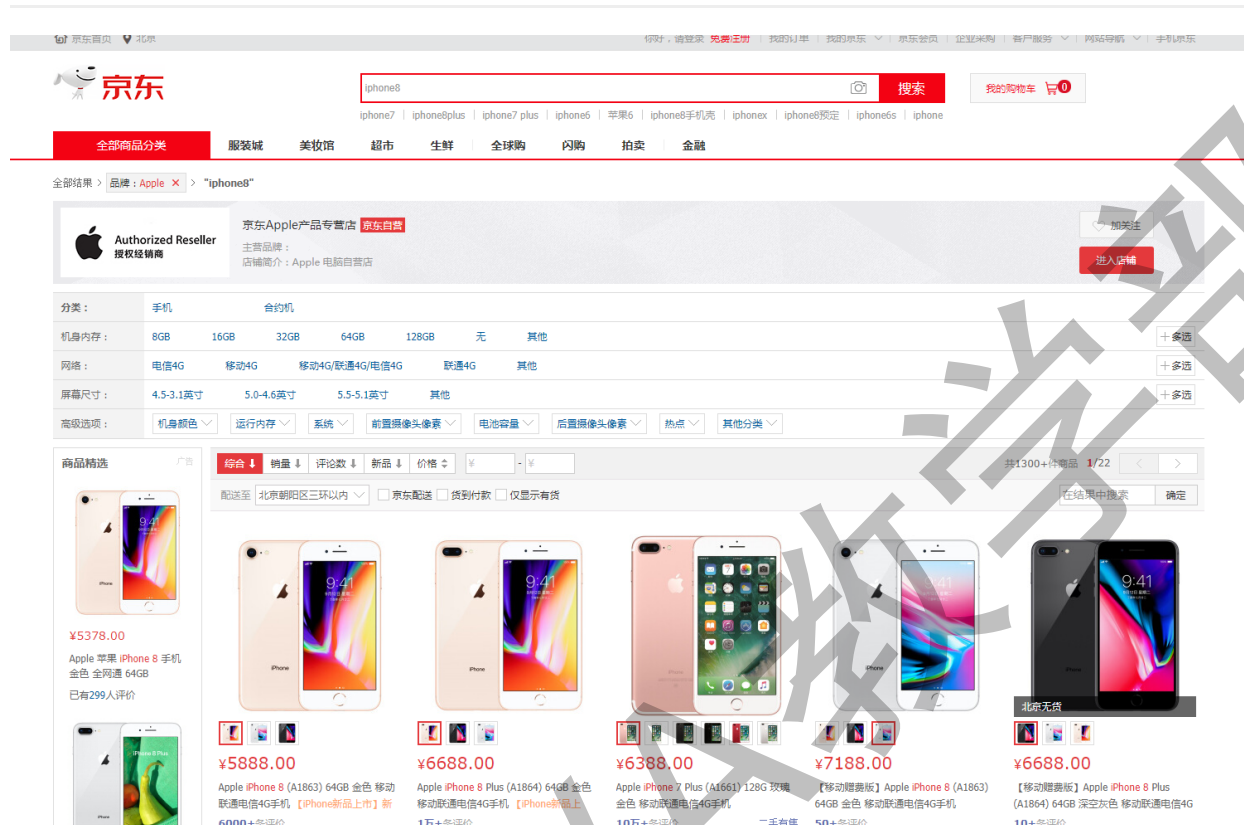


全文搜索技术——Solr



1. 今天内容安排

使用Solr实现信息搜索功能，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。

2. 需求分析

2.1. 实现方法

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。

2.1.1. 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。

2.1.2. 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。

2.2. 什么是solr

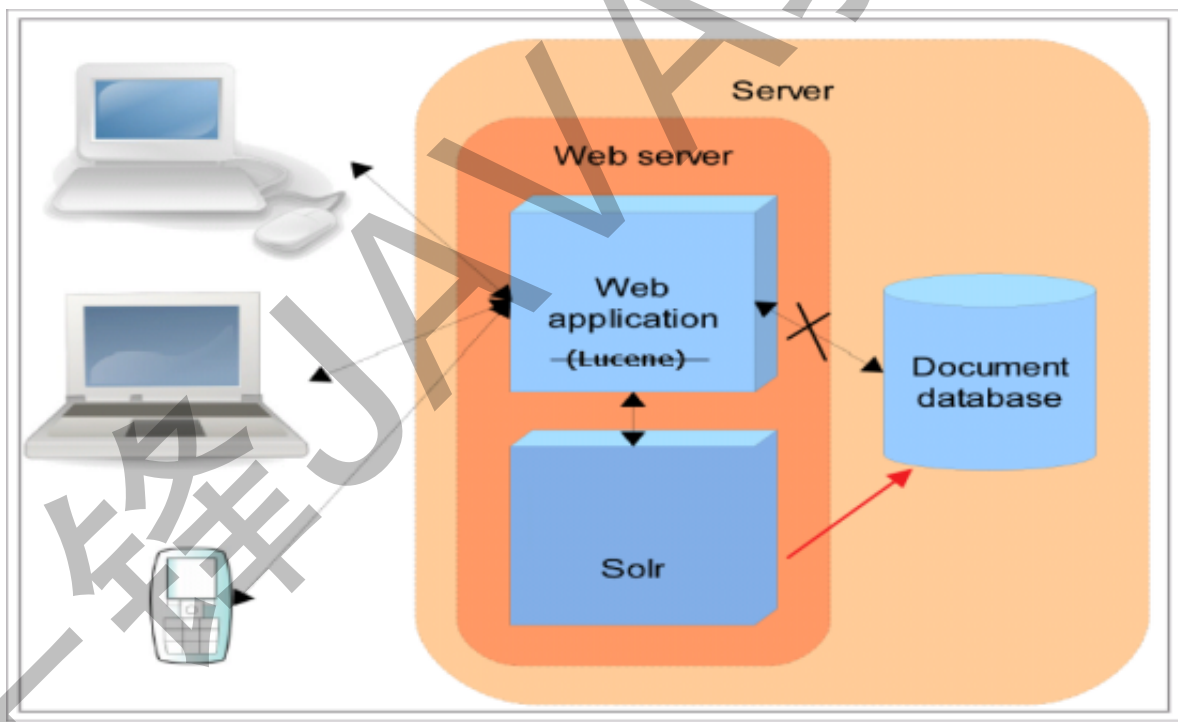
Solr 是Apache下的一个顶级开源项目，采用Java开发，它是基于Lucene的全文搜索服务器。Solr提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展，并对索引、搜索性能进行了优化。

