

Solr

全文检索服务

1 企业站内搜索技术选型

在一些大型门户网站、电子商务网站等都需要站内搜索功能，使用传统的数据库查询方式实现搜索无法满足一些高级的搜索需求，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，这里就需要使用全文检索技术实现搜索功能。

1.1 单独使用Lucene实现

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、搜索性能优化等，因此不建议采用。

1.2 使用Google或Baidu接口

通过第三方搜索引擎提供的接口实现站内搜索，这样和第三方引擎系统依赖紧密，不方便扩展，不建议采用。

1.3 使用Solr实现

基于Solr实现站内搜索扩展性较好并且可以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，因此在门户、论坛等系统中常用此方案。

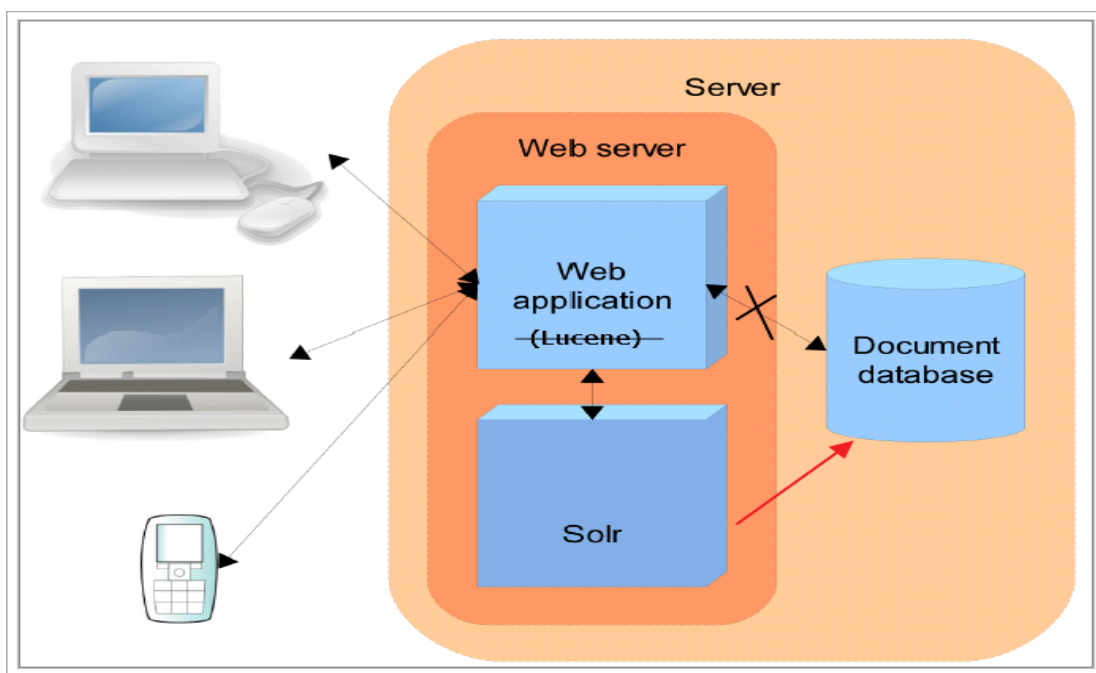
2 什么是Solr

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。



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，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。

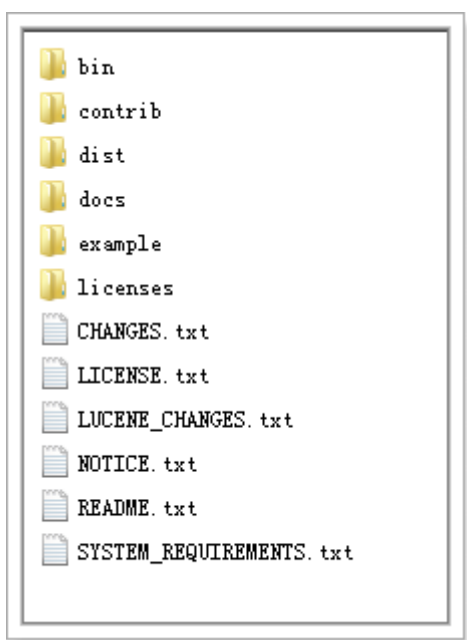
3 Solr安装配置

下载

从Solr官方网站（<http://lucene.apache.org/solr/>）下载Solr4.10.3，根据Solr的运行环境，Linux下需要下载lucene-4.10.3.tgz，windows下需要下载lucene-4.10.3.zip。

Solr使用指南可参考：<https://wiki.apache.org/solr/FrontPage>。

下载lucene-4.10.3.zip并解压：



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: 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。

docs: 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的多核中设置的多个Core目录。

- **example/webapps:**

该目录中包括一个solr.war，该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，Solr4.10.3要求jdk使用1.7以上，Solr默认提供Jetty（java写的Servlet容器），本教程使用Tomcat作为Servlet容器，环境如下：

Solr: Solr4.10.3

Jdk: jdk1.7.0_72

Tomcat: apache-tomcat-7.0.53

Solr与Tomcat整合配置

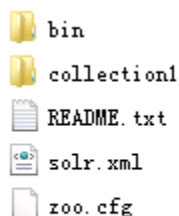
Solr Home与SolrCore

创建一个Solr

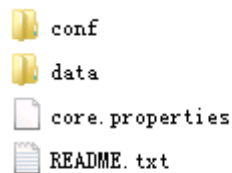
home目录，目录中包括了运行Solr实例所有的配置文件和数据文件，SolrHome是Solr运行的主目录，一个SolrHome可以包括多个SolrCore（Solr实例），每个SolrCore提供单独的搜索和索引服务。

