**畅购电商系统开发**

**第6天**

传智播客.黑马程序员.深圳

# 学习目标

* 条件筛选
* 多条件搜索[品牌、规格条件搜索]
* 规格过滤
* 价格区间搜索
* 搜索分页
* 搜索排序
* 搜索高亮

# 品牌统计



用户搜索的时候，除了使用分类搜索外，还有可能使用品牌搜索，所以我们还需要显示品牌数据和规格数据，品牌数据和规格数据的显示比较容易，都可以考虑使用分类统计的方式进行分组实现。

## 品牌统计分析

看下面的SQL语句，我们在执行搜索的时候，第1条SQL语句是执行搜，第2条语句是根据品牌名字分组查看有多少品牌，大概执行了2个步骤就可以获取数据结果以及品牌统计，我们可以发现他们的搜索条件完全一样。

-- 查询所有

**SELECT** \* **FROM** tb\_sku **WHERE** name **LIKE** '%手机%';

-- 根据品牌名字分组查询

**SELECT** brand\_name **FROM** tb\_sku **WHERE** name **LIKE** '%手机%' **GROUP** **BY** brand\_name;

我们每次执行搜索的时候，需要显示商品品牌名称，这里要显示的品牌名称其实就是符合搜素条件的所有商品的品牌集合，我们可以按照上面的实现思路，使用ES根据分组名称做一次分组查询即可实现。

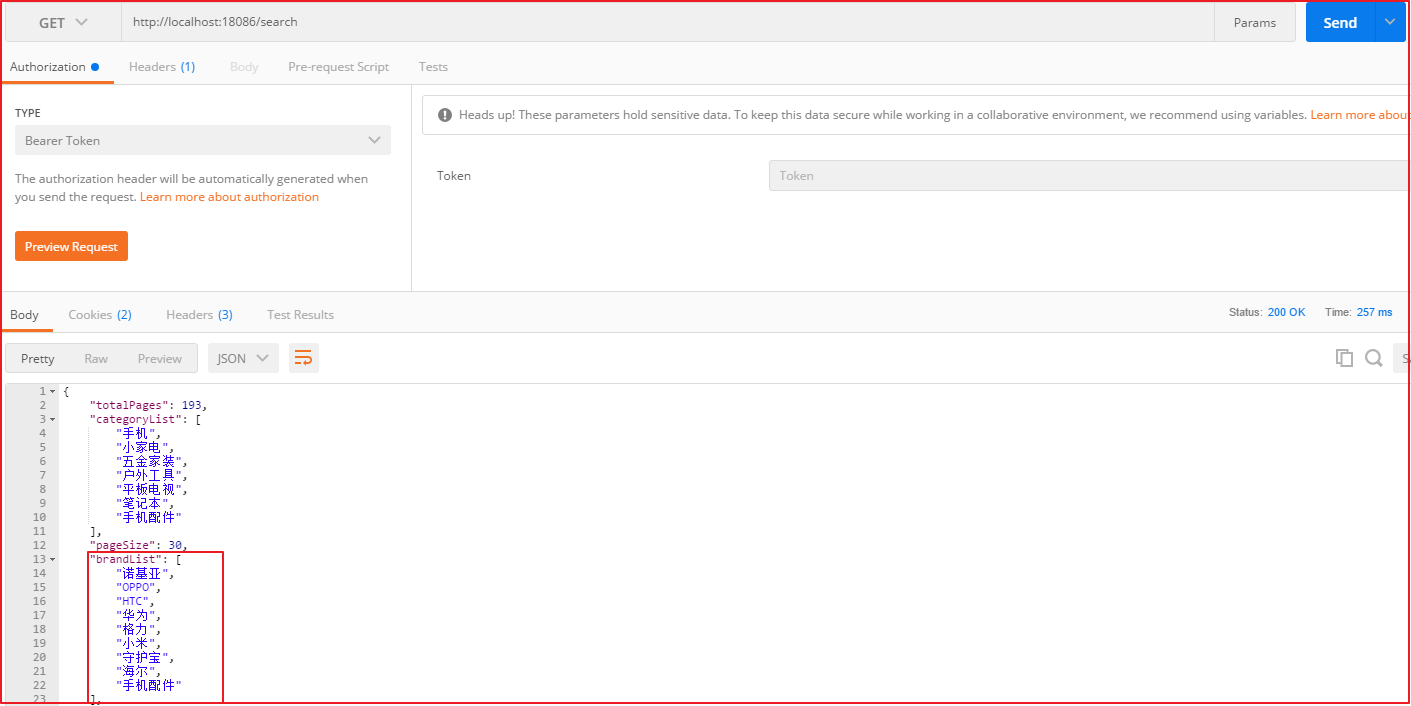
## 品牌分组统计实现

修改search微服务的com.changgou.search.service.impl.SkuServiceImpl类，添加一个品牌分组搜索，品牌查询逻辑与昨天课程的商品分类查询一致，可以使用昨天的代码修改一下：

@Autowired  
**private** ElasticsearchTemplate **esTemplate**;  
@Override  
**public** Map search(Map<String, String> searchMap) {  
 Map map = **new** HashMap();  
 *//1、构建基本查询条件* NativeSearchQueryBuilder builder = builderBasicQuery(searchMap);  
 *//2、根据查询条件-搜索商品列表* searchList(map, builder);  
 *//3、跟据查询条件-分组查询商品分类列表* searchCategoryList(map,builder);  
 *//4、跟据查询条件-分组查询品牌列表* searchBrandList(map,builder);  
 **return** map;  
}  
  
*/\*\*  
 \* 跟据查询条件-分组查询品牌列表  
 \** ***@param map*** *结果集包装  
 \** ***@param builder*** *查询条件构建器  
 \*/***private void** searchBrandList(Map map,NativeSearchQueryBuilder builder){  
 *//1.设置分组域名-termsAggregationBuilder = AggregationBuilders.terms(别名).field(域名);* TermsAggregationBuilder termsAggregationBuilder = AggregationBuilders.*terms*(**"group\_brand"**).field(**"brandName"**);  
 *//2.添加分组查询参数-builder.addAggregation(termsAggregationBuilder)* builder.addAggregation(termsAggregationBuilder);  
 *//3.执行搜索-esTemplate.queryForPage(builder.build(), SkuInfo.class)* AggregatedPage<SkuInfo> page = **esTemplate**.queryForPage(builder.build(), SkuInfo.**class**);  
 *//4.获取所有分组查询结果集-page.getAggregations()* Aggregations aggregations = page.getAggregations();  
 *//5.提取分组结果数据-stringTerms = aggregations.get(填入刚才查询时的别名)* StringTerms stringTerms = aggregations.get(**"group\_brand"**);  
 *//6.定义分类名字列表-categoryList = new ArrayList<String>()* List<String> brandList = **new** ArrayList<String>();  
 *//7.遍历读取分组查询结果-stringTerms.getBuckets().for* **for** (StringTerms.Bucket bucket : stringTerms.getBuckets()) {  
 *//7.1获取分类名字，并将分类名字存入到集合中-bucket.getKeyAsString()* brandList.add(bucket.getKeyAsString());  
 }  
 *//8.返回分类数据列表-map.put("categoryList", categoryList)* map.put(**"brandList"**, brandList);  
}

## 测试

使用PostMan请求<http://localhost:18086/search>



# 规格统计



用户搜索的时候，除了使用分类、品牌搜索外，还有可能使用规格搜索，所以我们还需要显示规格数据，规格数据的显示相比上面2种实现略微较难一些，需要对数据进行处理，我们也可以考虑使用分类统计和品牌统计的方式进行分组实现。

## 规格统计分析

看下面的SQL语句，我们在执行搜索的时候，第1条SQL语句是执行搜，第2条语句是根据规格分组查看有多少规格，大概执行了2个步骤就可以获取数据结果以及规格统计，我们可以发现他们的搜索条件完全一样。

