第7章 商品搜索

学习目标

- 根据搜索关键字查询
- 条件筛选
- 规格过滤
- 价格区间搜索
- 分页查询
- 排序查询
- 高亮查询

1 根据关键字查询

(1) changgou_service_search项目创建SearchService接口

```
public interface SearchService {

    /**
     * 全文检索
     * @param paramMap  查询参数
     * @return
     */
    public Map search(Map<String, String> paramMap) throws Exception;
}
```

(2) changgou_service_search项目创建SearchService接口实现类SearchServiceImpl

```
@Service
public class SearchServiceImpl implements SearchService {

    @Autowired
    private ElasticsearchTemplate esTemplate;



    //设置每页查询条数据
    public final static Integer PAGE_SIZE = 20;

    @Override
    public Map search(Map<String, String> searchMap) throws Exception {
        Map<String, Object> resultMap = new HashMap<>();

        //有条件才查询Es
        if (null != searchMap) {
            //组合条件对象
            BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
            //0:关键词
            if (!StringUtils.isEmpty(searchMap.get("keywords"))) {
                boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));

            }

            //4. 原生搜索实现类
            NativeSearchQueryBuilder nativeSearchQueryBuilder = new
NativeSearchQueryBuilder();
            nativeSearchQueryBuilder.withQuery(boolQuery);

            //10: 执行查询，返回结果对象
            AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
                @Override
                public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {


                    List<T> list = new ArrayList<>();
```

```
                    SearchHits hits = searchResponse.getHits();
                    if (null != hits) {
                        for (SearchHit hit : hits) {
                            SkuInfo skuInfo =
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);

                            list.add((T) skuInfo);
                        }
                    }
                    return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
                }
            });

            //11. 总条数
            resultMap.put("total", aggregatedPage.getTotalElements());
            //12. 总页数
            resultMap.put("totalPages", aggregatedPage.getTotalPages());
            //13. 查询结果集合
            resultMap.put("rows", aggregatedPage.getContent());

            return resultMap;
        }
        return null;
    }

}
```

(3) changgou_service_search项目创建SearchController

```
@RestController
@RequestMapping("/sku_search")
public class SearchController {

    @Autowired
    private EsManagerService esManagerService;

    @Autowired
    private SearchService searchService;

     //对搜索入参带有特殊符号进行处理
    public void handlerSearchMap(Map<String,String> searchMap){

        if(null != searchMap){
            Set<Map.Entry<String, String>> entries =
searchMap.entrySet();
            for (Map.Entry<String, String> entry : entries) {
                if(entry.getKey().startsWith("spec_")){

searchMap.put(entry.getKey(),entry.getValue().replace("+","%2B"));
                }
            }
        }

    }

    /**
     * 全文检索
     * @return
     */
    @GetMapping
    public Map search(@RequestParam Map<String, String> paramMap) throws
Exception {
        //特殊符号处理
        handlerSearchMap(searchMap);
        Map resultMap = searchService.search(paramMap);
        return resultMap;
    }
}
```

(4) 测试

使用postmain访问 http://localhost:9009/sku_search?keywords=手机

2 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实现，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。

