谷粒商城

版本：V 1.0



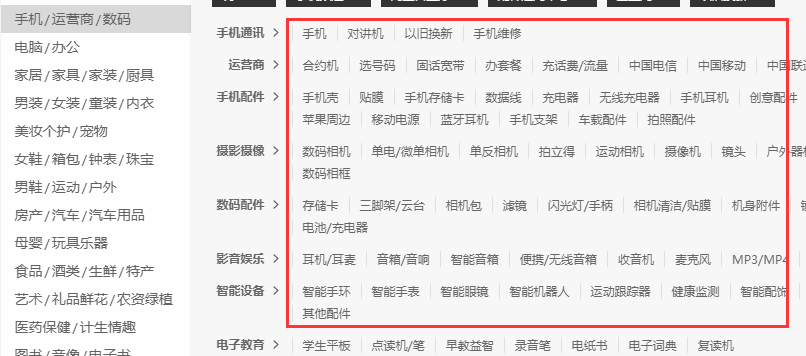
**当前的位置**

# 一、利用elasticSearch开发电商的搜索列表功能

## 1、功能简介

### 入口： 两个

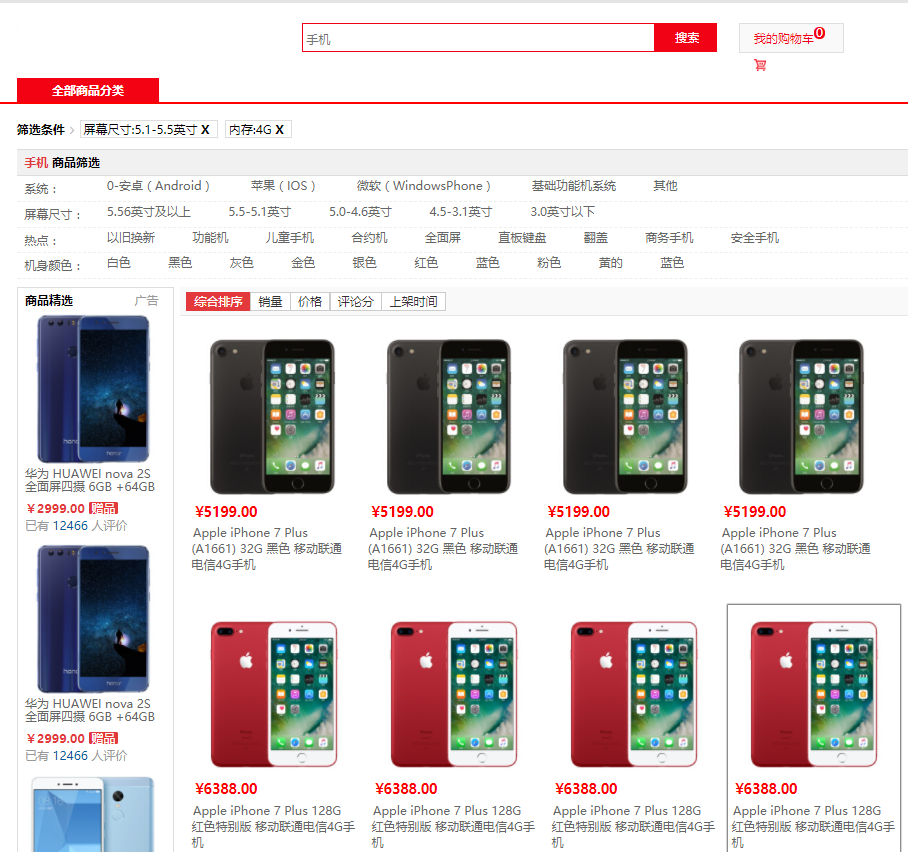
首页的分类



搜索栏



### 列表展示页面



## 2 根据业务搭建数据结构

建立mapping！

这时我们要思考三个问题：

1. 哪些字段需要分词
   1. 例如：商品名称[不desc是skuName]
2. 我们用哪些字段进行过滤
   1. 平台属性[三级分类Id]【真正的过滤应该是通过平台属性值进行过滤】
3. 哪些字段我们需要通过搜索查询出来。
   1. 商品名称,价格等。

|  |  |  |
| --- | --- | --- |
| 需要分词的字段 | **名称** | 分词 |
| 需要用于过滤的字段 | 三级分类、平台属性值 | 不分词 |
| 需要查询的字段 | Sku\_id,价格,**名称(关键词高亮)**，图片地址 | 显示的内容 |

