第09天

品优购电商系统开发

# 上节内容回顾

1. 内容系统和门户系统
2. SpringData + redis
3. 缓存业务

# 今日学习内容(学习内容按视频的顺序进行归纳总结)

1. solr环境安装、中文分析器和业务域的配置
2. Spring Data Solr完成增删改查操作
3. 批量数据导入功能
4. 按关键字搜索功能

# 课程介绍及需求(视频01) [了解]

## 今日目标

主要完成查询功能，查询功能用的是Solr搜索引擎。所以今天的重点是Solr搜索引擎的使用。

# solr的安装及域(视频02-08) [了解]

## 什么是solr

大多数搜索引擎应用都必须具有某种搜索功能，问题是搜索功能往往是巨大的资源消耗并且它们由于沉重的数据库加载而拖垮你的应用的性能。

这就是为什么转移负载到一个外部的搜索服务器是一个不错的主意，Apache Solr是一个流行的开源搜索服务器，它通过使用类似REST的HTTP API，这就确保你能从几乎任何编程语言来使用solr。

Solr是一个开源搜索平台，用于构建搜索应用程序。 它建立在[Lucene](http://www.yiibai.com/lucene/" \o "Lucene)(全文搜索引擎)之上。 Solr是企业级的，快速的和高度可扩展的。 使用Solr构建的应用程序非常复杂，可提供高性能。

为了在CNET网络的公司网站上添加搜索功能，Yonik Seely于2004年创建了Solr。并在2006年1月，它成为Apache软件基金会下的一个开源项目。并于2016年发布最新版本Solr 6.0，支持并行SQL查询的执行。

Solr可以和[Hadoop](http://www.yiibai.com/hadoop/" \o "Hadoop)一起使用。由于Hadoop处理大量数据，Solr帮助我们从这么大的源中找到所需的信息。不仅限于搜索，Solr也可以用于存储目的。像其他NoSQL数据库一样，它是一种非关系数据存储和处理技术。

总之，Solr是一个可扩展的，可部署，搜索/存储引擎，优化搜索大量以文本为中心的数据。

## solr安装

1：安装 Tomcat，解压缩即可。

2：解压 solr。

3：把 solr 下的dist目录solr-4.10.3.war部署到 Tomcat\webapps下(去掉版本号)。

4：启动 Tomcat解压缩 war 包

5：把solr下example/lib/ext 目录下的所有的 jar 包，添加到 solr 的工程中(\WEB-INF\lib目录下)。

6：创建一个 solrhome 。solr 下的/example/solr 目录就是一个 solrhome。复制此目录到D盘改名为solrhome

7：关联 solr 及 solrhome。需要修改 solr 工程的 web.xml 文件。

|  |
| --- |
| <env-entry>  <env-entry-name>solr/home</env-entry-name>  <env-entry-value>d:\solrhome</env-entry-value>  <env-entry-type>java.lang.String</env-entry-type>  </env-entry> |

8：启动 Tomcat

http://IP:8080/solr/

## solr中文分析器

IK Analyzer简介

IK Analyzer 是一个开源的，基亍 java 语言开发的轻量级的中文分词工具包。从 2006年 12 月推出 1.0 版开始， IKAnalyzer 已经推出了 4 个大版本。最初，它是以开源项目Luence 为应用主体的，结合词典分词和文法分析算法的中文分词组件。从 3.0 版本开始，IK 发展为面向 Java 的公用分词组件，独立亍 Lucene 项目，同时提供了对 Lucene 的默认优化实现。在 2012 版本中，IK 实现了简单的分词歧义排除算法，标志着 IK 分词器从单纯的词典分词向模拟语义分词衍化。