2.1 品牌筛选

2.1.1 需求分析

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。

2.1.2 代码实现

修改搜索微服务com.changgou.service.SearchServiceImpl的搜索方法，添加品牌过滤，代码如下：

代码如下：

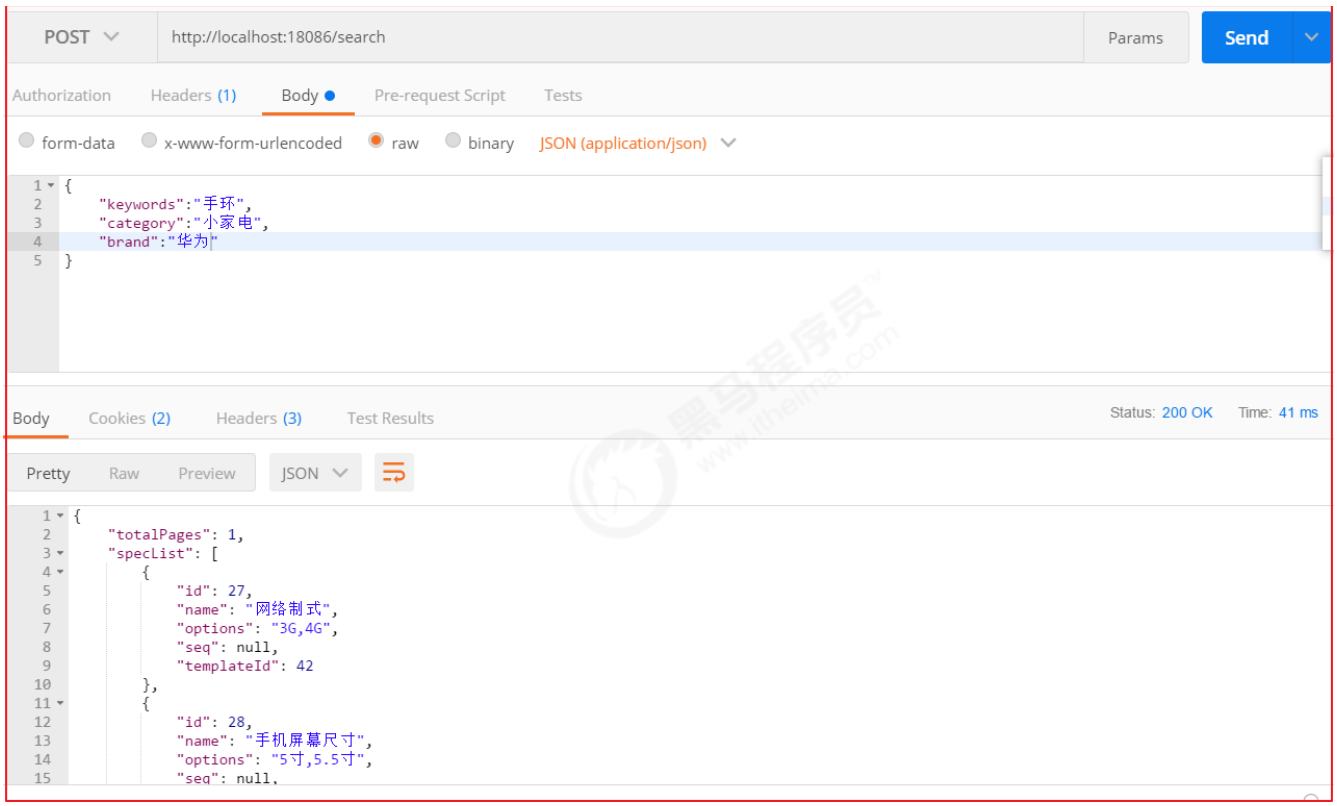
```
@Override
public Map search(Map<String, String> searchMap) throws Exception {
    Map<String, Object> resultMap = new HashMap<>();

    //有条件才查询Es
    if (null != searchMap) {
        //组合条件对象
        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
        //0:关键词
        if (StringUtils.isNotEmpty(searchMap.get("keywords"))) {
            boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));
        }
        //1:条件 品牌
        if (StringUtils.isNotEmpty(searchMap.get("brand"))) {
            boolQuery.filter(QueryBuilders.termQuery("brandName",
searchMap.get("brand")));
        }

        //4. 原生搜索实现类
        NativeSearchQueryBuilder nativeSearchQueryBuilder = new
NativeSearchQueryBuilder();
        nativeSearchQueryBuilder.withQuery(boolQuery);

        //6. 品牌聚合(分组)查询
        String skuBrand = "skuBrand";
nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuBran
d).field("brandName"));

        //10: 执行查询, 返回结果对象
        AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
            @Override
            public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {

                List<T> list = new ArrayList<>();


                SearchHits hits = searchResponse.getHits();
```

```
                if (null != hits) {
                    for (SearchHit hit : hits) {
                        SkuInfo skuInfo =
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);

                        list.add((T) skuInfo);
                    }
                }
                return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
            }
        });

        //11. 总条数
        resultMap.put("total", aggregatedPage.getTotalElements());
        //12. 总页数
        resultMap.put("totalPages", aggregatedPage.getTotalPages());
        //13. 查询结果集合
        resultMap.put("rows", aggregatedPage.getContent());

        //14. 获取品牌聚合结果
        StringTerms brandTerms = (StringTerms)
aggregatedPage.getAggregation(skuBrand);
        List<String> brandList =
brandTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
        resultMap.put("brandList", brandList);

        return resultMap;
    }
    return null;
}
```

2.1.3 测试

测试效果如下：

访问地址：http://localhost:9009/sku_search?keywords=手机



此时只能搜到华为手环设备

2.2 规格过滤

2.2.1 需求分析

规格这一部分，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：

`spechMap.规格名字.keyword`

2.2.2 代码实现

修改com.changgou.service.SearchServiceImpl的搜索方法，增加规格查询操作，代码如下：

代码如下：

```
@Override
public Map search(Map<String, String> searchMap) throws Exception {
    Map<String, Object> resultMap = new HashMap<>();

    //有条件才查询Es
    if (null != searchMap) {
        //组合条件对象
        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
        //0:关键词
        if (!StringUtils.isEmpty(searchMap.get("keywords"))) {
            boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));
        }
        //1:条件 品牌
        if (!StringUtils.isEmpty(searchMap.get("brand"))) {
            boolQuery.filter(QueryBuilders.termQuery("brandName",
searchMap.get("brand")));
        }

        //2:条件 规格
        for (String key : searchMap.keySet()) {
            if (key.startsWith("spec_")) {
                String value = searchMap.get(key).replace("%2B", "+");
                boolQuery.filter(QueryBuilders.termQuery("specMap." +
key.substring(5) + ".keyword",value));
            }
        }

        //4. 原生搜索实现类
        NativeSearchQueryBuilder nativeSearchQueryBuilder = new
NativeSearchQueryBuilder();
        nativeSearchQueryBuilder.withQuery(boolQuery);


        //6. 品牌聚合(分组)查询
        String skuBrand = "skuBrand";
 nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuBra
nd).field("brandName"));


        //7. 规格聚合(分组)查询
```

