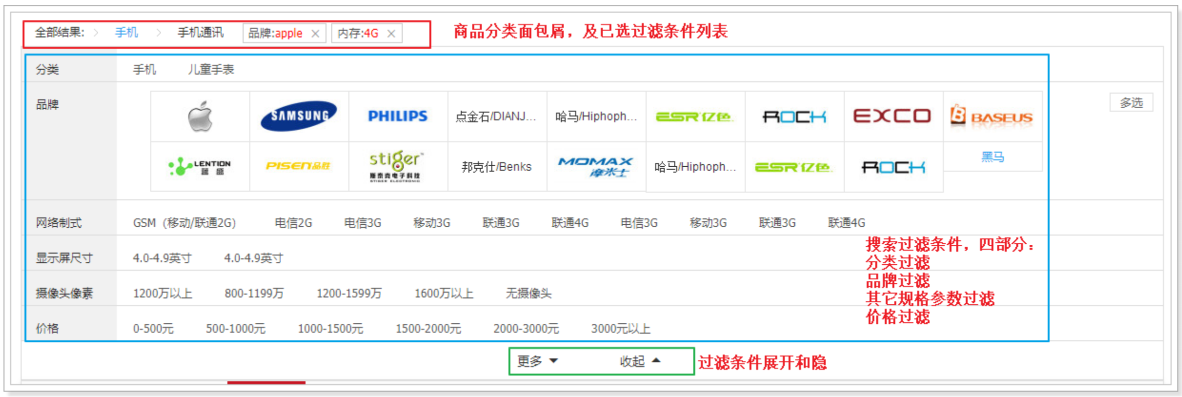
1 学习目标

- 了解过滤功能的基本思路
- 独立实现分类和品牌展示
- 了解规格参数展示
- 实现过滤条件筛选
- 实现已选过滤项回显
- 实现取消选择过滤项

2 过滤功能分析

首先看下页面要实现的效果：



全部结果: 手机 手机通讯 品牌:apple × 内存:4G ×
商品分类面包屑，及已选过滤条件列表
分类 手机 儿童手表
品牌
SAMSUNG PHILIPS 点金石/DIANJ... 哈马/Hiphoph... ESR亿色 ROCK EXCO BASEUS
LENTION PISEN品胜 stiger 邦克仕/Benks MOMAX摩米士 哈马/Hiphoph... ESR亿色 ROCK 黑马
多选
网络制式 GSM（移动/联通2G） 电信2G 电信3G 移动3G 联通3G 联通4G 电信3G 移动3G 联通3G 联通4G
显示屏尺寸 4.0-4.9英寸 4.0-4.9英寸
摄像头像素 1200万以上 800-1199万 1200-1599万 1600万以上 无摄像头
价格 0-500元 500-1000元 1000-1500元 1500-2000元 2000-3000元 3000元以上
搜索过滤条件，四部分：
分类过滤
品牌过滤
其它规格参数过滤
价格过滤
更多 收起
过滤条件展开和隐

整个过滤部分有3块：

- 顶部的导航，已经选择的过滤条件展示：
  - 商品分类面包屑，根据用户选择的商品分类变化
  - 其它已选择过滤参数
- 过滤条件展示，又包含3部分
  - 商品分类展示
  - 品牌展示
  - 其它规格参数
- 展开或收起的过滤条件的按钮

顶部导航要展示的内容跟用户选择的过滤条件有关。

- 比如用户选择了某个商品分类，则面包屑中才会展示具体的分类
- 比如用户选择了某个品牌，列表中才会有品牌信息。

所以，这部分需要依赖第二部分：过滤条件的展示和选择。因此我们先不着急去做。

展开或收起的按钮是否显示，取决于过滤条件现在有多少，如果有很多，那么就没必要展示。所以也是跟第二部分的过滤条件有关。

这样分析来看，我们必须先做第二部分：过滤条件展示。

3 生成分类和品牌 筛选条件

先来看分类和品牌。在我们的数据库中已经有所有的分类和品牌信息。在这个位置，是不是把所有的分类和品牌信息都展示出来呢?