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安装及配置

3.1. Solr的下载

从Solr官方网站 (<http://lucene.apache.org/solr/>) 下载Solr7.1.0

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

3.2. Solr的文件夹结构

将solr-7.1.0.zip解压：

solr-7.1.0		mapic	
名称		修改日期	大小
▶ bin		2017年10月13日 下午4:16	
▶ contrib		2017年10月13日 下午4:16	
▶ dist		今天 上午10:11	
▶ docs		今天 上午10:11	
▶ example		今天 上午10:15	
▶ licenses		今天 上午10:11	
▶ server		今天 上午10:15	
CHANGES.txt		2017年10月13日 下午4:16	7
LICENSE.txt		2017年10月13日 下午4:16	
LUCENE_CHANGES.txt		2017年10月13日 下午4:16	6
NOTICE.txt		2017年10月13日 下午4:16	
README.txt		2017年10月13日 下午4:16	

bin: solr的运行脚本

contrib: solr的一些软件/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：

server:solr 运行的 war 文件和core

licenses: solr相关的一些许可信息

3.3. 运行环境

solr 需要运行在一个Servlet容器中，Solr7.1.0要求jdk使用1.8以上，tomcat8.0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: Solr7.1.0

Jdk: jdk1.8._131

Tomcat: apache-tomcat-8.5

3.4. Solr整合tomcat

3.4.1. Solr Home与SolrCore

创建一个Solr home目录，SolrHome是Solr运行的主目录，目录中包括了运行Solr实例所有的配置文件和数据文件，Solr实例就是SolrCore，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。

server\solr是一个solr home目录结构，如下：

名称
▶ configsets
README.txt
solr.xml
zoo.cfg

上图中“configsets里面的sample_techproducts_configs”是一个SolrCore（Solr实例）目录 不过缺少一个 core.properties， 可以手动创建一个, 也可以从目录solr-7.1.0/example/example-DIH/solr/solr中复制内容如下所示：

另外solr-7.1.0/example/example-DIH/solr/solr 这个目录也是一个 solrCore 目录,也可以用这个来代替上面的sample_techproducts_configs

名称	修改日期
▶ _default	今天 上午10:38
▼ sample_techproducts_configs	今天 上午10:43
▶ conf	今天 上午10:41
core.properties	2017年10月13日 下午4:16

说明：

Solr： 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 。

data目录存放索引文件需要创建, 需要手动创建

3.4.2. 整合步骤

第一步：安装tomcat

第二步：把solr的webapp复制到tomcat 的webapp目录下。

把/solr-7.1.0/server/solr-webapp/webapp 重命名为 solr复制到tomcat的webapps下,重命名不是必须的,只是为了好区分

第三步：把/solr-7.1.0/server/lib目录下的所有的jar包(包括 ext 里面的)添加到solr的 web工程 中的 lib 下

第四步：配置solrHome和solrCore。

在 solr7.1中有两个位置可以找到 solrCore, 但是每个都缺少一个文件,文件都在另外一个中, 分别是solr-7.1.0\example\example-DIH\solr\solr和solr-7.1.0\server\solr

1) 将/solr-7.1.0/server下的 solr复制到/ usr/local 下 并重命名为 solrhome, 重命名不是必须的,只是为了好分辨,因为内部的 configsets 中的core 缺少一个文件,所以删除configsets目录,另行拷贝 core

2) 复制 solrcore把solr-7.1.0/example/example-DIH/solr/solr文件夹复制到/usr/local/solrhome路径下, 并编辑内部的 core.properties ,添加 name=collection1; collection1属于任意内容

在 solrhome 内部创建一个 lib 目录,然后将/solr-7.1.0/dist 下的 solr-dataimport 开头的两个依赖包复制进去

3) 一个solrcore相当于mysql中一个数据库。Solrcore之间是相互隔离。注意core 里面得有 core.properties 文件,core.properties 添加内容 name=collection1

i. 在solrcore中有一个文件夹叫做conf, 包含了索引solr实例的配置信息。

ii. 在conf文件夹下有一个solrconfig.xml。配置实例的相关信息。如果使用默认配置可以不用做任何修改。

Xml的配置信息：

Lib： solr服务依赖的扩展包，默认的路径是collection1\lib文件夹，如果没有 就创建一个

dataDir： 配置了索引库的存放路径。默认路径是collection1\data文件夹，如 果data文件夹不存在，会自动创建。

requestHandler：

```

<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>
    <!-- Default search field
       <str name="df">text</str>
    -->
    <!-- Change from JSON to XML format (the default prior to Solr 7.0)
       <str name="wt">xml</str>
    -->
    <!-- Controls the distribution of a query to shards other than itself.
         Consider making 'preferLocalShards' true when:
         1) maxShardsPerNode > 1
         2) Number of shards > 1
         3) CloudSolrClient or LbHttpSolrServer is used by clients.
         Without this option, every core broadcasts the distributed query to
         a replica of each shard where the replicas are chosen randomly.
         This option directs the cores to prefer cores hosted locally, thus
         preventing network delays between machines.
         This behavior also immunizes a bad/slow machine from slowing down all
         the good machines (if those good machines were querying this bad one).

         Specify this option=false for clients connecting through HttpSolrServer
    -->
    <bool name="preferLocalShards">false</bool>
  </lst>

  <!-- The default high-performance update handler -->
  <updateHandler class="solr.DirectUpdateHandler2">

    <!-- Enables a transaction log, used for real-time get, durability, and
         and solr cloud replica recovery. The log can grow as big as
         uncommitted changes to the index, so use of a hard autoCommit
         is recommended (see below).
         "dir" - the target directory for transaction logs, defaults to the
                  solr data directory.
         "numVersionBuckets" - sets the number of buckets used to keep
         track of max version values when checking for re-ordered
         updates; increase this value to reduce the cost of
         synchronizing access to version buckets during high-volume
         indexing, this requires 8 bytes (long) * numVersionBuckets
         of heap space per Solr core.
    -->
    <updateLog>
      <str name="dir">${solr.ulog.dir}</str>
      <int name="numVersionBuckets">${solr.ulog.numVersionBuckets:65536}</int>
    </updateLog>
  </updateHandler>