以上分析的所有显示，以及分词，过滤的字段都应该在es中出现。Es中如何保存这些数据呢？

“根据上述的字段描述，应该建立一个mappings对应的存上上述字段描述的信息！”

根据以上制定出如下结构：mappings

Index：gmall\_sku\_info

type：\_doc

document: properties - rows

field: id,price,skuName…

Es中index默认是true。

|  |
| --- |
| PUT gmall\_sku\_info  {  "mappings": {  "\_doc":{  "properties": {  "id":{  "type": "keyword"  , "index": false  },  "price":{  "type": "double"  },  "skuName":{  "type": "text",  "analyzer": "ik\_max\_word"  },  "catalog3Id":{  "type": "keyword"  },  "skuDefaultImg":{  "type": "keyword",  "index": false  },  "skuAttrValueList":{  "properties": {  "valueId":{  "type":"keyword"  }  }  }  }  }  }  } |

注意：ik\_max\_word 中文词库必须有！

skuAttrValueList：平台属性值的集合，主要用于平台属性值过滤。

## 3 sku数据保存到ES

思路：

回顾一下，es数据保存的dsl javaBean == json格式的数据。

|  |
| --- |
| PUT /movie\_index/movie/1  { "id":1,  "name":"operation red sea",  "doubanScore":8.5,  "actorList":[  {"id":1,"name":"zhang yi"},  {"id":2,"name":"hai qing"},  {"id":3,"name":"zhang han yu"}  ]  } |

es存储数据是以json格式保存的，那么如果一个javabean的结构刚好跟要求的json格式吻合，我们就可以直接把javaBean序列化为json保持到es中，所以我们要制作一个与es中json格式一致的javabean.

### 3.1 JavaBean

把es中所有的字段封装到skuLsInfo中。

|  |
| --- |
| **public class** SkuLsInfo **implements** Serializable {   String **id**;   BigDecimal **price**;   String **skuName**;   String **catalog3Id**;   String **skuDefaultImg**;   Long **hotScore=0L**; // 綜合分數   List<SkuLsAttrValue> **skuAttrValueList**;  } |

SkuLsAttrValue

|  |
| --- |
| **public class** SkuLsAttrValue **implements** Serializable {   String **valueId**;  } |

添加get，set方法

### 3.2 保存sku数据的业务实现类

自行添加gmall-interface中增加接口方法

|  |
| --- |
| **package** com.atguigu.gmall.service;  **import** com.atguigu.gmall.bean.SkuLsInfo;  **public interface** ListService {    **public void** saveSkuInfo(SkuLsInfo skuLsInfo); } |

在gmall-list-service模块中增加业务实现类listServiceImpl

|  |
| --- |
| @Service **public class** ListServiceImpl **implements** ListService {   @Autowired  JestClient **jestClient**;   **public static final** String ***ES\_INDEX***=**"gmall0401\_sku\_info"**;   **public static final** String ***ES\_TYPE***=**"\_doc"**;   @Override  **public void** saveSkuInfo(SkuLsInfo skuLsInfo) {  *// 保存数据* Index index = **new** Index.Builder(skuLsInfo).index(***ES\_INDEX***).type(***ES\_TYPE***).id(skuLsInfo.getId()).build();  **try** {  DocumentResult documentResult = **jestClient**.execute(index);  } **catch** (IOException e) {  e.printStackTrace();  }  } } |

电商业务：分为前台，后台管理

前台：显示，以及购买流程

后台：商品的管理

管理：spu，sku，商品的上架，下架。

下架：实际就是从es中删除。

上架：将最新的产品新增到es上。

苹果3,4,4s,5,5s,5c,se,6,6s…

### 3.3 在后台管理的sku保存中，调用该方法

|  |
| --- |
| AttManageController 中  @RequestMapping(value = **"onSale"**,method = RequestMethod.***GET***) @ResponseBody **public void** onSale(String skuId){  SkuInfo skuInfo = **manageService**.getSkuInfo(skuId);  SkuLsInfo skuLsInfo = **new** SkuLsInfo();  *// 属性拷贝* **try** {  BeanUtils.*copyProperties*(skuLsInfo,skuInfo);  } **catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (InvocationTargetException e) {  e.printStackTrace();  }  **listService**.saveSkuInfo(skuLsInfo); } |
|  |
| 测试：  <http://localhost:8082/onSale?skuId=3> |
| 查询  GET gmall0401\_sku\_info/\_doc/3 |

## 4 查询数据的后台方法

### 4.1 分析

首先先观察功能页面，咱们一共要用什么查询条件，查询出什么数据？

查询条件：

1. 关键字
2. 可以通过**分类ID**进入列表页面
3. 平台属性值:过滤
4. 分页页码

查询结果：

1 sku的列表(关键字高亮显示)