显然不是，用户搜索的条件会对商品进行过滤，而在搜索结果中，不一定包含所有的分类和品牌，直接展示出所有商品分类，让用户选择显然是不合适的。

无论是分类信息，还是品牌信息，都应该从e s搜索的结果商品中进行聚合得到。

3.1 聚合商品分类和品牌

我们修改搜索的业务逻辑，对分类和品牌聚合。

因为索引库中只有id，所以我们根据id聚合，然后再根据id去查询完整数据。

3.2 实现

3.2.1 brandService和categoryService分别提供通过ids查询数据的接口

3.2.1.1 BrandService

```
    @ApiOperation(value="通过品牌id集合获取品牌")
    @GetMapping(value = "brand/getBrandByIds")
    Result<List<BrandEntity>> getBrandByIds(@RequestParam String brandIds);
```

3.2.1.2 CategoryService

```
    @ApiOperation(value = "通过id集合查询分类信息")
    @GetMapping(value = "category/getCateByIds")
    Result<List<CategoryEntity>> getCateByIds(@RequestParam String cateIds);
```

3.2.1.3 BrandServiceImpl

```
    @Override
    public Result<List<BrandEntity>> getBrandByIds(String brandIds) {

        List<Integer> brandIdsArr = Arrays.asList(brandIds.split(","))
                .stream().map(idStr ->
Integer.parseInt(idStr)).collect(Collectors.toList());
        List<BrandEntity> list = brandMapper.selectByIdList(brandIdsArr);
        return this.setResultSuccess(list);
    }
```

3.2.1.4 CategoryServiceImpl

```
    @Override
    public Result<List<CategoryEntity>> getCateByIds(String cateIds) {

        List<Integer> cateIdsArr = Arrays.asList(cateIds.split(","))
                .stream().map(idStr ->
Integer.parseInt(idStr)).collect(Collectors.toList());

        List<CategoryEntity> list = categoryMapper.selectByIdList(cateIdsArr);
        return this.setResultSuccess(list);
    }
```

3.2.2 提供扩展的response

原来我们查询的方法的返回值是Result,result里面封装了code,msg,和data 等信息

还需要total,totalPage(放在了meaage字段中)

现在还需要品牌和分类的信息,显然result已经支撑不住了

所以我们扩展一下result,顺便让大家继续学一下继承

3.2.2.1 mingrui-shop-service-api-search/pom.xml

```
        <!--需要api-xxx中的一些类-->
        <dependency>
            <groupId>com.baidu</groupId>
            <artifactId>mingrui-shop-service-api-xxx</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
```

3.2.2.2 api-search新建response包并新建GoodsResponse

```
@Data
@NoArgsConstructor
public class GoodsResponse extends Result<List<GoodsDoc>> {

    private Integer total;

    private Integer totalPage;

    private List<BrandEntity> brandList;

    private List<CategoryEntity> categoryList;

    public GoodsResponse(Integer total, Integer totalPage, List<BrandEntity>
brandList, List<CategoryEntity> categoryList, List<GoodsDoc> goodsDocs){

        super(HTTPStatus.OK,HTTPStatus.OK + "",goodsDocs);
        this.total = total;
        this.totalPage = totalPage;
        this.brandList = brandList;
        this.categoryList = categoryList;
    }

}
```

3.2.3 service-search项目新建BrandFeign和CategoryFeign

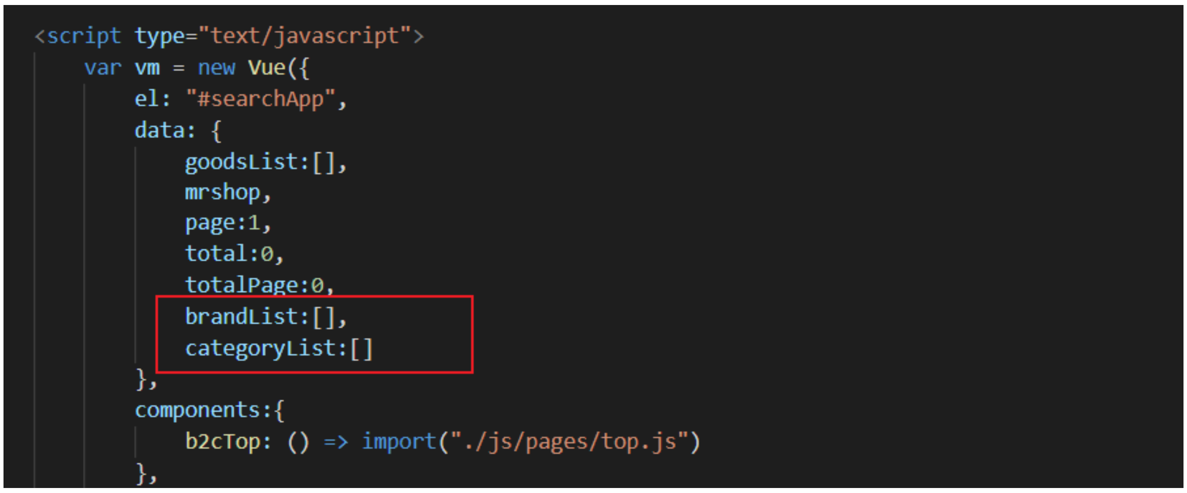
```
@FeignClient(contextId = "BrandService", value = "xxx-service")
public interface BrandFeign extends BrandService {
}
```

```
@FeignClient(contextId = "CategoryService", value = "xxx-service")
public interface CategoryFeign extends CategoryService {
}
```