```

查询用的 handler

更新 handler

第六步：告诉solr服务器配置文件也就是solrHome的位置。修改web.xml使用jndi的方式告诉solr服务器。

Solr/home名称必须是固定的。

```
<!-- People who want to hardcode their "Solr Home" directly into the  
WAR File can set the JNDI property here...
```

```
-->
```

```
<!--
```

```
<env-entry>
```

```
<env-entry-name>solr/home</env-entry-name>
```

```
<env-entry-value>/put/your/solr/home/here</env-entry-value>
```

```
<env-entry-type>java.lang.String</env-entry-type>
```

```
</env-entry>
```

```
-->
```

```
<env-entry>
```

```
<env-entry-name>solr/home</env-entry-name>
```

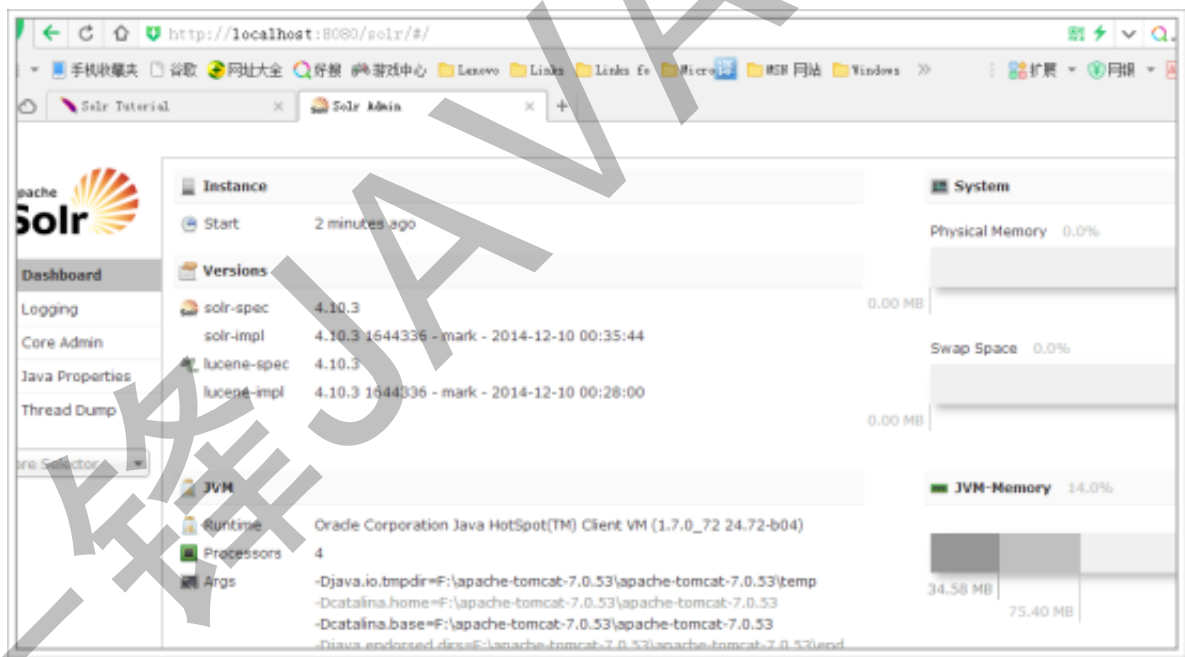
```
<env-entry-value>/usr/local/solrhome</env-entry-value>
```

```
<env-entry-type>java.lang.String</env-entry-type>
```

```
</env-entry>
```

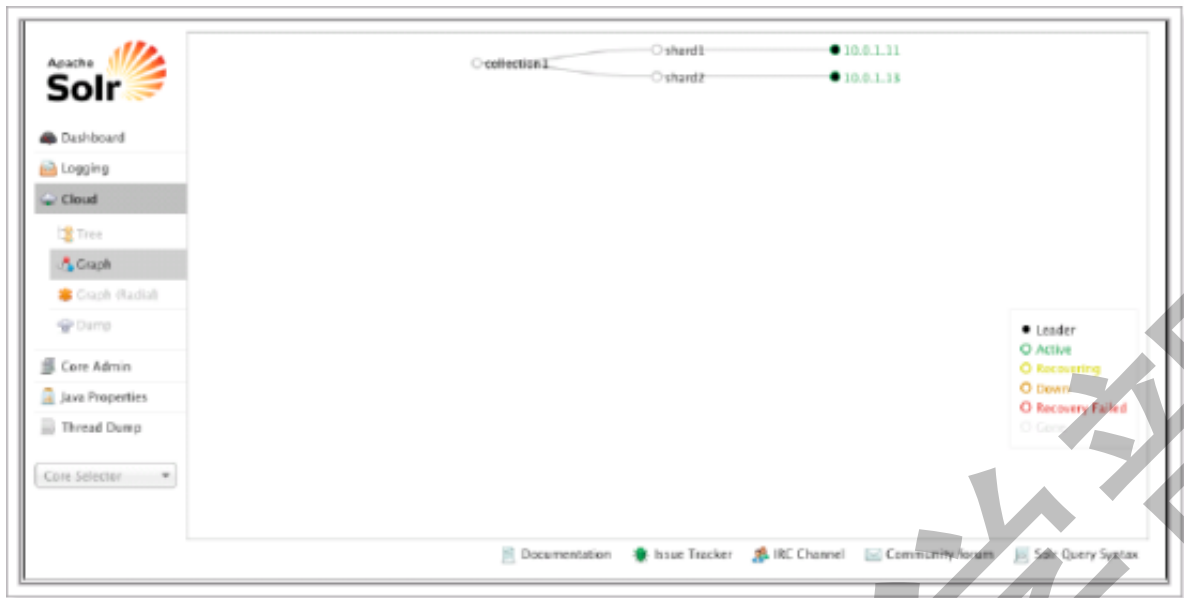
第七步：启动tomcat

第八步：访问<http://localhost:8080/solr/index.html>



3.5. Solr后台管理

3.5.1. 管理界面



3.5.2. 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。

3.5.3. Logging

Solr运行日志信息

3.5.4. Cloud

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

3.5.5. 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。

添加solrcore：

- 第一步：复制collection1改名为collection2
- 第二步：修改core.properties。name=collection2
- 第三步：重启tomcat