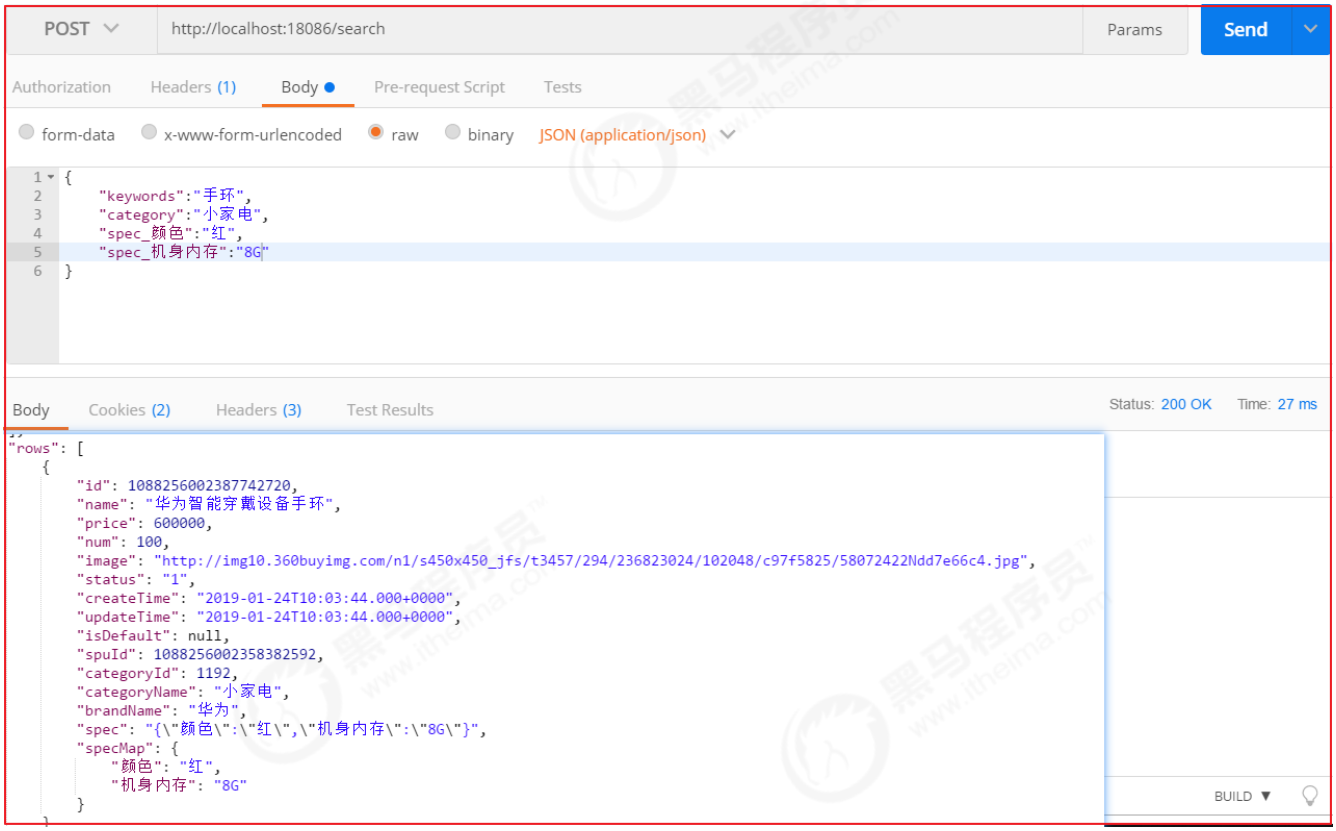
```
        String skuSpec = "skuSpec";
 nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuSpe
c).field("spec.keyword"));

        //10: 执行查询，返回结果对象
        AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
            @Override
            public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {

                List<T> list = new ArrayList<>();

                SearchHits hits = searchResponse.getHits();
                if (null != hits) {
                    for (SearchHit hit : hits) {
                        SkuInfo skuInfo =
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);

                        list.add((T) skuInfo);
                    }
                }
                return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
            }
        });

        //11. 总条数
        resultMap.put("total", aggregatedPage.getTotalElements());
        //12. 总页数
        resultMap.put("totalPages", aggregatedPage.getTotalPages());
        //13. 查询结果集合
        resultMap.put("rows", aggregatedPage.getContent());

        //14. 获取品牌聚合结果
        StringTerms brandTerms = (StringTerms)
aggregatedPage.getAggregation(skuBrand);



        List<String> brandList =
```

```
brandTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
        resultMap.put("brandList", brandList);

        //15. 获取规格聚合结果
        StringTerms specTerms = (StringTerms)
aggregatedPage.getAggregation(skuSpec);
        List<String> specList =
specTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
        resultMap.put("specList", specList(specList));

        return resultMap;
    }
    return null;
}

//处理规格集合
public Map<String, Set<String>> specList(List<String> specList) {

    Map<String, Set<String>> specMap = new HashMap<>();

    if (null != specList && specList.size() > 0) {

        for (String spec : specList) {

            Map<String, String> map = JSON.parseObject(spec, Map.class);
            Set<Map.Entry<String, String>> entries = map.entrySet();
            for (Map.Entry<String, String> entry : entries) {
                String key = entry.getKey();
                String value = entry.getValue();

                Set<String> specValues = specMap.get(key);
                if (null == specValues) {
                    specValues = new HashSet<>();
                }
                specValues.add(value);
                specMap.put(key, specValues);
            }
        }

    }
```

```
        return specMap;
    }
```

