**全文检索技术**

**Lucene&Solr**

**Part3**

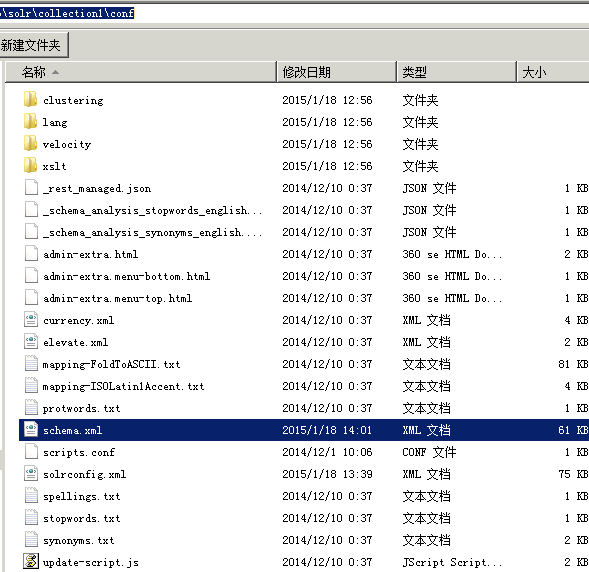
# 课程计划

1. Solr配置中文分析器
   1. Schema.xml的配置
   2. 配置IKAnalyzer
2. DataimportHandler插件
3. Solrj的复杂查询
   1. 后台复杂查询
   2. solrJ实现复杂查询
4. 京东案例

# 配置中文分析器

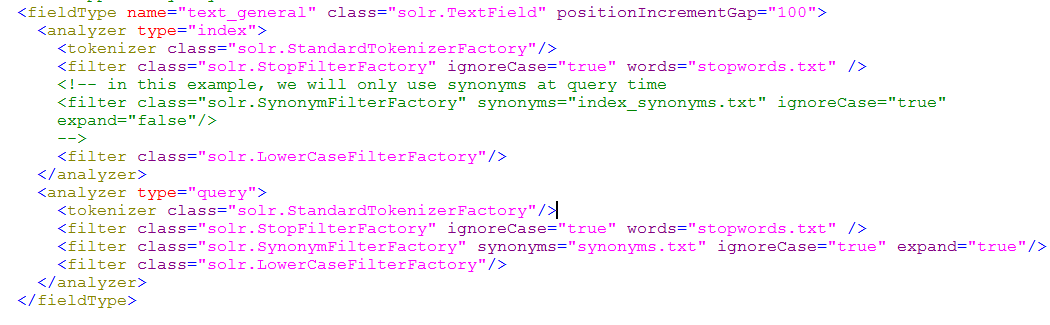
## Schema.xml

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。



### 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

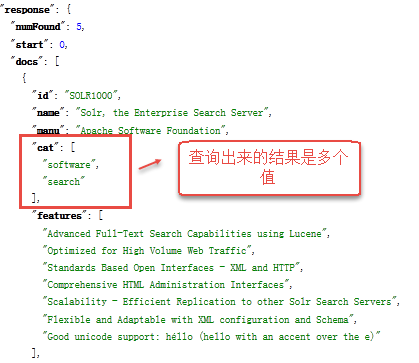
在fields结点内定义具体的Field，filed定义包括name,type（为之前定义过的各种FieldType）,indexed（是否被索引）,stored（是否被储存），multiValued（是否存储多个值）等属性。