3.5.6. java properties

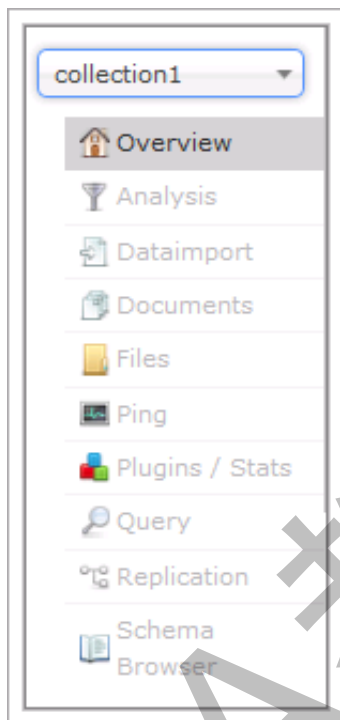
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

3.5.7. Tread Dump

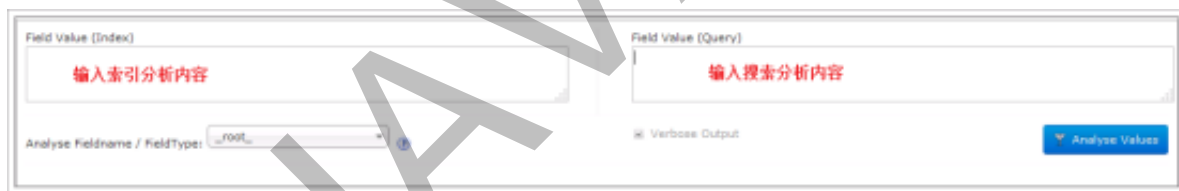
显示Solr Server中当前活跃线程信息，同时也可以跟踪线程运行栈信息。

3.5.8. Core selector

选择一个SolrCore进行详细操作，如下：



3.5.9. Analysis



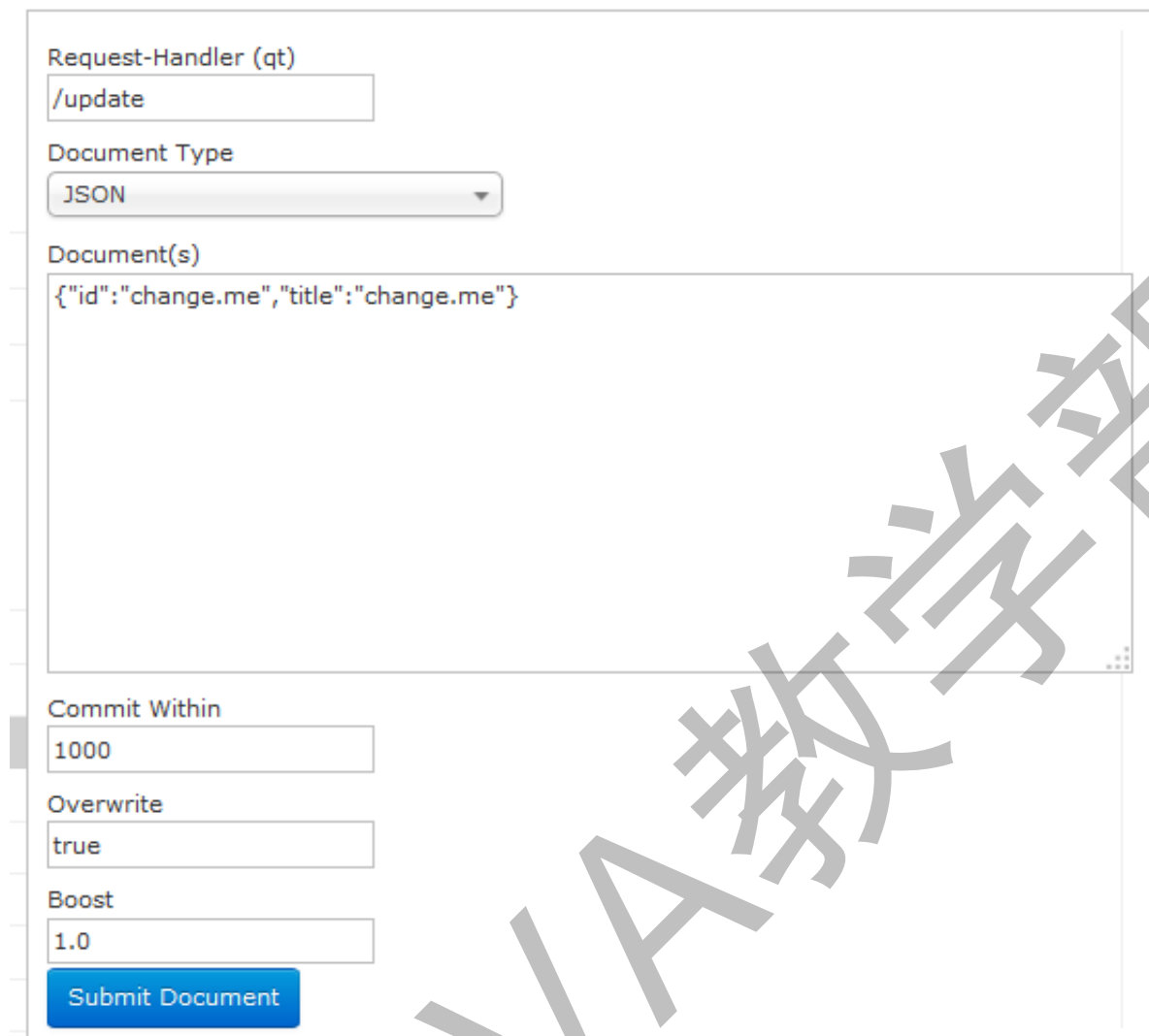
通过此界面可以测试索引分析器和搜索分析器的执行情况。

3.5.10. 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。

3.5.11. 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)

/update

Document Type

JSON

Document(s)

