**Solr**

**全文搜索服务**

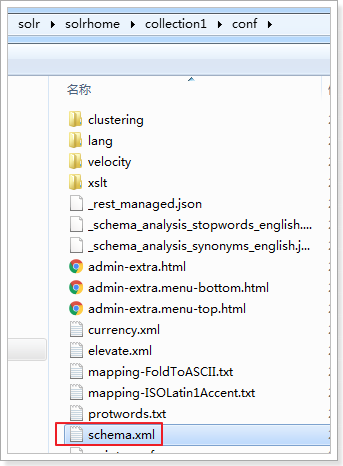
# 课程计划

1. solr的使用
   1. schma.xml文件
   2. 配置中文分词器
   3. 配置业务域
   4. DataimportHandler插件
2. Solrj的复杂查询
   1. solr的查询语法
   2. solrj的复杂查询
3. 京东案例

# solr基本使用

## schema.xml

schema.xml文件在SolrCore的conf目录下，在此配置文件中定义了域以及域的类型等一些配置。在solr中域必须先定义后使用。



### field

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

* Name：域的名称
* Type：域的类型
* Indexed：是否索引
* Stored：是否存储
* Required：是否必须
* multiValued：是否是多值，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）

### dynamicField（动态域）

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

Name：动态域的名称，是一个表达式，\*匹配任意字符，只要域的名称和表达式的规则能够匹配就可以使用。

例如：搜索时查询条件[product\_i：钻石]就可以匹配这个动态域，可以直接使用，不用单独再定义一个product\_i域。

### uniqueKey

<uniqueKey>id</uniqueKey>

相当于主键，每个文档中必须有一个id域。

### copyField（复制域）

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

可以将多个Field复制到一个Field中，以便进行统一的检索。当创建索引时，solr服务器会自动的将源域的内容复制到目标域中。

* source：源域
* dest：目标域，搜索时，指定目标域为默认搜索域，可以提高查询效率。

定义目标域：

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

目标域必须要使用：multiValued="true"

### fieldType（域类型）

<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" />

<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"/>

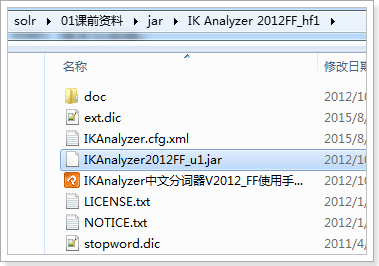
</analyzer>

</fieldType>

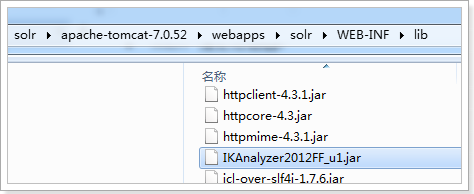
* name：域类型的名称
* class：指定域类型的solr类型。
* analyzer：指定分词器。在FieldType定义的时候最重要的就是定义这个类型的数据在建立索引和进行查询的时候要使用的分析器analyzer，包括分词和过滤。
* type：index和query。Index 是创建索引，query是查询索引。
* tokenizer：指定分词器
* filter：指定过滤器