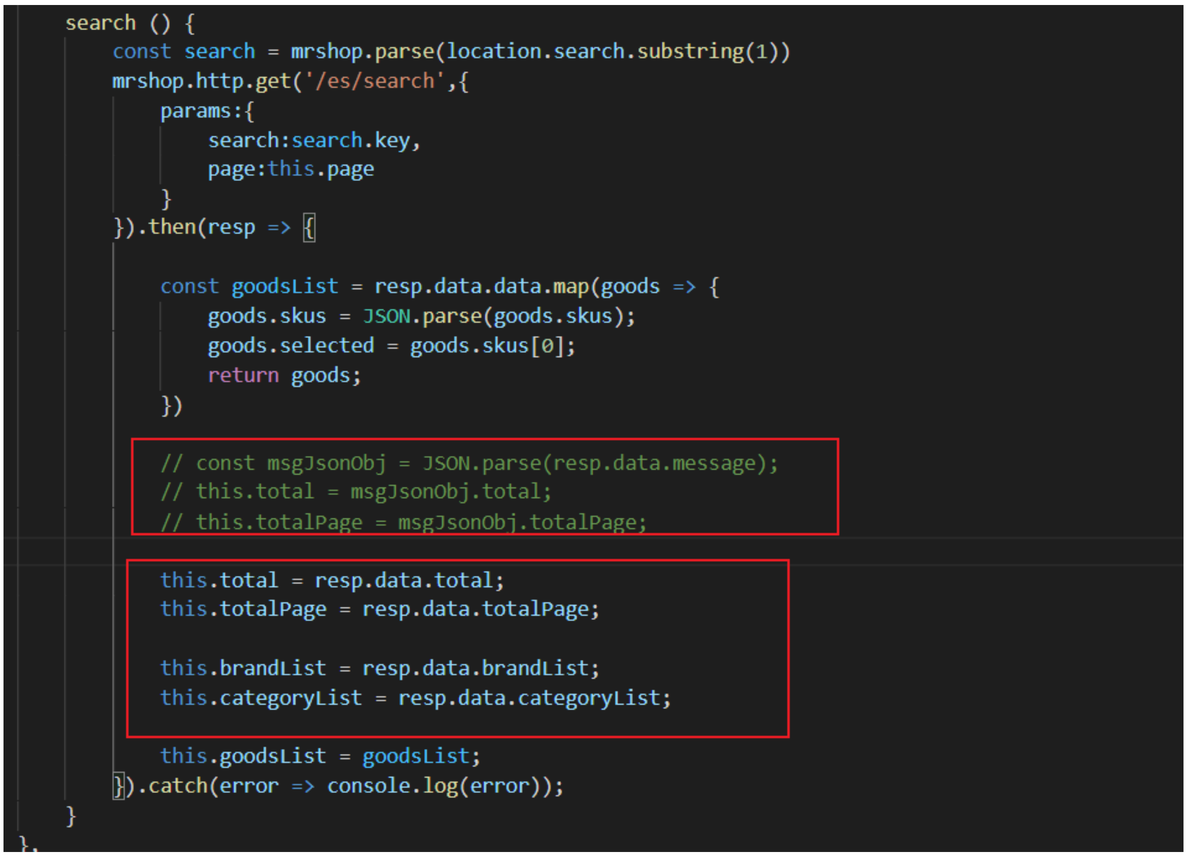
3.2.4 ShopElasticsearchServiceImpl

```
    @Override
    public GoodsResponse search(String search, Integer page) {

        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();

        if (!StringUtils.isEmpty(search)) {
            //多字段同时查询

 queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search,"title","brandName"
,"categoryName"));
        }

        queryBuilder.withPageable(PageRequest.of(page-1,10));

        //设置查询出来的内容,页面上做多只需要id,title,skus
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]
{"id","title","skus"}, null));
        //设置高亮字段

 queryBuilder.withHighlightBuilder(ESHighLightUtil.getHighlightBuilder("title"))
;

        //聚合

 queryBuilder.addAggregation(AggregationBuilders.terms("cate_agg").field("cid3")
);

 queryBuilder.addAggregation(AggregationBuilders.terms("brand_agg").field("brand
Id"));

        SearchHits<GoodsDoc> hits =
elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);

        List<SearchHit<GoodsDoc>> highLightHit =
ESHighLightUtil.getHighLightHit(hits.getSearchHits());

        List<GoodsDoc> goodsDocs = highLightHit.stream().map(searchHit ->
searchHit.getContent()).collect(Collectors.toList());

//        long total = hits.getTotalHits();//总条数 47
//        Double totalD = Long.valueOf(total).doubleValue();//doule类型的总条数
//        double totalPageD = Math.ceil(totalD ;//如果有小数直接想上取整
//         int totalPage = Double.valueOf(totalPageD).intValue();//将double类型的值
转为int类型

        //获取聚合数据
        Aggregations aggregations = hits.getAggregations();
        Terms brand_agg = aggregations.get("brand_agg");
        Terms cate_agg = aggregations.get("cate_agg");

        List<? extends Terms.Bucket> brandBuckets = brand_agg.getBuckets();
        List<String> brandIdList = brandBuckets.stream().map(brandbuckt -> {
            Number keyAsNumber = brandbuckt.getKeyAsNumber();
            Integer brandId = Integer.valueOf(keyAsNumber.intValue());
            return brandId + "";//得到品牌id,并且且转为String类型,方便接下来的操作
        }).collect(Collectors.toList());

        List<? extends Terms.Bucket> cateBuckets = cate_agg.getBuckets();
        List<String> cateIdList = cateBuckets.stream().map(cateBucket -> {
            Number keyAsNumber = cateBucket.getKeyAsNumber();
            Integer cateId = Integer.valueOf(keyAsNumber.intValue());

            return cateId + "";
        }).collect(Collectors.toList());

        //通过brandid获取brand详细数据
        //String.join(分隔符,List<String>),将list集合转为,分隔的字符串
        Result<List<BrandEntity>> brandResult =
brandFeign.getBrandByIds(String.join(",",brandIdList));

        //通过分类id获取分类详细数据
        Result<List<CategoryEntity>> cateResult =
categoryFeign.getCateByIds(String.join(",",cateIdList));

        /*Map<String, Integer> map = new HashMap<>();
        map.put("total",Long.valueOf(hits.getTotalHits()).intValue());

 map.put("totalPage",Double.valueOf(Math.ceil(Long.valueOf(hits.getTotalHits()).
doubleValue()/ 10)).intValue());

        String message = JSONUtil.toJsonString(map);*/

        GoodsResponse goodsResponse = new
GoodsResponse(Long.valueOf(hits.getTotalHits()).intValue()
                ,
Double.valueOf(Math.ceil(Long.valueOf(hits.getTotalHits()).doubleValue() /
10)).intValue()
                , brandResult.getData(), cateResult.getData(), goodsDocs);

        return goodsResponse;
    }
```