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

Commit Within

1000

Overwrite

true

Boost

1.0

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。

3.5.12. Query

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。

3.6. 配置中文分析器

3.6.1. managed-schema

managed-schema，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。

名称	修改日期
_rest_managed.json	2017年10月13日 下午4:16
_schema_analysis_stopwords_english.json	2017年10月13日 下午4:16
_schema_analysis_synonyms_english.json	2017年10月13日 下午4:16
admin-extra.html	2017年10月13日 下午4:16
admin-extra.menu-bottom.html	2017年10月13日 下午4:16
admin-extra.menu-top.html	2017年10月13日 下午4:16
clustering	2017年10月13日 下午4:16
currency.xml	2017年10月13日 下午4:16
elevate.xml	2017年10月13日 下午4:16
lang	今天 上午10:11
managed-schema	2017年10月13日 下午2:59
mapping-FoldToASCII.txt	2017年10月13日 下午4:16
mapping-ISOLatin1Accent.txt	2017年10月13日 下午4:16
params.json	2017年10月13日 下午4:16
protowords.txt	2017年10月13日 下午4:16
solrconfig.xml	2017年10月13日 下午4:16
spellings.txt	2017年10月13日 下午4:16
stopwords.txt	2017年10月13日 下午4:16
synonyms.txt	2017年10月13日 下午4:16
update-script.js	2017年10月13日 下午4:16
velocity	今天 上午10:11
xslt	今天 上午10:11

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 solr索引字段在solrhome\collection1\conf\schema.xml配置文件中，类似下面这些，包含在<field>之间的。

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,也就是是否能根据该字段进行搜索）,stored（是否被储存,也就是搜索后的结果中是否会显示该字段的内容），multiValued（是否存储多个值,代表是否可以被分词）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",
        "owner": "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: héllo (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。

注意在创建索引时必须指定唯一约束。

copyField复制域 copyField复制域，可以将多个Field复制到一个Field中，以便进行统一的检索：

比如，输入关键字搜索title标题内容content，

定义title、content、text的域：

```
<field name="content" type="text_general" indexed="false" stored="true" 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schem.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：

The screenshot shows a REST client interface with the following details:

- 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:** (button)
- Status:** success
- Response:** {"responseHeader": {"status": 0, "qtime": 3}}

搜索索引：

Request-Handler (qt)

/select

— common —

q

id:100033

fq

sort

start, rows

0

10

fl

df

Raw Query Parameters

key1=val1&key2=val2

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
      }
    ]
  }
}

```

3.6.2. 安装中文分词器

使用IKAnalyzer中文分析器。

解压分词器压缩包

第一步：把ik-analyzer-solr5-5.x.jar添加到solr/WEB-INF/lib目录下。

第二步：复制IKAnalyzer的配置文件IKAnalyzer.cfg.xml和自定义词典ext.dic和停用词词典stopword.dic到solr的classpath下。(就是 solrweb 项目的 classes 目录中)

第三步：在managed-schema.xml中添加一个自定义的fieldType，使用中文分析器。

<!-- IKAnalyzer定义一个域类型,这个类型相当于数据库中表中列的类型,比如 int,varchar等,每种类型有不同的处理机制-->

```

<fieldType name="text_ik" class="solr.TextField">
  <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>
</fieldType>

```

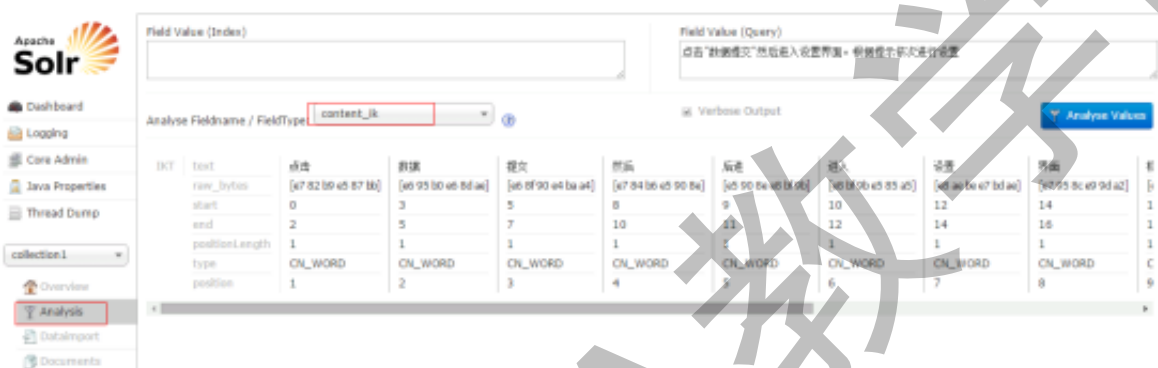
第四步：定义field，指定field的type属性为text_ik

<!--IKAnalyzer Field 定义一个域,指定类型,和其他属性,域就相当于表中的列,类型就相当于列的类型-->

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

第四步：重启tomcat

测试：



3.7. 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：，此处可以使用自己的配置,这里仅做演示用,下面的数据导入中的3.8.2第三部配置和当前位置匹配,其实可以使用一两列来演示数据,取决于自己

```
<!--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"/>
```

3.8. 维护索引

3.8.1. 添加/更新文档

添加单个文档

Request-Handler (qt)
/update

Document Type
JSON

Document(s)
{
 "id": "a001",
 "title_kk": "第一个索引文档",
 "content_kk": "点击"数据提交"然后进入设置界面。根据提示依次进行设置"
}

Commit Within
1000

Overwrite
true

Boost
1.0
















Submit Document

Status: success
Response:
{
 "responseHeader": {
 "status": 0,
 "QTime": 177
 }
}

3.8.2. 批量导入数据

使用dataimport插件批量导入数据。

第一步：把dataimport插件依赖的jar(dist 目录下)包添加到solrcore (collection1\lib) 中,或者是 solr 的 web 项目 lib 中,并且复制数据库连接依赖包

名称	修改日期
 solr-analysis-extras-7.1.0.jar	2017年10月13日 下午4:15
 solr-analytics-7.1.0.jar	2017年10月13日 下午4:15
 solr-cell-7.1.0.jar	2017年10月13日 下午4:15
 solr-clustering-7.1.0.jar	2017年10月13日 下午4:15
 solr-core-7.1.0.jar	2017年10月13日 下午4:16
 solr-dataimporthandler-7.1.0.jar	2017年10月13日 下午4:15
 solr-dataimporthandler-extras-7.1.0.jar	2017年10月13日 下午4:15
 solr-langid-7.1.0.jar	2017年10月13日 下午4:15
 solr-ltr-7.1.0.jar	2017年10月13日 下午4:15
 solr-solrj-7.1.0.jar	2017年10月13日 下午4:16
 solr-test-framework-7.1.0.jar	2017年10月13日 下午4:15
 solr-uima-7.1.0.jar	2017年10月13日 下午4:15
 solr-velocity-7.1.0.jar	2017年10月13日 下午4:16
▶  solrj-lib	今天 上午10:11
▶  test-framework	今天 上午10:11

