实例目录分别是solr.xml所在文件夹下的core0和core1目录。而且，就算没有solr.xml文件，也不会报错的，因为都有默认值。

源码片段如下：

public static final String *DEFAULT\_DEFAULT\_CORE\_NAME* = "collection1";

public static final String *DEF\_SOLR\_XML* = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n"

+ "<solr persistent=\"false\">\n"

+ " <cores adminPath=\"/admin/cores\" defaultCoreName=\""

+ *DEFAULT\_DEFAULT\_CORE\_NAME*

+ "\""

+ " host=\"${host:}\" hostPort=\"${hostPort:}\" hostContext=\"${hostContext:}\" zkClientTimeout=\"${zkClientTimeout:15000}\""

+ ">\n"

+ " <core name=\""

+ *DEFAULT\_DEFAULT\_CORE\_NAME*

+ "\" shard=\"${shard:}\" collection=\"${collection:collection1}\" instanceDir=\"collection1\" />\n"

+ " </cores>\n" + "</solr>";

### 5.2 模式配置Schema.xml

schema.xml这个配置文件可以在你下载solr包的安装解压目录的\solr\example\solr\collection1\conf中找到，它就是solr模式关联的文件。打开这个配置文件，会发现有详细的注释。模式组织主要分为三个重要配置

#### 5.2.1. types 部分

是一些常见的可重用定义，定义了 Solr（和 Lucene）如何处理 Field。也就是添加到索引中的xml文件属性中的类型，如int、text、date等.

<fieldType name="string" class="solr.StrField" sortMissingLast="true"/>

<fieldType name="boolean" class="solr.BoolField" sortMissingLast="true"/>

<fieldType name="int" class="solr.TrieIntField" precisionStep="0" positionIncrementGap="0"/>

<fieldType name="text\_general" class="solr.TextField" positionIncrementGap="100">

<analyzer type="index">

<tokenizer class="solr.StandardTokenizerFactory"/>

<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" enablePositionIncrements="true" />

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="solr.StandardTokenizerFactory"/>

<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" enablePositionIncrements="true" />

<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>

<filter class="solr.LowerCaseFilterFactory"/>

</analyzer>

</fieldType>

参数说明:

|  |  |
| --- | --- |
| 属性 | 描述 |
| name | 标识而已 |
| class | 和其他属性决定了这个fieldType的实际行为。 |
| sortMissingLast | 设置成true没有该field的数据排在有该field的数据之后，而不管请求时的排序规则, 默认是设置成false。 |
| sortMissingFirst | 跟上面倒过来，默认是设置成false |
| analyzer | 字段类型指定的分词器 |
| type | 当前分词用用于的操作.index代表生成索引时使用的分词器query代码在查询时使用的分词器 |
| tokenizer | 分词器类 |
| filter | 分词后应用的过滤器 过滤器调用顺序和配置相同. |

#### 5.2.2. fileds

是你添加到索引文件中出现的属性名称，而声明类型就需要用到上面的types

<field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false"/>

<field name="path" type="text\_smartcn" indexed="false" stored="true" multiValued="false" termVector="true" />

<field name="content" type="text\_smartcn" indexed="false" stored="true" multiValued="false" termVector="true"/>

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

<field name ="pinyin" type ="text\_pinyin" indexed ="true" stored ="false" multiValued ="false"/>

<field name="\_version\_" type="long" indexed="true" stored="true"/>

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

<dynamicField name="\*\_l" type="long" indexed="true" stored="true"/>

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

* field: 固定的字段设置
* dynamicField: 动态的字段设置,用于后期自定义字段,\*号通配符.例如: test\_i就是int类型的动态字段.

还有一个特殊的字段copyField,一般用于检索时用的字段这样就只对这一个字段进行索引分词就行了,copyField的dest字段如果有多个source一定要设置multiValued=true,否则会报错的。

另外，使用copyField将多个field都复制到一个总的field上，然后对它进行搜索，可以最小化索引字段并且提高搜索的效率。

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

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

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

字段属性说明:

|  |  |
| --- | --- |
| 属性 | 描述 |
| name | 字段类型名 |
| class | java类名 |
| indexed | 缺省true。 说明这个数据应被搜索和排序，如果数据没有indexed，则stored应是true。 |
| stored | 缺省true。说明这个字段被包含在搜索结果中是合适的。如果数据没有stored,则indexed应是true。 |
| omitNorms | 字段的长度不影响得分和在索引时不做boost时，设置它为true。  一般文本字段不设置为true。 |
| termVectors | 如果字段被用来做more like this 和highlight的特性时应设置为true。 |
| compressed | 字段是压缩的。这可能导致索引和搜索变慢，但会减少存储空间，只有StrField和TextField是可以压缩，这通常适合字段的长度超过200个字符。 |
| multiValued | 字段多于一个值的时候，可设置为true。 |
| positionIncrementGap | 和multiValued一起使用，设置多个值之间的虚拟空白的数量 |

注意:\_version\_ 是一个特殊字段,不能删除,是记录当前索引版本号的.

#### 5.2.3. 其他配置

uniqueKey: 唯一键，这里配置的是上面出现的fileds，一般是id、url等不重复的。在更新、删除的时候可以用到。

defaultSearchField:默认搜索属性，如q=solr就是默认的搜索那个字段

solrQueryParser:查询转换模式，是并且还是或者（AND/OR必须大写）

### 5.3. solr配置solrconfig.xml

solrconfig.xml这个配置文件可以在你下载solr包的安装解压目录的solr\example\solr\collection1\conf中找到，这个配置文件内容有点多,主要内容有:使用的lib配置,包含依赖的jar和Solr的一些插件;组件信息配置;索引配置和查询配置。下面详细说明：

#### 5.3.1 索引indexConfig

Solr 性能因素，来了解与各种更改相关的性能权衡。 下表概括了可控制 Solr 索引处理的各种因素：

|  |  |
| --- | --- |
| 属性 | 描述 |
| useCompoundFile | 通过将很多 Lucene 内部文件整合到一个文件来减少使用中的文件的数量。这可有助于减少 Solr 使用的文件句柄数目，代价是降低了性能。除非是应用程序用完了文件句柄，否则 false 的默认值应该就已经足够。 |
| ramBufferSizeMB | 在添加或删除文档时，为了减少频繁的更些索引，Solr会选缓存在内存中，当内存中的文件大于设置的值，才会更新到索引库（也就是<maxBufferedDocs>制定缓存的文档数目达到多少时会自动建立硬盘索引文件）。较大的值可使索引时间变快但会牺牲较多的内存。如两个值同时设置，满足一个就会进行刷新索引 |
| maxBufferedDocs |
| mergeFactor | 决定低水平的 Lucene 段被合并的频率。较小的值（最小为 2）使用的内存较少但导致的索引时间也更慢。较大的值可使索引时间变快但会牺牲较多的内存。例如：<meigeFactor>10</mergeFactor>表示，最低级的索引文件数量达到10时会合并为1个索引文件，当这样合成索引文件数目达到10个时，又会合并为更高级的索引文件 |
| maxIndexingThreads | indexWriter生成索引时使用的最大线程数 |
| unlockOnStartup | unlockOnStartup 告知 Solr 忽略在多线程环境中用来保护索引的锁定机制。在某些情况下，索引可能会由于不正确的关机或其他错误而一直处于锁定，这就妨碍了添加和更新。将其设置为 true 可以禁用启动锁定，进而允许进行添加和更新。 |
| lockType | single: 在只读索引或是没有其它进程修改索引时使用.  native: 使用操作系统本地文件锁,不能使用多个Solr在同一个JVM中共享一个索引.  simple :使用一个文本文件锁定索引. |

#### 5.3.2 查询配置query

|  |  |
| --- | --- |
| 属性 | 描述 |
| maxBooleanClauses | 最大的BooleanQuery数量. 当值超出时，抛出 TooManyClausesException.注意这个是全局的,如果是多个SolrCore都会使用一个值,每个Core里设置不一样的化,会使用最后一个的. |
| filterCache | filterCache存储了无序的lucene document id集合，存储了filter queries(“fq”参数)得到的document id集合结果，还可用于facet，如果配置了useFilterForSortedQuery，那么如果查询有filter，则使用filterCache |
| queryResultCache | 缓存搜索结果,一个文档ID列表 |
| documentCache | 缓存Lucene的Document对象,不会自热 |
| fieldValueCache | 字段缓存使用文档ID进行快速访问。默认情况下创建fieldValueCache即使这里没有配置。 |
| enableLazyFieldLoading | 若应用程序预期只会检索 Document 上少数几个 Field，那么可以将属性设置为 true。延迟加载的一个常见场景大都发生在应用程序返回和显示一系列搜索结果的时候，用户常常会单击其中的一个来查看存储在此索引中的原始文档。初始的显示常常只需要显示很短的一段信息。若考虑到检索大型 Document 的代价，除非必需，否则就应该避免加载整个文档。 |
| queryResultWindowSize | 一次查询中存储最多的doc的id数目. |
| queryResultMaxDocsCached | 查询结果doc的最大缓存数量, 例如要求每页显示10条,这里设置是20条,也就是说缓存里总会给你多出10条的数据.让你点示下一页时很快拿到数据. |
| listener | 选项定义 newSearcher 和 firstSearcher 事件，您可以使用这些事件来指定实例化新搜索程序或第一个搜索程序时应该执行哪些查询。如果应用程序期望请求某些特定的查询，那么在创建新搜索程序或第一个搜索程序时就应该反注释这些部分并执行适当的查询。 |
| useColdSearcher | 是否使用冷搜索, 为false时使用“预热”后的searcher |
| maxWarmingSearchers | 最大“预热”searcher数量 |

#### 5.3.3 <dataDir>

用于替换默认的索引数据目录(./data)，如果不是绝对路径，将使用servlet容器当前工作目录下的相对路径。

#### 5.3.4 更新处理器的配置<updateHandler>

该元素用于对更新处理器进行配置，更新处理器是用于更新索引数据的类，主要涉及底层的关于如何更新处理内部的信息。（不要跟高层次的配置参数Request Handlers对处理发自客户端的更新相混淆）

常用配置参数：

<updateHandler class="solr.DirectUpdateHandler2">

<autoCommit>

<maxDocs>10000</maxDocs>

<maxTime>1000</maxTime>

</autoCommit>

<listener event="postCommit" class="solr.RunExecutableListener">

<str name="exe">snapshooter</str>

<str name="dir">solr/bin</str>

<bool name="wait">true</bool>

<arr name="args"> <str>arg1</str> <str>arg2</str> </arr>

<arr name="env"> <str>MYVAR=val1</str> </arr>

</listener>

</updateHandler>

参数说明：

|  |  |
| --- | --- |
| 属性 | 描述 |
| class | 使用哪个类进行索引更新操作 |
| autoCommit | 在何时对索引数据进行提交（就是将内存中的索引写到硬盘）。上面的配置参数指明在内存中缓存的文档数目达到10000或时间达到1000时进行提交。 |
| listener | 配置一个“监听器”，这个“监听器”在特定的事件发生时将被触发 |

#### 5.3.5 ****<requestDispatcher>****

这里配置可以被HTTP的requestDispatcher使用的一些参数。

#### ****5.3.6 <requestHandler>****

可以定义多个请求处理器，这些处理器可以在查询和索引数据更新的时候使用。

#### 5.3.7 ****<queryResponseWriter>****

可以定义多个结果处理器，这些结果处理器会以不同的方式返回结果。

### 5.4Solr加入中文分词器

中文分词在solr里面是没有默认开启的，需要我们自己配置一个中文分词器。目前可用的分词器有smartcn，IK，Jeasy，庖丁。其实主要是两种，一种是基于中科院ICTCLAS的隐式马尔科夫HMM算法的中文分词器，如smartcn，ictclas4j，优点是分词准确度高，缺点是不能使用用户自定义词库；另一种是基于最大匹配的分词器，如IK ，Jeasy，庖丁，优点是可以自定义词库，增加新词，缺点是分出来的垃圾词较多。各有优缺点看应用场合自己衡量选择吧。

下面给出两种分词器的安装方法，任选其一即可，推荐第一种，因为smartcn就在solr发行包的contrib/analysis-extras/lucene-libs/下，就是lucene-analyzers-smartcn-4.7.0.jar,首选在solrconfig.xml中加一句引用analysis-extras的配置,这样我们自己加入的分词器才会引用到solr中.