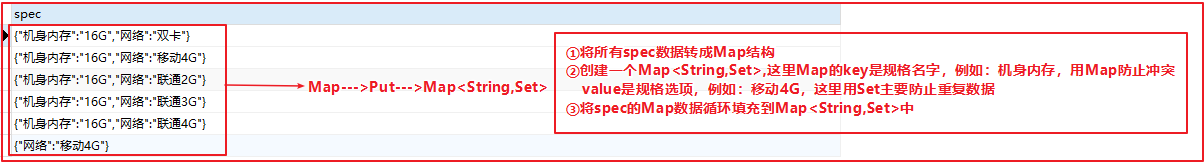
-- 查询所有

**SELECT** \* **FROM** tb\_sku **WHERE** name **LIKE** '%手机%';

-- 根据规格名字分组查询

**SELECT** spec **FROM** tb\_sku **WHERE** name **LIKE** '%手机%' **GROUP** **BY** spec;

上述SQL语句执行后的结果如下图：



获取到的规格数据我们发现有重复，不过也可以解决，解决思路如下：

1.获取分组查询所有规格数据

2.将所有规格数据转换成Map

3.定义一个Map<String,Set>,key是规格名字，防止重复所以用Map，valu是规格值，规格值有多个，所以用集合，为了防止规格重复，用Set去除重复

4.循环规格的Map，将数据填充到定义的Map<String,Set>中

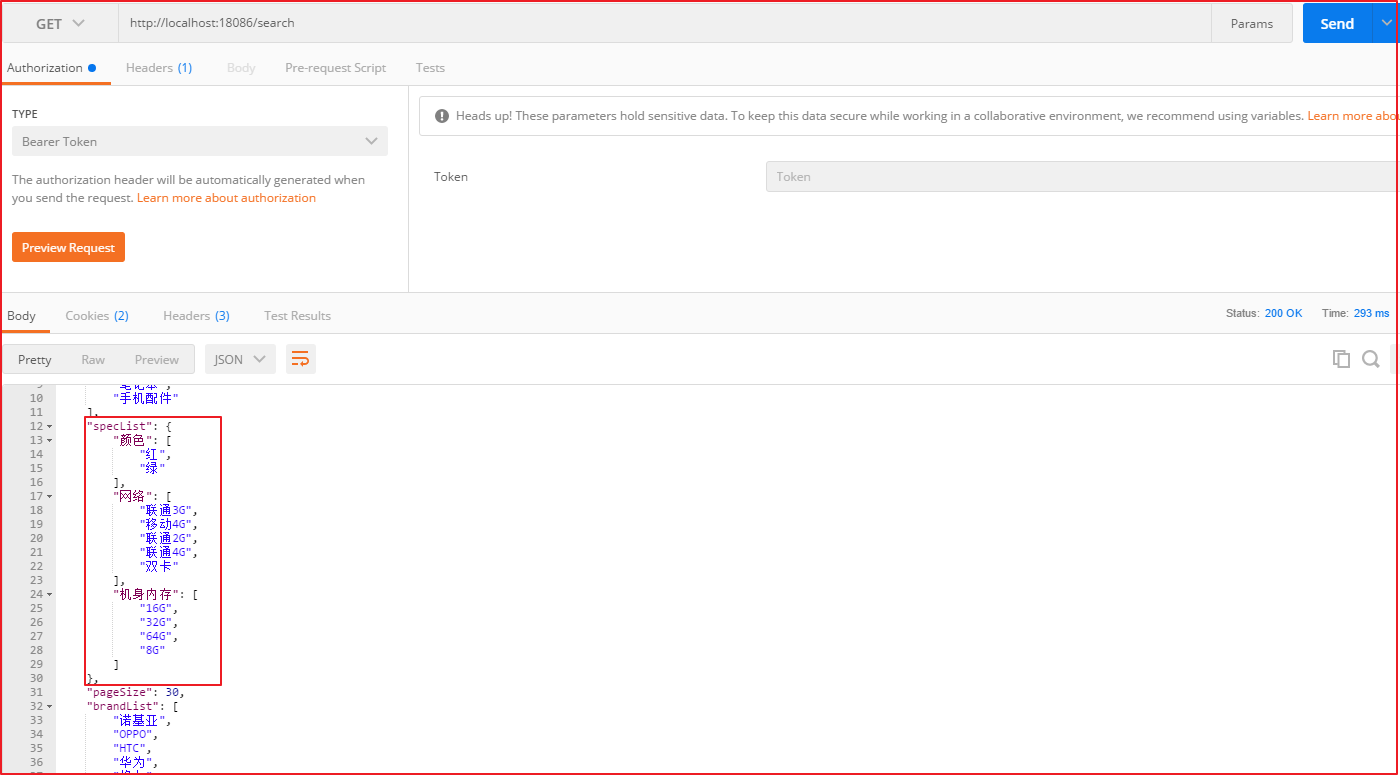
我们每次执行搜索的时候，需要显示商品规格数据，这里要显示的规格数据其实就是符合搜素条件的所有商品的规格集合，我们可以按照上面的实现思路，使用ES根据分组名称做一次分组查询，并去除重复数据即可实现。

## 代码实现

**public** Map search(Map<String, String> searchMap) {  
 Map map = **new** HashMap();  
 *//1、构建基本查询条件* NativeSearchQueryBuilder builder = builderBasicQuery(searchMap);  
 *//2、根据查询条件-搜索商品列表* searchList(map, builder);  
 *//3、跟据查询条件-分组查询商品分类列表* searchCategoryList(map,builder);  
 *//4、跟据查询条件-分组查询品牌列表* searchBrandList(map,builder);  
 *//5、跟据查询条件-分组查询规格列表* searchSpec(map,builder);  
 **return** map;  
}  
  
*/\*\*  
 \* 跟据查询条件-分组查询规格列表  
 \** ***@param map*** *结果集包装  
 \** ***@param builder*** *查询条件构建器  
 \*/***private void** searchSpec(Map map,NativeSearchQueryBuilder builder){  
 *//1.设置分组域名-termsAggregationBuilder = AggregationBuilders.terms(别名).field(域名).size(查询记录数);* TermsAggregationBuilder termsAggregationBuilder = AggregationBuilders.*terms*(**"group\_spec"**).field(**"spec.keyword"**).size(10000);  
 *//2.添加分组查询参数-builder.addAggregation(termsAggregationBuilder)* builder.addAggregation(termsAggregationBuilder);  
 *//3.执行搜索-esTemplate.queryForPage(builder.build(), SkuInfo.class)* AggregatedPage<SkuInfo> page = **esTemplate**.queryForPage(builder.build(), SkuInfo.**class**);  
 *//4.获取所有分组查询结果集-page.getAggregations()* Aggregations aggregations = page.getAggregations();  
 *//5.提取分组结果数据-stringTerms = aggregations.get(填入刚才查询时的别名)* StringTerms stringTerms = aggregations.get(**"group\_spec"**);  
 *//6.定义分类名字列表-categoryList = new ArrayList<String>()* List<String> specList = **new** ArrayList<String>();  
 *//7.遍历读取分组查询结果-stringTerms.getBuckets().for* **for** (StringTerms.Bucket bucket : stringTerms.getBuckets()) {  
 *//7.1获取分类名字，并将分类名字存入到集合中-bucket.getKeyAsString()* specList.add(bucket.getKeyAsString());  
 }  
  
 *//所有规格列表* Map<String, Set<String>> specMap = **new** HashMap<>();  
 *//包装List<Spec>，组装成Map<String,Set>* **for** (String spec : specList) {  
 *//{"电视音响效果":"小影院","电视屏幕尺寸":"20英寸","尺码":"165"}  
 //把spec的json串转换成Map<String,String>* Map<String, String> tempMap = JSON.*parseObject*(spec, Map.**class**);  
 *//循环读取key与value，组装到结果集中* **for** (String key : tempMap.keySet()) {  
 Set<String> values = specMap.get(key);  
 *//如果当前key是第一次组装* **if**(values == **null**){  
 values = **new** HashSet<String>();  
 }  
 *//向结果value中追加一个元素* values.add(tempMap.get(key));  
 *//规格结果集追加元素* specMap.put(key, values);  
 }  
 }  
 *//8.返回分类数据列表-map.put("categoryList", categoryList)* map.put(**"specMap"**, specMap);  
}