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

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

    <lst name="defaults">
<!--执行数据导入的配置文件-->
        <str name="config">data-config.xml</str>

    </lst>
</requestHandler>
```

第三步：创建一个data-config.xml，保存到collection1\conf\目录下

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

<dataConfig>
<!--配置数据库连接-->
    <dataSource type="JdbcDataSource"

        driver="com.mysql.jdbc.Driver"

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

        user="root"

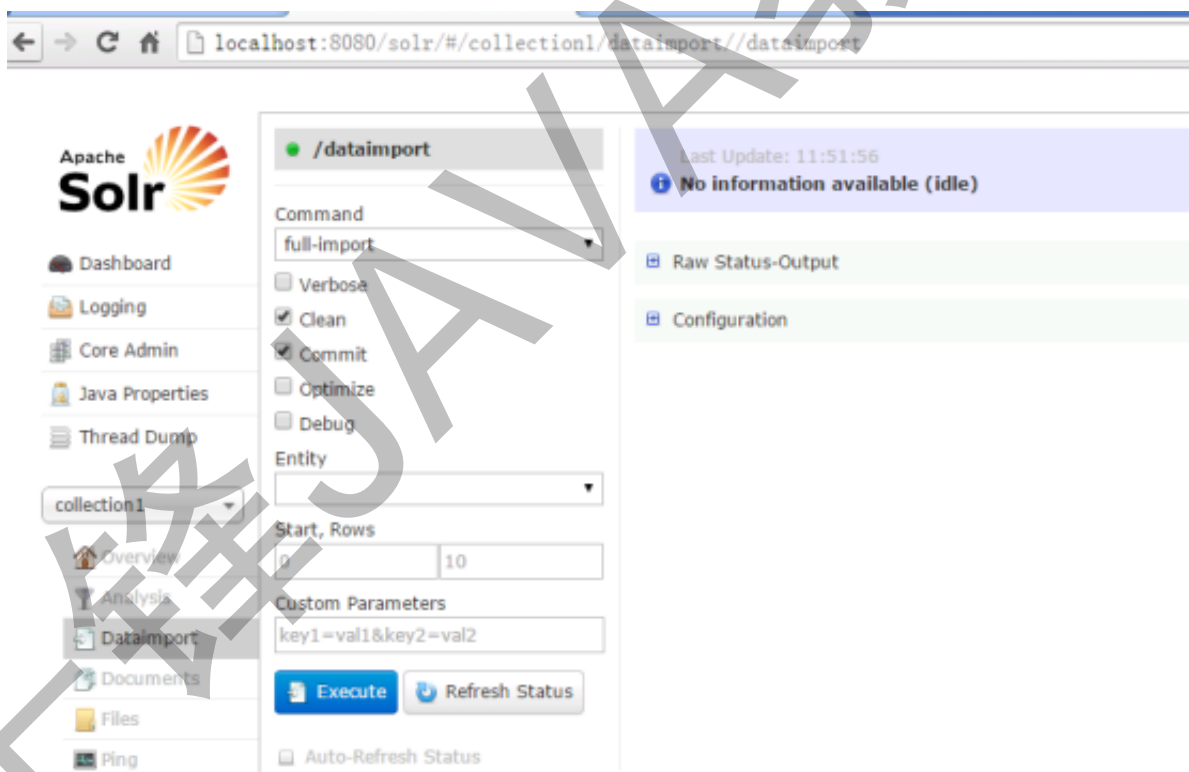
        password="qishimeiyoumima"/>

    <document>
<!--配置实体，此处的配置就类似于 hibernate 的映射,主要是执行一条 sql 语句,然后将返回的列
和 solr 中的域进行对应,这样查询完成后会自动将对应列的数据放到solr 的对应域中-->
```

```
<entity name="product" query="SELECT
pid,name,catalog_name,price,description,picture FROM products ">
  <!--column 代表 sql 语句返回的列名字,如果有别名,代表别名, name 代表的是在 solr
中用于存放该数据的域名 也就是 field 名字
此处和上面3.7步骤的配置域对应,为了演示,此处和上面做自己的对应即可,不一定非得照着来,毕竟表的
列越多,写起来越麻烦
-->
  <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>
```

第四步：重启tomcat



第五步：点击“execute”按钮导入数据 到入数据前会先清空索引库，然后再导入。

3.8.3. 删除文档(注意需要提交)