## 配置中文分析器

使用IKAnalyzer中文分析器

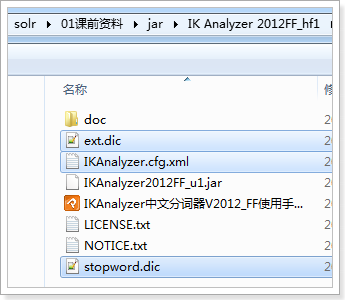


第一步：把IKAnalyzer2012FF\_u1.jar添加到solr/WEB-INF/lib目录下。

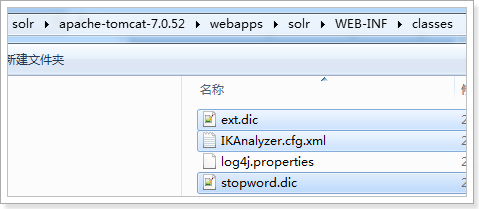


第二步：复制IKAnalyzer的配置文件和自定义词典和停用词词典到solr的solr/WEB-INF/classes目录下。

复制IK分词器配置文件、自定义词典、停用词词典



粘贴到Tomcat的solr的/WEB-INF/classes目录下



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

<!-- IKAnalyzer-->

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

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

</fieldType>

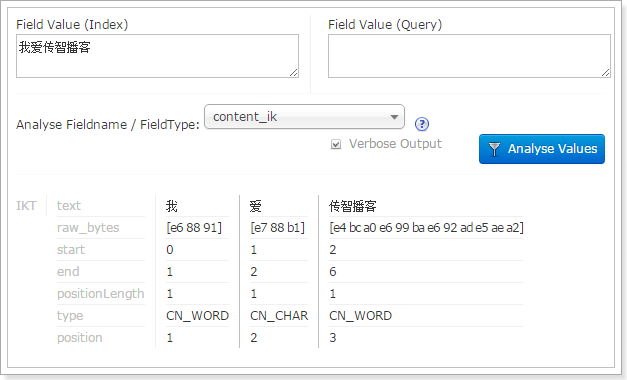
第四步：在schema.xml中添加field，指定field的type属性为text\_ik

<!--IKAnalyzer Field-->

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

第五步：重启tomcat

效果：



## 配置业务Field

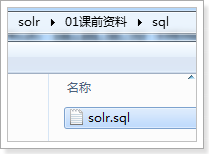
### 需求

要使用solr实现网站中商品搜索，需要 将mysql数据库中数据在solr中创建索引。

1. 需要在solr的schema.xml文件定义要存储的商品Field。
2. 需要把MySQL的数据导入到solr索引库中
3. 开发搜索功能

### 数据库添加数据

在数据库中运行solr.sql脚本

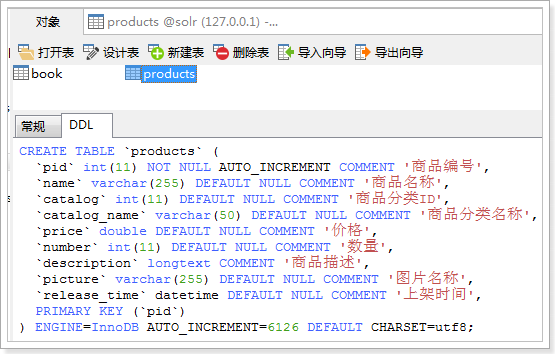


### 定义Field

先确定定义的商品document的Field域有哪些？

可以根据mysql数据库中商品表的字段来确定：

products商品表：



Schema.xml中配置业务域

<!--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"/>

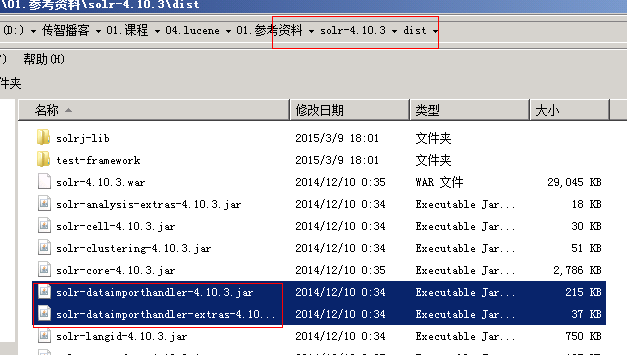
<copyField source="product\_name" dest="product\_keywords"/>