目录结构

example\solr是一个solr home目录结构，如下：



上图中“collection1”是一个SolrCore（Solr实例）目录，目录内容如下所示：



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。

conf是SolrCore的配置文件目录。

配置

创建目录 F:\develop\solr作为Solr Home，

1. 将example\solr目录 拷贝至 F:\develop\solr目录下并改名为solrhome

`solrconfig.xml`

`solrconfig.xml`, 在SolrCore的`conf`目录下, 它是SolrCore运行的配置文件。

加载jar包

将`contrib`和`dist`两个目录拷贝到`F:\develop\solr`下, 修改`solrconfig.xml`文件:

```
<lib dir="${solr.install.dir:../..}/contrib/extraction/lib" regex=".*\.jar" />
<lib dir="${solr.install.dir:../..}/dist/" regex="solr-cell-\d.*\.jar" />

<lib dir="${solr.install.dir:../..}/contrib/clustering/lib/" regex=".*\.jar" />
<lib dir="${solr.install.dir:../..}/dist/" regex="solr-clustering-\d.*\.jar" />

<lib dir="${solr.install.dir:../..}/contrib/langid/lib/" regex=".*\.jar" />
<lib dir="${solr.install.dir:../..}/dist/" regex="solr-langid-\d.*\.jar" />

<lib dir="${solr.install.dir:../..}/contrib/velocity/lib" regex=".*\.jar" />
<lib dir="${solr.install.dir:../..}/dist/" regex="solr-velocity-\d.*\.jar" />
```

dataDir

配置SolrCore的数据目录, 数据目录下包括了`index`索引目录和`tlog`日志文件目录, 数据目录默认在`solrCore`下的`data`目录, 也可以更改目录地址, 如下:

```
<dataDir>${solr.data.dir:F:/develop/solr/collection1/data}</dataDir>
```

requestHandler

`requestHandler`请求处理器, 定义了索引和搜索的访问方式。通过`/update`维护索引, 可以完成索引的添加、修改、删除操作。

```
<requestHandler name="/update" class="solr.UpdateRequestHandler">
```

提交`xml`、`json`数据完成索引维护, 索引维护小节详细介绍。

通过`/select`搜索索引。

```
<requestHandler name="/select" class="solr.SearchHandler">
```

设置搜索参数完成搜索, 搜索参数也可以设置一些默认值, 如下:

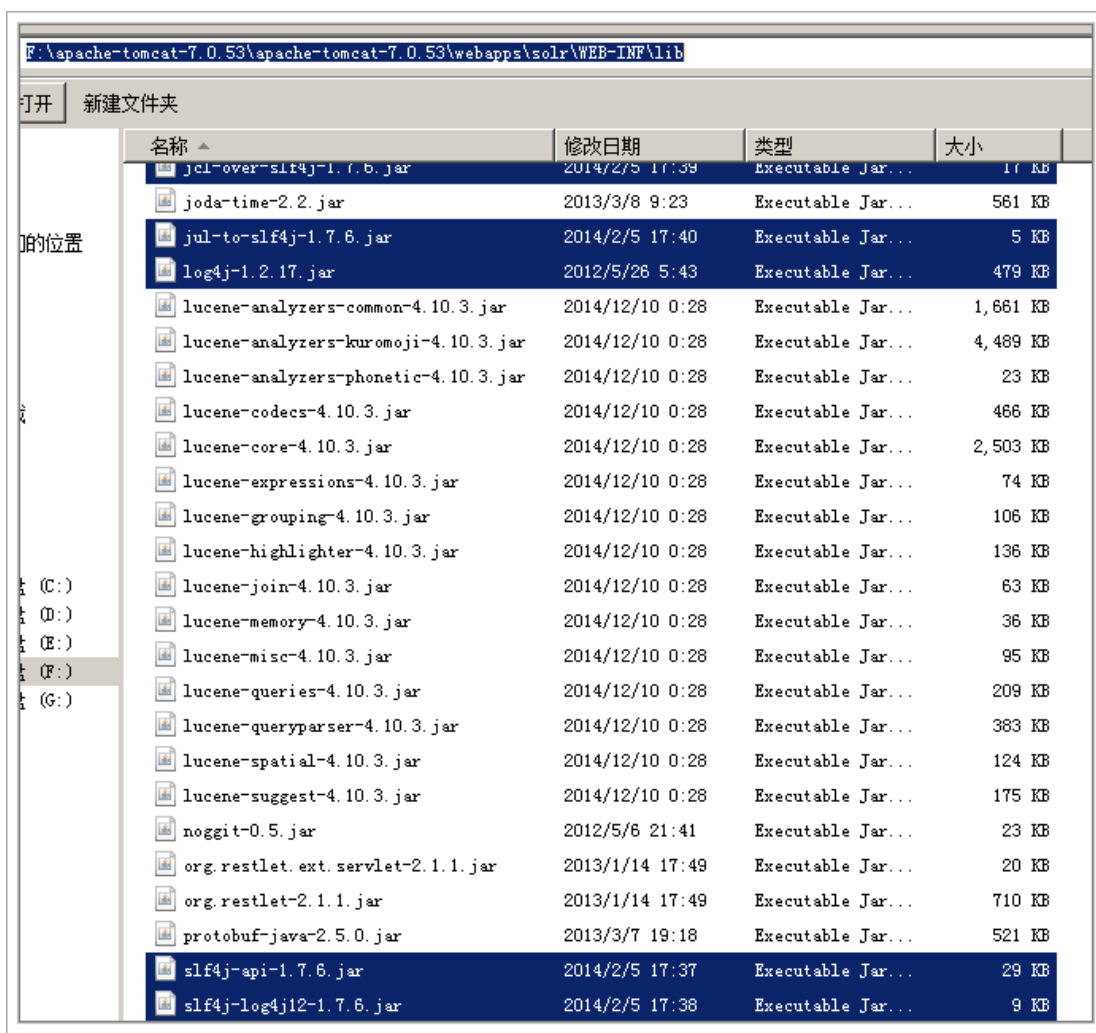
```
<requestHandler name="/select" class="solr.SearchHandler">
  <!-- 设置默认的参数值, 可以在请求地址中修改这些参数-->
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int><!--显示数量-->
```

```
<str name="wt">json</str><!--显示格式-->
<str name="df">text</str><!--默认搜索字段-->
</lst>
</requestHandler>
```