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：



### uniqueKey

Solr中默认定义唯一主键key为id域，如下：



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，

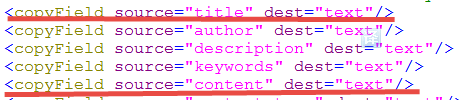
定义title、content、text的域：







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：



### dynamicField（动态字段）

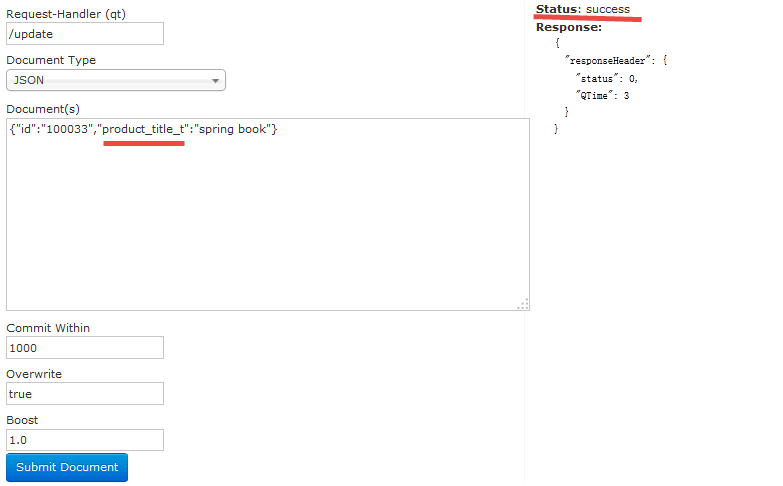
动态字段就是不用指定具体的名称，只要定义字段名称的规则，例如定义一个 dynamicField，name 为\*\_i，定义它的type为text，那么在使用这个字段的时候，任何以\_i结尾的字段都被认为是符合这个定义的，例如：name\_i，gender\_i，school\_i等。

自定义Field名为：product\_title\_t，“product\_title\_t”和scheam.xml中的dynamicField规则匹配成功，如下：

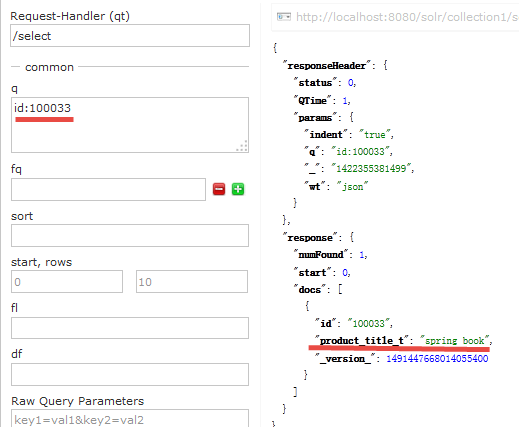


“product\_title\_t”是以“\_t”结尾。

创建索引：



搜索索引：



## 安装中文分词器

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classpath下。

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

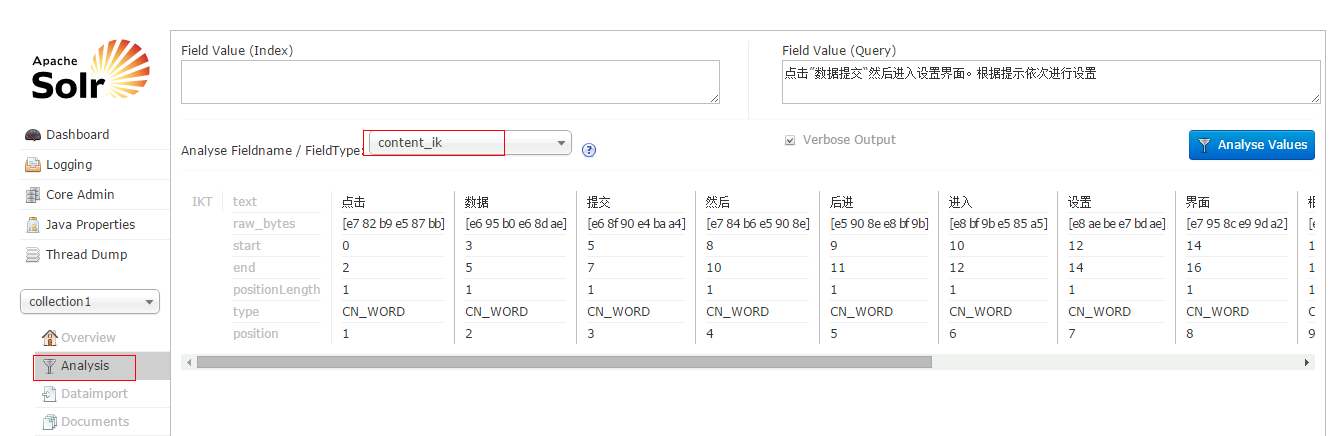
|  |
| --- |
| <!-- IKAnalyzer-->  <fieldType name="text\_ik" class="solr.TextField">  <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>  </fieldType> |