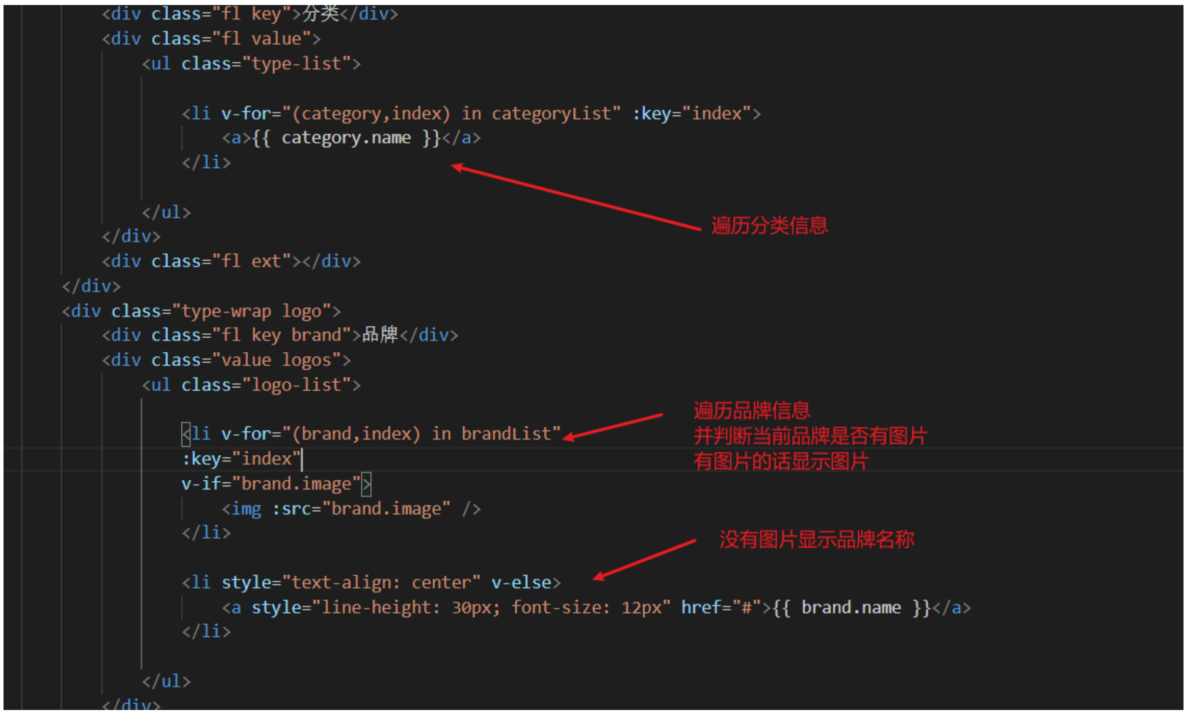
3.2.5 search.html



```
<script type="text/javascript">
    var vm = new Vue({
        el: "#searchApp",
        data: {
            goodsList:[],
            mrshop,
            page:1,
            total:0,
            totalPage:0,
            brandList:[],
            categoryList:[]
        },
        components:{
            b2cTop: () => import("./js/pages/top.js")
        },
```



```
    search () {
        const search = mrshop.parse(location.search.substring(1))
        mrshop.http.get('/es/search',{
            params:{
                search:search.key,
                page:this.page
            }
        }).then(resp => {

            const goodsList = resp.data.data.map(goods => {
                goods.skus = JSON.parse(goods.skus);
                goods.selected = goods.skus[0];
                return goods;
            })

            // const msgJsonObj = JSON.parse(resp.data.message);
            // this.total = msgJsonObj.total;
            // this.totalPage = msgJsonObj.totalPage;

            this.total = resp.data.total;
            this.totalPage = resp.data.totalPage;

            this.brandList = resp.data.brandList;
            this.categoryList = resp.data.categoryList;

            this.goodsList = goodsList;
        }).catch(error => console.log(error));
    }
},
```