IK Analyzer配置

步骤：

1、把IKAnalyzer2012FF\_u1.jar 添加到 solr 工程的 lib 目录下

2、创建WEB-INF/classes文件夹 把扩展词典、停用词词典、配置文件放到 solr 工程的 WEB-INF/classes 目录下。

3、修改 Solrhome 的 schema.xml 文件，配置一个 FieldType，使用 IKAnalyzer

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

## solr域-1

什么是域？就是我们要用什么属性值进行搜索？用什么属性进行搜索这个属性就是域。

## solr域-2

修改solrhome的schema.xml 文件 设置业务系统 Field

|  |
| --- |
| <field name="item\_goodsid" type="long" indexed="true" stored="true"/>  <field name="item\_title" type="text\_ik" indexed="true" stored="true"/>  <field name="item\_price" type="double" indexed="true" stored="true"/>  <field name="item\_image" type="string" indexed="false" stored="true" />  <field name="item\_category" type="string" indexed="true" stored="true" />  <field name="item\_seller" type="text\_ik" indexed="true" stored="true" />  <field name="item\_brand" type="string" indexed="true" stored="true" /> |

* name：指定域的名称
* type：指定域的类型
* indexed：是否索引
* stored：是否存储
* required：是否必须
* multiValued：是否多值，比如商品信息中，一个商品有多张图片，一个Field像存储多个值的话，必须将multiValued设置为true。

## solr域-3

复制域

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

<copyField source="item\_title" dest="item\_keywords"/>

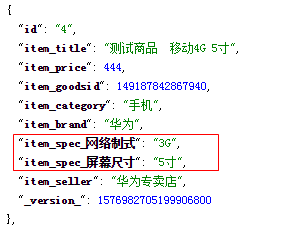
<copyField source="item\_category" dest="item\_keywords"/>

<copyField source="item\_seller" dest="item\_keywords"/>

<copyField source="item\_brand" dest="item\_keywords"/>

## solr域-4

当我们需要动态扩充字段时，我们需要使用动态域。对于品优购，规格的值是不确定的，所以我们需要使用动态域来实现。需要实现的效果如下：



配置：

|  |
| --- |
| <dynamicField name="item\_spec\_\*" type="string" indexed="true" stored="true" /> |

# Spring Data Solr(视频09-17) [重点]

## SpringDataSolr简介

虽然支持任何编程语言的能力具有很大的市场价值，你可能感兴趣的问题是：我如何将Solr的应用集成到Spring中？可以，Spring Data Solr就是为了方便Solr的开发所研制的一个框架，其底层是对SolrJ（官方API）的封装。

## springdataSolr入门-准备

1. 创建maven工程，pom.xml中引入依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-solr</artifactId>  <version>1.5.5.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.2.4.RELEASE</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>  </dependencies> |

（2）在src/main/resources下创建 applicationContext-solr.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:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:solr=*"http://www.springframework.org/schema/data/solr"*  xsi:schemaLocation=*"http://www.springframework.org/schema/data/solr*  *http://www.springframework.org/schema/data/solr/spring-solr-1.0.xsd*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>  <!-- solr服务器地址 -->  <solr:solr-server id=*"solrServer"* url=*"http://127.0.0.1:8080/solr"* />  <!-- solr模板，使用solr模板可对索引库进行CRUD的操作 -->  <bean id=*"solrTemplate"* class=*"org.springframework.data.solr.core.SolrTemplate"*>  <constructor-arg ref=*"solrServer"* />  </bean>  </beans> |

## springdataSolr入门-注解

创建 cn.itcast.pojo 包，将品优购的TbItem实体类拷入本工程 ,属性使用@Field注解标识 。

如果属性与配置文件定义的域名称不一致，需要在注解中指定域名称。

|  |
| --- |
| **public** **class** TbItem **implements** Serializable{  @Field  **private** Long id;  @Field("item\_title")  **private** String title;  @Field("item\_price")  **private** BigDecimal price;  @Field("item\_image")  **private** String image;  @Field("item\_goodsid")  **private** Long goodsId;  @Field("item\_category")  **private** String category;  @Field("item\_brand")  **private** String brand;  @Field("item\_seller")  **private** String seller;  .......  } |

## springdataSolr入门-增加修改

创建测试类TestTemplate.java

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations="classpath:applicationContext-solr.xml")  **public** **class** TestTemplate {  @Autowired  **private** SolrTemplate solrTemplate;  @Test  **public** **void** testAdd(){  TbItem item=**new** TbItem();  item.setId(1L);  item.setBrand("华为");  item.setCategory("手机");  item.setGoodsId(1L);  item.setSeller("华为2号专卖店");  item.setTitle("华为Mate9");  item.setPrice(**new** BigDecimal(2000));  solrTemplate.saveBean(item);  solrTemplate.commit();  }  } |