第四步：定义field，指定field的type属性为text\_ik

|  |
| --- |
| <!--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"/> |

第四步：重启tomcat

测试：



## 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"/>

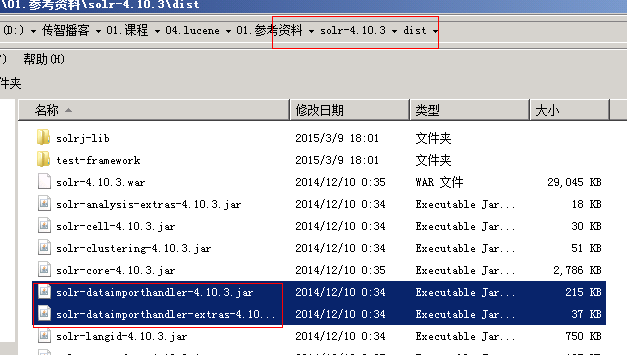
<copyField source="product\_name" dest="product\_keywords"/>

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

# 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。

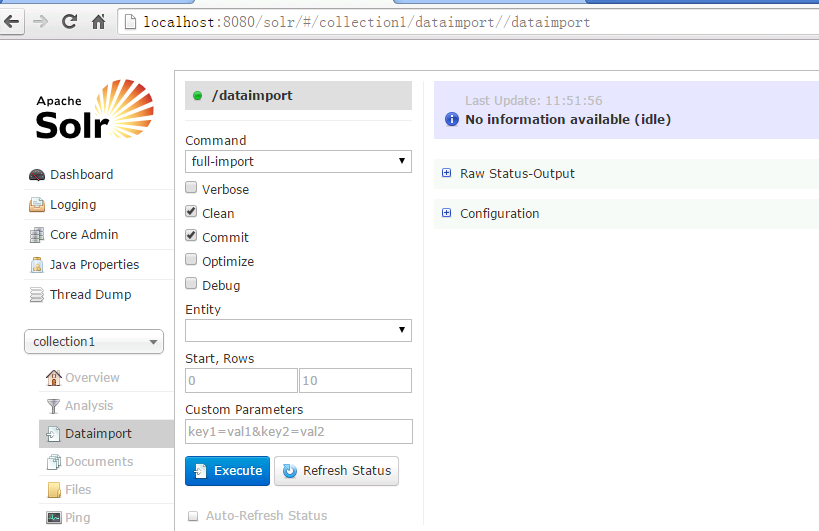
第二步：配置solrconfig.mxl文件，添加一个requestHandler。

|  |
| --- |
| <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

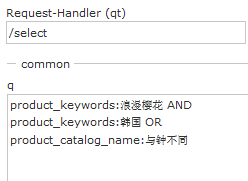
**导入数据前会先清空索引库，然后再导入。**

# Solr复杂查询

## 使用后台查询

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\*:\*。

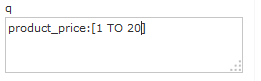


1. fq - （filter query）过虑查询，作用：在q查询符合结果中同时是fq查询符合的，例如：：



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：



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]