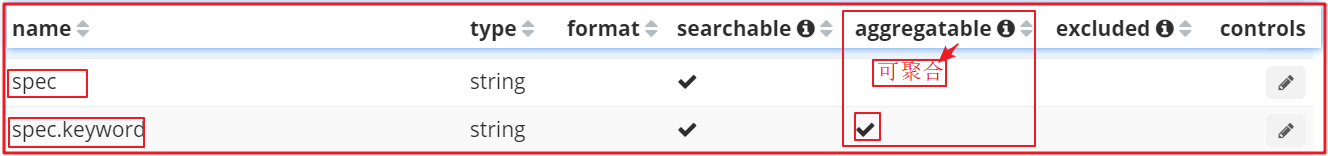
## 测试

使用Postman测试访问http://localhost:18086/search 效果如下：

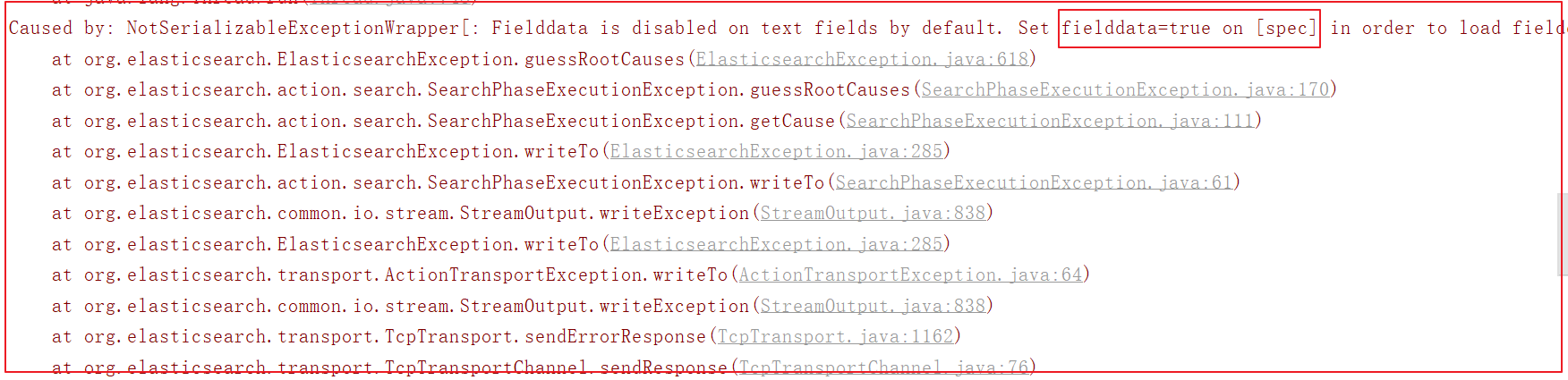


## 注意与小结

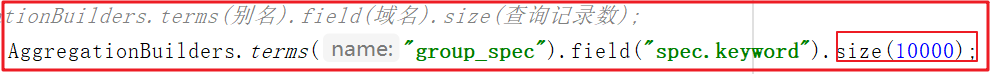
* 分组聚合(terms)查询中，分组的域存储内容如果比较复杂时，需要使用es生成的聚合域名(.keyword)



如果不使用此域名查询会报错：



* 分组聚合(terms)查询中，如果聚合后结果太多时，搜索结果会出现误差，可以使用.size()设置查询记录



# 分组综合优化(作业)

上面的代码看起来比较臃肿，能否进行简化呢？

如果按照下面这种方式，将多个分组查询写到一起，实现一次查询效率会更高，同学们可以去实现一次：

@Override  
**public** Map search(Map<String, String> searchMap) {  
 Map map = **new** HashMap();  
 *//1、构建基本查询条件* NativeSearchQueryBuilder builder = builderBasicQuery(searchMap);  
 *//2、根据查询条件-搜索商品列表* searchList(map, builder);  
   
 */\*//3、跟据查询条件-分组查询商品分类列表  
 searchCategoryList(map,builder);  
 //4、跟据查询条件-分组查询品牌列表  
 searchBrandList(map,builder);  
 //5、跟据查询条件-分组查询规格列表  
 searchSpec(map,builder);\*/  
  
 //一次分组查询分类、品牌与规格* searchGroup(map,builder);  
 **return** map;  
}  
  
*/\*\*  
 \* 跟据查询条件-分组查询商品分类、品牌、规格列表  
 \** ***@param map*** *结果集包装  
 \** ***@param builder*** *查询条件构建器  
 \*/***private void** searchGroup(Map map,NativeSearchQueryBuilder builder){  
 *//1.设置分组域名-termsAggregationBuilder = AggregationBuilders.terms(别名).field(域名).size(查询记录数);  
 //TermsAggregationBuilder termsAggregationBuilder = AggregationBuilders.terms("group\_spec").field("spec.keyword").size(10000);  
 //2.添加分组查询参数-builder.addAggregation(termsAggregationBuilder)  
 //一次设置三个聚合分组条件，一起查询* builder.addAggregation(AggregationBuilders.*terms*(**"group\_category"**).field(**"categoryName"**));  
 builder.addAggregation(AggregationBuilders.*terms*(**"group\_brand"**).field(**"brandName"**));  
 builder.addAggregation(AggregationBuilders.*terms*(**"group\_spec"**).field(**"spec.keyword"**).size(10000));  
 *//3.执行搜索-esTemplate.queryForPage(builder.build(), SkuInfo.class)* AggregatedPage<SkuInfo> page = **esTemplate**.queryForPage(builder.build(), SkuInfo.**class**);  
 *//4.获取所有分组查询结果集-page.getAggregations()* Aggregations aggregations = page.getAggregations();  
  
 *//1、提取分类结果* List<String> categoryList = getGroupResult(aggregations, **"group\_category"**);  
 map.put(**"categoryList"**, categoryList);  
  
 *//2、提取品牌结果* List<String> brandList = getGroupResult(aggregations, **"group\_brand"**);  
 map.put(**"brandList"**, brandList);  
  
 *//3、提取规格结果* List<String> specList = getGroupResult(aggregations, **"group\_spec"**);  
 *//所有规格列表* Map<String, Set<String>> specMap = **new** HashMap<>();  
 *//包装List<Spec>，组装成Map<String,Set>* **for** (String spec : specList) {  
 *//{"电视音响效果":"小影院","电视屏幕尺寸":"20英寸","尺码":"165"}  
 //把spec的json串转换成Map<String,String>* Map<String, String> tempMap = JSON.*parseObject*(spec, Map.**class**);  
 *//循环读取key与value，组装到结果集中* **for** (String key : tempMap.keySet()) {  
 Set<String> values = specMap.get(key);  
 *//如果当前key是第一次组装* **if**(values == **null**){  
 values = **new** HashSet<String>();  
 }  
 *//向结果value中追加一个元素* values.add(tempMap.get(key));  
 *//规格结果集追加元素* specMap.put(key, values);  
 }  
 }  
 *//8.返回分类数据列表-map.put("categoryList", categoryList)* map.put(**"specMap"**, specMap);  
}  
  
*/\*\*  
 \* 提取分组聚合结果  
 \** ***@param aggregations*** *聚合结果对象  
 \** ***@param group\_name****分组域的别名**\** ***@return*** *提取的结果集  
 \*/***private** List<String> getGroupResult(Aggregations aggregations, String group\_name) {  
 *//5.提取分组结果数据-stringTerms = aggregations.get(填入刚才查询时的别名)* StringTerms stringTerms = aggregations.get(group\_name);  
 *//6.定义分类名字列表-categoryList = new ArrayList<String>()* List<String> specList = **new** ArrayList<String>();  
 *//7.遍历读取分组查询结果-stringTerms.getBuckets().for* **for** (StringTerms.Bucket bucket : stringTerms.getBuckets()) {  
 *//7.1获取分类名字，并将分类名字存入到集合中-bucket.getKeyAsString()* specList.add(bucket.getKeyAsString());  
 }  
 **return** specList;  
}

# 条件筛选



用户有可能会根据分类搜索、品牌搜索，还有可能根据规格搜索，以及价格搜索和排序操作。根据分类和品牌搜索的时候，可以直接根据指定域搜索，而规格搜索的域数据是不确定的，价格是一个区间搜索，所以我们可以分为三段时间，先实现分类、品牌搜素，再实现规格搜索，然后实现价格区间搜索。