## springdataSolr入门-根据主键查询删除

按主键查询

|  |
| --- |
| @Test  **public** **void** testFindOne(){  TbItem item = solrTemplate.getById(1, TbItem.**class**);  System.***out***.println(item.getTitle());  } |

按主键删除

|  |
| --- |
| @Test  **public** **void** testDelete(){  solrTemplate.deleteById("1");  solrTemplate.commit();  } |

## springdataSolr入门-批量插入数据

首先循环插入100条测试数据

|  |
| --- |
| @Test  **public** **void** testAddList(){  List<TbItem> list=**new** ArrayList();  **for**(**int** i=0;i<100;i++){  TbItem item=**new** TbItem();  item.setId(i+1L);  item.setBrand("华为");  item.setCategory("手机");  item.setGoodsId(1L);  item.setSeller("华为2号专卖店");  item.setTitle("华为Mate"+i);  item.setPrice(**new** BigDecimal(2000+i));  list.add(item);  }  solrTemplate.saveBeans(list);  solrTemplate.commit();  } |

批量添加用的是saveBeans,比上面的多了一个S

## springdataSolr入门-分页查询

要查询的结果，如果过多的话，我们不会把它全部都查询出来

@Test

**public** **void** testPageQuery(){

Query query=**new** SimpleQuery("\*:\*");

query.setOffset(20);//开始索引（默认0）

query.setRows(20);//每页记录数(默认10)

ScoredPage<TbItem> page = solrTemplate.queryForPage(query, TbItem.**class**);

System.***out***.println("总记录数："+page.getTotalElements());

List<TbItem> list = page.getContent();

showList(list);

}

//显示记录数据

**private** **void** showList(List<TbItem> list){

**for**(TbItem item:list){

System.***out***.println(item.getTitle() +item.getPrice());

}

}

## springdataSolr入门-条件查询

Criteria 用于对条件的封装：

|  |
| --- |
| @Test  **public** **void** testPageQueryMutil(){  Query query=**new** SimpleQuery("\*:\*");  Criteria criteria=**new** Criteria("item\_title").contains("2");  criteria=criteria.and("item\_title").contains("5");  query.addCriteria(criteria);  //query.setOffset(20);//开始索引（默认0）  //query.setRows(20);//每页记录数(默认10)  ScoredPage<TbItem> page = solrTemplate.queryForPage(query, TbItem.**class**);  System.***out***.println("总记录数："+page.getTotalElements());  List<TbItem> list = page.getContent();  showList(list);  } |

## springdataSolr入门-全部删除

没有全部删除的方法，但我们可以用查询删除

|  |
| --- |
| @Test  **public** **void** testDeleteAll() {  solrTemplate.delete(**new** SimpleQuery("\*:\*"));  solrTemplate.commit();  } |

# 批量导入(视频18-20) [撑握]

**需求分析：**编写专门的导入程序，将商品数据导入到Solr系统中

## 批量数据导入-商品列表数据

批量导入数据这件事情是必须要做的，但是只做“一次”，那只做一次是不是可以把代码用完成就删除了呢？不行，测试的时候，上线的时候等还要用。

批量导入分两部,一部分是查询出来，另一部分是导入进去。先进行第一部分查询出来数据：

## 查询商品数据列表

**工程搭建**

（1）创建pinyougou-solr-util（jar）,引入pinyougou-dao 以及spring 相关依赖

（2）创建spring 配置文件



内容为：

|  |
| --- |
| <context:component-scan base-package=*"com.pinyougou.solrUtil"*>  </context:component-scan> |