1. 这些sku涉及了哪些属性和属性值
2. 命中个数[total]，用于分页

基于以上

### **4.2 编写DSL语句** ：

主要目的：是从es中取得数据！

|  |
| --- |
| GET gmall/SkuInfo/\_search  {  "query": {  "bool": {  "filter": [  {"term":{ "skuAttrValueList.valueId": "13"}},  {"term":{ "skuAttrValueList.valueId": "80"}},  {"term":{"catalog3Id":"61"}}  ],  "must":  { "match": { "skuName": "小米" } }  }  }  , "highlight": {  "fields": {"skuName":{}}  },  "from": 0,  "size": 2,  "sort":{"hotScore":{"order":"desc"}},  "aggs": {  "groupby\_attr": {  "terms": {  "field": "skuAttrValueList.valueId"  }  }  }  } |

**Es 匹配根据中文分词解析库进行匹配**

### 4.3 制作传入参数的类

在gmall-bean 项目中添加如下实体类 传入的参数是根据dsl语句得到

|  |
| --- |
| **public class** SkuLsParams **implements** Serializable {   String **keyword**;   String **catalog3Id**;   String[] **valueId**;   **int pageNo**=1;   **int pageSize**=20;  } |

### 4.4 返回结果的类

|  |
| --- |
| **public class** SkuLsResult **implements** Serializable {   List<SkuLsInfo> **skuLsInfoList**;   **long total**;   **long totalPages**;   List<String> **attrValueIdList**;  } |

### 4.5 基于这个DSL查询编写Java代码

|  |
| --- |
| **接口 ListService**  **public** SkuLsResult search(SkuLsParams skuLsParams); |
| **实现类**  **public** SkuLsResult search(SkuLsParams skuLsParams){   String query=makeQueryStringForSearch(skuLsParams);   Search search= **new** Search.Builder(query).addIndex(***ES\_INDEX***).addType(***ES\_TYPE***).build();  SearchResult searchResult=**null**;  **try** {  searchResult = **jestClient**.execute(search);  } **catch** (IOException e) {  e.printStackTrace();  }   SkuLsResult skuLsResult = makeResultForSearch(skuLsParams, searchResult);   **return** skuLsResult;  } |

#### 4.5.1 构造查询DSL

查询的过程很简单，但是要构造查询的query这个字符串有点麻烦，主要是这个Json串中的数据都是动态的。要拼接这个字符串，需要各种循环判断，处理标点符号等等。操作麻烦，可读性差。

但是JestClient这个客户端包，提供了一组builder工具。这个工具可以比较方便的帮程序员组合复杂的查询Json。

|  |
| --- |
| **public** String makeQueryStringForSearch(SkuLsParams skuLsParams){  *// 创建查询bulid* SearchSourceBuilder searchSourceBuilder = **new** SearchSourceBuilder();   BoolQueryBuilder boolQueryBuilder = QueryBuilders.*boolQuery*();   **if** (skuLsParams.getKeyword()!=**null**){  MatchQueryBuilder ma = **new** MatchQueryBuilder(**"skuName"**, skuLsParams.getKeyword());  boolQueryBuilder.must(ma);  *// 设置高亮* HighlightBuilder highlightBuilder = **new** HighlightBuilder();  *// 设置高亮字段* highlightBuilder.field(**"skuName"**);  highlightBuilder.preTags(**"<span style='color:red'>"**);  highlightBuilder.postTags(**"</span>"**);  *// 将高亮结果放入查询器中* searchSourceBuilder.highlight(highlightBuilder);   }  *// 设置三级分类* **if** (skuLsParams.getCatalog3Id()!=**null**){  TermQueryBuilder termQueryBuilder = **new** TermQueryBuilder(**"catalog3Id"**, skuLsParams.getCatalog3Id());  boolQueryBuilder.filter(termQueryBuilder);  }  *// 设置属性值* **if** (skuLsParams.getValueId()!=**null** && skuLsParams.getValueId().**length**>0){  **for** (**int** i=0;i<skuLsParams.getValueId().**length**;i++){  String valueId = skuLsParams.getValueId()[i];  TermQueryBuilder termsQueryBuilder = **new** TermQueryBuilder(**"skuAttrValueList.valueId"**, valueId);  boolQueryBuilder.filter(termsQueryBuilder);  }  }  searchSourceBuilder.query(boolQueryBuilder);  *// 设置分页* **int** form = (skuLsParams.getPageNo()-1)\*skuLsParams.getPageSize();  searchSourceBuilder.from(form);  searchSourceBuilder.size(skuLsParams.getPageSize());  *// 设置按照热度* searchSourceBuilder.sort(**"hotScore"**, SortOrder.***DESC***);   *// 设置聚合* TermsBuilder groupby\_attr = AggregationBuilders.*terms*(**"groupby\_attr"**).field(**"skuAttrValueList.valueId"**);  searchSourceBuilder.aggregation(groupby\_attr);   String query = searchSourceBuilder.toString();  System.***out***.println(**"query="**+query);  **return** query; } |