2.2.3 测试

访问地址：http://localhost:9009/sku_search?keywords=手机



2.3 价格区间查询

2.3.1 需求分析

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`。

2.3.2 代码实现

修改com.changgou.service.impl.SearchServiceImpl的搜索方法，增加价格区间查询操作，代码如下：

代码如下：

```
@Override
public Map search(Map<String, String> searchMap) throws Exception {
    Map<String, Object> resultMap = new HashMap<>();

    //有条件才查询Es
    if (null != searchMap) {
        //组合条件对象
        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
        //0:关键词
        if (!StringUtils.isEmpty(searchMap.get("keywords"))) {
            boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));
        }
        //1:条件 品牌
        if (!StringUtils.isEmpty(searchMap.get("brand"))) {
            boolQuery.filter(QueryBuilders.termQuery("brandName",
searchMap.get("brand")));
        }

        //2:条件 规格
        for (String key : searchMap.keySet()) {
            if (key.startsWith("spec_")) {
                String value = searchMap.get(key).replace("%2B", "+");
                boolQuery.filter(QueryBuilders.termQuery("specMap." +
key.substring(5) + ".keyword",value));
            }
        }
        //3:条件 价格
        if (StringUtils.isNotEmpty(searchMap.get("price"))) {
            String[] p = searchMap.get("price").split("-");
           if (p.length == 2) {

boolQuery.filter(QueryBuilders.rangeQuery("price").lte(p[1]));
            }
boolQuery.filter(QueryBuilders.rangeQuery("price").gte(p[0]));
        }

        //4. 原生搜索实现类


        NativeSearchQueryBuilder nativeSearchQueryBuilder = new
```

```
NativeSearchQueryBuilder();
        nativeSearchQueryBuilder.withQuery(boolQuery);

        //6. 品牌聚合(分组)查询
        String skuBrand = "skuBrand";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuBran
d).field("brandName"));

        //7. 规格聚合(分组)查询
        String skuSpec = "skuSpec";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuSpec
).field("spec.keyword"));

        //10: 执行查询, 返回结果对象
        AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
            @Override
            public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {

                List<T> list = new ArrayList<>();

                SearchHits hits = searchResponse.getHits();
                if (null != hits) {
                    for (SearchHit hit : hits) {
                        SkuInfo skuInfo =
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);

                        list.add((T) skuInfo);
                    }
                }
                return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
            }
        });

        //11. 总条数

        resultMap.put("total", aggregatedPage.getTotalElements());
```

```
        //12. 总页数
        resultMap.put("totalPages", aggregatedPage.getTotalPages());
        //13. 查询结果集合
        resultMap.put("rows", aggregatedPage.getContent());

        //14. 获取品牌聚合结果
        StringTerms brandTerms = (StringTerms)
aggregatedPage.getAggregation(skuBrand);
        List<String> brandList =
brandTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
        resultMap.put("brandList", brandList);

        //15. 获取规格聚合结果
        StringTerms specTerms = (StringTerms)
aggregatedPage.getAggregation(skuSpec);
        List<String> specList =
specTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
        resultMap.put("specList", specList(specList));

        return resultMap;
    }

    return null;
}
```

2.3.3 测试

访问地址：http://localhost:9009/sku_search?keywords=手机

POST http://localhost:18086/search Params Send

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)


```
{
    "keywords":"手环",
    "spec_颜色":"红",
    "price":"300-1000"
}
```


Body Cookies (2) Headers (3) Test Results Status: 200 OK Time: 28 ms

Pretty Raw Preview JSON


```
        {
            "id": 18374,
            "name": "小米",
            "image": "http://img10.360buyimg.com/popshop/jfs/t7084/169/439244907/4647/724c7958/598042c9N6e4e79e5.jpg",
            "letter": "X",
            "seq": null
        }
    ],
    "rows": [
        {
            "id": 1088256019328536576,
            "name": "守护宝幼儿安全手环",
            "price": 500,
            "num": 100,
            "image": "http://img10.360buyimg.com/n1/s450x450_jfs/t3457/294/236823024/102048/c97f5825/58072422Ndd7e66c4.jpg",
```


效果如下(部分数据)：

```
[
        {
            "id": 1088256019328536576,
            "name": "守护宝幼儿安全手环",
            "price": 500,
            "num": 100,
            "image":
"http://img10.360buyimg.com/n1/s450x450_jfs/t3457/294/236823024/102048/c97f5825/58072422Ndd7e66c4.jpg",
            "status": "1",
            "createTime": "2019-01-24T10:03:48.000+0000",
            "updateTime": "2019-01-24T10:03:48.000+0000",
            "isDefault": null,
            "spuId": 1088256019315953664,
            "categoryId": 1108,
            "categoryName": "户外工具",
            "brandName": "守护宝",
            "spec": "{\"颜色\":\"红\",\"机身内存\":\"64G\"}",
            "specMap": {
                "颜色": "红",
                "机身内存": "64G"
            }
        },
        {
            "id": 1088256014043713536,
            "name": "计步器小米手环，适用老人、小孩",
            "price": 800,
            "num": 100,
            "image":
"http://img10.360buyimg.com/n1/s450x450_jfs/t3457/294/236823024/102048/c97f5825/58072422Ndd7e66c4.jpg",
            "status": "1",
            "createTime": "2019-01-24T10:03:47.000+0000",
            "updateTime": "2019-01-24T10:03:47.000+0000",
            "isDefault": null,
            "spuId": 1088256014026936320,
            "categoryId": 1192,
            "categoryName": "小家电",
            "brandName": "小米",

            "spec": "{\"颜色\":\"红\",\"机身内存\":\"64G\"}",
```

```
            "specMap": {
                "颜色": "红",
                "机身内存": "64G"
            }
        }
    ]
```