<lib dir="../../../contrib/analysis-extras/lib" regex=".\*\.jar" />

#### 5.4.1. smartcn 分词器的安装

首选将发行包的contrib/analysis-extras/lucene-libs/ lucene-analyzers-smartcn-4.7.0.jar复制到\solr\contrib\analysis-extras\lib下,在solr本地应用文件夹下，打开/solr/conf/scheme.xml，编辑text字段类型如下，添加以下代码到scheme.xml中的相应位置，就是找到fieldType定义的那一段，在下面多添加这一段就好啦

<fieldType name="text\_smartcn" class="solr.TextField" positionIncrementGap="0">

<analyzer type="index">

<tokenizer class="org.apache.lucene.analysis.cn.smart.SmartChineseSentenceTokenizerFactory"/>

<filter class="org.apache.lucene.analysis.cn.smart.SmartChineseWordTokenFilterFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="org.apache.lucene.analysis.cn.smart.SmartChineseSentenceTokenizerFactory"/>

<filter class="org.apache.lucene.analysis.cn.smart.SmartChineseWordTokenFilterFactory"/>

</analyzer>

</fieldType>

如果需要检索某个字段，还需要在scheme.xml下面的field中，添加指定的字段，用text\_ smartcn作为type的名字，来完成中文分词。如 text要实现中文检索的话，就要做如下的配置：

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

#### 5.4.2. IK 分词器的安装

首选要去下载IKAnalyzer的发行包.下载地址: <http://ik-analyzer.googlecode.com/files/IK%20Analyzer%202012FF_hf1.zip>.

下载后解压出来文件中的三个复制到\solr\contrib\analysis-extras\lib目录中.

IKAnalyzer2012FF\_u1.jar 分词器jar包

IKAnalyzer.cfg.xml 分词器配置文件

Stopword.dic 分词器停词字典,可自定义添加内容

复制后就可以像smartcn一样的进行配置scheme.xml了.

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

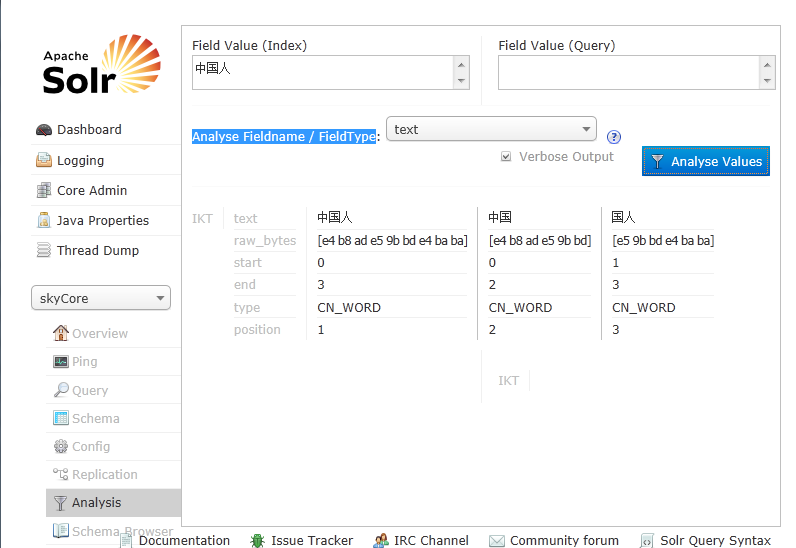
<analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>

</fieldType>

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

现在来验证下是否添加成功,首先使用StartSolrJetty来启动solr服务,启动过程中如果配置出错,一般有两个原因:一是配置的分词器jar找不到,也就是你没有复制jar包到\solr\contrib\analysis-extras\lib目前下;二是分词器版本不对导致的分词器接口API不一样出的错,要是这个错的话就在检查分词器的相关文档,看一下支持的版本是否一样.

如果在启动过程中没有报错的话说明配置成功了.我们可以进入到http://localhost:8080/solr地址进行测试一下刚加入的中文分词器.在首页的Core Selector中选择你配置的Croe后点击下面的Analysis,在Analyse Fieldname / FieldType里选择你刚才设置的字段名称或是分词器类型,在Field Value(index)中输入:中国人,点击右面的分词就行了.



## 6.Solr功能应用

这里使用SolrJ和浏览器请求url方式进行介绍Solr的一些基本应用。

使用SolrJ加上EmbeddedSolrServer(嵌入式服务器),方便进行代码跟踪调试.在功能上和其它服务器都是一样的,它们都是继承的SolrServer来提供服务API的. EmbeddedSolrServer优点是不用起http协议,直接加载SolrCore进行操作,性能上应该是最快的,方便用于把Solr单结点服务嵌入到项目中使用.下面开始介绍Solr的功能的应用.EmbeddedSolrServer初始化:

System.setProperty("solr.solr.home", "D: \\solr-4.7.0-src\\solr\\example\\solr");

CoreContainer.Initializer initializer = new CoreContainer.Initializer();

CoreContainer coreContainer = initializer.initialize();

SolrServer server = new EmbeddedSolrServer(coreContainer, "");

### 6.1维护索引

在一般系统中维护的都是增删改,在Solr中的维护功能是增删和优化功能,在Solr中的修改操作就是先删掉再添加.在做索引维护之前,首先要做的是配置schema.xml主要是按上面章节中的说明设置好字段信息(名称,类型,索引,存储,分词等信息),大概就像在数据库中新建一个表一样.设置好schema.xml就可以进行索引相关操作了.

#### 6.1.1增加索引

在增加索引之前先可构建好SolrInputDocument对象.主要操作就是给文档添加字段和值.代码如下:

SolrInputDocument doc = new SolrInputDocument();

doc.setField("id", "ABC");

doc.setField("content", "中华人民共和国");

构建好文档后添加的上面初始化好的server里就行了.

server.add(doc);

server.commit();//这句一般不用加因为我们可以通过在配置文件中的

//autoCommit来提高性能

Solr在add文档时.如果文档不存在就直接添加,如果文档存在就删除后添加,这也就是修改功能了.判断文档是否存在的依据是定义好的uniqueKey字段.

#### 6.1.2删除索引

删除指令有两种：

1） 通过文档ID进行删除，用 <id></id> 包装：

id 值是在schema.xml的uniqueKey所指字段的值，而不是索引内部的docId

2） 通过查询到的结果进行删除，用<query></query> 包装:

query 值是查询串，如：title:"solr"

方式1：url删除，使用stream相关参数

[http://localhost:8080/solr/db/update?stream.body=%3Cdelete%3E%3Cquery%3E\*:\*%3C/query%3E%3C/delete%3E&stream.contentType=text/xml;charset=utf-8&commit=true](http://localhost:8080/solr/db/update?stream.body=%3Cdelete%3E%3Cquery%3E*:*%3C/query%3E%3C/delete%3E&stream.contentType=text/xml;charset=utf-8&commit=true)

* stream相关参数还有：
  + stream.file=（服务器本地文件）
  + stream.url 分别指到你的删除文本
* 这里是直接字符串内容用 stream.body 参数。
* commit 参数是指提交，提交了才能看到删除效果。

方式2：使用post.jar删除

删除id为3的doc

java -Ddata=args -Durl=http://localhost:8080/solr/libDirTest/update -jar post.jar "<delete><id>3</id></delete>"

删除所有

java -Ddata=args -Durl=http://localhost:8080/solr/libDirTest/update -jar post.jar "<delete><query>\*:\*</query></delete>"

方式3：solrj

通过ID删除方式代码:

server.deleteById(id);

//或是使用批量删除

server.deleteById(ids);

通过查询删除方式代码:

server.deleteByQuery("\*.\*");//这样就删除了所有文档索引

//”\*.\*”就查询所有内容的,介绍查询时会详细说明.

#### 6.1.2优化索引

优化Lucene 的索引文件以改进搜索性能。索引完成后执行一下优化通常比较好。如果更新比较频繁，则应该在使用率较低的时候安排优化。一个索引无需优化也可以正常地运行。优化是一个耗时较多的过程。

server.optimize();//不要频繁的调用..尽量在无人使用时调用.

### 6.2查询索引

Solr在不修改任务配置的情况下就可以使用查询功能，在web项目中应用可以直接URL进行访问Solr服务器例如：

[http://localhost:8080/solr/ collection1/select?q=\*%3A\*&wt=xml&indent=true](http://localhost:8983/skyCore/select?q=*%3A*&wt=xml&indent=true)

上面的意思就是查询名为collection1的SolrCore的所有内容用xml格式返回并且有缩进。

返回结果如下:

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

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">0</int>

<lst name="params">

<str name="indent">true</str>

<str name="q">\*:\*</str>

<str name="wt">xml</str>

</lst>

</lst>

<result name="response" numFound="17971" start="0">

<doc>

<str name="path">E:\Reduced\军事\1539.txt</str>

<str name="category\_s">2</str>

<int name="browseCount\_i">-1423701734</int>

<long name="modified\_l">1162438568000</long>

<long name="releasedate\_l">1162438568000</long>

<str name="content"> [俄罗斯lenta网站2006年2月9日报道]俄空军副总司令比热耶夫中将称，2006年春天独联体国家防空系统打击范围向西推进150千米，侦察范围向西推进400千米。　　2006年3月白俄罗斯4个S-300PS防空导弹营担负战斗任务，使独联体防空系统作战范围得以向西推进。比热耶夫中将还宣布，近期乌兹别克斯坦可能加入独联体防空系统。　　独联体国家防空系统建于9年前，共有9个国家参加该组织。目前只有亚美尼亚、白俄罗斯、哈萨克斯坦、吉尔吉斯、俄罗斯和塔吉克斯坦支持该体系。　　乌克兰、乌兹别克斯坦与俄罗斯在双边基础上合作，格鲁吉亚和土库曼最近7年不参加独联体国家对空防御。</str>

<str name="id">E3798D82-EAB6-2BEA-D7E2-79FBD102E845</str>

<long name="\_version\_">1436361868021071872</long></doc>

…

</result>

</response>

上面所看到的就是用xml格式返回的查询结果,其中的doc就是一个文档,在doc里面的那个就是我们开始在schema.xml中定义的字段.