<copyField source="product\_description" dest="product\_keywords"/>

## dataimportHandler插件

使用dataimport插件批量导入数据。

第一步：把dataimport插件依赖的jar包添加到solrcore（collection1\lib）中, 还需要mysql的数据库驱动。



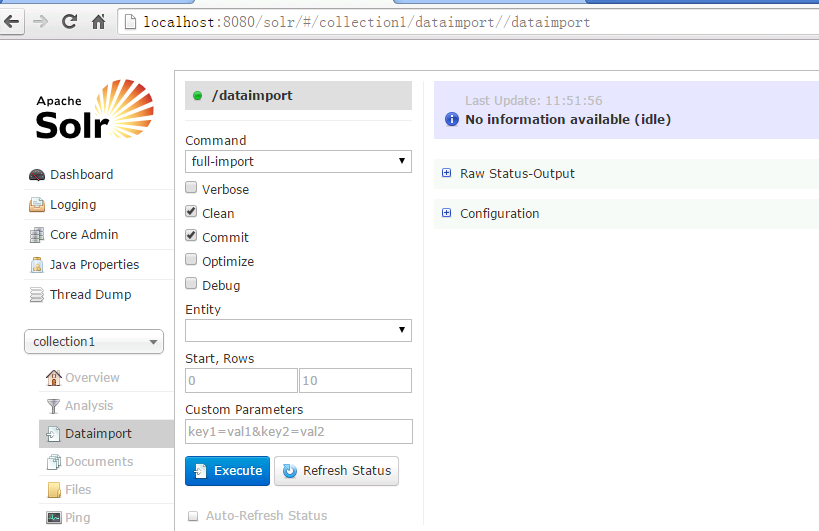
第二步：配置solrconfig.mxl文件，添加一个requestHandler。

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

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

|  |
| --- |
| <?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="root"/>  <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> |

第四步：重启tomcat



第五步：点击“execute”按钮导入数据

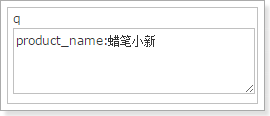
**注意:导入数据前会先清空索引库，然后再导入。**

# solrj的复杂查询

## solr的查询语法

1. q： 查询关键字，必须的。

请求的q是字符串，如果查询所有使用\*:\*

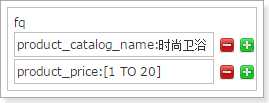


1. fq: (filter query)过滤查询

作用：在q查询符合结果中同时是fq查询符合的

请求fq是一个数组（多个值）

过滤查询价格从1到20的记录。



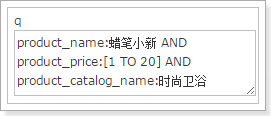
也可以使用“\*”表示无限，例如：

20以上：product\_price:[20 TO \*]

20以下：product\_price:[\* TO 20]

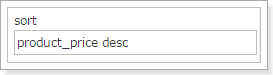
也可以在“q”查询条件中使用product\_price:[1 TO 20]，

如下效果和上面一样：



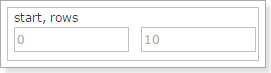
1. sort: 排序，desc代表降序，asc代表升序

按照价格升序排

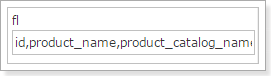


1. start: 分页显示使用，开始记录下标，从0开始

rows: 指定返回结果最多有多少条记录，配合start来实现分页。

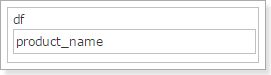


1. fl: (Field List)指定返回那些字段内容，用逗号或空格分隔多个。

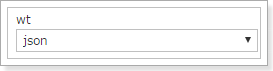


显示商品id、商品名称、商品分类名称

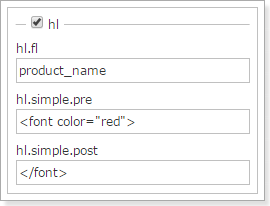
1. df: 指定默认搜索Field



1. wt: (writer type)指定输出格式，可以有 xml, json, php, phps

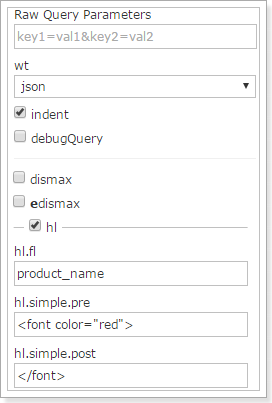
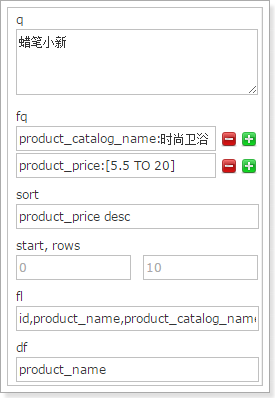


1. hl: 是否高亮 ,设置高亮Field，设置格式前缀和后缀。



## solrj的复杂查询

页面的查询条件，复杂查询条件和页面的查询条件一致



代码实现：

|  |
| --- |
| //复杂查询索引  @Test  **public** **void** queryIndex2() **throws** Exception {  //创建连接  SolrServer solrServer = **new** HttpSolrServer("http://localhost:8080/solr");  //创建一个query对象  SolrQuery query = **new** SolrQuery();  //设置查询条件  query.setQuery("钻石");  //过滤条件  query.setFilterQueries("product\_catalog\_name:幽默杂货");  //排序条件  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 = solrServer.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"));    }  } |