## 分类、品牌筛选

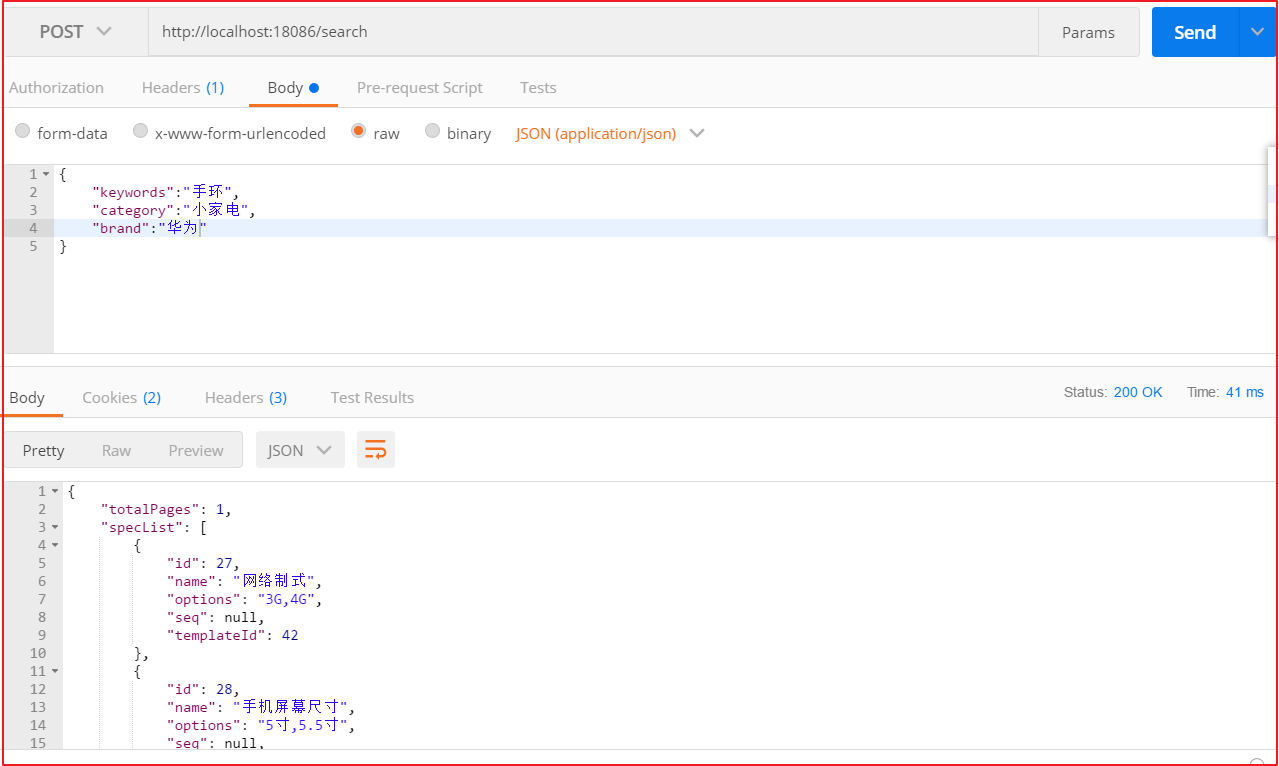
页面每次向后台传入对应的分类和品牌，后台据分类和品牌进行条件过滤即可。

### 实现筛选

修改搜索微服务com.changgou.search.service.impl.SkuServiceImpl的buildBasicQuery方法，添加分类和品牌过滤，代码如下：

*/\*\*  
 \* 构建基本查询条件  
 \** ***@param searchMap*** *用户传入的查询参数  
 \** ***@return*** *查询条件构建器  
 \*/***private** NativeSearchQueryBuilder builderBasicQuery(Map<String, String> searchMap) {  
 *//1、创建查询条件构建器-builder = new NativeSearchQueryBuilder()* NativeSearchQueryBuilder builder = **new** NativeSearchQueryBuilder();  
 *//2、组装查询条件* **if**(searchMap != **null**){  
 BoolQueryBuilder boolQueryBuilder = QueryBuilders.*boolQuery*();  
 *//2.1关键字搜索-builder.withQuery(QueryBuilders.matchQuery(域名，内容))* String keywords = searchMap.get(**"keywords"**) == **null** ? **""** : searchMap.get(**"keywords"**);  
 *//如果用户传入了关键字* **if**(StringUtils.*isNotEmpty*(keywords)){  
 *//查询name域  
 //builder.withQuery(QueryBuilders.matchQuery("name", keywords));* boolQueryBuilder.must(QueryBuilders.*matchQuery*(**"name"**, keywords));  
 }  
 *//2.2 分类查询* String category = searchMap.get(**"category"**) == **null** ? **""** : searchMap.get(**"category"**);  
 *//如果用户传入了分类* **if**(StringUtils.*isNotEmpty*(category)){  
 *//查询category域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"categoryName"**, category));  
 }  
 *//2.3 品牌查询* String brand = searchMap.get(**"brand"**) == **null** ? **""** : searchMap.get(**"brand"**);  
 *//如果用户传入了品牌* **if**(StringUtils.*isNotEmpty*(brand)){  
 *//查询brand域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"brandName"**, brand));  
 }  
 *//追加多条件匹配搜索* builder.withQuery(boolQueryBuilder);  
 }  
 **return** builder;  
}

### 测试

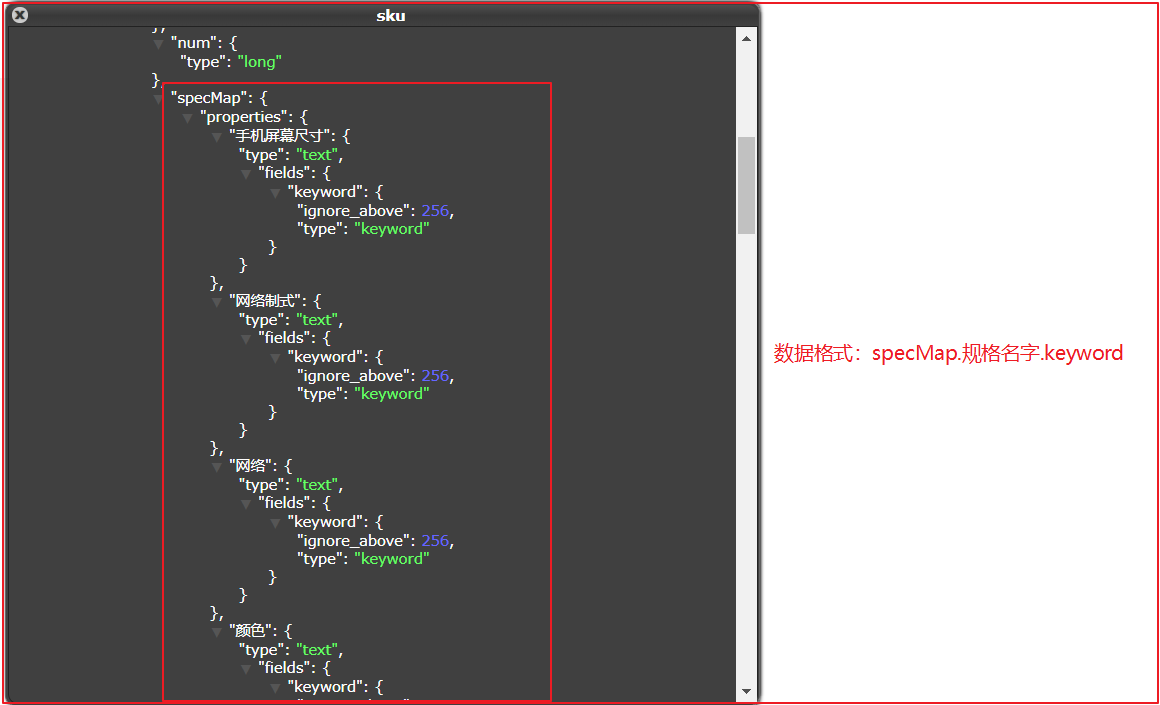


## 规格筛选



规格这一块，需要向后台发送规格名字以及规格值，我们可以按照一定要求来发送数据，例如规格名字以特殊前缀提交到后台：spec\_网络制式：电信4G、spec\_显示屏尺寸：4.0-4.9英寸