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>
```

3.9. 查询索引

通过/select搜索索引，Solr制定一些参数完成不同需求的搜索：

3.9.1. 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:与钟不同

3.9.2. 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]

3.9.3. sort

- 排序，格式：sort=+[,+]... 。示例：

按价格降序

sort

product_price desc

3.9.4. start

- 分页显示使用，开始记录下标，从0开始

3.9.5. 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	
<input type="text" value="0"/>	<input type="text" value="10"/>

显示前10条。

3.9.6. fl

- 指定返回那些字段内容，用逗号或空格分隔多个。

fl
<input type="text" value="duct_picture,product_name,product_price"/>

显示商品图片、商品名称、商品价格

3.9.7. df

-指定一个搜索Field

df
<input type="text" value="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

3.9.8. 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。

3.9.9. hl

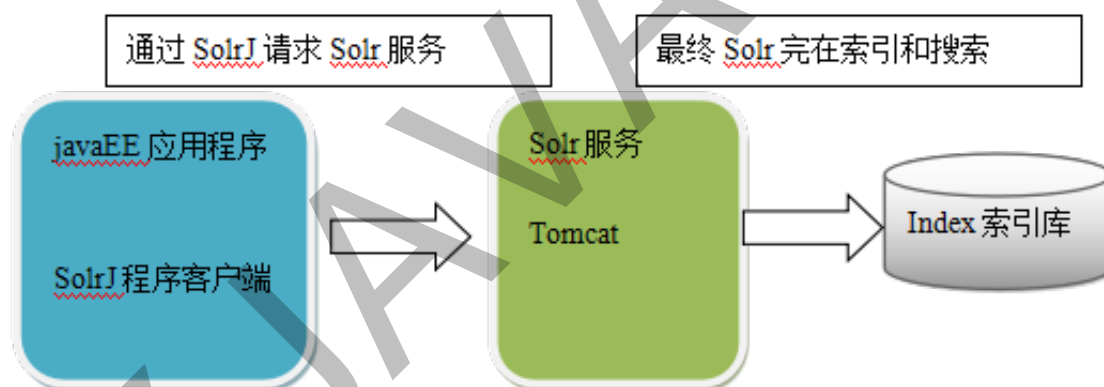
-是否高亮,设置高亮Field, 设置格式前缀和后缀。

hl	<input checked="" type="checkbox"/>
hl.fl	product_name
hl.simple.pre	
hl.simple.post	
















4. 使用Solrj管理索引库

4.1. 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:



4.2. 依赖的jar包

名称	修改日期	大小
 solr-analysis-extras-7.1.0.jar	2017年10月13日 下午4:15	
 solr-analytics-7.1.0.jar	2017年10月13日 下午4:15	
 solr-cell-7.1.0.jar	2017年10月13日 下午4:15	
 solr-clustering-7.1.0.jar	2017年10月13日 下午4:15	
 solr-core-7.1.0.jar	2017年10月13日 下午4:16	
 solr-dataimporthandler-7.1.0.jar	2017年10月13日 下午4:15	
 solr-dataimporthandler-extras-7.1.0.jar	2017年10月13日 下午4:15	
 solr-langid-7.1.0.jar	2017年10月13日 下午4:15	
 solr-ltr-7.1.0.jar	2017年10月13日 下午4:15	
 solr-solrj-7.1.0.jar	2017年10月13日 下午4:16	
 solr-test-framework-7.1.0.jar	2017年10月13日 下午4:15	
 solr-uima-7.1.0.jar	2017年10月13日 下午4:15	
 solr-velocity-7.1.0.jar	2017年10月13日 下午4:16	
▶  solrj-lib	今天 上午10:11	
▶  test-framework	今天 上午10:11	

4.3. 添加文档

4.3.1. 实现步骤

第一步：创建一个java工程

第二步：导入jar包。包括solrj的jar包。还需要 server中的lib下的ext中的日志包

第三步：和Solr服务器建立连接。HttpClient对象建立连接。

第四步：创建一个SolrInputDocument对象，然后添加域。

第五步：将SolrInputDocument添加到索引库。

第六步：提交。

4.3.2. 代码实现

```
//向索引库中添加索引

@Test

public void addDocument() throws Exception {

    //和solr服务器创建连接

    //参数：solr服务器的地址

    //SolrServer solrserver = new HttpSolrServer("http://localhost:8080/solr"); //
    老版本是 server 而且不需要加 core
    HttpSolrClient client=new
    HttpSolrClient.Builder().withBaseSolrUrl("http://10.0.114.174:8080/solr/collection
    1").build();
```

```

//创建一个文档对象

SolrInputDocument document = new SolrInputDocument();

//向文档中添加域

//第一个参数：域的名称，域的名称必须是在schema.xml中定义的

//第二个参数：域的值

document.addField("id", "c0001");

document.addField("title_ik", "使用solrJ添加的文档");

document.addField("content_ik", "文档的内容");

document.addField("product_name", "商品名称");

//把document对象添加到索引库中

client.add(document);

//提交修改

client.commit();

}

```

4.4. 删除文档

4.4.1. 根据id删除

```

//删除文档，根据id删除

@Test

public void deleteDocumentById() throws Exception {

    //创建连接
    ///SolrServer solrserver = new HttpSolrServer("http://localhost:8080/solr"); //老
    版本是 server 而且不需要加 core
    HttpSolrClient client=new
    HttpSolrClient.Builder().withBaseSolrUrl("http://10.0.114.174:8080/solr/collection
    1").build(); //老版本是 server 而且不需要加 core

```

```
//根据id删除文档

client.deleteById("c0001");

//提交修改

client.commit();

}
```

4.4.2. 根据查询删除

查询语法完全支持Lucene的查询语法。

```
//根据查询条件删除文档

@Test

public void deleteDocumentByQuery() throws Exception {

//创建连接

//SolrServer solrserver = new HttpSolrServer("http://localhost:8080/solr"); //
老版本是 server 而且不需要加 core
HttpSolrClient client=new
HttpSolrClient.Builder().withBaseSolrUrl("http://10.0.114.174:8080/solr/collection
1").build();

//根据查询条件删除文档

client.deleteByQuery("*:*");

//提交修改

client.commit();

}
```

4.5. 修改文档

在solrj中修改没有对应的update方法，只有add方法，只需要添加一条新的文档，和被修改的文档id一致就，可以修改了。本质上就是先删除后添加。

4.6. 查询文档

4.6.1. 简单查询

```
//查询索引

@Test

public void queryIndex() throws Exception {

    //创建连接

    //SolrServer solrserver = new HttpSolrServer("http://localhost:8080/solr"); //
    老版本是 server 而且不需要加 core
    HttpSolrClient client=new
    HttpSolrClient.Builder().withBaseSolrUrl("http://10.0.114.174:8080/solr/collection
    1").build();

    //创建一个query对象

    SolrQuery query = new SolrQuery();

    //设置查询条件

    query.setQuery("*:*");

    //执行查询

    QueryResponse queryResponse = client.query(query);

    //取查询结果

    SolrDocumentList solrDocumentList = queryResponse.getResults();

    //共查询到商品数量

    System.out.println("共查询到商品数量:" + solrDocumentList.getNumFound());

    //遍历查询的结果

    for (SolrDocument solrDocument : solrDocumentList) {

        System.out.println(solrDocument.get("id"));
```

```

System.out.println(solrDocument.get("product_name"));

System.out.println(solrDocument.get("product_price"));

System.out.println(solrDocument.get("product_catalog_name"));

System.out.println(solrDocument.get("product_picture"));

    }

}

```

4.6.2. 复杂查询

```

//其中包含查询、过滤、分页、排序、高亮显示等处理。

//复杂查询索引

@Test

public void queryIndex2() throws Exception {

    //创建连接

    //SolrServer solrserver = new HttpSolrServer("http://localhost:8080/solr"); //
    老版本是 server 而且不需要加 core
    HttpSolrClient client=new
    HttpSolrClient.Builder().withBaseSolrUrl("http://10.0.114.174:8080/solr/collection
    1").build();

    //创建一个query对象

    SolrQuery query = new SolrQuery();