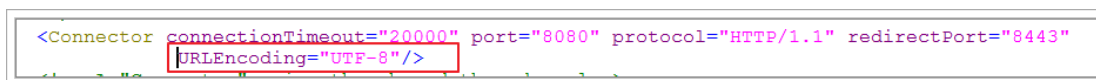
Solr工程部署

1. 将dist\solr-4.10.3.war拷贝到Tomcat的webapp目录下改名为solr.war
2. 启动tomcat后, solr.war自动解压, 将原来的solr.war删除。
3. 拷贝example\lib\ext 目录下所有jar包到Tomcat的webapp\solr\WEB-INF\lib目录下





4. 修改Tomcat的url字符集，修改conf/server.xml文件



5. 修改Tomcat目录下webapp\solr\WEB-INF\web.xml文件，如下所示： 设置Solr home

```
<env-entry>
  <env-entry-name>solr/home</env-entry-name>
  <env-entry-value>F:\develop\solr\solrhome</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

6. 拷贝log4j.properties文件

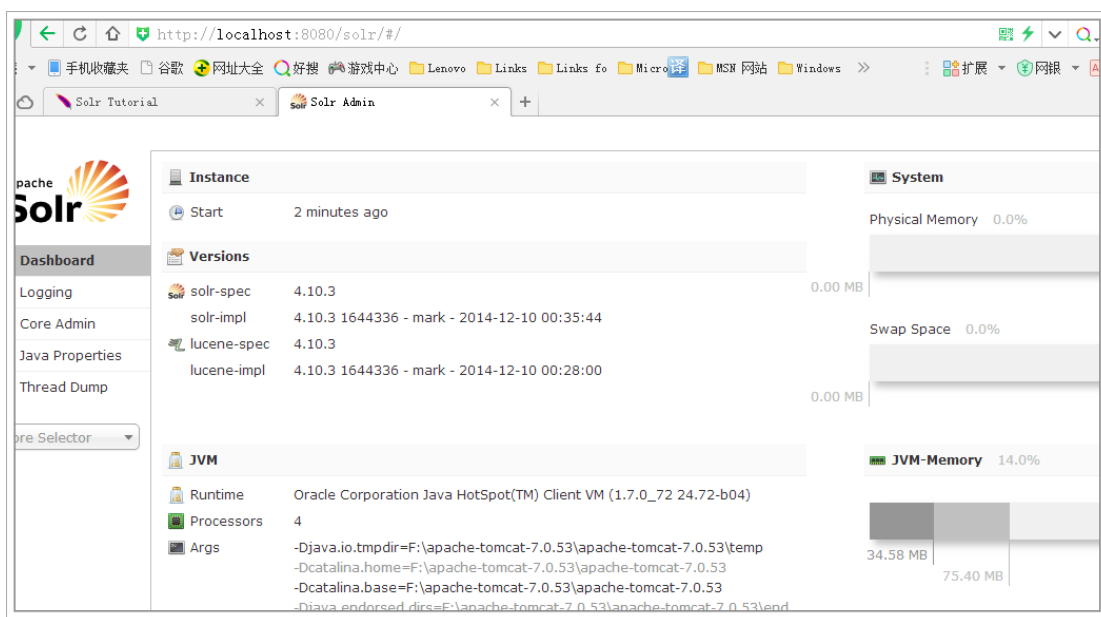
在 Tomcat下webapps\solr\WEB-INF目录中创建文件 classes文件夹，

复制Solr目录下example\resources\log4j.properties至Tomcat下webapps\solr\WEB-INF\classes目录

启动Tomcat

访问<http://localhost:8080/solr>

管理界面



Dashboard:

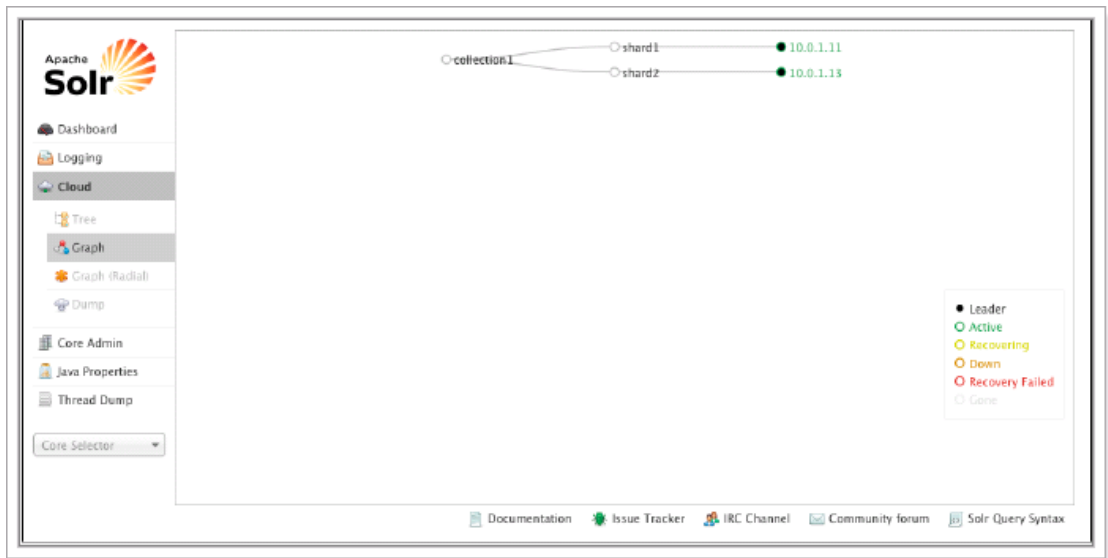
仪表盘，显示了该Solr实例开始启动运行的时间、版本、系统资源、jvm等信息。

Logging:

Solr运行日志信息

Cloud:

Cloud即SolrCloud，即Solr云（集群），当使用Solr Cloud模式运行时显示此菜单，如下图是Solr Cloud的管理界面：



Core Admin:

Solr Core的管理界面。Solr Core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。

java properties

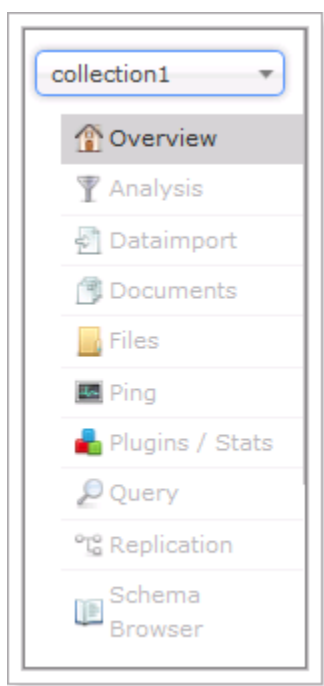
Solr在JVM运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

Tread Dump

显示Solr Server中当前活跃线程信息，同时也可以跟踪线程运行栈信息。

Core selector

选择一个SolrCore进行详细操作，如下：



Analysis

A screenshot of the Solr Analysis UI. It features two main input fields: 'Field Value (Index)' on the left and 'Field Value (Query)' on the right. Both fields contain the placeholder text '输入索引分析内容' and '输入搜索分析内容' respectively. Below the 'Field Value (Index)' field is a dropdown menu labeled 'Analyse Fieldname / FieldType:' with the value '_root_'. To the right of the 'Field Value (Query)' field is a checkbox labeled 'Verbose Output'. At the bottom right of the form is a blue button labeled 'Analyse Values'.

通过此界面可以测试索引分析器和搜索分析器的执行情况。

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

通过此菜单可以创建索引、更新索引、删除索引等操作，界面如下：

Request-Handler (qt)

Document Type

JSON

Document(s)