如果使用SolrJ进行调用的话代码如下：

SolrQuery query = new SolrQuery();

query.set("q","\*.\*");

QueryResponse rsp =server.query(query)

SolrDocumentList list = rsp.getResults();

返回结果在SolrDocumentList中在这个对象中遍历取出值来:

for (int i = 0; i < list.size(); i++) {

SolrDocument sd = list.get(i);

String id = (String) sd.getFieldValue("id");

System.out.println(id);

}

#### 6.2.1查询参数

|  |  |
| --- | --- |
| 名称 | 描述 |
| q | 查询字符串，必须的。 |
| fq | filter query。使用Filter Query可以充分利用Filter Query Cache，提高检索性能。作用：在q查询符合结果中同时是fq查询符合的，例如：q=mm&fq=date\_time:[20081001 TO 20091031]，找关键字mm，并且date\_time是20081001到20091031之间的。 |
| fl | field list。指定返回结果字段。以空格“ ”或逗号“,”分隔。 |
| start | 用于分页定义结果起始记录数，默认为0。 |
| rows | 用于分页定义结果每页返回记录数，默认为10。 |
| sort | 排序，格式:sort=<field name>+<desc|asc>[,<field name>+<desc|asc>]… 。示例：（inStock desc, price asc）表示先 “inStock” 降序, 再 “price” 升序，默认是相关性降序。 |
| df | 默认的查询字段，一般默认指定。 |
| q.op | 覆盖schema.xml的defaultOperator（有空格时用"AND"还是用"OR"操作逻辑），一般默认指定。必须大写 |
| wt | writer type。指定查询输出结构格式，默认为“xml”。在solrconfig.xml中定义了查询输出格式：xml、json、python、ruby、php、phps、custom。 |
| qt | query type，指定查询使用的Query Handler，默认为“standard”。 |
| explainOther | 设置当debugQuery=true时，显示其他的查询说明。 |
| defType | 设置查询解析器名称。 |
| timeAllowed | 设置查询超时时间。 |
| omitHeader | 设置是否忽略查询结果返回头信息，默认为“false”。 |
| indent | 返回的结果是否缩进，默认关闭，用 indent=true|on 开启，一般调试json,php,phps,ruby输出才有必要用这个参数。 |
| version | 查询语法的版本，建议不使用它，由服务器指定默认值。 |
| debugQuery | 设置返回结果是否显示Debug信息。 |

#### 6.2.2查询语法

1.匹配所有文档：\*:\*

2.强制、阻止和可选查询：

1) Mandatory：查询结果中必须包括的(for example, only entry name containing the word make)

Solr/Lucene Statement：+make, +make +up ,+make +up +kiss

2) prohibited：(for example, all documents except those with word believe)

Solr/Lucene Statement：+make +up -kiss

3) optional：

Solr/Lucene Statement：+make +up kiss

3.布尔操作：AND、OR和NOT布尔操作（必须大写）与Mandatory、optional和prohibited相似。

1) make AND up ＝ +make +up :AND左右两边的操作都是mandatory

2) make || up ＝ make OR up＝make up :OR左右两边的操作都是optional

3) +make +up NOT kiss ＝ +make +up –kiss

4) make AND up OR french AND Kiss不可以达到期望的结果，因为AND两边的操作都是mandatory的。

4. 子表达式查询（子查询）：可以使用“()”构造子查询。

示例：(make AND up) OR (french AND Kiss)

5.子表达式查询中阻止查询的限制：

示例：make (-up):只能取得make的查询结果；要使用make (-up \*:\*)查询make或者不包括up的结果。

6.多字段fields查询：通过字段名加上分号的方式（fieldName:query）来进行查询

示例：entryNm:make AND entryId:3cdc86e8e0fb4da8ab17caed42f6760c

7.通配符查询（wildCard Query）：

1) 通配符？和\*：“\*”表示匹配任意字符；“？”表示匹配出现的位置。

示例：ma?\*（ma后面的一个位置匹配），ma??\*(ma后面两个位置都匹配)

2) 查询字符必须要小写:+Ma +be\*\*可以搜索到结果；+Ma +Be\*\*没有搜索结果.

3) 查询速度较慢，尤其是通配符在首位：主要原因一是需要迭代查询字段中的每个term，判断是否匹配；二是匹配上的term被加到内部的查询，当terms数量达到1024的时候，查询会失败。

4) Solr中默认通配符不能出现在首位（可以修改QueryParser，设置

setAllowLeadingWildcard为true）

5) set setAllowLeadingWildcard to true.

8.模糊查询、相似查询：不是精确的查询，通过对查询的字段进行重新插入、删除和转换来取得得分较高的查询解决（由Levenstein Distance Algorithm算法支持）。

1) 一般模糊查询：示例：make-believ~

2) 门槛模糊查询：对模糊查询可以设置查询门槛，门槛是0~1之间的数值，门槛越高表面相似度越高。示例：make-believ~0.5、make-believ~0.8、make-believ~0.9

9.范围查询（Range Query）：Lucene支持对数字、日期甚至文本的范围查询。结束的范围可以使用“\*”通配符。

示例：

1) 日期范围（ISO-8601 时间GMT）：sa\_type:2 AND a\_begin\_date:[1990-01-01T00:00:00.000Z TO 1999-12-31T24:59:99.999Z]

2) 数字：salary:[2000 TO \*]

3) 文本：entryNm:[a TO a]

10.日期匹配：YEAR, MONTH, DAY, DATE (synonymous with DAY) HOUR, MINUTE, SECOND, MILLISECOND, and MILLI (synonymous with MILLISECOND)可以被标志成日期。

示例：

1) r\_event\_date:[\* TO NOW-2YEAR]：2年前的现在这个时间

2) r\_event\_date:[\* TO NOW/DAY-2YEAR]：2年前前一天的这个时间

#### 6.2.3函数查询（Function Query）

函数查询 可以利用 numeric字段的值 或者 与字段相关的的某个特定的值的函数，来对文档进行评分。

1. 使用函数查询的方法

这里主要有三种方法可以使用函数查询，这三种s方法都是通过solr http接口的。

1) 使用FunctionQParserPlugin。ie: q={!func}log(foo)

2) 使用“\_val\_”内嵌方法

内嵌在正常的solr查询表达式中。即，将函数查询写在 q这个参数中，这时候，我们使用“\_val\_”将函数与其他的查询加以区别。

ie：entryNm:make && \_val\_:ord(entryNm)

3) 使用dismax中的bf参数

使用明确为函数查询的参数，比如说dismax中的bf（boost function）这个参数。 注意：bf这个参数是可以接受多个函数查询的，它们之间用空格隔开，它们还可以带上权重。所以，当我们使用bf这个参数的时候，我们必须保证单个函数中是没有空格出现的，不然程序有可能会以为是两个函数。

示例：

q=dismax&bf="ord(popularity)^0.5 recip(rord(price),1,1000,1000)^0.3

2. 函数的格式（Function Query Syntax)

目前，function query 并不支持 a+b 这样的形式，我们得把它写成一个方法形式，这就是 sum(a,b).

3. 使用函数查询注意事项

1) 用于函数查询的field必须是被索引的；

2) 字段不可以是多值的（multi-value）

4. 可以利用的函数 （available function）

1) constant：支持有小数点的常量； 例如：1.5 ；SolrQuerySyntax:\_val\_:1.5

2) fieldvalue：这个函数将会返回numeric field的值，这个字段必须是indexd的，非multiValued的。格式很简单，就是该字段的名字。如果这个字段中没有这样的值，那么将会返回0。

3) ord：对于一个字段，它所有的值都将会按照字典顺序排列，这个函数返回你要查询的那个特定的值在这个顺序中的排名。这个字段，必须是非multiValued的，当没有值存在的时候，将返回0。例如：某个特定的字段只能去三个值，“apple”、“banana”、“pear”，那么ord（“apple”）=1，ord（“banana”）=2，ord（“pear”）=3.需要注意的是，ord（）这个函数，依赖于值在索引中的位置，所以当有文档被删除、或者添加的时候，ord（）的值就会发生变化。当你使用MultiSearcher的时候，这个值也就是不定的了。

4) rord：这个函数将会返回与ord相对应的倒排序的排名。

格式: rord(myIndexedField)。

5) sum：这个函数的意思就显而易见啦，它就是表示“和”啦。

格式：sum(x,1) 、sum(x,y)、 sum(sqrt(x),log(y),z,0.5)

6) product：product(x,y,...)将会返回多个函数的乘积。格式：product(x,2)、product(x,y)

7) div：div(x,y)表示x除以y的值，格式：div（1,x）、div(sum(x,100),max(y,1))

8) pow：pow表示幂值。pow(x,y) =x^y。例如：pow(x,0.5) 表示开方pow(x,log(y))

9) abs：abs(x)将返回表达式的绝对值。格式：abs(-5)、 abs(x)

10) log：log(x)将会返回基数为10，x的对数。格式： log(x)、 log(sum(x,100))

11) Sqrt：sqrt(x) 返回 一个数的平方根。格式：sqrt（2）、sqrt(sum(x,100))

12) Map：如果 x>=min,且x<=max,那么map(x,min,max,target)=target.如果 x不在[min,max]这个区间内，那么map(x,min,max,target)=x.

格式：map(x,0,0,1)

13) Scale：scale(x,minTarget,maxTarget) 这个函数将会把x的值限制在[minTarget,maxTarget]范围内。

14) query ：query(subquery,default)将会返回给定subquery的分数，如果subquery与文档不匹配，那么将会返回默认值。任何的查询类型都是受支持的。可以通过引用的方式，也可以直接指定查询串。

例子：q=product(popularity, query({!dismax v='solr rocks'}) 将会返回popularity和通过dismax 查询得到的分数的乘积。

q=product(popularity, query($qq)&qq={!dismax}solr rocks 跟上一个例子的效果是一样的。不过这里使用的是引用的方式

q=product(popularity, query($qq,0.1)&qq={!dismax}solr rocks 在前一个例子的基础上又加了一个默认值。

15) linear： inear(x,m,c)表示 m\*x+c ,其中m和c都是常量，x是一个变量也可以是一个函数。例如： linear(x,2,4)=2\*x+4.

16) Recip：recip(x,m,a,b)=a/(m\*x+b)其中，m、a、b是常量，x是变量或者一个函数。当a=b，并且x>=0的时候，这个函数的最大值是1，值的大小随着x的增大而减小。例如：recip(rord(creationDate),1,1000,1000)

17) Max： max(x,c)将会返回一个函数和一个常量之间的最大值。

例如：max(myfield,0)

### 6.3高亮显示

我们经常使用搜索引擎，比如在baidu 搜索 java ，会出现如下结果，结果中与关键字匹配的地方是红色显示与其他内容区别开来。



solr 默认已经配置了highlight 组件(详见 SOLR\_HOME/conf/sorlconfig.xml)。通常我出只需要这样请求：

http://localhost:8983/solr/ collection1 /select? q=%E4%B8%AD%E5%9B%BD&start=0&rows=1&fl=content+path+&wt=xml&indent=true&hl=true&hl.fl=content

可以看到与比一般的请求多了两个参数"hl=true"和"hl.fl=content "，"hl=true" 是开启高亮，"hl.fl= content " 是告诉solr 对 name 字段进行高亮(如果你想对多个字段进行高亮，可以继续添加字段，字段间用逗号隔开，如 "hl.fl=name,name2,name3")。 高亮内容与关键匹配的地方，默认将会被 "<em>" 和 "</em>" 包围。还可以使用hl.simple.pre" 和 "hl.simple.post"参数设置前后标签。

使用SolrJ方法基本一样也是设置这些个参数,只不过是SolrJ封装起来了,代码如下:

SolrQuery query = new SolrQuery();

query.set("q","\*.\*");

query.setHighlight(true); // 开启高亮组件

query.addHighlightField("content");// 高亮字段

query.setHighlightSimplePre(PRE\_TAG);// 标记

query.setHighlightSimplePost(POST\_TAG);

QueryResponse rsp =server.query(query)

//…上面取结果的代码

//取出高亮结果

if (rsp.getHighlighting() != null) {

if (rsp.getHighlighting().get(id) != null) {//先通过结果中的ID到高亮集合中取出文档高亮信息

Map<String, List<String>> map = rsp.getHighlighting().get(id);//取出高亮片段

if (map.get(name) != null) {

for (String s : map.get(name)) {

System.out.println(s);