    //设置查询条件
    /**q :查询字符串，这个是必须的。如果查询所有*:* ，根据指定字段查询（Name:张三 AND
    Address:北京） 注意：AND要大写 否则会被当做默认OR*/

    query.setQuery("钻石");

    /**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：q=查询全部&fq=只要title得值为:伟帅你是我偶吧*/
    query.setFilterQueries("product_catalog_name:幽默杂货");

```

```
//排序条件
/*sort :
 * 排序, 格式: sort=<field name>+<desc|asc>[,<field name>+<desc|asc>]... 。
 * 示例: (score desc, price asc) 表示先 “score” 降序, 再 “price” 升序, 默认
是相关性降序。
```

```
SortClause sort=new SortClause("id", ORDER.desc);
query.setSort(sort);//可以添加集合*/
query.setSort("product_price", ORDER.asc);
```

```
//分页处理
```

```
query.setStart(0);
```

```
query.setRows(10);
```

```
//结果中域的列表
```

```
query.setFields("id","product_name","product_price","product_catalog_name","product_picture");
```

```
//设置默认搜索域
```

```
query.set("df", "product_keywords");
```

```
//高亮显示
```

```
query.setHighlight(true);
```

```
//高亮显示的域
```

```
query.addHighlightField("product_name");
```

```
//高亮显示的前缀
```

```
query.setHighlightSimplePre("<em>");
```

```
//高亮显示的后缀
```

```
query.setHighlightSimplePost("</em>");
```

```
//执行查询
```

```
QueryResponse queryResponse = client.query(query);
```

```
//取查询结果
```

```
SolrDocumentList solrDocumentList = queryResponse.getResults();
```



```
//共查询到商品数量

System.out.println("共查询到商品数量:" + solrDocumentList.getNumFound());

//遍历查询的结果

for (SolrDocument solrDocument : solrDocumentList) {

    System.out.println(solrDocument.get("id"));

    //取高亮显示

    String productName = "";

    Map<String, Map<String, List<String>>> highlighting =
    queryResponse.getHighlighting();

    List<String> list =
    highlighting.get(solrDocument.get("id")).get("product_name");

    //判断是否有高亮内容

    if (null != list) {

        productName = list.get(0);

    } else {

        productName = (String) solrDocument.get("product_name");

    }

    System.out.println(productName);

    System.out.println(solrDocument.get("product_price"));

    System.out.println(solrDocument.get("product_catalog_name"));

    System.out.println(solrDocument.get("product_picture"));

}

}

/*【注：以上是比较常用的参数，当然具体的参数使用还是多看Solr官方的技术文档以及一些大神的博文日志，这里只是抛砖引玉】*/

/*二、 Solr运算符
```

1. “:” 指定字段查指定值，如返回所有值*:*
2. “?” 表示单个任意字符的通配
3. “*” 表示多个任意字符的通配（不能在检索的项开始使用*或者?符号）
4. “~” 表示模糊检索，如检索拼写类似于”roam”的项这样写：roam~将找到形如foam和rooms的单词；roam~0.8，检索返回相似度在0.8以上的记录。
5. 邻近检索，如检索相隔10个单词的”apache”和”jakarta”，”jakarta apache”~10
6. “^” 控制相关度检索，如检索jakarta apache，同时希望去让”jakarta”的相关度更加好，那么在其后加上”^”符号和增量值，即jakarta^4 apache
7. 布尔操作符AND、||
8. 布尔操作符OR、&&
9. 布尔操作符NOT、!、- （排除操作符不能单独与项使用构成查询）
10. “+” 存在操作符，要求符号”+”后的项必须在文档相应的域中存在
11. () 用于构成子查询
12. [] 包含范围检索，如检索某时间段记录，包含头尾，date:[200707 TO 200710]
13. { } 不包含范围检索，如检索某时间段记录，不包含头尾
date:{200707 TO 200710}
14. / 转义操作符，特殊字符包括+ - && || ! () { } [] ^ ” ~ * ? : /

注：①“+”和”-“表示对单个查询单元的修饰，and 、or 、 not 是对两个查询单元是否做交集或者做差集还是取反的操作的符号

比如:AB:china +AB:america ,表示的是AB:china忽略不计可有可无，必须满足第二个条件才是对的,而不是你所认为的必须满足这两个搜索条件

如果输入:AB:china AND AB:america ,解析出来的结果是两个条件同时满足，即+AB:china AND +AB:america或+AB:china +AB:america

总而言之，查询语法： 修饰符 字段名:查询关键词 AND/OR/NOT 修饰符 字段名:查询关键词
*/

5 错误

1.java.lang.NoClassDefFoundError: com/codahale/metrics/MetricSet

缺少 jar 包,复制server/lib 下面的依赖包到项目即可

2. Unsupported major.minor version 52.0

jdk 版本问题,使用 jdk1.8即可,最好不要使用 openjdk

3. HTTP Status 403 - Access to the requested resource has been denied

没有权限, solr7.1.0 默认是需要权限访问的,需要配置权限

可以通过删除 solr web.xml 中最下面的<auth-constraint/>,或者指定权限的用户,添加对应权限用户,每次使用用户登录

例如:

修改tomcat的tomcat-user.xml ,</tomcat-user>上面添加:

```
<role rolename="solr"/>
```

```
<user username="admin" password="admin" roles="solr"/>
```

然后在 solr的 web.xml 中最下面的auth-constraint添加指定角色和用户

```
<auth-constraint>
```

```
    <role-name>solr</role-name>
```

```
    <role-name>admin</role-name>
```

```
</auth-constraint>
```

4. HTTP Status 500 - Filter execution threw an exception

```
java.lang.NoSuchMethodError: javax.servlet.ServletInputStream.isFinished()Z
```

```
org.apache.solr.servlet.SolrDispatchFilter.consumeInputFully(SolrDispatchFilter.java:409)
```

```
org.apache.solr.servlet.SolrDispatchFilter.doFilter(SolrDispatchFilter.java:399)
```

```
org.apache.solr.servlet.SolrDispatchFilter.doFilter(SolrDispatchFilter.java:326)
```

使用8.5版本的 tomcat

5.HTTP Status 404 – Not Found

如果在前面条件都 ok 的情况下 访问 solr 网址无法访问,需要注意, solr7.1.0需要使用/index.html 来访问,不能直接访问项目名