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

按价格降序

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



显示前10条。

1. fl - 指定返回那些字段内容，用逗号或空格分隔多个。

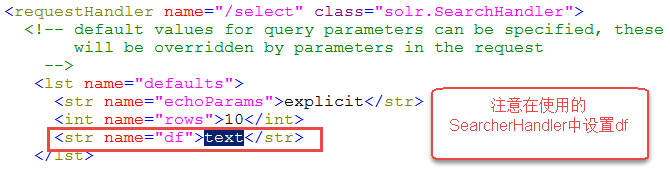


显示商品图片、商品名称、商品价格

1. df-指定一个搜索Field



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。



1. wt - (writer type)指定输出格式，可以有 xml, json, php, phps等
2. hl 是否高亮 ,设置高亮Field，设置格式前缀和后缀。



## 使用solrj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");  //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"));    }  } |

# 案例实现

## 原型分析



## 系统架构

spring容器

表现层

springmvc

Service层

Dao层

SolrJ

Solr索引库

mysql数据库

商品信息

Tomcat

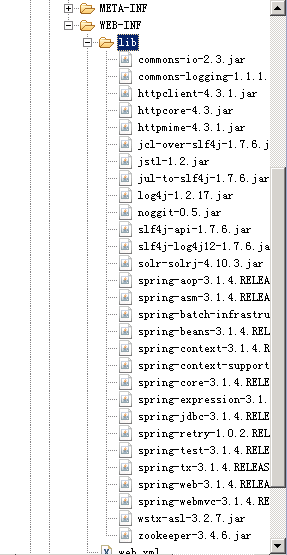
Solr服务

索引、搜索请求

## 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



### Web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"WebApp\_ID"* version=*"2.5"*>  <!-- POST提交过滤器 UTF-8 -->  <filter>  <filter-name>encoding</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encoding</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- 前端控制器 -->  <servlet>  <servlet-name>springmvc</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>\*.action</url-pattern>  </servlet-mapping>  </web-app> |

### Springmvc.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.2.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.2.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.2.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-4.2.xsd"*>  <!-- 配置扫描 器 -->  <context:component-scan base-package=*"com.itheima.controller"*/>  <!-- 配置处理器映射器 适配器 -->  <mvc:annotation-driven/>    <!-- 配置视图解释器 jsp -->  <bean id=*"jspViewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"*/>  <property name=*"suffix"* value=*".jsp"*/>  </bean>    </beans> |

## 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<Product>，还需要返回查询结果的总数量。

返回：PageBean

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

商品对象模型：

**public** **class** Product {

// 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** PageBean {

// 商品列表

**private** List<Product> 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** PageBean queryProduct(SolrQuery query) **throws** Exception {    PageBean pageBean = **new** PageBean();  //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  pageBean.setRecordCount(solrDocumentList.getNumFound());  List<Product> productList = **new** ArrayList<>();  //遍历查询结果  **for** (SolrDocument solrDocument : solrDocumentList) {  //取商品信息  Product product = **new** Product();  product.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");  }  product.setName(productName);  product.setPrice((**float**) solrDocument.get("product\_price"));  product.setCatalog\_name((String) solrDocument.get("product\_catalog\_name"));  product.setPicture((String) solrDocument.get("product\_picture"));  //添加到商品列表  productList.add(product);  }  //商品列表添加到resultmodel中  pageBean.setProductList(productList);  **return** pageBean;  }  } |

## 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：PageBean

方法定义：PageBean 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** PageBean 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  PageBean pageBean = productDao.queryProduct(query);  //计算总页数  **long** recordCount = pageBean.getRecordCount();  **int** pages = (**int**) (recordCount/Commons.*PAGE\_SIZE*);  **if** (recordCount % Commons.*PAGE\_SIZE* > 0) {  pages ++;  }  pageBean.setPageCount(pages);  pageBean.setCurPage(page);    **return** pageBean;  }  } |

## 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** ProductAction {    @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  PageBean pageBean = productService.queryProduct(queryString, catalog\_name, price, sort, page);  //列表传递给jsp  model.addAttribute("result", pageBean);  //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。