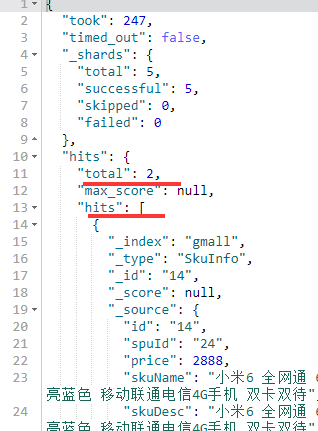
#### 4.5.2 处理返回值

思路：所有的返回值其实都在这个searchResult中

|  |
| --- |
| searchResult = **jestClient**.execute(search); |

它的结构其实可以在kibana 中观察一下：

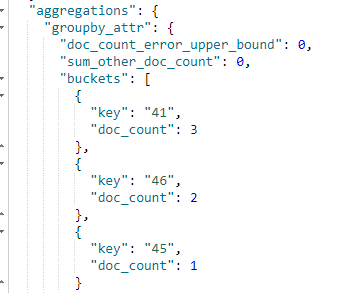
**命中的结果**



**高亮显示**



**分组统计结果：**

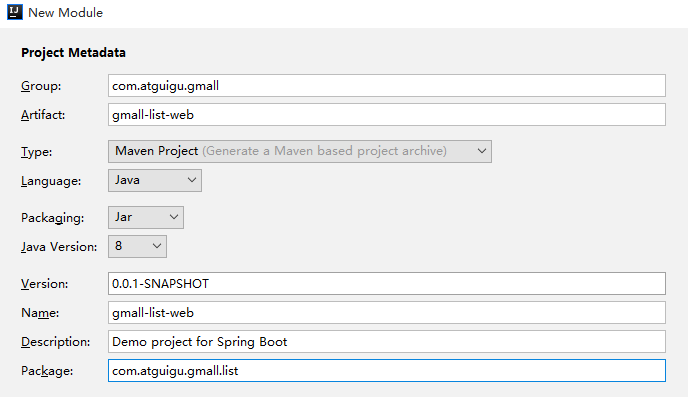


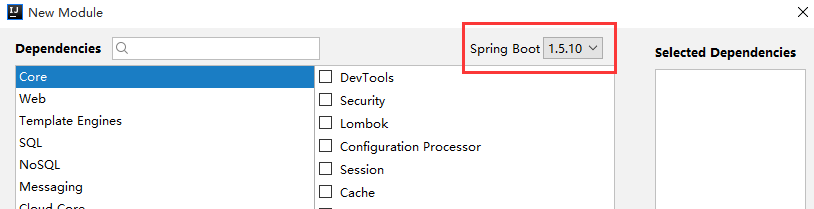
针对这三个部分来解析searchResult

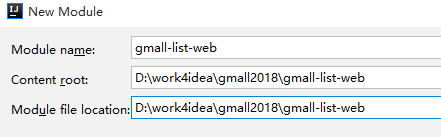
|  |
| --- |
| **private** SkuLsResult makeResultForSearch(SkuLsParams skuLsParams,SearchResult searchResult){  SkuLsResult skuLsResult=**new** SkuLsResult();  List<SkuLsInfo> skuLsInfoList=**new** ArrayList<>(skuLsParams.getPageSize());   *//获取sku列表* List<SearchResult.Hit<SkuLsInfo, Void>> hits = searchResult.getHits(SkuLsInfo.**class**);  **for** (SearchResult.Hit<SkuLsInfo, Void> hit : hits) {  SkuLsInfo skuLsInfo = hit.**source**;  **if**(hit.**highlight**!=**null**&&hit.**highlight**.size()>0){  List<String> list = hit.**highlight**.get(**"skuName"**);  *//把带有高亮标签的字符串替换skuName* String skuNameHl = list.get(0);  skuLsInfo.setSkuName(skuNameHl);  }  skuLsInfoList.add(skuLsInfo);  }  skuLsResult.setSkuLsInfoList(skuLsInfoList);  skuLsResult.setTotal(searchResult.getTotal());   *//取记录个数并计算出总页数* **long totalPage= (searchResult.getTotal() + skuLsParams.getPageSize() -1) / skuLsParams.getPageSize();**  skuLsResult.setTotalPages(totalPage);   *//取出涉及的属性值id* List<String> attrValueIdList=**new** ArrayList<>();  MetricAggregation aggregations = searchResult.getAggregations();  TermsAggregation groupby\_attr = aggregations.getTermsAggregation(**"groupby\_attr"**);  **if**(groupby\_attr!=**null**){  List<TermsAggregation.Entry> buckets = groupby\_attr.getBuckets();  **for** (TermsAggregation.Entry bucket : buckets) {  attrValueIdList.add( bucket.getKey()) ;  }  skuLsResult.setAttrValueIdList(attrValueIdList);  }  **return** skuLsResult; } |