}

}

}

### 6.4拼写检查

首先配置 solrconfig.xml，文件可能已经有这两个元素(如果没有添加即可)，需要根据我们自己的系统环境做些适当的修改。

<searchComponent name="spellcheck" class="solr.SpellCheckComponent">

<str name="queryAnalyzerFieldType">text\_spell</str>

<lst name="spellchecker">

<str name="name">direct</str>

<str name="field">spell</str>

<str name="classname">solr.DirectSolrSpellChecker</str>

<str name="distanceMeasure">internal</str>

<float name="accuracy">0.5</float>

<int name="maxEdits">2</int>

<int name="minPrefix">1</int>

<int name="maxInspections">5</int>

<int name="minQueryLength">2</int>

<float name="maxQueryFrequency">0.01</float>

</lst>

</searchComponent>

<requestHandler name="/spell" class="solr.SearchHandler" startup="lazy">

<lst name="defaults">

<str name="spellcheck.dictionary">direct</str>

<str name="spellcheck">on</str>

<str name="spellcheck.collate">true</str>

<str name="spellcheck.collateExtendedResults">true</str>

</lst>

<arr name="last-components">

<str>spellcheck</str>

</arr>

</requestHandler>

配置完成之后，我们进行一下测试,重启Solr后，访问如下链接

[http://localhost:8080/solr/ collection1/spell?wt=xml&indent=true&spellcheck=true&spellcheck.q=%E4%B8%AD%E5%9B%BD](http://localhost:8080/solr/%20collection1/spell?wt=xml&indent=true&spellcheck=true&spellcheck.q=%E4%B8%AD%E5%9B%BD)

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

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">0</int>

</lst>

<result name="response" numFound="0" start="0"/>

<lst name="spellcheck">

<lst name="suggestions">

<lst name="beijink">

<int name="numFound">1</int>

<int name="startOffset">0</int>

<int name="endOffset">3</int>

<arr name="suggestion">

<str>beijing</str>

</arr>

</lst>

</lst>

</lst>

</response>

使用SolrJ时也同样加入参数就可以

SolrQuery query = new SolrQuery();

query.set("q","\*.\*");

query.set("qt", "/spell");

QueryResponse rsp =server.query(query)

//…上面取结果的代码

SpellCheckResponse spellCheckResponse = rsp.getSpellCheckResponse();

if (spellCheckResponse != null) {

String collation = spellCheckResponse.getCollatedResult();

}

### 6.5检索建议

检索建议目前是各大搜索的标配应用，主要作用是避免用户输入错误的搜索词，同时将用户引导到相应的关键词搜索上。Solr内置了检索建议功能，它在Solr里叫做Suggest模块.该模块可选择基于提示词文本做检索建议，还支持通过针对索引的某个字段建立索引词库做检索建议。在诸多文档中都推荐使用基于索引来做检索建议，因此我们目前的实现也是采取该方案。

现在我们开始配置Suggest模块,首先在solrconfig.xml文件中配置Suggest依赖的SpellChecker模块，然后再配置Suggest模块,所以这两个都需要配置。

<searchComponent name="suggest" class="solr.SpellCheckComponent">

<str name="queryAnalyzerFieldType">string</str>

<lst name="spellchecker">

<str name="name">suggest</str>

<str name="classname">org.apache.solr.spelling.suggest.Suggester</str>

<str name="lookupImpl">org.apache.solr.spelling.suggest.tst.TSTLookup</str>

<str name="field">text</str>

<float name="threshold">0.0001</float>

<str name="spellcheckIndexDir">spellchecker</str>

<str name="comparatorClass">freq</str>

<str name="buildOnOptimize">true</str>

<!--<str name="buildOnCommit">true</str>-->

</lst>

</searchComponent>

<requestHandler name="/suggest" class="solr.SearchHandler" startup="lazy">

<lst name="defaults">

<str name="spellcheck">true</str>

<str name="spellcheck.dictionary">suggest</str>

<str name="spellcheck.onlyMorePopular">true</str>

<str name="spellcheck.extendedResults">false</str>

<str name="spellcheck.count">10</str>

<str name="spellcheck.collate">true</str>

</lst>

<arr name="components">

<str>suggest</str>

</arr>

</requestHandler>

配置完成之后，我们进行一下测试,重启Solr后，访问如下链接

[http://localhost:8080/solr/ collection1/suggest?wt=xml&indent=true&spellcheck=true&spellcheck.q=%E4%B8%AD%E5%9B%BD](http://localhost:8080/solr/%20collection1/suggest?wt=xml&indent=true&spellcheck=true&spellcheck.q=%E4%B8%AD%E5%9B%BD)

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

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">4</int>

</lst>

<lst name="spellcheck">

<lst name="suggestions">

<lst name="中国">

<int name="numFound">4</int>

<int name="startOffset">0</int>

<int name="endOffset">2</int>

<arr name="suggestion">

<str>中国队</str>

<str>中国证监会</str>

<str>中国足协</str>

<str>中国银行</str>

</arr>

</lst>

</lst>

</lst>

</response>

使用SolrJ时也同样加入参数就可以

SolrQuery query = new SolrQuery();

query.set("q", token);

query.set("qt", "/suggest");

query.set("spellcheck.count", "10");

QueryResponse response = server.query(query);

SpellCheckResponse spellCheckResponse = response.getSpellCheckResponse();

if (spellCheckResponse != null) {

List<SpellCheckResponse.Suggestion> suggestionList = spellCheckResponse.getSuggestions();

for (SpellCheckResponse.Suggestion suggestion : suggestionList) {

List<String> suggestedWordList = suggestion.getAlternatives();

for (int i = 0; i < suggestedWordList.size(); i++) {

String word = suggestedWordList.get(i);

}

}

return results;

}

通过threshold参数来限制一些不常用的词不出现在智能提示列表中，当这个值设置过大时，可能导致结果太少，需要引起注意。目前主要存在的问题是使用freq排序算法，返回的结果完全基于索引中字符的出现次数，没有兼顾用户搜索词语的频率，因此无法将一些热门词排在更靠前的位置。这块可定制SuggestWordScoreComparator来实现，目前还没有着手做这件事情。

### 6.6分组统计

我这里实现分组统计的方法是使用了Solr的Facet组件, Facet组件是Solr默认集成的一个组件.

#### 6.6.1 Facet简介

Facet是solr的高级搜索功能之一,可以给用户提供更友好的搜索体验.在搜索关键字的同时,能够按照Facet的字段进行分组并统计



#### 6.6.2 Facet字段

1. 适宜被Facet的字段

一般代表了实体的某种公共属性,如商品的分类,商品的制造厂家,书籍的出版商等等.

2. Facet字段的要求

Facet的字段必须被索引.一般来说该字段无需分词,无需存储.

无需分词是因为该字段的值代表了一个整体概念,如电脑的品牌”联想”代表了一个整 体概念,如果拆成”联”,”想”两个字都不具有实际意义.另外该字段的值无需进行大小 写转换等处理,保持其原貌即可.

无需存储是因为一般而言用户所关心的并不是该字段的具体值,而是作为对查询结果进 行分组的一种手段,用户一般会沿着这个分组进一步深入搜索.

3. 特殊情况

对于一般查询而言,分词和存储都是必要的.比如CPU类型”Intel 酷睿2双核 P7570”, 拆分成”Intel”,”酷睿”,”P7570”这样一些关键字并分别索引,可能提供更好的搜索 体验.但是如果将CPU作为Facet字段,最好不进行分词.这样就造成了矛盾,解决方法为, 将CPU字段设置为不分词不存储,然后建立另外一个字段为它的COPY,对这个COPY的 字段进行分词和存储.

<types>

<fieldType name="string" class="solr.StrField" omitNorms="true"/>

<fieldType name="tokened" class="solr.TextField" >

<analyzer>

……

</analyzer>

</fieldType>

</types>

<fields>

<field name="cpu" type="string" indexed="true" stored="false"/>

<field name="cpuCopy” type=" tokened" indexed="true" stored="true"/>

</fields>

<copyField source="cpu" dest="cpuCopy"/>

#### 6.6.2 Facet组件

Solr的默认requestHandler已经包含了Facet组件(solr.FacetComponent).如果自定义requestHandler或者对默认的requestHandler自定义组件列表,那么需要将Facet加入到组件列表中去.

<requestHandler name="standard" class="solr.SearchHandler" default="true">

……

<arr name="components">

<str>自定义组件名</str>

<str>facet</str>

……

</arr>

</requestHandler>

#### 6.6.2 Facet查询

进行Facet查询需要在请求参数中加入facet=on或者facet=true只有这样Facet组件才起作用.

1. Field Facet

Facet字段通过在请求中加入facet.field参数加以声明,如果需要对多个字段进行Facet查询,那么将该参数声明多次.例如:

[http://localhost:8080/solr/ collection1/select?q=\*%3A\*&start=0&rows=1&wt=xml&indent=true&facet=true&facet.field=category\_s&facet.field=modified\_l](http://localhost:8080/solr/%20collection1/select?q=*%3A*&start=0&rows=1&wt=xml&indent=true&facet=true&facet.field=category_s&facet.field=modified_l)

返回结果:

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

<response>

<lst name="responseHeader">

<int name="status">0</int>

<int name="QTime">1</int>

<lst name="params">

<str name="facet">true</str>

<str name="indent">true</str>

<str name="start">0</str>

<str name="q">\*:\*</str>

<arr name="facet.field">

<str>category\_s</str>

<str>modified\_l</str>

</arr>

<str name="wt">xml</str>

<str name="rows">0</str>

</lst>

</lst>

<result name="response" numFound="17971" start="0">

</result>

<lst name="facet\_counts">

<lst name="facet\_queries"/>

<lst name="facet\_fields">

<lst name="category\_s">

<int name="0">5991</int>

<int name="1">5990</int>

<int name="2">5990</int>

</lst>

<lst name="modified\_l">

<int name="1162438554000">951</int>

<int name="1162438556000">917</int>

<int name="1162438548000">902</int>

<int name="1162438546000">674</int>

</lst>

</lst>

<lst name="facet\_dates"/>

<lst name="facet\_ranges"/>

</lst>

</response>

各个Facet字段互不影响,且可以针对每个Facet字段设置查询参数.以下介绍的参数既可以应用于所有的Facet字段,也可以应用于每个单独的Facet字段.应用于单独的字段时通过

这种方式调用.比如facet.prefix参数应用于cpu字段,可以采用如下形式

f.字段名.参数名=参数值

f.cpu.facet.prefix=Intel

1.1 facet.prefix

表示Facet字段值的前缀.比如facet.field=cpu&facet.prefix=Intel,那么对cpu字段进行Facet查询,返回的cpu都是以Intel开头的, AMD开头的cpu型号将不会被统计在内.

1.2 facet.sort

表示Facet字段值以哪种顺序返回.可接受的值为true(count)|false(index,lex). true(count)表示按照count值从大到小排列. false(index,lex)表示按照字段值的自然顺序(字母,数字的顺序)排列.默认情况下为true(count).当facet.limit值为负数时,默认facet.sort= false(index,lex).

1.3 facet.limit

限制Facet字段返回的结果条数.默认值为100.如果此值为负数,表示不限制.

1.4 facet.offset

返回结果集的偏移量,默认为0.它与facet.limit配合使用可以达到分页的效果.

1.5 facet.mincount

限制了Facet字段值的最小count,默认为0.合理设置该参数可以将用户的关注点集中在少数比较热门的领域.

1.6 facet.missing

默认为””,如果设置为true或者on,那么将统计那些该Facet字段值为null的记录.

1.7 facet.method

取值为enum或fc,默认为fc.该字段表示了两种Facet的算法,与执行效率相关.

enum适用于字段值比较少的情况,比如字段类型为布尔型,或者字段表示中国的所有省份.Solr会遍历该字段的所有取值,并从filterCache里为每个值分配一个filter(这里要求solrconfig.xml里对filterCache的设置足够大).然后计算每个filter与主查询的交集.

fc(表示Field Cache)适用于字段取值比较多,但在每个文档里出现次数比较少的情况.Solr会遍历所有的文档,在每个文档内搜索Cache内的值,如果找到就将Cache内该值的count加1.