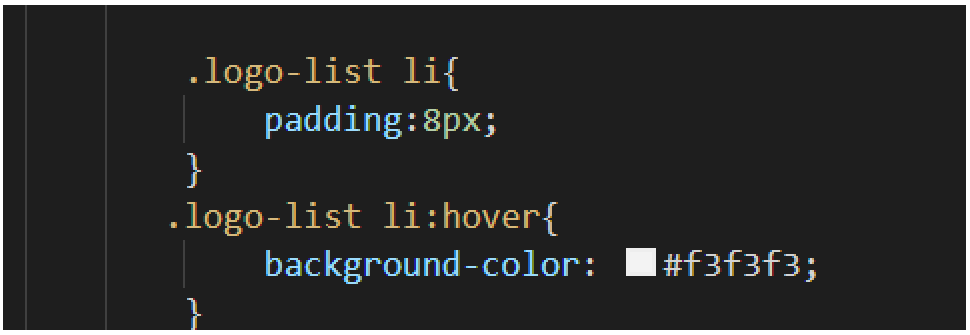


```
        <div class="fl key">分类</div>
        <div class="fl value">
            <ul class="type-list">

                <li v-for="(category,index) in categoryList" :key="index">
                    <a>{{ category.name }}</a>
                </li>

            </ul>
        </div>
        <div class="fl ext"></div>
    </div>
    <div class="type-wrap logo">
        <div class="fl key brand">品牌</div>
        <div class="value logos">
            <ul class="logo-list">

                <li v-for="(brand,index) in brandList"
                :key="index"
                v-if="brand.image">
                    <img :src="brand.image" />
                </li>

                <li style="text-align: center" v-else>
                    <a style="line-height: 30px; font-size: 12px" href="#">{{ brand.name }}</a>
                </li>

            </ul>
        </div>
```

```
                  searchEsData() {
                      const search = mrshop.parse(location.search.substring(1));//
将key=value转为key:value
                      //查询请求
                      mrshop.http.get('es/search', {
                          params: {
                              search: search.key,//将查询的内容传递到后台
                              page: this.page//当前页
                          }
                      }).then(resp => {

                          //处理sku
                          const goodsList = resp.data.data.map(goods => {

                              goods.skus = JSON.parse(goods.skus);//将字符串转为json
                              goods.selected = goods.skus[0];//设置当前选中
                              return goods;
                          })

                          this.total = resp.data.total;//总条数
                          this.totalPage = resp.data.totalPage//总页数

                          this.brandList = resp.data.brandList;
                          this.categoryList = resp.data.categoryList;
                          //var totalObj = JSON.parse(resp.data.message);

                          //this.total = totalObj.obj;//总条数
                          //this.totalPage = totalObj.totalPage//总页数
                          this.goodsList = goodsList;

                      }).catch(error => console.log(error))
```

```
                      <div class="type-wrap">
                          <div class="fl key">分类</div>
                          <div class="fl value">
                              <ul class="type-list">
                                  <!--遍历分类信息-->
                                  <li v-for="(category,index) in categoryList"
:key="index">
                                      <a>{{ category.name }}</a>
                                  </li>
                              </ul>
                          </div>
                          <div class="fl ext"></div>
                      </div>
                      <div class="type-wrap logo">
                          <div class="fl key brand">品牌</div>
                          <div class="value logos">
                              <ul class="logo-list">
                                  <!--遍历品牌信息
                                  有图片的显示图片-->
                                  <li
                                  v-for="(brand,index) in brandList"
                                  v-if="brand.image"
                                  :key="index">
                                      <img :src="brand.image" />
```

```
                                        </li>
                                        <!--没有图片的显示文字-->
                                        <li v-else style="text-align: center"><a
style="line-height: 30px; font-size: 12px"
                                                href="#">{{ brand.name }}</a></li>
                                    </ul>
                                </div>
                                <div class="fl ext">
                                    <a href="javascript:void(0);" class="sui-btn">多选
</a>
                                </div>
                            </div>
```



```
    .logo-list li{
        padding:8px;
    }
   .logo-list li:hover{
        background-color: #f3f3f3;
    }
```