```
{ "id": "change.me", "title": "change.me" }
```

Commit Within

Overwrite

Boost

Submit Document

/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

A screenshot of the Solr Admin web interface. The 'Request-Handler (qt)' dropdown is set to '/select'. Below it is a 'common' section. The 'q' (query) field contains the text '*.*' and is highlighted with a red box and the red Chinese text '查询表达式' (Query Expression). Below 'q' are fields for 'fq' (facet query), 'sort', 'start, rows' (with '0' and '10' entered), 'fl' (facet list), and 'df' (facet display). The 'Raw Query Parameters' field contains 'key1=val1&key2=val2'. The 'wt' (write method) dropdown is set to 'json'. At the bottom, there are checkboxes for 'indent' (checked) and 'debugQuery' (unchecked).

Request-Handler (qt)

/select

— common —

q

.

查询表达式

fq

sort

start, rows

0 10

fl

df

Raw Query Parameters

key1=val1&key2=val2

wt

json

☒ indent























☐ debugQuery

通过/select执行搜索索引，必须指定“q”查询条件方可搜索。

4 Solr索引

schema.xml

schema.xml，在SolrCore的conf目录下，它是Solr数据表配置文件，它定义了加入索引的数据的数据类型的。主要包括FieldTypes、Fields和其他的一些缺省设置。

\solr\collection1\conf			
新建文件夹			
名称 ^	修改日期	类型	大小
 clustering	2015/1/18 12:58	文件夹	
 lang	2015/1/18 12:58	文件夹	
 velocity	2015/1/18 12:58	文件夹	
 xslt	2015/1/18 12:58	文件夹	
 _rest_managed.json	2014/12/10 0:37	JSON 文件	1 KB
 _schema_analysis_stopwords_english...	2014/12/10 0:37	JSON 文件	1 KB
 _schema_analysis_synonyms_english...	2014/12/10 0:37	JSON 文件	1 KB
 admin-extra.html	2014/12/10 0:37	360 se HTML Do...	2 KB
 admin-extra.menu-bottom.html	2014/12/10 0:37	360 se HTML Do...	1 KB
 admin-extra.menu-top.html	2014/12/10 0:37	360 se HTML Do...	1 KB
 currency.xml	2014/12/10 0:37	XML 文档	4 KB
 elevate.xml	2014/12/10 0:37	XML 文档	2 KB
 mapping-FoldToASCII.txt	2014/12/10 0:37	文本文档	81 KB
 mapping-ISOLatin1Accent.txt	2014/12/10 0:37	文本文档	4 KB
 protowords.txt	2014/12/10 0:37	文本文档	1 KB
 schema.xml	2015/1/18 14:01	XML 文档	61 KB
 scripts.conf	2014/12/1 10:06	CONF 文件	1 KB
 solrconfig.xml	2015/1/18 13:39	XML 文档	75 KB
 spellings.txt	2014/12/10 0:37	文本文档	1 KB
 stopwords.txt	2014/12/10 0:37	文本文档	1 KB
 synonyms.txt	2014/12/10 0:37	文本文档	2 KB
 update-script.js	2014/12/10 0:37	JScript Script...	2 KB

FieldType域类型定义

下边“text_general”是Solr默认提供的FieldType，通过它说明FieldType定义的内容：

```

<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" />
    <!-- in this example, we will only use synonyms at query time
    <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true"
    expand="false"/>
    -->
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>

```

FieldType子结点包括: 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。

在FieldType定义的时候最重要的就是定义这个类型的数据在建立索引和进行查询的时候要使用的分析器analyzer,包括分词和过滤

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

搜索分析器中: 使用solr.StandardTokenizerFactory标准分词器, solr.StopFilterFactory停用词过滤器, 这里还用到了solr.SynonymFilterFactory同义词过滤器。

Field定义

在fields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FieldType),indexed (是否被索引),stored (是否被储存), multiValued (是否存储多个值) 等属性。

如下:

```

<field name="name" type="text_general" indexed="true" stored="true"/>
<field name="features" type="text_general" indexed="true" stored="true"
multiValued="true"/>

```

multiValued: 该Field如果要存储多个值时设置为true, solr允许一个Field存储多个值, 比如存储一个用户的好友id (多个), 商品的图片 (多个, 大图和小图), 通过使用solr查询要看出返回给客户端是数组:

```

"response": {
  "numFound": 5,
  "start": 0,
  "docs": [
    {
      "id": "SOLR1000",
      "name": "Solr, the Enterprise Search Server",
      "manu": "Apache Software Foundation",
      "cat": [
        "software",
        "search"
      ],
      "features": [
        "Advanced Full-Text Search Capabilities using Lucene",
        "Optimized for High Volume Web Traffic",
        "Standards Based Open Interfaces - XML and HTTP",
        "Comprehensive HTML Administration Interfaces",
        "Scalability - Efficient Replication to other Solr Search Servers",
        "Flexible and Adaptable with XML configuration and Schema",
        "Good unicode support: hello (hello with an accent over the e)"
      ]
    }
  ]
}

```

查询出来的结果是多个值

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。

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：

```

<field name="title" type="text_general" indexed="true" stored="true" multiValued="true"/>
<field name="content" type="text_general" indexed="false" stored="true" multiValued="true"/>
...
<field name="text" type="text_general" indexed="true" stored="false" multiValued="true"/>

```

根据关键字只搜索text域的内容就相当于搜索title和content，将title和content复制到text中，如下：

```

<copyField source="title" dest="text"/>
<copyField source="author" dest="text"/>
<copyField source="description" dest="text"/>
<copyField source="keywords" dest="text"/>
<copyField source="content" dest="text"/>

```


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，例如定义一个 dynamicField，name 为 *_i，定义它的 type 为 text，那么在使用这个字段的时候，任何以 _i 结尾的字段都被认为是符合这个定义的，例如：name_i，gender_i，school_i 等。

自定义 Field 名为：product_title_t，“product_title_t”和 schema.xml 中的 dynamicField 规则匹配成功，如下：

```
<dynamicField name="*_t" type="text_general" indexed="true" stored="true"/>
```

“product_title_t”是以“_t”结尾。

创建索引：

Request-Handler (qt)
/update

Document Type
JSON

Document(s)
{\"id\": \"100033\", \"product_title_t\": \"spring book\"}

Commit Within
1000

Overwrite
true

Boost
1.0

Submit Document

Status: success
Response:
{
 \"responseHeader\": {
 \"status\": 0,
 \"QTime\": 3
 }
}

搜索索引：

自定义中文分词器类型及字段

5 配置IK域类型,修改schema.xml文件

修改Solr的schema.xml文件，添加FieldType：

```
<!-- IKAnalyzer-->  
<fieldType name="text_ik" class="solr.TextField">  
  <analyzer type="index" isMaxWordLength="false"  
class="org.wltea.analyzer.lucene.IKAnalyzer"/>  
  <analyzer type="query" isMaxWordLength="true"  
class="org.wltea.analyzer.lucene.IKAnalyzer"/>  
</fieldType>
```

根据需求配置自定义域

```
<!--IKAnalyzer Field-->
```

```
<field name="product_name" type="text_ik" indexed="true" stored="true" />
```

Request-Handler (qt)

— common —

q

fq

sort

start, rows

fl

df

Raw Query Parameters

<http://localhost:8080/solr/collection1/s>

```

{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "indent": "true",
      "q": "id:100033",
      "_": "1422355381499",
      "wt": "json"
    }
  },
  "response": {
    "numFound": 1,
    "start": 0,
    "docs": [
      {
        "id": "100033",
        "product_title_t": "spring book",
        "_version_": 1491447668014055400
      }
    ]
  }
}

```

Analyzer

安装中文分词器

IKAnalyzer部署

拷贝IKAnalyzer的文件到Tomcat下Solr目录 中

将IKAnalyzer2012FF_u1.jar拷贝到 Tomcat的webapps/solr/WEB-INF/lib 下。

在Tomcat的webapps/solr/WEB-INF/下创建classes目录

将IKAnalyzer.cfg.xml、ext_stopword.dic mydict.dic copy到 Tomcat的

webapps/solr/WEB-INF/classes

注意: ext_stopword.dic 和mydict.dic必须保存成无BOM的utf-8类型。

修改schema.xml文件

1. FieldType

首先需要在types结点内定义一个FieldType子结点, 包括name,class,等参数, name就是这个FieldType的名称, class指向org.apache.solr.analysis包里面对应的class名称, 用来定义这个类型的行为。在FieldType定义的时候最重要的就是定义这个类型的数据在建立索引和进行查询的时候要使用的分析器analyzer, 包括分词和过滤

修改Solr的schema.xml文件, 添加FieldType:

```
<!-- IKAnalyzer-->
  <fieldType name="text_ik" class="solr.TextField">
    <analyzer type="index" isMaxWordLength="false"
class="org.wltea.analyzer.lucene.IKAnalyzer"/>
    <analyzer type="query" isMaxWordLength="true"
class="org.wltea.analyzer.lucene.IKAnalyzer"/>
  </fieldType>
```

2. Field:

FieldType定义好后就可以在fields结点内定义具体的field, field定义包括name,type (即FieldType),indexed (是否被索引),stored (是否被储存), multi Valued (是否有多个值) 等

```
<!--IKAnalyzer Field-->
  <field name="title_ik" type="text_ik" indexed="true" stored="true" />
  <field name="content_ik" type="text_ik" indexed="true" stored="false" multiValued="true"/>
```

3. 测试

Field Value (Index)

中国人传智播客

Field Value (Query)

Analyse Fieldname / FieldType:

title_ik

?

☒ Verbose Output

Analyse Values

IKT	text	中国人	中国	国人	传智播客
raw_bytes		[e4 b8 ad e5 9b bd e4 ba ba]	[e4 b8 ad e5 9b bd]	[e5 9b bd e4 ba ba]	[e4 bc a0 e6 99 ba e6 92 ad e5 ae a2]
start		0	0	1	3
end		3	2	3	7
positionLength		1	1	1	1
type		CN_WORD	CN_WORD	CN_WORD	CN_WORD
position		1	2	3	4

5.1 设置业务系统Field

如果不使用Solr提供的Field可以针对具体的业务需要自定义一套Field，如下是商品信息Field：

```
<!--product-->
  <field name="product_name" type="text_ik" indexed="true" stored="true"/>
  <field name="product_price" type="float" indexed="true" stored="true"/>
  <field name="product_description" type="text_ik" indexed="true"
stored="false" />
  <field name="product_picture" type="string" indexed="false" stored="true" />
  <field name="product_catalog_name" type="string" indexed="true"
stored="true" />