后台接到数据后，可以根据前缀spec\_来区分是否是规格，如果以spec\_xxx开始的数据则为规格数据，需要根据指定规格找信息。



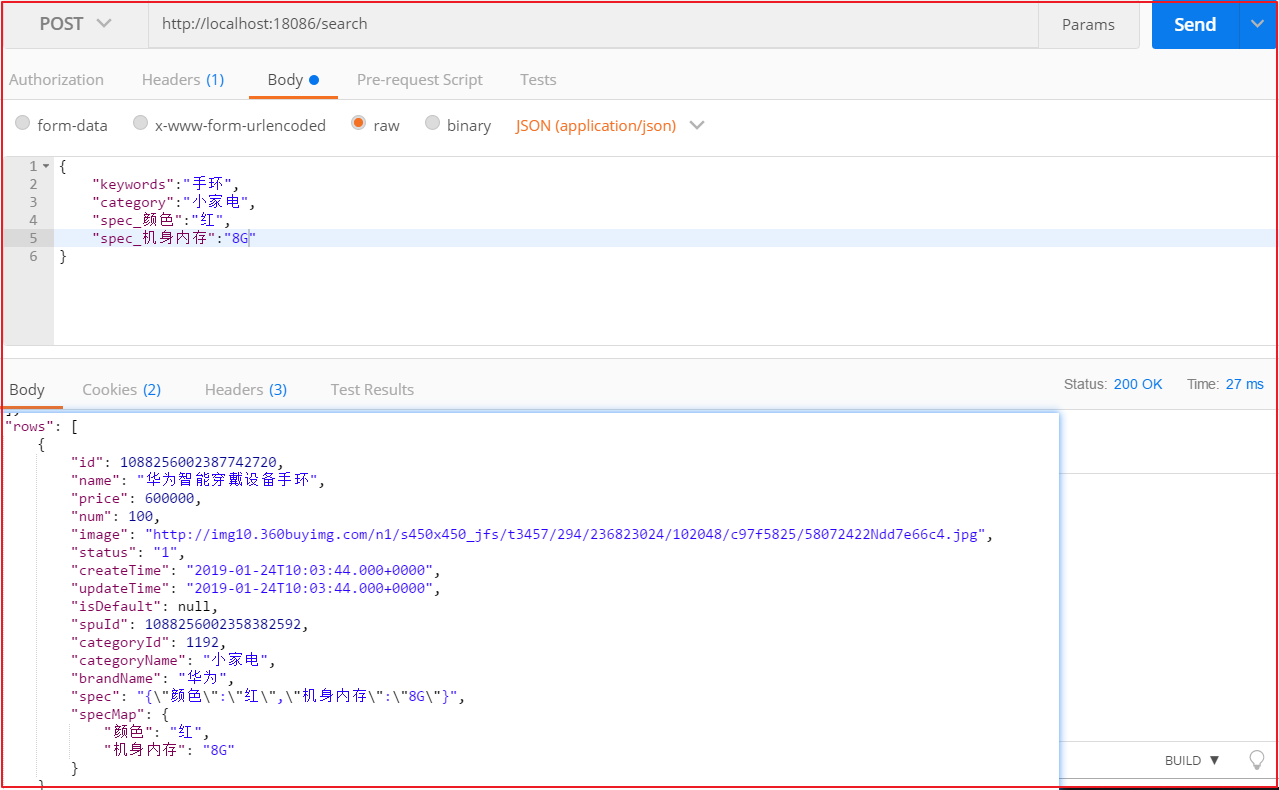
上图是规格的索引存储格式，真实数据在spechMap.规格名字.keyword中，所以找数据也是按照如下格式去找：specMap.规格名字.keyword

### 代码实现

修改com.changgou.search.service.impl.SkuServiceImpl的buildBasicQuery方法，增加规格查询，代码如下：

*/\*\*  
 \* 构建基本查询条件  
 \** ***@param searchMap*** *用户传入的查询参数  
 \** ***@return*** *查询条件构建器  
 \*/***private** NativeSearchQueryBuilder builderBasicQuery(Map<String, String> searchMap) {  
 *//1、创建查询条件构建器-builder = new NativeSearchQueryBuilder()* NativeSearchQueryBuilder builder = **new** NativeSearchQueryBuilder();  
 *//2、组装查询条件* **if**(searchMap != **null**){  
 BoolQueryBuilder boolQueryBuilder = QueryBuilders.*boolQuery*();  
 *//2.1关键字搜索-builder.withQuery(QueryBuilders.matchQuery(域名，内容))* String keywords = searchMap.get(**"keywords"**) == **null** ? **""** : searchMap.get(**"keywords"**);  
 *//如果用户传入了关键字* **if**(StringUtils.*isNotEmpty*(keywords)){  
 *//查询name域  
 //builder.withQuery(QueryBuilders.matchQuery("name", keywords));* boolQueryBuilder.must(QueryBuilders.*matchQuery*(**"name"**, keywords));  
 }  
 *//2.2 分类查询* String category = searchMap.get(**"category"**) == **null** ? **""** : searchMap.get(**"category"**);  
 *//如果用户传入了分类* **if**(StringUtils.*isNotEmpty*(category)){  
 *//查询category域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"categoryName"**, category));  
 }  
 *//2.3 品牌查询* String brand = searchMap.get(**"brand"**) == **null** ? **""** : searchMap.get(**"brand"**);  
 *//如果用户传入了品牌* **if**(StringUtils.*isNotEmpty*(brand)){  
 *//查询brand域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"brandName"**, brand));  
 }  
 *//2.4 规格查询  
 //读取用户传入的所有参数的key* **for** (String key : searchMap.keySet()) {  
 *//识别规格:用户传入，spec\_网络制式：电信4G* **if**(key.startsWith(**"spec\_"**)){  
 *//specMap.规格名字.keyword* String specField = **"specMap."** + key.substring(5) + **".keyword"**;  
 *//规格域* boolQueryBuilder.must(QueryBuilders.*termQuery*(specField, searchMap.get(key)));  
 }  
 }  
 *//追加多条件匹配搜索* builder.withQuery(boolQueryBuilder);  
 }  
 **return** builder;  
}

### 测试



## 价格区间筛选



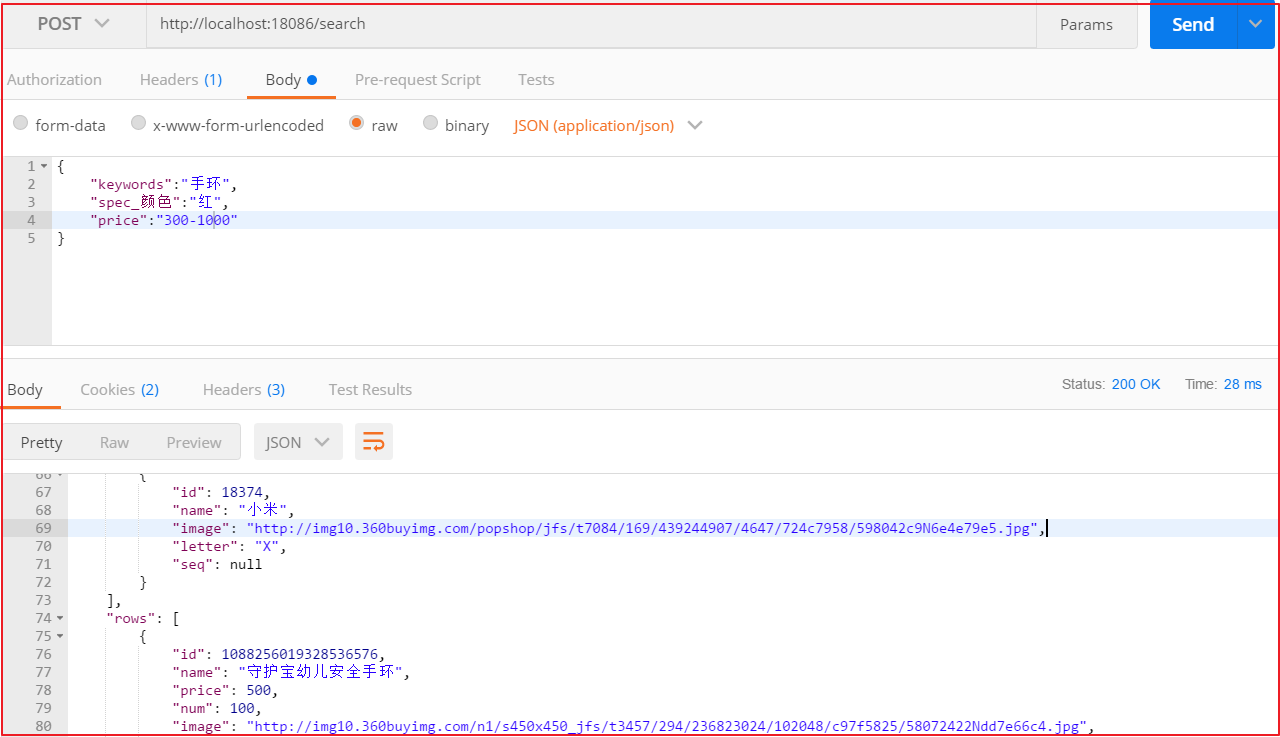
价格区间查询，每次需要将价格传入到后台，前端传入后台的价格大概是price=0-500或者price=500-1000依次类推，最后一个是price=3000,后台可以根据-分割，如果分割得到的结果最多有2个，第1个表示x<price，第2个表示price<=y。