3.3 问题

当前程序是做完了,好像也并没有什么问题

但是我可以告诉你们一个非常大的问题

用户体验

现在我们是本地开发,如果项目上线的话

当前的搜索效率有点慢,尤其是加上聚合后(我们还有一个规格没有加上呢)

所以我们拆一下方法,说白了就是拆接口,分发请求

拆请求的话会增加大家电脑的压力,大家看一下代码就可以了,不需要实现

3.3.1 ShopElasticsearchService

```
    @ApiOperation(value = "搜索")
    @GetMapping(value = "es/search")
    Result<List<GoodsDoc>> search(@RequestParam String search, @RequestParam
Integer page);

    @ApiOperation(value = "品牌过滤")
    @GetMapping(value = "es/searchBrand")
    Result<List<BrandEntity>> getBrandInfo(String search);

    @ApiOperation(value = "分类过滤")
    @GetMapping(value = "es/searchCategory")
    Result<List<CategoryEntity>> getCategoryInfo(String search);
```

3.3.2 ShopElasticsearchServiceImpl

```
    @Override
    public Result<List<GoodsDoc>> search(String search, Integer page) {

        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();

        if (!StringUtils.isEmpty(search)) {
            //多字段同时查询

 queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search,"title","brandName"
,"categoryName"));
        }

        queryBuilder.withPageable(PageRequest.of(page-1,10));

        //设置查询出来的内容,页面上做多只需要id,title,skus
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]
{"id","title","skus"}, null));
        //设置高亮字段

 queryBuilder.withHighlightBuilder(ESHighLightUtil.getHighlightBuilder("title"))
;

        SearchHits<GoodsDoc> hits =
elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);

        List<SearchHit<GoodsDoc>> highLightHit =
ESHighLightUtil.getHighLightHit(hits.getSearchHits());

        List<GoodsDoc> goodsDocs = highLightHit.stream().map(searchHit ->
searchHit.getContent()).collect(Collectors.toList());


        Map<String, Integer> map = new HashMap<>();
        map.put("total",Long.valueOf(hits.getTotalHits()).intValue());

 map.put("totalPage",Double.valueOf(Math.ceil(Long.valueOf(hits.getTotalHits()).
doubleValue()/ 10)).intValue());

        String message = JSONUtil.toJsonString(map);

        return this.setResult(HTTPStatus.OK,message,goodsDocs);
    }

    @Override
    public Result<List<BrandEntity>> getBrandInfo(String search){
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();

        if (!StringUtils.isEmpty(search)) {
            //多字段同时查询

 queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search,"title","brandName"
,"categoryName"));
        }

 queryBuilder.addAggregation(AggregationBuilders.terms("brand_agg").field("brand
Id"));

        SearchHits<GoodsDoc> hits =
elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);
```

```
        Aggregations aggregations = hits.getAggregations();
        Terms brand_agg = aggregations.get("brand_agg");

        List<? extends Terms.Bucket> brandBuckets = brand_agg.getBuckets();
        List<String> brandIdList = brandBuckets.stream().map(brandbuckt -> {
            Number keyAsNumber = brandbuckt.getKeyAsNumber();
            Integer brandId = Integer.valueOf(keyAsNumber.intValue());
            return brandId + "";//得到品牌id,并且且转为String类型,方便接下来的操作
        }).collect(Collectors.toList());

        Result<List<BrandEntity>> brandResult =
brandFeign.getBrandByIds(String.join(",",brandIdList));

        return this.setResultSuccess(brandResult.getData());
    }

    @Override
    public Result<List<CategoryEntity>> getCategoryInfo(String search){
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();

        if (!StringUtils.isEmpty(search)) {
            //多字段同时查询

 queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search,"title","brandName"
,"categoryName"));
        }

 queryBuilder.addAggregation(AggregationBuilders.terms("cate_agg").field("cid3")
);

        SearchHits<GoodsDoc> hits =
elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);

        Aggregations aggregations = hits.getAggregations();
        Terms cate_agg = aggregations.get("cate_agg");

        List<? extends Terms.Bucket> cateBuckets = cate_agg.getBuckets();
        List<String> cateIdList = cateBuckets.stream().map(cateBucket -> {
            Number keyAsNumber = cateBucket.getKeyAsNumber();
            Integer cateId = Integer.valueOf(keyAsNumber.intValue());

            return cateId + "";
        }).collect(Collectors.toList());

        Result<List<CategoryEntity>> cateResult =
categoryFeign.getCateByIds(String.join(",",cateIdList));

        return this.setResultSuccess(cateResult.getData());
    }
```