# 案例

## 需求

使用Solr实现电商网站中商品信息搜索功能，可以根据关键字搜索商品信息，根据商品分类、价格过滤搜索结果，也可以根据价格进行排序，实现分页。

界面如下：



### 架构分析

spring容器

表现层

springmvc

Service层

Dao层

Solr索引库

mysql数据库

Tomcat

Solr服务

索引、搜索请求

架构分为：

* 1. solr服务器
  2. 自己的web服务器（需要开发）
  3. 数据库mysql

自己开发的应用

* 1. Controller

获取搜索条件，并响应搜索结果到前台页面。

* 1. Service

使用solrj来调用solr的服务进行索引和搜索

Service调用dao进行商品数据的维护时，要同步更新索引库（不实现）

* 1. Dao（本案例不实现）

对商品数据进行维护和查询

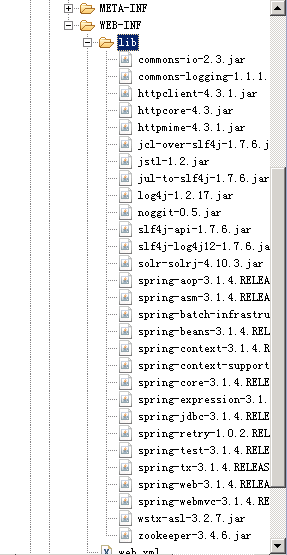
## 环境准备

* Solr：4.10.3
* Jdk环境：1.7
* IDE环境：eclipse Mars2
* 服务器：Tomcat 7

## 工程搭建

创建一个web工程导入jar包

1. springmvc的相关jar包
2. solrJ的jar包
3. Example\lib\ext下的jar包



## Dao

功能：接收service层传递过来的参数，根据参数查询索引库，返回查询结果。

参数：SolrQuery对象

返回值：一个商品列表List<ProductModel>，还需要返回查询结果的总数量。

返回：ResultModel

方法定义：ResultModel queryProduct(SolrQuery query) throws Exception;

商品对象模型：

**public** **class** ProductModel {

// 商品编号

**private** String pid;

// 商品名称

**private** String name;

// 商品分类名称

**private** String catalog\_name;

// 价格

**private** **float** price;

// 商品描述

**private** String description;

// 图片名称

**private** String picture;

}

返回值对象模型

**public** **class** ResultModel {

// 商品列表

**private** List<ProductModel> productList;

// 商品总数

**private** Long recordCount;

// 总页数

**private** **int** pageCount;

// 当前页

**private** **int** curPage;

}

|  |
| --- |
| @Repository  **public** **class** ProductDaoImpl **implements** ProductDao {  @Autowired  **private** SolrServer solrServer;    @Override  **public** ResultModel queryProduct(SolrQuery query) **throws** Exception {    ResultModel resultModel = **new** ResultModel();  //根据query对象查询商品列表  QueryResponse queryResponse = solrServer.query(query);  SolrDocumentList solrDocumentList = queryResponse.getResults();  //取查询结果的总数量  resultModel.setRecordCount(solrDocumentList.getNumFound());  List<ProductModel> productList = **new** ArrayList<>();  //遍历查询结果  **for** (SolrDocument solrDocument : solrDocumentList) {  //取商品信息  ProductModel productModel = **new** ProductModel();  productModel.setPid((String) 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");  **if** (**null** != list) {  productName = list.get(0);  } **else** {  productName = (String) solrDocument.get("product\_name");  }  productModel.setName(productName);  productModel.setPrice((**float**) solrDocument.get("product\_price"));  productModel.setCatalog\_name((String) solrDocument.get("product\_catalog\_name"));  productModel.setPicture((String) solrDocument.get("product\_picture"));  //添加到商品列表  productList.add(productModel);  }  //商品列表添加到resultmodel中  resultModel.setProductList(productList);  **return** resultModel;  }  } |