  <field name="product_keywords" type="text_ik" indexed="true" stored="false"
multiValued="true"/>
  <copyField source="product_name" dest="product_keywords"/>
  <copyField source="product_description" dest="product_keywords"/>
```

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：

Request-Handler (qt)

Document Type

Document(s)

```
<add>
<doc>
<field name="id">change.me</field>
<field name="title">change.me</field>
</doc>
</add>
```

Commit Within

Overwrite

Submit Document

overwrite="true" :

solr在做索引的时候，如果文档已经存在，就用xml中的文档进行替换

commitWithin="10000" : solr

在做索引的时候，每个10000（10秒）毫秒，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>后边添加“<commit/>”，如下：

```
<add>
<doc>
<field name="id">change.me</field>
<field name="title">change.me</field>
</doc>
</add>
<commit/>
```

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。

请求xml格式如下:

```
<add>
<doc>
<field name="id">change.me</field>
<field name=" ? ? " > ? ? </field>
. . .
</doc>
</add>
```

说明: 唯一标识 Field必须有, 这里使用Solr默认 id。

删除索引

删除索引格式如下:

1) 删除制定ID的索引

```
<delete>
  <id>8</id>
. . .
</delete>
```

2) 删除查询到的索引数据

```
<delete>
  <query>product_catalog_name:幽默杂货</query>
</delete>
```

3) 删除所有索引数据

```
<delete>
  <query>*:*</query>
</delete>
```

SolrJ完成索引维护

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

使用SolrJ创建索引，通过调用SolrJ提供的API请求Solr服务，Document通过SolrInputDocument进行构建。

```
// 创建索引
    public void testCreateIndex() throws SolrServerException,
IOException {
        SolrServer solrServer = new
HttpSolrServer(urlString);
        SolrInputDocument document = new
SolrInputDocument();
        document.addField("id", "c0001");
        document.addField("product_name",
"solr全文检索");//商品名称
        document.addField("product_price",
86.5f);//商品价格
        document.addField("product_picture",
"382782828.jpg");//商品图片
        document.addField("product_description",
"这是一本关于solr的书籍!");//商品描述
        document.addField("product_catalog_name",
"javabook");//商品分类

        UpdateResponse response =
solrServer.add(document);
        // 提交
        solrServer.commit();
    }
```

说明：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。

删除索引

上边介绍的删除方法，使用SolrJ也可以完成，代码如下：

```
// 删除索引
@Test
```

```

    public void testDeleteIndex() throws SolrServerException,
IOException {
        SolrServer solrServer = new
HttpSolrServer(urlString);

        //根据id删除
        UpdateResponse response =
solrServer.deleteById("c0001");
        //根据多个id删除
        // solrServer.deleteById(ids);
        //自动查询条件删除
        //
solrServer.deleteByQuery("product_keywords:教程");
        // 提交
        solrServer.commit();
    }

```

说明: deleteById(String id)根据id删除索引, 此方法为重载方法, 也可以传个多个id批量删除, 也可以调用deleteByQuery() 根据查询条件删除

dataimport-handler

安装dataimport-Handler从关系数据库将数据导入到索引库。

第一步: 向SolrCore中加入jar包

在SolrCore目录中创建lib目录, 将dataimportHandler和mysql数据库驱动的jar拷贝至lib下:

dataimportHandler在solr安装目录的dist 下:

D:\solr\solr-4.10.3\solr-4.10.3\dist				
名称	修改日期	类型	大小	
solrj-lib	2014/12/10 0:37	文件夹		
test-framework	2014/12/10 0:37	文件夹		
solr-4.10.3.war	2014/12/10 0:35	WAR 文件	29,045 KB	
solr-analysis-extras-4.10.3.jar	2014/12/10 0:34	Executable Jar...	18 KB	
solr-cell-4.10.3.jar	2014/12/10 0:34	Executable Jar...	30 KB	
solr-clustering-4.10.3.jar	2014/12/10 0:34	Executable Jar...	51 KB	
solr-core-4.10.3.jar	2014/12/10 0:35	Executable Jar...	2,786 KB	
solr-dataimporthandler-4.10.3.jar	2014/12/10 0:34	Executable Jar...	215 KB	
solr-dataimporthandler-extras-4.10.3.jar	2014/12/10 0:34	Executable Jar...	37 KB	
solr-langid-4.10.3.jar	2014/12/10 0:34	Executable Jar...	750 KB	

第二步: 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>

```

第三步：编辑data-config.xml文件，存放在SolrCore的conf目录

```

<?xml version="1.0" encoding="UTF-8" ?>
<dataConfig>
<dataSource type="JdbcDataSource"
  driver="com.mysql.jdbc.Driver"
  url="jdbc:mysql://localhost:3306/lucene"
  user="root"
  password="mysql"/>
<document>
  <entity name="product" query="SELECT
pid,name,catalog_name,price,description,picture FROM products ">
    <field column="pid" name="id"/>
    <field column="name" name="product_name"/>
    <field column="catalog_name" name="product_catalog_name"/>
    <field column="price" name="product_price"/>
    <field column="description" name="product_description"/>
    <field column="picture" name="product_picture"/>
  </entity>
</document>
</dataConfig>

```

说明：

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

下边以“product_”开头的Field都是在schema.xml中自定义的商品信息Field。

第四步：重启Tomcat，进入管理界面--》SolrCore--》dataimport下执行导入

/dataimport

Command

full-import

☐ Verbose

☒ Clean

☒ Commit

☐ Optimize

☐ Debug

Entity

Start, Rows

010

Custom Parameters

key1=val1&key2=val2

Execute

Refresh Status

第五步：查看导入结果
进入管理界面--》SolrCore--》dataimport下

/dataimport

Command

full-import

☐ Verbose

☒ Clean

☒ Commit

☐ Optimize

☐ Debug

Entity

Start, Rows

010

Custom Parameters

key1=val1&key2=val2

Execute

Refresh Status

Last Update: 22:36:59

✓ Indexing completed. Added/Updated: 3803 documents. Deleted 0 documents. (Duration: 13s)