测试后台程序…

#### 4.5.3 创建gmall-list-web模块

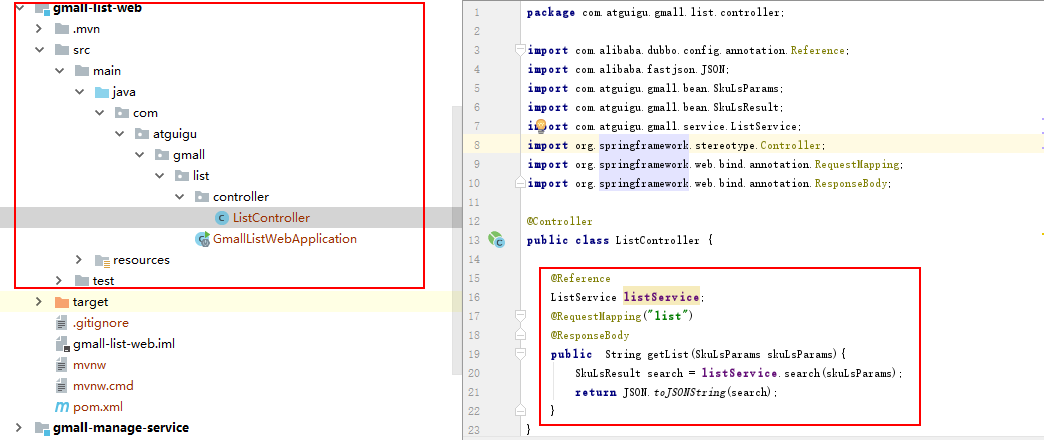






5.1.1 pom.xml

|  |
| --- |
| <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>gmall-parent</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**> </**parent**>   <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>gmall-interface</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**> </**dependency**>  <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>gmall-web-util</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**> </**dependency**> </**dependencies**> |



Controller

|  |
| --- |
| @Controller **public class** ListController {   @Reference  **private** ListService **listService**;  @RequestMapping(**"list.html"**)  @ResponseBody  **public** String getList(SkuLsParams skuLsParams){  SkuLsResult search = **listService**.search(skuLsParams);  **return** JSON.*toJSONString*(search);  } } |

# application.properties 配置文件

|  |
| --- |
| **server.port**=**8086 spring.thymeleaf.cache**=**false spring.thymeleaf.mode**=**LEGACYHTML5 spring.dubbo.application.name**=**list-web spring.dubbo.registry.protocol**=**zookeeper spring.dubbo.registry.address**=**192.168.67.202:2181 spring.dubbo.base-package**=**com.atguigu.gmall spring.dubbo.protocol.name**=**dubbo spring.dubbo.consumer.timeout**=**10000 spring.dubbo.consumer.check**=**false** |

# 域名更改：

1.修改nginx.conf 配置

# vim /usr/local/nginx/conf/nginx.conf

2 具体修改

|  |
| --- |
| upstream manage.gmall.com {  server 192.168.67.1:8082;  }  server {  listen 80;  server\_name manage.gmall.com;  location /{  proxy\_pass http://manage.gmall.com;  }  }  upstream list.gmall.com{  server 192.168.67.1:8086;  }  server {  listen 80;  server\_name list.gmall.com;  location /{  proxy\_pass http://list.gmall.com;  }  } |

在hosts 配置文件中添加

3 重启nginx

./nginx -s quit ：重启

./nginx

quit 退出重启

./nginx -s reload ：重启

启动状态下 重新读取nginx.conf reload

# hosts 配置文件



如果出现错误，先从query dsl 语句开始！

测试：直接http://www.gmall.com/ 输入检索条件即可！