### 代码实现

修改com.changgou.search.service.impl.SkuServiceImpl的buildBasicQuery方法，增加价格区间查询操作，代码如下：

*/\*\*  
 \* 构建基本查询条件  
 \** ***@param searchMap*** *用户传入的查询参数  
 \** ***@return*** *查询条件构建器  
 \*/***private** NativeSearchQueryBuilder builderBasicQuery(Map<String, String> searchMap) {  
 *//1、创建查询条件构建器-builder = new NativeSearchQueryBuilder()* NativeSearchQueryBuilder builder = **new** NativeSearchQueryBuilder();  
 *//2、组装查询条件* **if**(searchMap != **null**){  
 BoolQueryBuilder boolQueryBuilder = QueryBuilders.*boolQuery*();  
 *//2.1关键字搜索-builder.withQuery(QueryBuilders.matchQuery(域名，内容))* String keywords = searchMap.get(**"keywords"**) == **null** ? **""** : searchMap.get(**"keywords"**);  
 *//如果用户传入了关键字* **if**(StringUtils.*isNotEmpty*(keywords)){  
 *//查询name域  
 //builder.withQuery(QueryBuilders.matchQuery("name", keywords));* boolQueryBuilder.must(QueryBuilders.*matchQuery*(**"name"**, keywords));  
 }  
 *//2.2 分类查询* String category = searchMap.get(**"category"**) == **null** ? **""** : searchMap.get(**"category"**);  
 *//如果用户传入了分类* **if**(StringUtils.*isNotEmpty*(category)){  
 *//查询category域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"categoryName"**, category));  
 }  
 *//2.3 品牌查询* String brand = searchMap.get(**"brand"**) == **null** ? **""** : searchMap.get(**"brand"**);  
 *//如果用户传入了品牌* **if**(StringUtils.*isNotEmpty*(brand)){  
 *//查询brand域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"brandName"**, brand));  
 }  
 *//2.4 规格查询  
 //读取用户传入的所有参数的key* **for** (String key : searchMap.keySet()) {  
 *//识别规格:用户传入，spec\_网络制式：电信4G* **if**(key.startsWith(**"spec\_"**)){  
 *//specMap.规格名字.keyword* String specField = **"specMap."** + key.substring(5) + **".keyword"**;  
 *//规格域* boolQueryBuilder.must(QueryBuilders.*termQuery*(specField, searchMap.get(key)));  
 }  
 }  
 *//2.5 价格区间查询* String price = searchMap.get(**"price"**) == **null** ? **""** : searchMap.get(**"price"**);  
 *//如果用户传入了价格* **if**(StringUtils.*isNotEmpty*(price)){  
 *//范围匹配搜索* RangeQueryBuilder rangeQueryBuilder = QueryBuilders.*rangeQuery*(**"price"**);  
 *//解析前端传入的价格：0-500 ,500-1000,3000* String[] split = price.split(**"-"**);  
 *//处理前面的价格:price >= 0* boolQueryBuilder.must(rangeQueryBuilder.gte(split[0]));  
 *//如果解析结果大小1，说明传入的不是3000* **if**(split.**length** > 1){  
 *//price <= 500* boolQueryBuilder.must(rangeQueryBuilder.lte(split[1]));  
 }  
 }  
 *//追加多条件匹配搜索* builder.withQuery(boolQueryBuilder);  
 }  
 **return** builder;  
}

### 测试



# 搜索分页



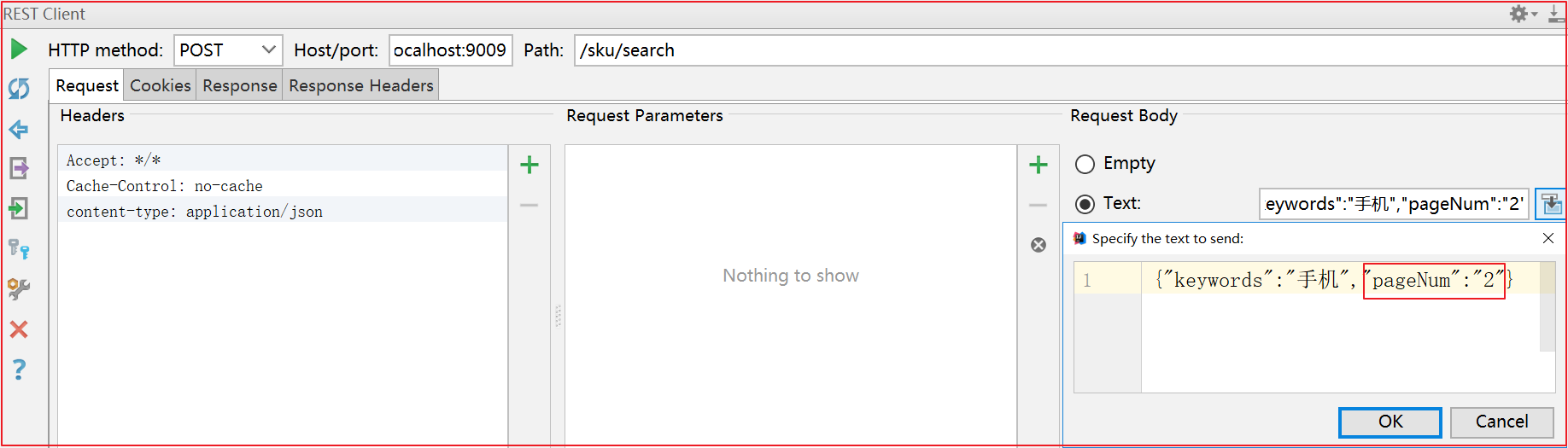
页面需要实现分页搜索，所以我们后台每次查询的时候，需要实现分页。用户页面每次会传入当前页和每页查询多少条数据，当然如果不传入每页显示多少条数据，默认查询30条即可。

## 分页实现

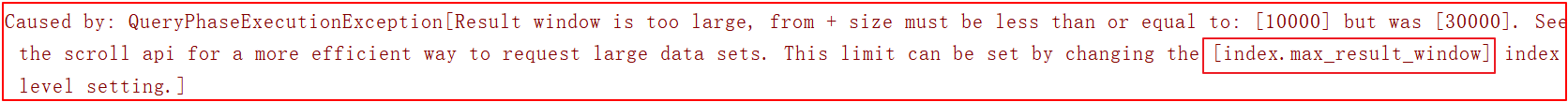
分页使用PageRequest.of( pageNo- 1, pageSize);实现，第1个参数表示第N页，从0开始，第2个参数表示每页显示多少条，实现代码如下：