3 搜索分页

3.1 分页分析

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。

3.2 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：

代码如下：

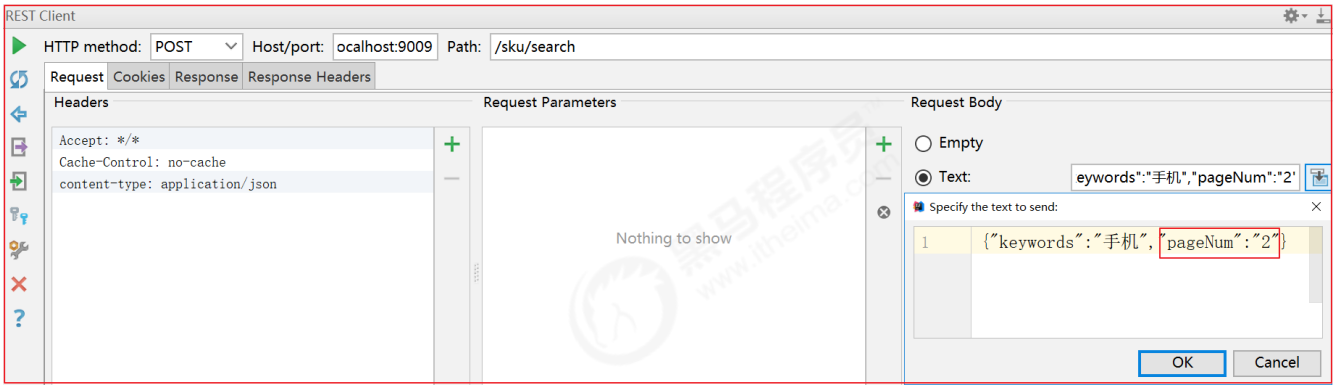
```
    @Override
    public Map search(Map<String, String> searchMap) throws Exception {
        Map<String, Object> resultMap = new HashMap<>();

        //有条件才查询Es
        if (null != searchMap) {
            //组合条件对象
            BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
            //0:关键词
            if (!StringUtils.isEmpty(searchMap.get("keywords"))) {
                boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));
            }
            //1:条件 品牌
            if (!StringUtils.isEmpty(searchMap.get("brand"))) {
                boolQuery.filter(QueryBuilders.termQuery("brandName",
searchMap.get("brand")));
            }

            //2:条件 规格
            for (String key : searchMap.keySet()) {
                if (key.startsWith("spec_")) {
                    String value = searchMap.get(key).replace("%2B",
"+");
                    boolQuery.filter(QueryBuilders.termQuery("specMap." +
key.substring(5) + ".keyword",value));
                }
            }
            //3:条件 价格
            if (!StringUtils.isEmpty(searchMap.get("price"))) {
                String[] p = searchMap.get("price").split("-");

boolQuery.filter(QueryBuilders.rangeQuery("price").gte(p[0]));
                if (p.length == 2) {

boolQuery.filter(QueryBuilders.rangeQuery("price").lte(p[1]));
                }
            }


            //4. 原生搜索实现类
```

```
            NativeSearchQueryBuilder nativeSearchQueryBuilder = new
NativeSearchQueryBuilder();
            nativeSearchQueryBuilder.withQuery(boolQuery);

            //6. 品牌聚合(分组)查询
            String skuBrand = "skuBrand";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuBran
d).field("brandName"));

            //7. 规格聚合(分组)查询
            String skuSpec = "skuSpec";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuSpec
).field("spec.keyword"));


            String pageNum = searchMap.get("pageNum");
            if (null == pageNum) {
                pageNum = "1";
            }
            //9: 分页

nativeSearchQueryBuilder.withPageable(PageRequest.of(Integer.parseInt(pag
eNum) - 1, Page.pageSize));

            //10: 执行查询, 返回结果对象
            AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
                @Override
                public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {

                    List<T> list = new ArrayList<>();

                    SearchHits hits = searchResponse.getHits();
                    if (null != hits) {
                        for (SearchHit hit : hits) {


                            SkuInfo skuInfo =
```

```
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);


                            list.add((T) skuInfo);
                        }
                    }
                    return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
                }
            });

            //11. 总条数
            resultMap.put("total", aggregatedPage.getTotalElements());
            //12. 总页数
            resultMap.put("totalPages", aggregatedPage.getTotalPages());
            //13. 查询结果集合
            resultMap.put("rows", aggregatedPage.getContent());

            //14. 获取品牌聚合结果
            StringTerms brandTerms = (StringTerms)
aggregatedPage.getAggregation(skuBrand);
            List<String> brandList =
brandTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
            resultMap.put("brandList", brandList);

            //15. 获取规格聚合结果
            StringTerms specTerms = (StringTerms)
aggregatedPage.getAggregation(skuSpec);
            List<String> specList =
specTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
            resultMap.put("specList", specList(specList));

            //16. 返回当前页
            resultMap.put("pageNum", pageNum);

            return resultMap;
        }


        return null;
```

```
    }
```