3.3.3 search.html

```
                searchEsData() {
                    const search = mrshop.parse(location.search.substring(1));//
将key=value转为key:value
```

```
                    //查询请求
                    mrshop.http.get('es/search', {
                        params: {
                            search: search.key,//将查询的内容传递到后台
                            page: this.page//当前页
                        }
                    }).then(resp => {
                        //处理sku
                        const goodsList = resp.data.data.map(goods => {

                            goods.skus = JSON.parse(goods.skus);//将字符串转为json
                            goods.selected = goods.skus[0];//设置当前选中
                            return goods;
                        })

                        // this.brandList = resp.data.brandList;
                        // this.categoryList = resp.data.categoryList;

                        var totalObj = JSON.parse(resp.data.message);
                        this.total = totalObj.obj;//总条数
                        this.totalPage = totalObj.totalPage//总页数

                        this.goodsList = goodsList;

                    }).catch(error => console.log(error));
                    //查询品牌信息
                    mrshop.http.get('es/searchBrand', {
                        params: {
                            search: search.key
                        }
                    }).then(resp => {
                        this.brandList = resp.data.data;
                    }).catch(error => console.log(error));

                    //查询分类信息
                    mrshop.http.get('es/searchCategory', {
                        params: {
                            search: search.key
                        }
                    }).then(resp => {
                        this.categoryList = resp.data.data;
                    }).catch(error => console.log(error))
                }
```

3.4 拆方法

search这个方法代码太多了......

拆方法还是遵循一个原则-从后往前拆

```
    @Override
    public GoodsResponse search(String search, Integer page) {

        NativeSearchQueryBuilder queryBuilder = this.getQueryBuilder(search,
page);
        SearchHits<GoodsDoc> hits =
elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);
```