1.8 facet.enum.cache.minDf

当facet.method=enum时,此参数其作用,minDf表示minimum document frequency.也就是文档内出现某个关键字的最少次数.该参数默认值为0.设置该参数可以减少filterCache的内存消耗,但会增加总的查询时间(计算交集的时间增加了).如果设置该值的话,官方文档建议优先尝试25-50内的值.

#### 6.6.3 Date Facet

日期类型的字段在文档中很常见,如商品上市时间,货物出仓时间,书籍上架时间等等.某些情况下需要针对这些字段进行Facet.不过时间字段的取值有无限性,用户往往关心的不是某个时间点而是某个时间段内的查询统计结果. Solr为日期字段提供了更为方便的查询统计方式.当然,字段的类型必须是DateField(或其子类型).

需要注意的是,使用Date Facet时,字段名,起始时间,结束时间,时间间隔这4个参数都必须提供.与Field Facet类似,Date Facet也可以对多个字段进行Facet.并且针对每个字段都可以单独设置参数.

facet.date:该参数表示需要进行Date Facet的字段名,与facet.field一样,该参数可以被设置多次,表示对多个字段进行Date Facet.

facet.date.start:起始时间,时间的一般格式为1995-12-31T23:59:59Z,另外可以使用NOW\YEAR\ MONTH等等,具体格式可以参考DateField的java doc.

facet.date.end:结束时间.

facet.date.gap:时间间隔.如果start为2009-1-1,end为2010-1-1.gap设置为+1MONTH表示间隔1个月,那么将会把这段时间划分为12个间隔段.

注意+因为是特殊字符所以应该用%2B代替.

facet.date.hardend:取值可以为true|false,默认为false.它表示gap迭代到end处采用何种处理.举例说明start为2009-1-1,end为2009-12-25,gap为+1MONTH,

hardend为false的话最后一个时间段为2009-12-1至2010-1-1;

hardend为true的话最后一个时间段为2009-12-1至2009-12-25.

facet.date.other:取值范围为before|after|between|none|all,默认为none.before会对start之前的值做统计.after会对end之后的值做统计.between会对start至end之间所有值做统计.如果hardend为true的话,那么该值就是各个时间段统计值的和.none表示该项禁用.all表示before,after,all都会统计.

举例:

&facet=on

&facet.date=date

&facet.date.start=2009-1-1T0:0:0Z

&facet.date.end=2010-1-1T0:0:0Z

&facet.date.gap=%2B1MONTH

&facet.date.other=all

返回结果:

<lst name="facet\_counts">

<lst name="facet\_queries"/>

<lst name="facet\_fields"/>

<lst name="facet\_dates">

<int name="2009-01-01T00:00:00Z">5</int>

<int name="2009-02-01T00:00:00Z">7</int>

<int name="2009-03-01T00:00:00Z">4</int>

<int name="2009-04-01T00:00:00Z">3</int>

<int name="2009-05-01T00:00:00Z">7</int>

<int name="2009-06-01T00:00:00Z">3</int>

<int name="2009-07-01T00:00:00Z">6</int>

<int name="2009-08-01T00:00:00Z">7</int>

<int name="2009-09-01T00:00:00Z">2</int>

<int name="2009-10-01T00:00:00Z">4</int>

<int name="2009-11-01T00:00:00Z">1</int>

<int name="2009-12-01T00:00:00Z">5</int>

<str name="gap">+1MONTH</str>

<date name="end">2010-01-01T00:00:00Z</date>

<int name="before">180</int>

<int name="after">5</int>

<int name="between">54</int>

</lst>

</lst>

#### 6.6.4 Facet Query

Facet Query利用类似于filter query的语法提供了更为灵活的Facet.通过facet.query参数,可以对任意字段进行筛选.

例1:

&facet=on

&facet.query=date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]

&facet.query=date:[2009-4-1T0:0:0Z TO 2009-5-1T0:0:0Z]

返回结果:

<lst name="facet\_counts">

<lst name="facet\_queries">

<int name="date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]">5</int>

<int name="date:[2009-4-1T0:0:0Z TO 2009-5-1T0:0:0Z]">3</int>

</lst>

<lst name="facet\_fields"/>

<lst name="facet\_dates"/>

</lst>

例2:

&facet=on

&facet.query=date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]

&facet.query=price:[\* TO 5000]

返回结果:

<lst name="facet\_counts">

<lst name="facet\_queries">

<int name="date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]">5</int>

<int name="price:[\* TO 5000]">116</int>

</lst>

<lst name="facet\_fields"/>

<lst name="facet\_dates"/>

</lst>

例3:

&facet=on

&facet.query=cpu:[A TO G]

返回结果:

<lst name="facet\_counts">

<lst name="facet\_queries">

<int name="cpu:[A TO G]">11</int>

</lst>

<lst name="facet\_fields"/>

<lst name="facet\_dates"/>

</lst>

#### 6.6.5 key操作符

可以用key操作符为Facet字段取一个别名.

例:

&facet=on

&facet.field={!key=中央处理器}cpu

&facet.field={!key=显卡}videoCard

返回结果:

<lst name="facet\_counts">

<lst name="facet\_queries"/>

<lst name="facet\_fields">

<lst name="中央处理器">

<int name="Intel 酷睿2双核 T6600">48</int>

<int name="Intel 奔腾双核 T4300">28</int>

<int name="Intel 酷睿2双核 P8700">18</int>

<int name="Intel 酷睿2双核 T6570">11</int>

<int name="Intel 酷睿2双核 T6670">11</int>

<int name="Intel 奔腾双核 T4400">9</int>

<int name="Intel 酷睿2双核 P7450">9</int>

<int name="Intel 酷睿2双核 T5870">8</int>

<int name="Intel 赛扬双核 T3000">7</int>

<int name="Intel 奔腾双核 SU4100">6</int>

<int name="Intel 酷睿2双核 P8400">6</int>

<int name="Intel 酷睿2双核 SU7300">5</int>

<int name="Intel 酷睿 i3 330M">4</int>

</lst>

<lst name="显卡">

<int name="ATI Mobility Radeon HD 4">63</int>

<int name="NVIDIA GeForce G 105M">24</int>

<int name="NVIDIA GeForce GT 240M">21</int>

<int name="NVIDIA GeForce G 103M">8</int>

<int name="NVIDIA GeForce GT 220M">8</int>

<int name="NVIDIA GeForce 9400M G">7</int>

<int name="NVIDIA GeForce G 210M">6</int>

</lst>

</lst>

<lst name="facet\_dates"/>

</lst>

#### 6.6.6 tag操作符和ex操作符

当查询使用filter query的时候,如果filter query的字段正好是Facet字段,那么查询结果往往被限制在某一个值内.

例:

&fq=screenSize:14

&facet=on

&facet.field=screenSize

返回结果:

<lst name="facet\_counts">

<lst name="facet\_queries"/>

<lst name="facet\_fields">

<lst name=" screenSize">

<int name="14.0">107</int>

<int name="10.2">0</int>

<int name="11.1">0</int>

<int name="11.6">0</int>

<int name="12.1">0</int>

<int name="13.1">0</int>

<int name="13.3">0</int>

<int name="14.1">0</int>

<int name="15.4">0</int>

<int name="15.5">0</int>

<int name="15.6">0</int>

<int name="16.0">0</int>

<int name="17.0">0</int>

<int name="17.3">0</int>

</lst>

</lst>

<lst name="facet\_dates"/>

</lst>

可以看到,屏幕尺寸(screenSize)为14寸的产品共有107件,其它尺寸的产品的数目都是0,这是因为在filter里已经限制了screenSize:14.这样,查询结果中,除了screenSize=14的这一项之外,其它项目没有实际的意义.有些时候,用户希望把结果限制在某一范围内,又希望查看该范围外的概况.比如上述情况,既要把查询结果限制在14寸屏的笔记本,又想查看一下其它屏幕尺寸的笔记本有多少产品.这个时候需要用到tag和ex操作符.tag就是把一个filter标记起来,ex(exclude)是在Facet的时候把标记过的filter排除在外.

例:

&fq={!tag=aa}screenSize:14

&facet=on

&facet.field={!ex=aa}screenSize

返回结果:

<lst name="facet\_counts">

<lst name="facet\_queries"/>

<lst name="facet\_fields">

<lst name=" screenSize">

<int name="14.0">107</int>

<int name="14.1">40</int>

<int name="13.3">34</int>

<int name="15.6">22</int>

<int name="15.4">8</int>

<int name="11.6">6</int>

<int name="12.1">5</int>

<int name="16.0">5</int>

<int name="15.5">3</int>

<int name="17.0">3</int>

<int name="17.3">3</int>

<int name="10.2">1</int>

<int name="11.1">1</int>

<int name="13.1">1</int>

</lst>

</lst>

<lst name="facet\_dates"/>

</lst>

这样其它屏幕尺寸的统计信息就有意义了.

#### 6.6.7 SolrJ对Facet的支持

//初始化查询对象

String q = “\*.\*”;

SolrQuery query = new SolrQuery(q);

query.setIncludeScore(false);//是否按每组数量高低排序

query.setFacet(true);//是否分组查询

query.setRows(0);//设置返回结果条数，如果你时分组查询，你就设置为0

query.addFacetField(“modified\_l”);//增加分组字段 q

query.addFacetQuery (“category\_s[0 TO 1]”);

QueryResponse rsp = server.query(query);

…

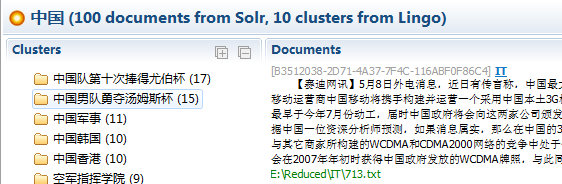
//取出结果

List<FacetField.Count> list = rsp.getFacetField(“modified\_l”).getValues();

Map<String, Integer> list = rsp.getFacetQuery();

### 6.7自动聚类

Solr 使用Carrot2完成了聚类功能,能够把检索到的内容自动分类, Carrot2聚类示例:



要想Solr支持聚类功能,首选要把Solr发行包的中的dist/ solr-clustering-4.2.0.jar, 复制到\solr\contrib\analysis-extras\lib下.然后打开solrconfig.xml进行添加配置:

<searchComponent name="clustering"

enable="${solr.clustering.enabled:true}"

class="solr.clustering.ClusteringComponent" >

<lst name="engine">

<str name="name">default</str>

<str name="carrot.algorithm">org.carrot2.clustering.lingo.LingoClusteringAlgorithm</str>

<str name="LingoClusteringAlgorithm.desiredClusterCountBase">30</str><!--2~100-->

<str name="LingoClusteringAlgorithm.clusterMergingThreshold">0.70</str><!--0~1-->

<str name="LingoClusteringAlgorithm.scoreWeight">0</str><!--0~1-->

<str name="LingoClusteringAlgorithm.labelAssigner">org.carrot2.clustering.lingo.SimpleLabelAssigner</str><!--org.carrot2.clustering.lingo.UniqueLabelAssigner -->

<str name="LingoClusteringAlgorithm.phraseLabelBoost">1.5</str><!--0~10-->

<str name="LingoClusteringAlgorithm.phraseLengthPenaltyStart">8</str><!--2~8-->

<str name="LingoClusteringAlgorithm.phraseLengthPenaltyStop">8</str><!--2~8-->

<str name="TermDocumentMatrixReducer.factorizationQuality">HIGH</str><!--LOW,MEDIUM,HIGH-->