Requests: 1 (0/s), Fetched: 3803 (293/s), Skipped: 0, Processed: 3803 (293/s)

Started: 6 minutes ago

Raw Status-Output

Configuration

6 Solr搜索

```
<requestHandler name="/select" class="solr.SearchHandler">
```

```

<!-- 设置默认的参数值，可以在请求地址中修改这些参数-->
<lst name="defaults">
  <str name="echoParams">explicit</str>
  <int name="rows">10</int><!--显示数量-->
  <str name="wt">json</str><!--显示格式-->
  <str name="df">text</str><!--默认搜索字段-->
</lst>
</requestHandler>

```

查询语法

通过/select搜索索引，Solr制定一些参数完成不同需求的搜索：

1. q - 查询字符串，必须的，如果查询所有使用*:*。

Request-Handler (qt)

/select

— common —

q

product_keywords:浪漫樱花 AND
 product_keywords:韩国 OR
 product_catalog_name:与转不同

2. fq - (filter query) 过滤查询，作用：在q查询符合结果中同时是fq查询符合的，例如：

fq

product_price:[1 TO 20]

-
+

过滤查询价格从1到20的记录。

也可以在“q”查询条件中使用product_price:[1 TO 20]，如下：

q

product_price:[1 TO 20]

也可以使用“*”表示无限，例如：

20以上：product_price:[20 TO *]

20以下：product_price:[* TO 20]

3. sort - 排序，格式：sort=<field name>+<desc|asc>[,<field name>+<desc|asc>]... 。示例：

sort

product_price desc

按价格降序

4. start - 分页显示使用, 开始记录下标, 从0开始
5. rows - 指定返回结果最多有多少条记录, 配合start来实现分页。

start, rows

0

10

显示前10条。

6. fl - 指定返回那些字段内容, 用逗号或空格分隔多个。

fl

duct_picture,product_name,product_price

显示商品图片、商品名称、商品价格

7. df-指定一个搜索Field

df

product_keywords

也可以在SolrCore目录中conf/solrconfig.xml文件中指定默认搜索Field, 指定后就可以直接在“q”查询条件中输入关键字。

```
<requestHandler name="/select" class="solr.SearchHandler">
  <!-- default values for query parameters can be specified, these
       will be overridden by parameters in the request
  -->
  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>
    <str name="df">text</str>
  </lst>
```

注意在使用的
SearcherHandler中设置df

8. wt - (writer type)指定输出格式, 可以有 xml, json, php, phps, 后面 solr 1.3增加的, 要用通知我们, 因为默认没有打开。
9. hl 是否高亮, 设置高亮Field, 设置格式前缀和后缀。

— ☒ hl —

hl.fl

hl.simple.pre

hl.simple.post

SolrJ完成搜索

搜索索引

```
// 搜索
@Test
public void testSearch() throws SolrServerException {

    SolrServer solr = new HttpSolrServer(urlString);
    // 查询对象
    SolrQuery query = new SolrQuery();
    //设置查询条件,名称“q”是固定的且必须 的

    //搜索product_keywords域, product_keywords是复制域包括product_name和product_description
    query.set("q", "product_keywords:java教程");
    // 请求查询
    QueryResponse response = solr.query(query);

    // 查询结果
    SolrDocumentList docs = response.getResults();
    // 查询文档总数
    System.out.println("查询文档总数" + docs.getNumFound());

    for (SolrDocument doc : docs) {
        //商品主键
        String id = (String) doc.getFieldValue("id");
        //商品名称
```

```
        String product_name = (String)
doc.getFieldValue("product_name");
        //商品价格
        Float product_price = (Float)
doc.getFieldValue("product_price");
        //商品图片
        String product_picture = (String)
doc.getFieldValue("product_picture");
        //商品分类
        String product_catalog_name = (String)
doc.getFieldValue("product_catalog_name");

        System.out.println("=====");
        System.out.println(id);
        System.out.println(product_name);
        System.out.println(product_price);
        System.out.println(product_picture);
        System.out.println(product_catalog_name);
    }
}
```

组合查询

```
//
根据商品分类、价格范围、关键字查询，查询结果按照价格降序排序
@Test
public void testSearch2() throws SolrServerException {

    SolrServer solr = new HttpSolrServer(urlString);
    // 查询对象
    SolrQuery query = new SolrQuery();
    //
    搜索product_keywords域，product_keywords是复制域包括product_name
    和product_description
    // 设置商品分类、关键字查询
```

```
// query.set("q", "product_keywords:挂钩 AND
product_catalog_name:幽默杂货");
query.setQuery("product_keywords:挂钩 AND
product_catalog_name:幽默杂货");

// 设置价格范围
query.set("fq", "product_price:[1 TO 20]");

// 查询结果按照价格降序排序
// query.set("sort", "product_price desc");
query.addSort("product_price", ORDER.desc);

// 请求查询
QueryResponse response = solr.query(query);

// 查询结果
SolrDocumentList docs = response.getResults();
// 查询文档总数
System.out.println("查询文档总数" +
docs.getNumFound());

for (SolrDocument doc : docs) {
    // 商品主键
    String id = (String) doc.getFieldValue("id");
    // 商品名称
    String product_name = (String)
doc.getFieldValue("product_name");
    // 商品价格
    Float product_price = (Float)
doc.getFieldValue("product_price");
    // 商品图片
    String product_picture = (String) doc
        .getFieldValue("product_picture");
    // 商品分类
    String product_catalog_name = (String) doc

        .getFieldValue("product_catalog_name");

    System.out.println("=====");
    System.out.println("id=" + id);
    System.out.println("product_name=" +
product_name);
```

```
        System.out.println("product_price=" +
product_price);
        System.out.println("product_picture=" +
product_picture);
        System.out.println("product_catalog_name=" +
product_catalog_name);
    }
}
```