## Service

功能：接收action传递过来的参数，根据参数拼装一个查询条件，调用dao层方法，查询商品列表。接收返回的商品列表和商品的总数量，根据每页显示的商品数量计算总页数。

参数：

1、查询条件：字符串

2、商品分类的过滤条件：商品的分类名称，字符串

3、商品价格区间：传递一个字符串，满足格式：“0-100、101-200、201-\*”

4、排序条件：页面传递过来一个升序或者降序就可以，默认是价格排序。0：升序1：降序

5、分页信息：每页显示的记录条数创建一个常量60条。传递一个当前页码就可以了。

返回值：ResultModel

方法定义：ResultModel queryProduct(String queryString, String caltalog\_name, String price, String sort, Integer page) throws Exception;

|  |
| --- |
| @Service  **public** **class** ProductServiceImpl **implements** ProductService {  @Autowired  **private** ProductDao productDao;    @Override  **public** ResultModel queryProduct(String queryString, String caltalog\_name,  String price, String sort, Integer page) **throws** Exception {  //拼装查询条件  SolrQuery query = **new** SolrQuery();  //查询条件  **if** (**null** != queryString && !"".equals(queryString)) {  query.setQuery(queryString);  } **else** {  query.setQuery("\*:\*");  }  //商品分类名称过滤  **if** (**null** != caltalog\_name && !"".equals(caltalog\_name)) {  query.addFilterQuery("product\_catalog\_name:" + caltalog\_name);  }  //价格区间过滤  **if** (**null** != price && !"".equals(price)) {  String[] strings = price.split("-");  query.addFilterQuery("product\_price:["+strings[0]+" TO "+strings[1]+"]");  }  //排序条件  **if** ("1".equals(sort)) {  query.setSort("product\_price", ORDER.*desc*);  } **else** {  query.setSort("product\_price", ORDER.*asc*);  }  //分页处理  **if** (**null** == page) {  page = 1;  }  //start  **int** start = (page-1) \* Commons.*PAGE\_SIZE*;  query.setStart(start);  query.setRows(Commons.*PAGE\_SIZE*);  //设置默认搜索域  query.set("df", "product\_keywords");  //高亮设置  query.setHighlight(**true**);  query.addHighlightField("product\_name");  query.setHighlightSimplePre("<span style=\"color:red\">");  query.setHighlightSimplePost("</span>");    //查询商品列表  ResultModel resultModel = productDao.queryProduct(query);  //计算总页数  **long** recordCount = resultModel.getRecordCount();  **int** pages = (**int**) (recordCount/Commons.*PAGE\_SIZE*);  **if** (recordCount % Commons.*PAGE\_SIZE* > 0) {  pages ++;  }  resultModel.setPageCount(pages);  resultModel.setCurPage(page);    **return** resultModel;  }  } |

## controller

功能：接收页面传递过来的参数调用service查询商品列表。将查询结果返回给jsp页面，还需要查询参数的回显。

参数：

1、查询条件：字符串

2、商品分类的过滤条件：商品的分类名称，字符串

3、商品价格区间：传递一个字符串，满足格式：“0-100、101-200、201-\*”

4、排序条件：页面传递过来一个升序或者降序就可以，默认是价格排序。0：升序1：降序

5、分页信息：每页显示的记录条数创建一个常量60条。传递一个当前页码就可以了。

6、Model：相当于request。

返回结果：String类型，就是一个jsp的名称。

String queryProduct(String queryString, String caltalog\_name, String price, String sort, Integer page, Model model) throws Exception;

|  |
| --- |
| @Controller  **public** **class** ProductController {    @Autowired  **private** ProductService productService;    @RequestMapping("/list")  **public** String queryProduct(String queryString, String catalog\_name, String price,  String sort, Integer page, Model model) **throws** Exception {  //查询商品列表  ResultModel resultModel = productService.queryProduct(queryString, catalog\_name, price, sort, page);  //列表传递给jsp  model.addAttribute("result", resultModel);  //参数回显  model.addAttribute("queryString", queryString);  model.addAttribute("caltalog\_name", catalog\_name);  model.addAttribute("price", price);  model.addAttribute("sort", sort);  model.addAttribute("page", page);    **return** "product\_list";  }  } |

## Jsp

参考资料。