<!--

org.carrot2.matrix.factorization.PartialSingularValueDecompositionFactory

org.carrot2.matrix.factorization.NonnegativeMatrixFactorizationEDFactory

org.carrot2.matrix.factorization.NonnegativeMatrixFactorizationKLFactory

org.carrot2.matrix.factorization.LocalNonnegativeMatrixFactorizationFactory

org.carrot2.matrix.factorization.KMeansMatrixFactorizationFactory

-->

<str name="TermDocumentMatrixReducer.factorizationFactory">org.carrot2.matrix.factorization.NonnegativeMatrixFactorizationEDFactory</str>

<str name="TermDocumentMatrixBuilder.maximumMatrixSize">37500</str><!--MinValue5000-->

<str name="TermDocumentMatrixBuilder.titleWordsBoost">2.0</str><!--2~10-->

<str name="TermDocumentMatrixBuilder.maxWordDf">0.9</str><!--0~1-->

<!--org.carrot2.text.vsm.LogTfIdfTermWeighting,org.carrot2.text.vsm.LinearTfIdfTermWeighting-->

<str name="TermDocumentMatrixBuilder.termWeighting">org.carrot2.text.vsm.TfTermWeighting</str>

<str name="MultilingualClustering.defaultLanguage">CHINESE\_SIMPLIFIED</str>

<str name="MultilingualClustering.languageAggregationStrategy">org.carrot2.text.clustering.MultilingualClustering.LanguageAggregationStrategy.FLATTEN\_MAJOR\_LANGUAGE </str><!--FLATTEN\_ALL,FLATTEN\_NONE-->

<str name="GenitiveLabelFilter.enabled">true</str>

<str name="StopWordLabelFilter.enabled">true</str>

<str name="NumericLabelFilter.enabled">true</str>

<str name="QueryLabelFilter.enabled">true</str>

<str name="MinLengthLabelFilter.enabled">true</str>

<str name="StopLabelFilter.enabled">true</str>

<str name="CompleteLabelFilter.enabled">true</str>

<str name="CompleteLabelFilter.labelOverrideThreshold">0.65</str><!--0~1-->

<str name="DocumentAssigner.exactPhraseAssignment">false</str>

<str name="DocumentAssigner.minClusterSize">2</str><!--1~100-->

<str name="merge-resources">true</str>

<str name="CaseNormalizer.dfThreshold">1</str><!--1~100-->

<str name="PhraseExtractor.dfThreshold">1</str><!--1~100-->

<str name="carrot.lexicalResourcesDir">clustering/carrot2</str>

<str name="SolrDocumentSource.solrIdFieldName">id</str>

</lst>

</searchComponent>

配好了聚类组件后,下面配置requestHandler:

<requestHandler name="/clustering"

startup="lazy"

enable="${solr.clustering.enabled:true}"

class="solr.SearchHandler">

<lst name="defaults">

<str name="echoParams">explicit</str>

<bool name="clustering">true</bool>

<str name="clustering.engine">default</str>

<bool name="clustering.results">true</bool>

<str name="carrot.title">category\_s</str>

<str name="carrot.snippet">content</str>

<str name="carrot.url">path</str>

<str name="carrot.produceSummary">true</str>

</lst>

<arr name="last-components">

<str>clustering</str>

</arr>

</requestHandler>

有两个参数要注意carrot.title, carrot.snippet是聚类的比较计算字段,这两个参数必须是stored="true".carrot.title的权重要高于carrot.snippet,如果只有一个做计算的字段carrot.snippet可以去掉(是去掉不是值为空).设完了用下面的URL就可以查询了

http://localhost:8080/skyCore/clustering?q=\*%3A\*&wt=xml&indent=true

### 6.8相似匹配

　　在我们使用网页搜索时，会注意到每一个结果都包含一个 “相似页面” 链接，单击该链接，就会发布另一个搜索请求，查找出与起初结果类似的文档。Solr 使用 MoreLikeThisComponent（MLT）和 MoreLikeThisHandler 实现了一样的功能。如上所述，MLT 是与标准 SolrRequestHandler 集成在一起的；MoreLikeThisHandler 与 MLT 结合在一起，并添加了一些其他选项，但它要求发布一个单一的请求。我将着重讲述 MLT，因为使用它的可能性更大一些。幸运的是，不需要任何设置就可以查询它，所以您现在就可以开始查询。

　　MLT 要求字段被储存或使用检索词向量，检索词向量以一种以文档为中心的方式储存信息。MLT 通过文档的内容来计算文档中关键词语，然后使用原始查询词语和这些新词语创建一个新的查询。提交新查询就会返回其他查询结果。所有这些都可以用检索词向量来完成：只需将 termVectors="true" 添加到 schema.xml 中的 <field> 声明。

MoreLikeThisComponent 参数：

|  |  |  |
| --- | --- | --- |
| **参数** | **说明** | **值域** |
| mlt | 在查询时，打开/关闭 MoreLikeThisComponent 的布尔值。 | true|false |
| mlt.count | 可选。每一个结果要检索的相似文档数。 | > 0 |
| mlt.fl | 用于创建 MLT 查询的字段。 | 任何被储存的或含有检索词向量的字段。 |
| mlt.maxqt | 可选。查询词语的最大数量。由于长文档可能会有很多关键词语，这样 MLT 查询可能会很大，从而导致反应缓慢或可怕的 TooManyClausesException，该参数只保留关键的词语。 | > 0 |

要想使用匹配相似首先在 solrconfig.xml 中配置 MoreLikeThisHandler

<requestHandler name="/mlt" class="solr.MoreLikeThisHandler">   
</requestHandler>

然后我就可以请求

<http://localhost:8080/skyCore/mlt?q=id%3A6F398CCD-2DE0-D3B1-9DD6-D4E532FFC531&mlt.true&mlt.fl=content&wt=xml&indent=true>

上面请求的意思查找 id 为 6F398CCD-2DE0-D3B1-9DD6-D4E532FFC531 的 document ,然后返回与此 document 在 name 字段上相似的其他 document。需要注意的是 mlt.fl 中的 field 的 termVector=true 才有效果

<field name="content" type="text\_smartcn" indexed="false" stored="true" multiValued="false" termVector="true"/>

使用SolrJ时也同样加入参数就可以

SolrQuery query = new SolrQuery();

query.set("qt", "/mlt");

query.set("mlt.fl","content");

query.set("fl", "id,");

query.set("q", "id: 6F398CCD-2DE0-D3B1-9DD6-D4E532FFC531");

query.setStart(0);

query.setRows(5);

QueryResponse rsp = server.query(query);

SolrDocumentList list = rsp.getResults();

### 6.9拼音检索

拼音检索中国人的专用检索,例如:中文内容为 中国 的输入zhongguo、zg、zhonggu 全拼、简拼、拼音的相邻的一部份都应该能检索出 中国 来。

想要实现拼音检索第一个就是拼音转换我这里用的是pinyin4j进行拼音转换。第二个就是N-Gram的题目，推敲到用户可能输入的既不是前缀也不是后缀，所以此处选择的是N-Gram技巧，但不同于常用的N-Gram，我应用的从一边开端的单向的N-Gram，Solr里的实现叫EdgeNGramTokenFilter，但是分的分的太细了，不需要这么复杂EdgeNGramTokenFilter,也就是说我们用的N-Gram不同于传统的N-Gram。

同样的例子使用EdgeNGramTokenFilter从前往后取2-Gram的结果是zh, 一般是取min–max之间的所有gram，所以使用EdgeNGramTokenFilter取2-20的gram结果就是zh,zho, zhon, zhong, zhongg, zhonggu, zhongguo, 从这个例子也不难理解为什么我要选择使用EdgeNGramTokenFilter而非一般意义上的N-Gram， 考虑到用户可能输入的不是前缀而是后缀，所以为了照顾这些用户，我选择了从前往后和从后往前使用了两次EdgeNGramTokenFilter，这样不只是前缀、后缀，二十任意的字串都考虑进去了，所以大幅度的提高了搜索体验.

现在思路明确了我们把它结合到Solr中，为了方便使用现在写了两个Filter进行处理拼音分词问题一个是拼音转换Filter（PinyinTransformTokenFilter）一个是拼音N-Gram的Filter(PinyinNGramTokenFilter),这样一来使用时就不用在添加索引前做拦音的转换了。而且PinyinTransformTokenFilter还有个好处就是它只使用中文分词器分过的词，也就是说做转换的词都是有用的不重复的，不会对没用的停词类的做拼音转换和重复拼音转换，这样大大的提高了拼音转换速度。

想要Solr支持拼音检索就要先把拼音分词（PinyinAnalyzer）的jar复制到\solr\contrib\analysis-extras\lib下，然后在schema.xml中配置一个拼音字段类型：

<fieldType name="text\_pinyin" class="solr.TextField" positionIncrementGap="0">

<analyzer type="index">

<tokenizer class="org.apache.lucene.analysis.cn.smart.SmartChineseSentenceTokenizerFactory"/>

<filter class="org.apache.lucene.analysis.cn.smart.SmartChineseWordTokenFilterFactory"/>

<filter class="com.shentong.search.analyzers.PinyinTransformTokenFilterFactory" minTermLenght="2" />

<filter class="com.shentong.search.analyzers.PinyinNGramTokenFilterFactory" minGram="1" maxGram="20" />

</analyzer>

<analyzer type="query">

<tokenizer class="org.apache.lucene.analysis.cn.smart.SmartChineseSentenceTokenizerFactory"/>

<filter class="org.apache.lucene.analysis.cn.smart.SmartChineseWordTokenFilterFactory"/>

<filter class="com.shentong.search.analyzers.PinyinTransformTokenFilterFactory" minTermLenght="2" />

<filter class="com.shentong.search.analyzers.PinyinNGramTokenFilterFactory" minGram="1" maxGram="20" />

</analyzer>

</fieldType>

minTermLenght：最小中文词长度，意思是小于这个值的中文词不会做拼音转换。

minGram：最小拼音切分长度。  
如果想使用简拼的话在拼音转换Filter 使用这个参数isFirstChar="true"就可以了

在这个拼音类型中我们使用了smartcn的中言语分词器，如果想使用其它的自己换掉就行了。现在我们在原来索引中加入一个拼音字段，因为只做索引,我们可以这样配置:

<field name ="pinyin" type ="text\_pinyin" indexed ="true" stored ="false" multiValued ="false"/>

加完后我们重新启动Solr测试一下看看



由于上面minTermLenght和minGram设置的值，现在出现了人没有进行拼音转换并且最小拼音切分是从1个开始的。

到这里我们的配置还有没完成呢，还要加几个copyFiled，这样就不用单独处理我们新加的拼音字段了。方便呀~~~

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

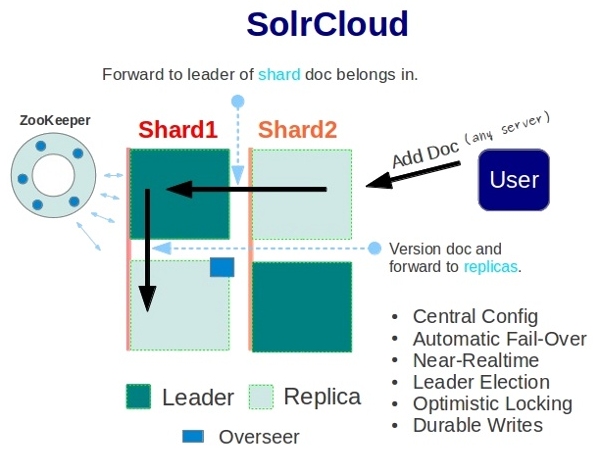
<copyField source="text" dest="spell"/>

到现在就可以使用拼音检索了。

拼音分词器jar点击并复制就可以粘出去了.