*/\*\*  
 \* 构建基本查询条件  
 \** ***@param searchMap*** *用户传入的查询参数  
 \** ***@return*** *查询条件构建器  
 \*/***private** NativeSearchQueryBuilder builderBasicQuery(Map<String, String> searchMap) {  
 *//1、创建查询条件构建器-builder = new NativeSearchQueryBuilder()* NativeSearchQueryBuilder builder = **new** NativeSearchQueryBuilder();  
 *//2、组装查询条件* **if**(searchMap != **null**){  
 BoolQueryBuilder boolQueryBuilder = QueryBuilders.*boolQuery*();  
 *//2.1关键字搜索-builder.withQuery(QueryBuilders.matchQuery(域名，内容))* String keywords = searchMap.get(**"keywords"**) == **null** ? **""** : searchMap.get(**"keywords"**);  
 *//如果用户传入了关键字* **if**(StringUtils.*isNotEmpty*(keywords)){  
 *//查询name域  
 //builder.withQuery(QueryBuilders.matchQuery("name", keywords));* boolQueryBuilder.must(QueryBuilders.*matchQuery*(**"name"**, keywords));  
 }  
 *//2.2 分类查询* String category = searchMap.get(**"category"**) == **null** ? **""** : searchMap.get(**"category"**);  
 *//如果用户传入了分类* **if**(StringUtils.*isNotEmpty*(category)){  
 *//查询category域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"categoryName"**, category));  
 }  
 *//2.3 品牌查询* String brand = searchMap.get(**"brand"**) == **null** ? **""** : searchMap.get(**"brand"**);  
 *//如果用户传入了品牌* **if**(StringUtils.*isNotEmpty*(brand)){  
 *//查询brand域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"brandName"**, brand));  
 }  
 *//2.4 规格查询  
 //读取用户传入的所有参数的key* **for** (String key : searchMap.keySet()) {  
 *//识别规格:用户传入，spec\_网络制式：电信4G* **if**(key.startsWith(**"spec\_"**)){  
 *//specMap.规格名字.keyword* String specField = **"specMap."** + key.substring(5) + **".keyword"**;  
 *//规格域* boolQueryBuilder.must(QueryBuilders.*termQuery*(specField, searchMap.get(key)));  
 }  
 }  
 *//2.5 价格区间查询* String price = searchMap.get(**"price"**) == **null** ? **""** : searchMap.get(**"price"**);  
 *//如果用户传入了价格* **if**(StringUtils.*isNotEmpty*(price)){  
 *//范围匹配搜索* RangeQueryBuilder rangeQueryBuilder = QueryBuilders.*rangeQuery*(**"price"**);  
 *//解析前端传入的价格：0-500 ,500-1000,3000* String[] split = price.split(**"-"**);  
 *//处理前面的价格:price >= 0* boolQueryBuilder.must(rangeQueryBuilder.gte(split[0]));  
 *//如果解析结果大小1，说明传入的不是3000* **if**(split.**length** > 1){  
 *//price <= 500* boolQueryBuilder.must(rangeQueryBuilder.lte(split[1]));  
 }  
 }  
 *//追加多条件匹配搜索* builder.withQuery(boolQueryBuilder);  
  
 *//当前页* Integer page = searchMap.get(**"pageNum"**) == **null** ? 0 : **new** Integer(searchMap.get(**"pageNum"**)) - 1;  
 Integer pageSize = 5; *//每页查询记录数  
 //PageRequest.of(当前页【0开始】，每页查询的条数)* PageRequest pageRequest = PageRequest.*of*(page, pageSize);  
 *//设置分页查询* builder.withPageable(pageRequest);  
 }  
 **return** builder;  
}

## 测试



注意问题: 当ES数据量较大，使用分页查询超过10000条（1000页）时，出现如下错误：



解决方法： 1. 使用滚动API；2. 直接设置[index.max\_result\_window]参数。

在Kibana控制面板里面，则执行下面这段：

PUT sku/\_settings

{

"index":{

"max\_result\_window":1000000

}

}

# 搜索排序



排序这里总共有根据价格排序、根据评价排序、根据新品排序、根据销量排序，排序要想实现非常简单，只需要告知排序的域以及排序方式即可实现。

* 价格排序：只需要根据价格高低排序即可，降序价格高->低，升序价格低->高
* 评价排序：评价分为好评、中评、差评，可以在数据库中设计3个列，用来记录好评、中评、差评的量，每次排序的时候，好评的比例来排序，当然还要有条数限制，评价条数需要超过N条。
* 新品排序：直接根据商品的发布时间或者更新时间排序。
* 销量排序：销量排序除了销售数量外，还应该要有时间段限制。

## 排序实现

这里我们不单独针对某个功能实现排序，我们只需要在后台接收2个参数，分别是排序域名字和排序方式，代码如下：

*/\*\*  
 \* 构建基本查询条件  
 \** ***@param searchMap*** *用户传入的查询参数  
 \** ***@return*** *查询条件构建器  
 \*/***private** NativeSearchQueryBuilder builderBasicQuery(Map<String, String> searchMap) {  
 *//1、创建查询条件构建器-builder = new NativeSearchQueryBuilder()* NativeSearchQueryBuilder builder = **new** NativeSearchQueryBuilder();  
 *//2、组装查询条件* **if**(searchMap != **null**){  
 BoolQueryBuilder boolQueryBuilder = QueryBuilders.*boolQuery*();  
 *//2.1关键字搜索-builder.withQuery(QueryBuilders.matchQuery(域名，内容))* String keywords = searchMap.get(**"keywords"**) == **null** ? **""** : searchMap.get(**"keywords"**);  
 *//如果用户传入了关键字* **if**(StringUtils.*isNotEmpty*(keywords)){  
 *//查询name域  
 //builder.withQuery(QueryBuilders.matchQuery("name", keywords));* boolQueryBuilder.must(QueryBuilders.*matchQuery*(**"name"**, keywords));  
 }  
 *//2.2 分类查询* String category = searchMap.get(**"category"**) == **null** ? **""** : searchMap.get(**"category"**);  
 *//如果用户传入了分类* **if**(StringUtils.*isNotEmpty*(category)){  
 *//查询category域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"categoryName"**, category));  
 }  
 *//2.3 品牌查询* String brand = searchMap.get(**"brand"**) == **null** ? **""** : searchMap.get(**"brand"**);  
 *//如果用户传入了品牌* **if**(StringUtils.*isNotEmpty*(brand)){  
 *//查询brand域* boolQueryBuilder.must(QueryBuilders.*termQuery*(**"brandName"**, brand));  
 }  
 *//2.4 规格查询  
 //读取用户传入的所有参数的key* **for** (String key : searchMap.keySet()) {  
 *//识别规格:用户传入，spec\_网络制式：电信4G* **if**(key.startsWith(**"spec\_"**)){  
 *//specMap.规格名字.keyword* String specField = **"specMap."** + key.substring(5) + **".keyword"**;  
 *//规格域* boolQueryBuilder.must(QueryBuilders.*termQuery*(specField, searchMap.get(key)));  
 }  
 }  
 *//2.5 价格区间查询* String price = searchMap.get(**"price"**) == **null** ? **""** : searchMap.get(**"price"**);  
 *//如果用户传入了价格* **if**(StringUtils.*isNotEmpty*(price)){  
 *//范围匹配搜索* RangeQueryBuilder rangeQueryBuilder = QueryBuilders.*rangeQuery*(**"price"**);  
 *//解析前端传入的价格：0-500 ,500-1000,3000* String[] split = price.split(**"-"**);  
 *//处理前面的价格:price >= 0* boolQueryBuilder.must(rangeQueryBuilder.gte(split[0]));  
 *//如果解析结果大小1，说明传入的不是3000* **if**(split.**length** > 1){  
 *//price <= 500* boolQueryBuilder.must(rangeQueryBuilder.lte(split[1]));  
 }  
 }  
 *//追加多条件匹配搜索* builder.withQuery(boolQueryBuilder);  
  
 *//当前页* Integer page = searchMap.get(**"pageNum"**) == **null** ? 1 : **new** Integer(searchMap.get(**"pageNum"**));  
 Integer pageSize = 5; *//每页查询记录数  
 //PageRequest.of(当前页【0开始】，每页查询的条数)* PageRequest pageRequest = PageRequest.*of*(page, pageSize);  
 *//设置分页查询* builder.withPageable(pageRequest);  
  