**代码编写**

创建com.pinyougou.solrutil包，创建类SolrUtil ,实现商品数据的查询(已审核商品)

|  |
| --- |
| @Component  **public** **class** SolrUtil {  @Autowired  **private** TbItemMapper itemMapper;  /\*\*  \* 导入商品数据  \*/  **public** **void** importItemData(){  TbItemExample example=**new** TbItemExample();  Criteria criteria = example.createCriteria();  criteria.andStatusEqualTo("1");//获取启用状态的商品项  List<TbItem> itemList = itemMapper.selectByExample(example);  System.***out***.println("===商品列表===");  **for**(TbItem item:itemList){  System.***out***.println(item.getTitle());  }  System.***out***.println("===结束===");  }  **public** **static** **void** main(String[] args) {  ApplicationContext context=**new** ClassPathXmlApplicationContext("classpath\*:spring/applicationContext\*.xml");  SolrUtil solrUtil= (SolrUtil) context.getBean("solrUtil");  solrUtil.importItemData();  }  } |

## 批量数据导入-批量导入

**实体类**

(1)将demo工程中添加了@Field注解的实体类拷贝到pinyougou-pojo中

(2)在pinyougou-pojo中引入依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-solr</artifactId>  <version>1.5.5.RELEASE</version>  </dependency> |

**添加Solr配置文件**

添加applicationContext-solr.xml到spring目录

|  |
| --- |
| <!-- solr服务器地址 -->  <solr:solr-server id=*"solrServer"* url=*"http://127.0.0.1:8080/solr"* />  <!-- solr模板，使用solr模板可对索引库进行CRUD的操作 -->  <bean id=*"solrTemplate"* class=*"org.springframework.data.solr.core.SolrTemplate"*>  <constructor-arg ref=*"solrServer"* />  </bean> |

**调用模板类导入solr**

修改pinyougou-solr-util的SolrUtil.java

|  |
| --- |
| @Autowired  **private** SolrTemplate solrTemplate;  /\*\*  \* 导入商品数据  \*/  **public** **void** importItemData(){  TbItemExample example=**new** TbItemExample();  Criteria criteria = example.createCriteria();  criteria.andStatusEqualTo("2");//已审核  List<TbItem> itemList = itemMapper.selectByExample(example);  System.***out***.println("===商品列表===");  **for**(TbItem item:itemList){  System.***out***.println(item.getTitle());  }  solrTemplate.saveBeans(itemList);  solrTemplate.commit();  System.***out***.println("===结束===");  } |

## 批量数据导入-规格数据

**Dynamic注解**

修改TbItem.java ，添加属性

|  |
| --- |
| @Dynamic  @Field("item\_spec\_\*")  **private** Map<String,String> specMap;  **public** Map<String, String> getSpecMap() {  **return** specMap;  }  **public** **void** setSpecMap(Map<String, String> specMap) {  **this**.specMap = specMap;  } |

**修改导入工具**

修改pinyougou-solr-util的SolrUtil.java

|  |
| --- |
| /\*\*  \* 导入商品数据  \*/  **public** **void** importItemData(){  TbItemExample example=**new** TbItemExample();  Criteria criteria = example.createCriteria();  criteria.andStatusEqualTo("1");//获取启用状态的商品项  List<TbItem> itemList = itemMapper.selectByExample(example);  System.***out***.println("===商品列表===");  **for**(TbItem item:itemList){  Map specMap= JSON.*parseObject*(item.getSpec());  //将spec字段中的json字符串转换为map  item.setSpecMap(specMap);//给带注解的字段赋值  System.***out***.println(item.getTitle());  }  solrTemplate.saveBeans(itemList);  solrTemplate.commit();  System.***out***.println("===结束===");  } |

# 基本搜索-关键字(视频21-28) [撑握]

**需求分析**

打开搜索页面，在搜索框输入要搜索的关键字，点击搜索按钮即可进行搜索，展示搜索结果



## 关键字搜索-后端-准备工作

（1）创建war工程pinyougou-search-service ，引入pinyougou-search-interface spring dubbox 等相关依赖（参加其它服务工程） Tomcat运行端口9004

（2）添加web.xml （参加其它服务工程）

（3）在src/main/resources/spring 下applicationContext-service.xml （参见其它服务工程）dubbo端口20884