### 6.10 SolrCloud

SolrCloud是基于Solr和Zookeeper的分布式搜索方案，是正在开发中的Solr4.0的核心组件之一，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,集中式的配置信息、自动容错 、近实时搜索 、查询时自动负载均衡。



基本可以用上面这幅图来概述，这是一个拥有4个Solr节点的集群，索引分布在两个Shard里面，每个Shard包含两个Solr节点，一个是Leader节点，一个是Replica节点，此外集群中有一个负责维护集群状态信息的Overseer节点，它是一个总控制器。集群的所有状态信息都放在Zookeeper集群中统一维护。从图中还可以看到，任何一个节点都可以接收索引更新的请求，然后再将这个请求转发到文档所应该属于的那个Shard的Leader节点，Leader节点更新结束完成，最后将版本号和文档转发给同属于一个Shard的replicas节点。这里就不多说SolrCloud了，等研究明白后再单写一个文档。

## 7. 案例：solr从数据库建立索引

这里需要3个文件，数据库配置文件db-data-config.xml，该文件的路径和名可以在solrConfig.xml中配置，而schema.xml配置一些字段的类型、属性以及如何处理该字段。

### 7.1 配置一个handler

这个handler首先要在solrConfig.xml中配置，如下所示：

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

<lst name="defaults">

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

</lst>

</requestHandler>

从它的名字上，可以发现，DataImportHandler正是requestHandler的实现，这里定义的请求处理器是用来索引数据和更新的。

可以看到，config为db-data-config.xml，而这个文件主要负责：

怎样获取数据？（查询语句，url等）

要读什么样的数据？（关系数据库中的列，或者xml的域）

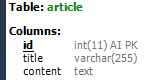
做什么样的处理？（修改、添加、删除）

下面就需要配置db-data-config.xml文件了。

### 7.2 数据库配置db-data-config.xml

这里需要配置数据源，字段，以及取得字段的方法。

当mysql数据库中表结构如下时：



可以这么配置：

<dataConfig>

<dataSource name="jdbc" driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost/test" user="root" password="root" />

<document>

<entity name="article" pk="id"

query="select id,title,content from article">

<field column="id" name="id" />

<field column="title" name="title" />

<field column="content" name="content" />

</entity>

</document>

</dataConfig>

* **dataSource**：数据源，也可配置在solrConfig.xml中
* **type**：指定了实现的类型，是可选的，默认是“JdbcDataSource”
* **name**：是dataSource的名字，当有多个datasource时，可使用name属性加以区分
* **document**：包含一个或多个entity
* **entity**：可理解为javabean或数据库中的表
* **field**：可理解为属性名或列名
* **query**：如何获取数据
* **column**：表的列名
* **name**：schema.xml中所配置的字段名

这些field在schema.xml中配置如下：

<field name="id" type="text\_general" indexed="true" stored="true" required="true" multiValued="false" />

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

<field name="content" type="text\_general" indexed="true" stored="false" multiValued="true"/>

这里id，title，content都是text\_general类型，都建立索引（indexed="true"），但是content一般内容较多因此不存储（stored="false"），id为唯一的（<uniqueKey>id</uniqueKey>），因此不允许多值。

配置已经完成，下面只需要将需要的jar包导入。

这里除mysql的jar包外还需要复制solr-4.7.0\dist下的

solr-dataimporthandler-4.7.0.jar和solr-dataimporthandler-extras-4.7.0.jar

至webapps\solr\WEB-INF\lib。

### 7.3 建立索引

使用url来建立索引：

[http://localhost:8080/solr/**db**/dataimport?command=full-import&commit=ture](http://localhost:8080/solr/db/dataimport?command=full-import&commit=ture)

上面的db表示使用db核心，上面所有的配置都是在db核心中进行的，因此这里也就是从数据库中建立索引，dataimport就是上面所建立的handler，command指明使用何种命令，command=full-import就表明全量导入，commit表示是否提交。

### 7.4 搜索

使用url搜索关键词为“test”的文档信息：

[http://localhost:8080/solr/**db**/select?q=test&wt=xml&indent=true](http://localhost:8080/solr/db/select?q=test&wt=xml&indent=true)

q表示搜索的关键词，wt（writer type）指定查询输出结构格式，indent表示返回的结果是否缩进。

### 7.5 配置参考

#### solrConfig.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <config>  <luceneMatchVersion>4.7</luceneMatchVersion>  <jmx />  <schemaFactory class="ClassicIndexSchemaFactory"/>  <!-- 对更新处理器进行配置，用于更新索引数据的类，  主要涉及底层的关于如何更新处理内部的信息。  （不要跟高层次的配置参数Request Handlers对处理  发自客户端的更新相混淆）-->  <updateHandler class="solr.DirectUpdateHandler2">  <maxPendingDeletes>100000</maxPendingDeletes>  </updateHandler>  <!-- 查询配置 -->  <query>  <!-- 最大的BooleanQuery数量. 当值超出时，抛出  TooManyClausesException.注意这个是全局的,如果  是多个SolrCore都会使用一个值,每个Core里设置不  一样的化,会使用最后一个的 -->  <maxBooleanClauses>1024</maxBooleanClauses>  <!-- filterCache存储了无序的lucene document id  集合，存储了filter queries(“fq”参数)得到的  document id集合结果，还可用于facet，如果配置了  useFilterForSortedQuery，那么如果查询有filter，  则使用filterCache -->  <filterCache  class="solr.LRUCache"  size="512"  initialSize="512"  autowarmCount="256"/>  <!-- 缓存搜索结果,一个文档ID列表 -->  <queryResultCache  class="solr.LRUCache"  size="512"  initialSize="512"  autowarmCount="256"/>  <!-- 缓存Lucene的Document对象,不会自热 -->  <documentCache  class="solr.LRUCache"  size="512"  initialSize="512"  autowarmCount="0"/>  <!-- 若应用程序预期只会检索 Document 上少数几个 Field，  那么可以将属性设置为 true。延迟加载的一个常见场景大都发  生在应用程序返回和显示一系列搜索结果的时候，用户常常会  单击其中的一个来查看存储在此索引中的原始文档。初始的显示  常常只需要显示很短的一段信息。若考虑到检索大型 Document  的代价，除非必需，否则就应该避免加载整个文档 -->  <enableLazyFieldLoading>true</enableLazyFieldLoading>  <!-- 一次查询中存储最多的doc的id数目 -->  <queryResultWindowSize>50</queryResultWindowSize>  <!-- 查询结果doc的最大缓存数量, 例如要求每页显示10条,这里设置是20条,  也就是说缓存里总会给你多出10条的数据.让你点示下一页时很快拿到数据. -->  <queryResultMaxDocsCached>200</queryResultMaxDocsCached>  <!-- 设置哈希表的大小为3000，当元素数目接近75%时，效率是最高的 -->  <HashDocSet maxSize="3000" loadFactor="0.75"/>  <!-- 选项定义 newSearcher 和 firstSearcher 事件，您可以使用这些事件  来指定实例化新搜索程序或第一个搜索程序时应该执行哪些查询。如果应用  程序期望请求某些特定的查询，那么在创建新搜索程序或第一个搜索程序时  就应该反注释这些部分并执行适当的查询 -->  <listener event="newSearcher" class="solr.QuerySenderListener">  <arr name="queries">  <lst> <str name="q">solr</str> <str name="start">0</str> <str name="rows">10</str> </lst>  <lst> <str name="q">rocks</str> <str name="start">0</str> <str name="rows">10</str> </lst>  <lst><str name="q">static newSearcher warming query from solrconfig.xml</str></lst>  </arr>  </listener>  <listener event="firstSearcher" class="solr.QuerySenderListener">  <arr name="queries">  </arr>  </listener>  <!-- 是否使用冷搜索, 为false时使用“预热”后的searcher -->  <useColdSearcher>false</useColdSearcher>  <!-- 最大“预热”searcher数量 -->  <maxWarmingSearchers>4</maxWarmingSearchers>  </query>  <!-- 这里配置可以被HTTP的requestDispatcher使用的一些参数 -->  <requestDispatcher handleSelect="true" >  <requestParsers enableRemoteStreaming="false" multipartUploadLimitInKB="2048" formdataUploadLimitInKB="2048" />  <httpCaching never304="true">  </httpCaching>  </requestDispatcher>  <!-- 可以定义多个请求处理器，这些处理器可以在查询和索引数据更新的时候使用 -->  <requestHandler name="standard" class="solr.StandardRequestHandler" default="true">  <lst name="defaults">  <str name="echoParams">explicit</str>  </lst>  </requestHandler>  <!-- 拼写检查，对不正确的拼写给出正确的拼写建议 -->  <requestHandler name="spellchecker" class="solr.SpellCheckerRequestHandler" startup="lazy">  <lst name="defaults">  <int name="suggestionCount">1</int>  <float name="accuracy">0.5</float>  </lst>  <str name="spellcheckerIndexDir">spell</str>  <str name="termSourceField">word</str>    </requestHandler>  <!-- 相似度的判断 -->  <requestHandler name="/mlt" class="solr.MoreLikeThisHandler">  <lst name="defaults">  <str name="mlt.fl">manu,cat</str>  <int name="mlt.mindf">1</int>  </lst>  </requestHandler>  <!-- 数据导入的请求处理器 -->  <requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">  <lst name="defaults">  <str name="config">db-data-config.xml</str>  </lst>  </requestHandler>  <!-- 查找的请求处理器 -->  <requestHandler name="/search" class="org.apache.solr.handler.component.SearchHandler">  <lst name="defaults">  <str name="echoParams">explicit</str>  </lst>  </requestHandler>  <requestHandler name="/update" class="solr.UpdateRequestHandler" />  <requestHandler name="/admin/" class="org.apache.solr.handler.admin.AdminHandlers" />  <!-- ping 性能检查 -->  <requestHandler name="/admin/ping" class="solr.PingRequestHandler">  <lst name="invariants">  <str name="q">solrpingquery</str>  </lst>  <lst name="defaults">  <str name="echoParams">all</str>  </lst>  </requestHandler>  <requestHandler name="/debug/dump" class="solr.DumpRequestHandler" >  <lst name="defaults">  <str name="echoParams">explicit</str> <!-- for all params (including the default etc) use: 'all' -->  <str name="echoHandler">true</str>  </lst>  </requestHandler>  <!-- 高亮的一些设置 -->  <highlighting>  <fragmenter name="gap" class="org.apache.solr.highlight.GapFragmenter" default="true">  <lst name="defaults">  <!-- 摘要长度 -->  <int name="hl.fragsize">100</int>  </lst>  </fragmenter>  <fragmenter name="regex" class="org.apache.solr.highlight.RegexFragmenter">  <lst name="defaults">  <int name="hl.fragsize">70</int>  <float name="hl.regex.slop">0.5</float>  <str name="hl.regex.pattern">[-\w ,/\n\"']{20,200}</str>  </lst>  </fragmenter>  <formatter name="html" class="org.apache.solr.highlight.HtmlFormatter" default="true">  <lst name="defaults">  <!-- 高亮结果添加的前缀 -->  <str name="hl.simple.pre"><![CDATA[<em>]]></str>  <!-- 高亮结果添加的后缀 -->  <str name="hl.simple.post"><![CDATA[</em>]]></str>  </lst>  </formatter>  </highlighting>  <!-- 结果处理器 -->  <queryResponseWriter name="xslt" class="solr.XSLTResponseWriter">  <int name="xsltCacheLifetimeSeconds">5</int>  </queryResponseWriter>  <admin>  <defaultQuery>\*:\*</defaultQuery>  </admin>  </config> |