测试如下：



4 搜索排序

4.1 排序分析

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。

4.2 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：

```
    @Override
    public Map search(Map<String, String> searchMap) throws Exception {
        Map<String, Object> resultMap = new HashMap<>();

        //有条件才查询Es
        if (null != searchMap) {
            //组合条件对象
            BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
            //0:关键词
            if (!StringUtils.isEmpty(searchMap.get("keywords"))) {
                boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));
            }
            //1:条件 品牌
            if (!StringUtils.isEmpty(searchMap.get("brand"))) {
                boolQuery.filter(QueryBuilders.termQuery("brandName",
searchMap.get("brand")));
            }

            //2:条件 规格
            for (String key : searchMap.keySet()) {
                if (key.startsWith("spec_")) {
                    String value = searchMap.get(key).replace("%2B",
"+");
                    boolQuery.filter(QueryBuilders.termQuery("specMap." +
key.substring(5) + ".keyword",value));
                }
            }
            //3:条件 价格
            if (!StringUtils.isEmpty(searchMap.get("price"))) {
                String[] p = searchMap.get("price").split("-");

boolQuery.filter(QueryBuilders.rangeQuery("price").gte(p[0]));
                if (p.length == 2) {

boolQuery.filter(QueryBuilders.rangeQuery("price").lte(p[1]));
                }
            }


            //4. 原生搜索实现类
```

```
            NativeSearchQueryBuilder nativeSearchQueryBuilder = new
NativeSearchQueryBuilder();
            nativeSearchQueryBuilder.withQuery(boolQuery);

            //6. 品牌聚合(分组)查询
            String skuBrand = "skuBrand";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuBran
d).field("brandName"));

            //7. 规格聚合(分组)查询
            String skuSpec = "skuSpec";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuSpec
).field("spec.keyword"));

            //8: 排序
            if (!StringUtils.isEmpty(searchMap.get("sortField"))) {
                if ("ASC".equals(searchMap.get("sortRule"))) {

nativeSearchQueryBuilder.withSort(SortBuilders.fieldSort(searchMap.get("s
ortField")).order(SortOrder.ASC));
                } else {


nativeSearchQueryBuilder.withSort(SortBuilders.fieldSort(searchMap.get("s
ortField")).order(SortOrder.DESC));
                }

            }

            String pageNum = searchMap.get("pageNum");
            if (null == pageNum) {
                pageNum = "1";
            }
            //9: 分页

nativeSearchQueryBuilder.withPageable(PageRequest.of(Integer.parseInt(pag
eNum) - 1, Page.pageSize));


            //10: 执行查询，返回结果对象
```

```
            AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
                @Override
                public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {

                    List<T> list = new ArrayList<>();

                    SearchHits hits = searchResponse.getHits();
                    if (null != hits) {
                        for (SearchHit hit : hits) {
                            SkuInfo skuInfo =
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);

                            list.add((T) skuInfo);
                        }
                    }
                    return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
                }
            });

            //11. 总条数
            resultMap.put("total", aggregatedPage.getTotalElements());
            //12. 总页数
            resultMap.put("totalPages", aggregatedPage.getTotalPages());
            //13. 查询结果集合
            resultMap.put("rows", aggregatedPage.getContent());

            //14. 获取品牌聚合结果
            StringTerms brandTerms = (StringTerms)
aggregatedPage.getAggregation(skuBrand);
            List<String> brandList =
brandTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
            resultMap.put("brandList", brandList);

            //15. 获取规格聚合结果


            StringTerms specTerms = (StringTerms)
```

```
aggregatedPage.getAggregation(skuSpec);
            List<String> specList =
specTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
            resultMap.put("specList", specList(specList));

            //16. 返回当前页
            resultMap.put("pageNum", pageNum);

            return resultMap;
        }

        return null;
    }
```

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"}
```

5 高亮显示

5.1 高亮分析

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。

5.2 高亮搜索实现步骤解析

将之前的搜索换掉，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

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。

5.3 高亮代码实现

删掉之前com.changgou.service.impl.SearchServiceImpl的搜索方法搜索代码，用下面高亮搜索代码替换：

代码如下：

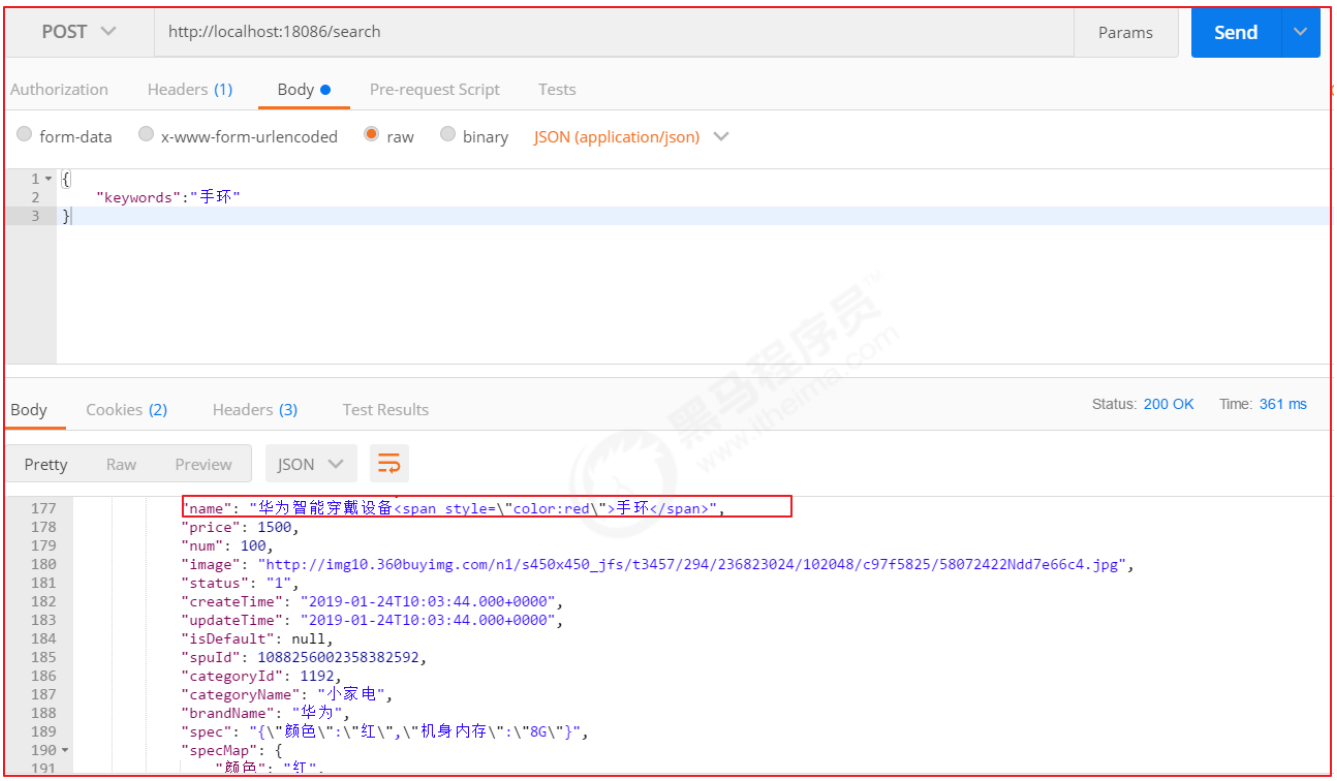
```
@Override
public Map search(Map<String, String> searchMap) throws Exception {
    Map<String, Object> resultMap = new HashMap<>();