（4）在src/main/resources/spring 下创建spring 配置文件applicationContext-solr.xml（同demo工程）

*<?***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:p="http://www.springframework.org/schema/p"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:solr="http://www.springframework.org/schema/data/solr"  
 xsi:schemaLocation="http://www.springframework.org/schema/data/solr   
 http://www.springframework.org/schema/data/solr/spring-solr-1.0.xsd  
 http://www.springframework.org/schema/beans   
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context   
 http://www.springframework.org/schema/context/spring-context.xsd"**>  
 *<!-- solr服务器地址 -->* <**solr:solr-server id="solrServer" url="http://127.0.0.1:8080/solr"** />  
 *<!-- solr模板，使用solr模板可对索引库进行CRUD的操作 -->* <**bean id="solrTemplate" class="org.springframework.data.solr.core.SolrTemplate"**>  
 <**constructor-arg ref="solrServer"** />  
 </**bean**>  
</**beans**>

## 关键字搜索-后端-服务接口

创建pinyougou-search-interface模块（搜索服务接口），依赖pinyougou-pojo

|  |
| --- |
| /\*\*  \* SKU搜索服务接口  \* **@author** Administrator  \*  \*/  **public** **interface** ItemSearchService {  /\*\*  \* 搜索  \* **@param** keywords  \* **@return**  \*/  **public** Map<String,Object> search(Map searchMap);  } |

查询的条件可能有很多，所以是多个，查询的结果集结构不同，所以是Map

## 关键字搜索-后端-服务实现层

编写服务实现类ItemSearchServiceImpl.java

|  |
| --- |
| @Service  **public** **class** ItemSearchServiceImpl **implements** ItemSearchService{  @Autowired  **private** SolrTemplate solrTemplate;  @Override  **public** Map<String, Object> search(Map searchMap) {  Query query=**new** SimpleQuery();  //添加查询条件  Criteria criteria=  **new** Criteria("item\_keywords").is(searchMap.get("keywords"));  query.addCriteria(criteria);  ScoredPage<TbItem> page = solrTemplate.queryForPage(query, TbItem.**class**);  Map<String,Object> map=**new** HashMap<>();  map.put("rows", page.getContent());  **return** map;  }  } |

## 关键字搜索-后端-控制层

（1）创建pinyougou-search-web 工程 ，引入依赖（参见其它web模块），tomcat运行端口9104

（2）添加web.xml （参加其它web工程）

（3）添加配置文件 （内容参加其它web工程）

## 关键字搜索-后端-控制层-2

添加一个controller类

@RestController

@RequestMapping("/itemsearch")

**public** **class** ItemSearchController {

@Reference

**private** ItemSearchService itemSearchService;

@RequestMapping("/search")

**public** Map<String, Object> search(@RequestBody Map searchMap ){

**return** itemSearchService.search(searchMap);

}

}

## 关键字搜索-前端-服务层和控制层

**拷贝资源**

将下列资源拷贝至pinyougou-search-web



将angularJS拷贝到插件文件夹，拷贝base.js到js文件夹

**服务层**

pinyougou-search-web工程searchService.js

|  |
| --- |
| //搜索服务层  app.service("searchService",**function**($http){  **this**.search=**function**(searchMap){  **return** $http.post('itemsearch/search.do',searchMap);  }  }); |

**控制层**

pinyougou-search-web工程searchController.js

|  |
| --- |
| app.controller('searchController',**function**($scope,searchService){  //搜索  $scope.search=**function**(){  searchService.search( $scope.searchMap ).success(  **function**(response){  $scope.resultMap=response;//搜索返回的结果  }  );  }  }); |

## 关键字搜索-前端-页面

pinyougou-search-web工程search.html

引入js

|  |
| --- |
| <script type=*"text/javascript"* src=*"plugins/angularjs/angular.min.js"*> </script>  <script type=*"text/javascript"* src=*"js/base.js"*> </script>  <script type=*"text/javascript"* src=*"js/service/searchService.js"*> </script>  <script type=*"text/javascript"* src=*"js/controller/searchController.js"*> </script> |