#### schema.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <schema name="db" version="1.1">  <types>  <!--  fieldType定义了字段类型；  class和其他的属性共同决定了这个字段类型的行为；  sortMissingLast设置成true没有该field的数据排在有该field的数据之后，  而不管请求时的排序规则, 默认是设置成false；  sortMissingFirst与sortMissingLast相反，默认是设置成false；  analyzer对字段类型指定分词器；  type表示当前分词用用于的操作，index代表生成索引时使用的分词器，query代码在查询时使用的分词器；  tokenizer指定分词器类；  filter指定分词后应用的过滤器，过滤器调用顺序和配置相同。  -->  <fieldtype name="string" class="solr.StrField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="long" class="solr.TrieLongField" precisionStep="0" positionIncrementGap="0"/>  <fieldType name="boolean" class="solr.BoolField" sortMissingLast="true"/>  <fieldType name="int" class="solr.TrieIntField" precisionStep="0" positionIncrementGap="0"/>  <fieldType name="text\_general" class="solr.TextField" positionIncrementGap="100">  <analyzer type="index">  <tokenizer class="solr.StandardTokenizerFactory"/>  </analyzer>  <analyzer type="query">  <tokenizer class="solr.StandardTokenizerFactory"/>  </analyzer>  </fieldType>  </types>  <!--  field声明了字段；  name是字段类型名；  type对应上面的fieldType；  indexed指定字段是否应被搜索和排序；  stored指定字段是否呗包含在搜索结果中；  required指定字段是否可为空；  mutiValued指定该字段是否允许多值。  -->  <fields>  <field name="id" type="text\_general" indexed="true" stored="true" required="true" multiValued="false" />  <field name="title" type="text\_general" indexed="true" stored="true"/>  <field name="content" type="text\_general" indexed="true" stored="false" multiValued="true"/>  <field name="\_version\_" type="long" indexed="true" stored="true"/>  </fields>  <!-- 唯一键，这里配置的是上面出现的fileds，一般是id、url等不重复的。在更新、删除的时候可以用到 -->  <uniqueKey>id</uniqueKey>  <!-- 默认搜索属性 -->  <defaultSearchField>content</defaultSearchField>  <!-- 查询转换模式，是ANd还是OR -->  <solrQueryParser defaultOperator="OR"/>  </schema> |

#### db-data-config.xml

|  |
| --- |
| <!—具体解释参照7.2 -->  <dataConfig>  <dataSource name="jdbc" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/test" user="root" password="root" />  <document>  <entity name="article" pk="id" query="select id,title,content from article">  <field column="id" name="id" />  <field column="title" name="title" />  <field column="content" name="content" />  </entity>  </document>  </dataConfig> |

## 8. 案例：solr从pdf、office文档中建立索引

使用solr从pdf、office文档中建立索引和从数据库中建立相似，只不过这里需要tika来解析这些文档。

### 8.1 配置一个handler

这个handler首先要在solrConfig.xml中配置，如下所示：

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

<lst name="defaults">

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

</lst>

</requestHandler>

这里和7.1几乎一样，只不过修改了配置文件名（当然也可以不修改）。

### 8.2 文档配置tika-data-config.xml

<dataConfig>

<dataSource type="BinFileDataSource" />

<document>

<entity name="files" dataSource="binary" rootEntity="false"

processor="FileListEntityProcessor"

baseDir="D:/temp" fileName=".\*.(doc)|(pdf)|(xls)|(ppt)|(docx)"

onError="skip"

recursive="true">

<field column="fileAbsolutePath" name="id" />

<field column="fileSize" name="size" />

<field column="fileLastModified" name="lastModified" />

<entity

name="documentImport"

processor="TikaEntityProcessor"

url="${files.fileAbsolutePath}"

format="text">

<field column="file" name="fileName"/>

<field column="Author" name="author" meta="true"/>

<field column="title" name="title" meta="true"/>

<field column="text" name="text"/>

</entity>

</entity>

</document>

</dataConfig>

这里只说上面（7.2）没有说过的属性。

* **fileName** :（必选）使用正则表达式匹配文件
* **baseDir** : (必选) 文件目录
* **recursive** : 是否递归的获取文件，默认false
* **rootEntity** :在这里必须是false(除非你只想索引文件名)。在默认情况下，document元素下就是根实体了，如果没有根实体的话，直接在实体下面的实体将会被看做跟实体。对于根实体对应的数据库中返回的数据的每一行，solr都将生成一个document
* **dataSource** :如果你是用solr1.3，那就必须设为“null”，因为它没使用任何dataSourde。不需要在solr1.4中指定它，它只是意味着我们不创建一个dataSource实例。在大多数情况下，只有一个DataSource（JdbcDataSource），当使用FileListEntityProcessor 的时候DataSource不是必须的
* **processor**:只有当datasource不是RDBMS时才是必须的
* **onError** :默认是“abort”，“skip”表示跳过当前文档，“continue”表示对错误视而不见

关于field在schema.xml中的定义，这里不再赘述。

### 8.3 建立索引

使用url来建立索引：

[http://localhost:8080/solr/**tika**/dataimport?command=full-import&commit=ture](http://localhost:8080/solr/tika/dataimport?command=full-import&commit=ture)

除了使用tika核心之外，都和7.3中相同。

### 8.4 搜索

使用url搜索关键词为“test”的文档信息：

[http://localhost:8080/solr/**tika**/select?q=test&wt=xml&indent=true](http://localhost:8080/solr/tika/select?q=test&wt=xml&indent=true)

除了使用tika核心之外，都和7.4中相同。

### 8.5 配置参考

#### solrConfig.xml

|  |
| --- |
| <!-- 7.5中solrConfig.xml中已做详细解释，或者参考5.3 -->  <?xml version="1.0" encoding="UTF-8" ?>  <config>  <luceneMatchVersion>4.7</luceneMatchVersion>  <schemaFactory class="ClassicIndexSchemaFactory"/>  <updateHandler class="solr.DirectUpdateHandler2">  <maxPendingDeletes>100000</maxPendingDeletes>  </updateHandler>  <query>  <maxBooleanClauses>1024</maxBooleanClauses>  <filterCache  class="solr.LRUCache"  size="512"  initialSize="512"  autowarmCount="256"/>  <queryResultCache  class="solr.LRUCache"  size="512"  initialSize="512"  autowarmCount="256"/>  <documentCache  class="solr.LRUCache"  size="512"  initialSize="512"  autowarmCount="0"/>  <enableLazyFieldLoading>true</enableLazyFieldLoading>  <queryResultWindowSize>50</queryResultWindowSize>    <queryResultMaxDocsCached>200</queryResultMaxDocsCached>  <HashDocSet maxSize="3000" loadFactor="0.75"/>  <listener event="newSearcher" class="solr.QuerySenderListener">  <arr name="queries">  <lst> <str name="q">solr</str> <str name="start">0</str> <str name="rows">10</str> </lst>  <lst> <str name="q">rocks</str> <str name="start">0</str> <str name="rows">10</str> </lst>  <lst><str name="q">static newSearcher warming query from solrconfig.xml</str></lst>  </arr>  </listener>  <listener event="firstSearcher" class="solr.QuerySenderListener">  <arr name="queries">  </arr>  </listener>  <useColdSearcher>false</useColdSearcher>  <maxWarmingSearchers>4</maxWarmingSearchers>  </query>  <requestDispatcher handleSelect="true" >  <requestParsers enableRemoteStreaming="false" multipartUploadLimitInKB="2048" formdataUploadLimitInKB="2048" />  <httpCaching never304="true">  </httpCaching>  </requestDispatcher>  <requestHandler name="standard" class="solr.StandardRequestHandler" default="true">  <lst name="defaults">  <str name="echoParams">explicit</str>  </lst>  </requestHandler>    <requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">  <lst name="defaults">  <str name="config">tika-data-config.xml</str>  </lst>  </requestHandler>    <requestHandler name="/search" class="org.apache.solr.handler.component.SearchHandler">  <lst name="defaults">  <str name="echoParams">explicit</str>  </lst>  </requestHandler>  <requestHandler name="/update" class="solr.UpdateRequestHandler" >  </requestHandler>    <requestHandler name="/admin/" class="org.apache.solr.handler.admin.AdminHandlers" />    <requestHandler name="/admin/ping" class="solr.PingRequestHandler">  <lst name="invariants">  <str name="q">solrpingquery</str>  </lst>  <lst name="defaults">  <str name="echoParams">all</str>  </lst>  </requestHandler>    <admin>  <defaultQuery>\*:\*</defaultQuery>  </admin>  </config> |

#### schema.xml

|  |
| --- |
| <!-- 7.5中schema.xml中已做详细解释，或者参考5.2 -->  <?xml version="1.0" encoding="UTF-8" ?>  <schema name="test" version="1.2">  <types>  <fieldType name="string" class="solr.StrField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="boolean" class="solr.BoolField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="integer" class="solr.IntField" omitNorms="true"/>  <fieldType name="long" class="solr.LongField" omitNorms="true"/>  <fieldType name="float" class="solr.FloatField" omitNorms="true"/>  <fieldType name="double" class="solr.DoubleField" omitNorms="true"/>  <fieldType name="sint" class="solr.SortableIntField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="slong" class="solr.SortableLongField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="sfloat" class="solr.SortableFloatField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="sdouble" class="solr.SortableDoubleField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="date" class="solr.DateField" sortMissingLast="true" omitNorms="true"/>  <fieldType name="random" class="solr.RandomSortField" indexed="true" />  <fieldType name="text\_ws" class="solr.TextField" positionIncrementGap="100">  <analyzer>  <tokenizer class="solr.WhitespaceTokenizerFactory"/>  </analyzer>  </fieldType>  <fieldType name="text" class="solr.TextField" positionIncrementGap="100">  <analyzer type="index">  <tokenizer class="solr.WhitespaceTokenizerFactory"/>  <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0" splitOnCaseChange="1"/>  <filter class="solr.LowerCaseFilterFactory"/>  <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>  </analyzer>  <analyzer type="query">  <tokenizer class="solr.WhitespaceTokenizerFactory"/>  <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0" splitOnCaseChange="1"/>  <filter class="solr.LowerCaseFilterFactory"/>  <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>  </analyzer>  </fieldType>  <fieldType name="text\_ik" class="solr.TextField">  <analyzer type="index" class="org.wltea.analyzer.lucene.IKAnalyzer" useSmart="true"/>  <analyzer type="query" class="org.wltea.analyzer.lucene.IKAnalyzer" useSmart="false"/>  </fieldType>  </types>  <fields>  <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" />  <field name="fileName" type="string" indexed="true" stored="true" />  <field name="author" type="string" indexed="true" stored="true" />  <field name="title" type="string" indexed="true" stored="true" />    <field name="size" type="long" indexed="true" stored="true" />  <field name="lastModified" type="date" indexed="true" stored="true" />    <field name="text" type="text\_ik" indexed="true" stored="false" multiValued="true"/>  </fields>  <defaultSearchField>text</defaultSearchField>  <solrQueryParser defaultOperator="OR"/>  </schema> |

#### tika-data-config.xml

|  |
| --- |
| <!—具体解释参照8.2 -->  <dataConfig>  <dataSource type="BinFileDataSource" />  <document>  <entity name="files" dataSource="binary" rootEntity="false"  processor="FileListEntityProcessor"  baseDir="D:/temp" fileName=".\*.(doc)|(pdf)|(xls)|(ppt)|(docx)"  onError="skip"  recursive="true">  <field column="fileAbsolutePath" name="id" />  <field column="fileSize" name="size" />  <field column="fileLastModified" name="lastModified" />    <entity  name="documentImport"  processor="TikaEntityProcessor"  url="${files.fileAbsolutePath}"  format="text">  <field column="file" name="fileName"/>  <field column="Author" name="author" meta="true"/>  <field column="title" name="title" meta="true"/>  <field column="text" name="text"/>    </entity>  </entity>  </document>  </dataConfig> |