    //有条件才查询Es
    if (null != searchMap) {
        //组合条件对象
        BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
        //0:关键词
        if (!StringUtils.isEmpty(searchMap.get("keywords"))) {
            boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));
        }
        //1:条件 品牌
        if (!StringUtils.isEmpty(searchMap.get("brand"))) {
            boolQuery.filter(QueryBuilders.termQuery("brandName",
searchMap.get("brand")));
        }

        //2:条件 规格
        for (String key : searchMap.keySet()) {
            if (key.startsWith("spec_")) {
                String value = searchMap.get(key).replace("%2B", "+");
                boolQuery.filter(QueryBuilders.termQuery("specMap." +
key.substring(5) + ".keyword",value));
            }
        }
        //3:条件 价格
        if (!StringUtils.isEmpty(searchMap.get("price"))) {
            String[] p = searchMap.get("price").split("-");

boolQuery.filter(QueryBuilders.rangeQuery("price").gte(p[0]));
            if (p.length == 2) {

boolQuery.filter(QueryBuilders.rangeQuery("price").lte(p[1]));
            }
        }

        //4. 原生搜索实现类

        NativeSearchQueryBuilder nativeSearchQueryBuilder = new
```

```
NativeSearchQueryBuilder();
        nativeSearchQueryBuilder.withQuery(boolQuery);

        //5:高亮
        HighlightBuilder.Field field = new HighlightBuilder
            .Field("name")
            .preTags("<span style='color:red'>")
            .postTags("</span>");
        nativeSearchQueryBuilder.withHighlightFields(field);

        //6. 品牌聚合(分组)查询
        String skuBrand = "skuBrand";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuBran
d).field("brandName"));

        //7. 规格聚合(分组)查询
        String skuSpec = "skuSpec";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuSpec
).field("spec.keyword"));

        //8: 排序
        if (!StringUtils.isEmpty(searchMap.get("sortField"))) {
            if ("ASC".equals(searchMap.get("sortRule"))) {

nativeSearchQueryBuilder.withSort(SortBuilders.fieldSort(searchMap.get("s
ortField")).order(SortOrder.ASC));
            } else {


nativeSearchQueryBuilder.withSort(SortBuilders.fieldSort(searchMap.get("s
ortField")).order(SortOrder.DESC));
            }

        }

        String pageNum = searchMap.get("pageNum");
        if (null == pageNum) {
            pageNum = "1";
        }
```

```
        //9: 分页

nativeSearchQueryBuilder.withPageable(PageRequest.of(Integer.parseInt(pag
eNum) - 1, Page.pageSize));

        //10: 执行查询，返回结果对象
        AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
            @Override
            public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {

                List<T> list = new ArrayList<>();

                SearchHits hits = searchResponse.getHits();
                if (null != hits) {
                    for (SearchHit hit : hits) {
                        SkuInfo skuInfo =
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);

                        Map<String, HighlightField> highlightFields =
hit.getHighlightFields();
                        if (null != highlightFields &&
highlightFields.size() > 0) {

skuInfo.setName(highlightFields.get("name").getFragments()
[0].toString());
                        }
                        list.add((T) skuInfo);
                    }
                }
                return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
            }
        });

        //11. 总条数
        resultMap.put("total", aggregatedPage.getTotalElements());
        //12. 总页数
        resultMap.put("totalPages", aggregatedPage.getTotalPages());
```

```
        //13. 查询结果集合
        resultMap.put("rows", aggregatedPage.getContent());

        //14. 获取品牌聚合结果
        StringTerms brandTerms = (StringTerms)
aggregatedPage.getAggregation(skuBrand);
        List<String> brandList =
brandTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
        resultMap.put("brandList", brandList);

        //15. 获取规格聚合结果
        StringTerms specTerms = (StringTerms)
aggregatedPage.getAggregation(skuSpec);
        List<String> specList =
specTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
        resultMap.put("specList", specList(specList));

        //16. 返回当前页
        resultMap.put("pageNum", pageNum);

        return resultMap;
    }

    return null;
}
```

5.4 测试



效果如下：

```
"name": "HTC M8Sd (E8) 波尔多红 电信4G<span style=\"color:red\">手机</span>双卡双待双通",
```

6 注：最终搜索业务代码如下

```
@Service
public class SearchServiceImpl implements SearchService {

    @Autowired
    private ElasticsearchTemplate esTemplate;



    //设置每页查询条数据
    public final static Integer PAGE_SIZE = 20;

    @Override
    public Map search(Map<String, String> searchMap) throws Exception {
        Map<String, Object> resultMap = new HashMap<>();