 *//排序  
 //排序方式:ASC|DESC* String sortRule = searchMap.get(**"sortRule"**) == **null** ? **""** : searchMap.get(**"sortRule"**);  
 *//排序域名* String sortField = searchMap.get(**"sortField"**) == **null** ? **""** : searchMap.get(**"sortField"**);  
 **if**(StringUtils.*isNotEmpty*(sortField)){  
 *//fieldSort(域名)，order(排序方式)* builder.withSort(SortBuilders.*fieldSort*(sortField).order(SortOrder.*valueOf*(sortRule)));  
 }  
 }  
 **return** builder;  
}

## 测试

根据价格降序：

{"keywords":"手机","pageNum":"1","sortRule":"DESC","sortField":"price"}

根据价格升序：

{"keywords":"手机","pageNum":"1","sortRule":"ASC","sortField":"price"}

# 高亮显示



高亮显示是指根据商品关键字搜索商品的时候，显示的页面对关键字给定了特殊样式，让它显示更加突出，如上图商品搜索中，关键字编程了红色，其实就是给定了红色样式。



## 高亮搜索实现步骤解析

将之前的搜索代码换掉，换成高亮搜索，我们需要做3个步骤：

1.指定高亮域，也就是设置哪个域需要高亮显示

设置高亮域的时候，需要指定前缀和后缀，也就是关键词用什么html标签包裹，再给该标签样式

2.高亮搜索实现

3.将非高亮数据替换成高亮数据

步骤解析

第1点，例如在百度中搜索数据的时候，会有2个地方高亮显示，分别是标题和描述，商城搜索的时候，只是商品名称高亮显示了。而高亮显示其实就是添加了样式，例如<span style="color:red;">笔记本</span>,而其中span开始标签可以称为前缀，span结束标签可以称为后缀。

第2点，高亮搜索使用ElasticsearchTemplate实现。

第3点，高亮搜索后，会搜出非高亮数据和高亮数据，高亮数据会加上第1点中的高亮样式，此时我们需要将非高亮数据换成高亮数据即可。例如非高亮:华为笔记本性能超强悍 高亮数据：华为<span style="color:red;"笔记本</span>性能超强悍,将非高亮的换成高亮的，到页面就能显示样式了。

## 高亮代码实现

ElasticsearchTemplate实现高亮搜索，需要把searchList(Map map, NativeSearchQueryBuilder builder)方法的查询逻辑换掉。

**API实现步骤：**

//h1.配置高亮查询信息-hField = new HighlightBuilder.Field()

//h1.1:设置高亮域名-在构造函数中设置

//h1.2：设置高亮前缀-hField.preTags

//h1.3：设置高亮后缀-hField.postTags

//h1.4：设置碎片大小-hField.fragmentSize

//h1.5：追加高亮查询信息-builder.withHighlightFields()

//h2.高亮数据读取-AggregatedPage<SkuInfo> page = esTemplate.queryForPage(query, SkuInfo.class, new SearchResultMapper(){})

//h2.1实现mapResults(查询到的结果,数据列表的类型,分页选项)方法

//h2.2 先定义一组查询结果列表-List<T> list = new ArrayList<T>()

//h2.3 遍历查询到的所有高亮数据-response.getHits().for

//h2.3.1 先获取当次结果的原始数据(无高亮)-hit.getSourceAsString()

//h2.3.2 把json串转换为SkuInfo对象-skuInfo = JSON.parseObject()

//h2.3.3 获取name域的高亮数据-nameHighlight = hit.getHighlightFields().get("name")

//h2.3.4 如果高亮数据不为空-读取高亮数据

//h2.3.4.1 定义一个StringBuffer用于存储高亮碎片-buffer = new StringBuffer()

//h2.3.4.2 循环组装高亮碎片数据- nameHighlight.getFragments().for(追加数据)

//h2.3.4.3 将非高亮数据替换成高亮数据-skuInfo.setName()

//h2.3.5 将替换了高亮数据的对象封装到List中-list.add((T) esItem)

//h2.4 返回当前方法所需要参数-new AggregatedPageImpl<T>(数据列表，分页选项,总记录数)

//h2.4 参考new AggregatedPageImpl<T>(list,pageable,response.getHits().getTotalHits())

**代码实现：**

*/\*\*  
 \* 根据查询条件-搜索商品列表  
 \** ***@param map*** *结果集  
 \** ***@param builder*** *查询条件构建器  
 \*/***private void** searchList(Map map, NativeSearchQueryBuilder builder) {  
 *//h1.配置高亮查询信息-hField = new HighlightBuilder.Field()  
 //h1.1:设置高亮域名-在构造函数中设置* HighlightBuilder.Field hField = **new** HighlightBuilder.Field(**"name"**);  
 *//h1.2：设置高亮前缀-hField.preTags* hField.preTags(**"<em style='color:red;'>"**);  
 *//h1.3：设置高亮后缀-hField.postTags* hField.postTags(**"</em>"**);  
 *//h1.4：设置碎片大小-hField.fragmentSize* hField.fragmentSize(100);  
 *//h1.5：追加高亮查询信息-builder.withHighlightFields()* builder.withHighlightFields(hField);  
  
 *//3、获取NativeSearchQuery搜索条件对象-builder.build()* NativeSearchQuery query = builder.build();  
 *//无高亮时的查询方式  
 /\*//4.查询数据-esTemplate.queryForPage(条件对象,搜索结果对象)  
 AggregatedPage<SkuInfo> page = esTemplate.queryForPage(query, SkuInfo.class);\*/  
  
 //高亮查询方式  
 //h2.高亮数据读取-AggregatedPage<SkuInfo> page = esTemplate.queryForPage(query, SkuInfo.class, new SearchResultMapper(){})* AggregatedPage<SkuInfo> page = **esTemplate**.queryForPage(query, SkuInfo.**class**, **new** SearchResultMapper() {  
 @Override  
 *//h2.1实现mapResults(查询到的结果,数据列表的类型,分页选项)方法* **public** <T> AggregatedPage<T> mapResults(SearchResponse response, Class<T> clazz, Pageable pageable) {  
 *//h2.2 先定义一组查询结果列表-List<T> list = new ArrayList<T>()* List<T> list = **new** ArrayList<T>();  
 *//h2.3 遍历查询到的所有高亮数据-response.getHits().for* **for** (SearchHit hit : response.getHits()) {  
 *//h2.3.1 先获取当次结果的原始数据(无高亮)-hit.getSourceAsString()* String json = hit.getSourceAsString();  
 *//h2.3.2 把json串转换为SkuInfo对象-skuInfo = JSON.parseObject()* SkuInfo skuInfo = JSON.*parseObject*(json, SkuInfo.**class**);  
 *//h2.3.3 获取name域的高亮数据-nameHighlight = hit.getHighlightFields().get("name")* HighlightField nameHighlight = hit.getHighlightFields().get(**"name"**);  
  
 *//h2.3.4 如果高亮数据不为空-读取高亮数据* **if**(nameHighlight != **null**){  
 *//h2.3.4.1 定义一个StringBuffer用于存储高亮碎片-buffer = new StringBuffer()* StringBuffer buffer = **new** StringBuffer();  
 *//h2.3.4.2 循环组装高亮碎片数据-nameHighlight.getFragments().for(追加数据)* **for** (Text fragment : nameHighlight.getFragments()) {  
 buffer.append(fragment);  
 }  
 *//h2.3.4.3 将非高亮数据替换成高亮数据-skuInfo.setName()* skuInfo.setName(buffer.toString());  
 }  
 *//h2.3.5 将替换了高亮数据的对象封装到List中-list.add((T) skuInfo)* list.add((T) skuInfo);  
 }  
 *//h2.4 返回当前方法所需要参数-new AggregatedPageImpl<T>(数据列表，分页选项,总记录数)  
 //h2.4 参考new AggregatedPageImpl<T>(list,pageable,response.getHits().getTotalHits())* **return new** AggregatedPageImpl<T>(list, pageable, response.getHits().getTotalHits());  
 }  
 });  
 *//5、包装结果并返回* map.put(**"rows"**, page.getContent());  
 map.put(**"total"**, page.getTotalElements());  
 map.put(**"totalPages"**, page.getTotalPages());  
}

## 测试