指定控制器

|  |
| --- |
| <body ng-app=*"pinyougou"* ng-controller=*"searchController"*> |

绑定搜索框

|  |
| --- |
| <div class=*"input-append"*>  <input type=*"text"* id=*"autocomplete"* ng-model=*"searchMap.keywords"* type=*"text"* class=*"input-error input-xxlarge"* />  <button class=*"sui-btn btn-xlarge btn-danger"* ng-click=*"search()"* type=*"button"*>搜索</button>  </div> |

循环显示数据

|  |
| --- |
| <li class=*"yui3-u-1-5"* ng-repeat=*"item in resultMap.rows"*>  <div class=*"list-wrap"*>  <div class=*"p-img"*>  <a href=*"item.html"* target=*"\_blank"*><img src=*"{{item.image}}"* /></a>  </div>  <div class=*"price"*>  <strong>  <em>¥</em>  <i>{{item.price}}</i>  </strong>  </div>  <div class=*"attr"*>  <em>{{item.title}}</em>  </div>  <div class=*"cu"*>  <em><span>促</span>满一件可参加超值换购</em>  </div>  <div class=*"commit"*>  <i class=*"command"*>已有2000人评价</i>  </div>  <div class=*"operate"*>  <a href=*"success-cart.html"* target=*"\_blank"* class=*"sui-btn btn-bordered btn-danger"*>加入购物车</a>  <a href=*"javascript:void(0);"* class=*"sui-btn btn-bordered"*>对比</a>  <a href=*"javascript:void(0);"* class=*"sui-btn btn-bordered"*>关注</a>  </div>  </div>  </li> |

## 关键字搜索-测试

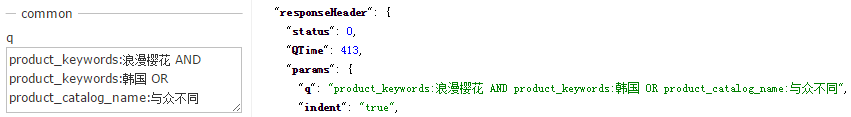
## 问题答疑

1. 我们引入DAO是为了使用mapper操作数据库，如果你想用JDBC也是可以的。
2. solr删除如果不提交，删除后的数据也是不能再用了，是避免脏数据的做法。

<http://localhost:8080/solr/#/collection1/query>使用

通过/select搜索索引，Solr制定一些参数完成不同需求的搜索：

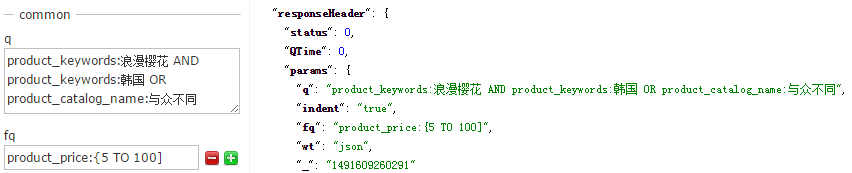
1. **q - 查询字符串，必须的，如果查询所有使用\*:\***



product\_keywords:浪漫樱花 AND product\_keywords:韩国

OR product\_catalog\_name:与众不同

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：**



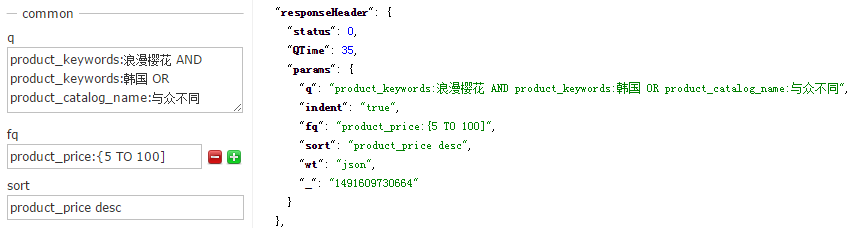
product\_price:{5 TO 100]

也可以使用“\*”表示无限，例如：

5以上：product\_price:[5 TO \*]

100以下：product\_price:[\* TO 100]

**3.sort - 排序，示例：**



product\_price desc

product\_price asc

**4.分页**

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。



**5 .fl - 指定返回那些字段内容，用逗号或空格分隔多个**

****

显示商品名称、商品价格

**6. df-指定一个默认搜索Field**