        //有条件才查询Es
        if (null != searchMap) {
            //组合条件对象
            BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
            //0:关键词
            if (!StringUtils.isEmpty(searchMap.get("keywords"))) {
                boolQuery.must(QueryBuilders.matchQuery("name",
searchMap.get("keywords")).operator(Operator.AND));
            }
            //1:条件 品牌
            if (!StringUtils.isEmpty(searchMap.get("brand"))) {
                boolQuery.filter(QueryBuilders.termQuery("brandName",
searchMap.get("brand")));
            }

            //2:条件 规格
            for (String key : searchMap.keySet()) {
                if (key.startsWith("spec_")) {
                    String value = searchMap.get(key).replace("%2B",
"+");
                    boolQuery.filter(QueryBuilders.termQuery("specMap." +
key.substring(5) + ".keyword",value));
                }
            }

            //3:条件 价格
```

```
            if (!StringUtils.isEmpty(searchMap.get("price"))) {
                String[] p = searchMap.get("price").split("-");

boolQuery.filter(QueryBuilders.rangeQuery("price").gte(p[0]));
                if (p.length == 2) {

boolQuery.filter(QueryBuilders.rangeQuery("price").lte(p[1]));
                }
            }

            //4. 原生搜索实现类
            NativeSearchQueryBuilder nativeSearchQueryBuilder = new
NativeSearchQueryBuilder();
            nativeSearchQueryBuilder.withQuery(boolQuery);

            //5:高亮
            HighlightBuilder.Field field = new HighlightBuilder
                    .Field("name")
                    .preTags("<span style='color:red'>")
                    .postTags("</span>");
            nativeSearchQueryBuilder.withHighlightFields(field);

            //6. 品牌聚合(分组)查询
            String skuBrand = "skuBrand";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuBran
d).field("brandName"));

            //7. 规格聚合(分组)查询
            String skuSpec = "skuSpec";

nativeSearchQueryBuilder.addAggregation(AggregationBuilders.terms(skuSpec
).field("spec.keyword"));

            //8: 排序
            if (!StringUtils.isEmpty(searchMap.get("sortField"))) {
                if ("ASC".equals(searchMap.get("sortRule"))) {

nativeSearchQueryBuilder.withSort(SortBuilders.fieldSort(searchMap.get("s
ortField")).order(SortOrder.ASC));

                } else {
```

```
nativeSearchQueryBuilder.withSort(SortBuilders.fieldSort(searchMap.get("s
ortField")).order(SortOrder.DESC));
                }

            }

            String pageNum = searchMap.get("pageNum");
            if (null == pageNum) {
                pageNum = "1";
            }
            //9: 分页

nativeSearchQueryBuilder.withPageable(PageRequest.of(Integer.parseInt(pag
eNum) - 1, Page.pageSize));

            //10: 执行查询，返回结果对象
            AggregatedPage<SkuInfo> aggregatedPage =
esTemplate.queryForPage(nativeSearchQueryBuilder.build(), SkuInfo.class,
new SearchResultMapper() {
                @Override
                public <T> AggregatedPage<T> mapResults(SearchResponse
searchResponse, Class<T> aClass, Pageable pageable) {

                    List<T> list = new ArrayList<>();

                    SearchHits hits = searchResponse.getHits();
                    if (null != hits) {
                        for (SearchHit hit : hits) {
                            SkuInfo skuInfo =
JSON.parseObject(hit.getSourceAsString(), SkuInfo.class);

                            Map<String, HighlightField> highlightFields =
hit.getHighlightFields();
                            if (null != highlightFields &&
highlightFields.size() > 0) {

skuInfo.setName(highlightFields.get("name").getFragments()
[0].toString());

                            }
```

```
                            list.add((T) skuInfo);
                        }
                    }
                    return new AggregatedPageImpl<T>(list, pageable,
hits.getTotalHits(), searchResponse.getAggregations());
                }
            });

            //11. 总条数
            resultMap.put("total", aggregatedPage.getTotalElements());
            //12. 总页数
            resultMap.put("totalPages", aggregatedPage.getTotalPages());
            //13. 查询结果集合
            resultMap.put("rows", aggregatedPage.getContent());

            //14. 获取品牌聚合结果
            StringTerms brandTerms = (StringTerms)
aggregatedPage.getAggregation(skuBrand);
            List<String> brandList =
brandTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
            resultMap.put("brandList", brandList);

            //15. 获取规格聚合结果
            StringTerms specTerms = (StringTerms)
aggregatedPage.getAggregation(skuSpec);
            List<String> specList =
specTerms.getBuckets().stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
            resultMap.put("specList", specList(specList));

            //16. 返回当前页
            resultMap.put("pageNum", pageNum);

            return resultMap;
        }

        return null;
    }
```

```
    //处理规格集合
    public Map<String, Set<String>> specList(List<String> specList) {

        Map<String, Set<String>> specMap = new HashMap<>();

        if (null != specList && specList.size() > 0) {

            for (String spec : specList) {

                Map<String, String> map = JSON.parseObject(spec,
Map.class);
                Set<Map.Entry<String, String>> entries = map.entrySet();
                for (Map.Entry<String, String> entry : entries) {
                    String key = entry.getKey();
                    String value = entry.getValue();

                    Set<String> specValues = specMap.get(key);
                    if (null == specValues) {
                        specValues = new HashSet<>();
                    }
                    specValues.add(value);
                    specMap.put(key, specValues);
                }
            }
        }
        return specMap;
    }

}
```