分页、高亮

```
// 分页和高亮
@Test
public void testSearch3() throws SolrServerException {

    SolrServer solr = new HttpSolrServer(urlString);
    // 查询对象
    SolrQuery query = new SolrQuery();

    // 设置商品分类、关键字查询
    query.setQuery("product_keywords:透明挂钩 ");

    // 分页参数
    // 每页显示记录数
    int pageSize = 2;
    // 当前页码
    int curPage = 2;

    // 开始记录下标
    int begin = pageSize * (curPage - 1);

    // 起始下标
    query.setStart(begin);
    // 结束下标
    query.setRows(pageSize);

    // 设置高亮参数
    query.setHighlight(true); // 开启高亮组件
    query.addHighlightField("product_name");// 高亮字段
    query.setHighlightSimplePre("<span color='red'>");//
前缀标记
    query.setHighlightSimplePost("</span>");// 后缀标记
```



```
// 请求查询
QueryResponse response = solr.query(query);

// 查询结果
SolrDocumentList docs = response.getResults();
// 查询文档总数
System.out.println("查询文档总数" +
docs.getNumFound());

    for (SolrDocument doc : docs) {
        // 商品主键
        String id = (String) doc.getFieldValue("id");
        // 商品名称
        String product_name = (String)
doc.getFieldValue("product_name");
        // 商品价格
        Float product_price = (Float)
doc.getFieldValue("product_price");
        // 商品图片
        String product_picture = (String) doc
            .getFieldValue("product_picture");
        // 商品分类
        String product_catalog_name = (String) doc

        .getFieldValue("product_catalog_name");

        System.out.println("=====");
        System.out.println("id=" + id);
        System.out.println("product_name=" +
product_name);
        System.out.println("product_price=" +
product_price);
        System.out.println("product_picture=" +
product_picture);
        System.out.println("product_catalog_name=" +
product_catalog_name);

        // 高亮信息
        if (response.getHighlighting() != null) {
            if (response.getHighlighting().get(id) !=
null) {
```

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

if (map.get("product_name") !=
null) {

    for (String s :
map.get("product_name")) {

        System.out.println(s);

    }

}

}

}

}
```

相关度排序

索引时设置boost

在创建索引时设置boost值：可以针对Field设置boost，也可以针对Document设置boost值，如下：

```
<add>
<doc>
<field name="id">a01</field>
<field name="product_name" boost="20.0">幸福一家人</field>
<field name="product_description">我们是幸福一家人</field>
</doc>
<doc boost="30.0">
<field name="id">a02</field>
<field name="product_name">幸福两家人</field>
<field name="product_description">我们是幸福两家人</field>
</doc>
</add>
<commit/>
```

针对Field设置boost必须搜索时匹配到该Field在计算相关度得分时才有效，针对Document设置Field则在搜索时只要匹配到该Document的任意Field在计算相关度得分都有效。

测试：

如果要搜索product_description，关键字“幸福”，“a02”的doc会排在前边，因为“a02”是整体设置了boost值。

搜索时设置boost

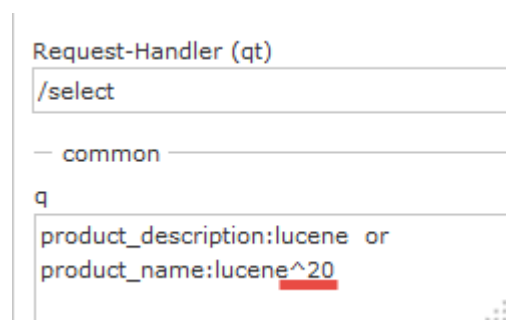
在创建搜索时设置域的boost值：搜索匹配到的域如果boost值设置的高则计算的相关度得分就可能会高，如下：

创建索引：

```
<add>
<doc>
<field name="id">a03</field>
<field name="product_name">lucene实战</field>
<field name="product_description"
>这是一本讲解全文检索技术的书籍</field>
</doc>
<doc>
<field name="id">a04</field>
<field name="product_name">全文检索</field>
<field name="product_description">Lucene是一个全文检索工具包</field>
</doc>
</add>
<commit/>
```

说明：“a03”文档的product_name中有lucene，product_description中没有，“a04”文档中的product_name中没有lucene，“product_description中有lucene。

product_name和product_description组合域搜索时设置product_name的boost值高点，如下：



```
Request-Handler (qt)
/select

— common —

q
product_description:lucene or
product_name:lucene^20
```

当product_name中匹配到重要性要比product_description高，查询结果是product_name中匹配到关键字的排在前边，如下：

```
"response": {
  "numFound": 2,
  "start": 0,
  "docs": [
    {
      "id": "a03",
      "product_name": [
        "lucene实战"
      ],
      "_version_": 1491526603554947000
    },
    {
      "id": "a04",
      "product_name": [
        "全文检索"
      ],
      "_version_": 1491526603571724300
    }
  ]
}
```

7 案例

需求

使用Solr实现电商网站中商品信息搜索功能，可以根据关键字、分类、价格搜索商品信息，也可以根据价格进行排序。
界面如下：

搜索商品

全部商品分类

▼

首页

服装城

食品

团购

夺宝岛

闪购

金融

服饰内衣 > 女装 > T恤

女装

T恤

衬衫

针织衫

雪纺衫

卫衣

马甲

连衣裙

半身裙

牛仔褲

休闲褲

打底褲

正裝褲

小西裝

短外套

风衣

毛呢大衣

真皮皮衣

棉服

T恤 - 商品筛选

商品类别:

幽默杂货

个性男人

新潮美容

益智手工

美味厨房

时尚卫浴

电脑周边

产品配件

环保餐具

保健按摩

另类文化

品质家电

雅致灯饰

闪亮纽扣

魅力女人

创意格架

品味茶杯

阳光车饰

手机饰品

巧妙收纳

四季用品

趣味纸抽

精品数码

与众不同

健康宝宝

布艺毛毯

理财铁罐

价格:

0-9

10-19

20-29

30-39

40-49

50以上

排序:

价格




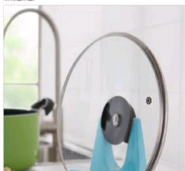
共301个商品

1/6

上一页

下一页

相关商品推荐:



系统架构