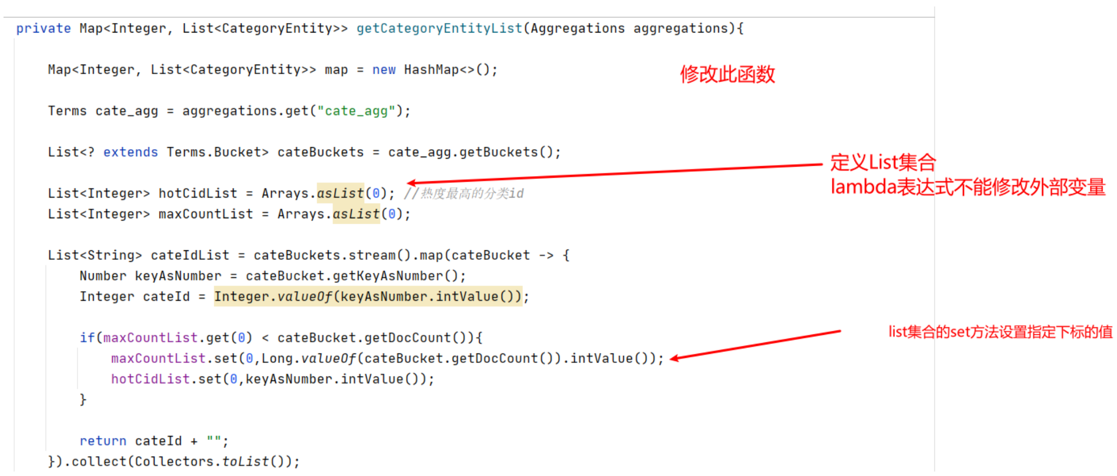
```
        List<SearchHit<GoodsDoc>> highLightHit =
ESHighLightUtil.getHighLightHit(hits.getSearchHits());
        List<GoodsDoc> goodsDocs = highLightHit.stream().map(searchHit ->
searchHit.getContent()).collect(Collectors.toList());

        //通过品牌id获取品牌详细数据
        List<BrandEntity> brandResult =
getBrandEntityList(hits.getAggregations());
        //通过分类id获取分类详细数据
        List<CategoryEntity> cateResult =
getCategoryEntityList(hits.getAggregations());

        GoodsResponse goodsResponse = new
GoodsResponse(Long.valueOf(hits.getTotalHits()).intValue()
                ,
Double.valueOf(Math.ceil(Long.valueOf(hits.getTotalHits()).doubleValue() /
10)).intValue()
                , brandResult, cateResult, goodsDocs);

        return goodsResponse;
    }

    private NativeSearchQueryBuilder getQueryBuilder(String search, Integer
page){
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();

        if (!StringUtils.isEmpty(search)) {
            //多字段同时查询

 queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search,"title","brandName"
,"categoryName"));
        }

        queryBuilder.withPageable(PageRequest.of(page-1,10));
        //设置查询出来的内容,页面上做多只需要id,title,skus
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]
{"id","title","skus"}, null));
        //设置高亮字段

 queryBuilder.withHighlightBuilder(ESHighLightUtil.getHighlightBuilder("title"))
;

        //聚合

 queryBuilder.addAggregation(AggregationBuilders.terms("cate_agg").field("cid3")
);

 queryBuilder.addAggregation(AggregationBuilders.terms("brand_agg").field("brand
Id"));

        return queryBuilder;
    }

    private List<BrandEntity> getBrandEntityList(Aggregations aggregations){
        Terms brand_agg = aggregations.get("brand_agg");

        List<? extends Terms.Bucket> brandBuckets = brand_agg.getBuckets();
        List<String> brandIdList = brandBuckets.stream().map(brandbuckt -> {
            Number keyAsNumber = brandbuckt.getKeyAsNumber();
            Integer brandId = Integer.valueOf(keyAsNumber.intValue());
            return brandId + "";//得到品牌id,并且且转为String类型,方便接下来的操作
        }).collect(Collectors.toList());

        Result<List<BrandEntity>> brandResult =
brandFeign.getBrandByIds(String.join(",",brandIdList));
        return brandResult.getData();
    }

    private List<CategoryEntity> getCategoryEntityList(Aggregations
aggregations){

        Terms cate_agg = aggregations.get("cate_agg");

        List<? extends Terms.Bucket> cateBuckets = cate_agg.getBuckets();
        List<String> cateIdList = cateBuckets.stream().map(cateBucket -> {
            Number keyAsNumber = cateBucket.getKeyAsNumber();
            Integer cateId = Integer.valueOf(keyAsNumber.intValue());

            return cateId + "";
        }).collect(Collectors.toList());

        Result<List<CategoryEntity>> cateResult =
categoryFeign.getCateByIds(String.join(",",cateIdList));

        return cateResult.getData();
    }
```

4 生成规格 筛选条件

```
GET /goods/_search
{
  "query": {
    "multi_match": {
      "query": "华为",
      "fields": ["title","brandName","categoryName"]
    }
  },
  "aggs": {
    "屏幕尺寸": {
      "terms": {
        "field": "specs.主屏幕尺寸（英寸）.keyword",
        "size": 10
      }
    },
    "内存":{
      "terms": {
        "field": "specs.内存.keyword",
        "size": 10
      }
    }
  }
}
```

> 问题：我们不可能将所有的规格参数全部取出来,规格参数是挂在某一个分类下的,
>
> 所以我们应该根据具体的分类id查询相应的规格参数
>
> 那我们怎么确定获取哪个分类下的id?
>
> 其实这个取决于公司,就比如说,我们可以查询热度最高的分类下的规格参数.
>
> 也可以查询分类下商品最多的分类下的规格参数
>
> 但是在学习阶段,热度最高我们现在没有办法实现
>
> 可以得到分类下商品最多的分类

4.1 修改获取分类信息的方法获取热度最高的分类



```
private Map<Integer, List<CategoryEntity>> getCategoryEntityList(Aggregations aggregations){

    Map<Integer, List<CategoryEntity>> map = new HashMap<>();

    Terms cate_agg = aggregations.get("cate_agg");

    List<? extends Terms.Bucket> cateBuckets = cate_agg.getBuckets();

    List<Integer> hotCidList = Arrays.asList(0); //热度最高的分类id
    List<Integer> maxCountList = Arrays.asList(0);

    List<String> cateIdList = cateBuckets.stream().map(cateBucket -> {
        Number keyAsNumber = cateBucket.getKeyAsNumber();
        Integer cateId = Integer.valueOf(keyAsNumber.intValue());

        if(maxCountList.get(0) < cateBucket.getDocCount()){
            maxCountList.set(0,Long.valueOf(cateBucket.getDocCount()).intValue());
            hotCidList.set(0,keyAsNumber.intValue());
        }

        return cateId + "";
    }).collect(Collectors.toList());
```

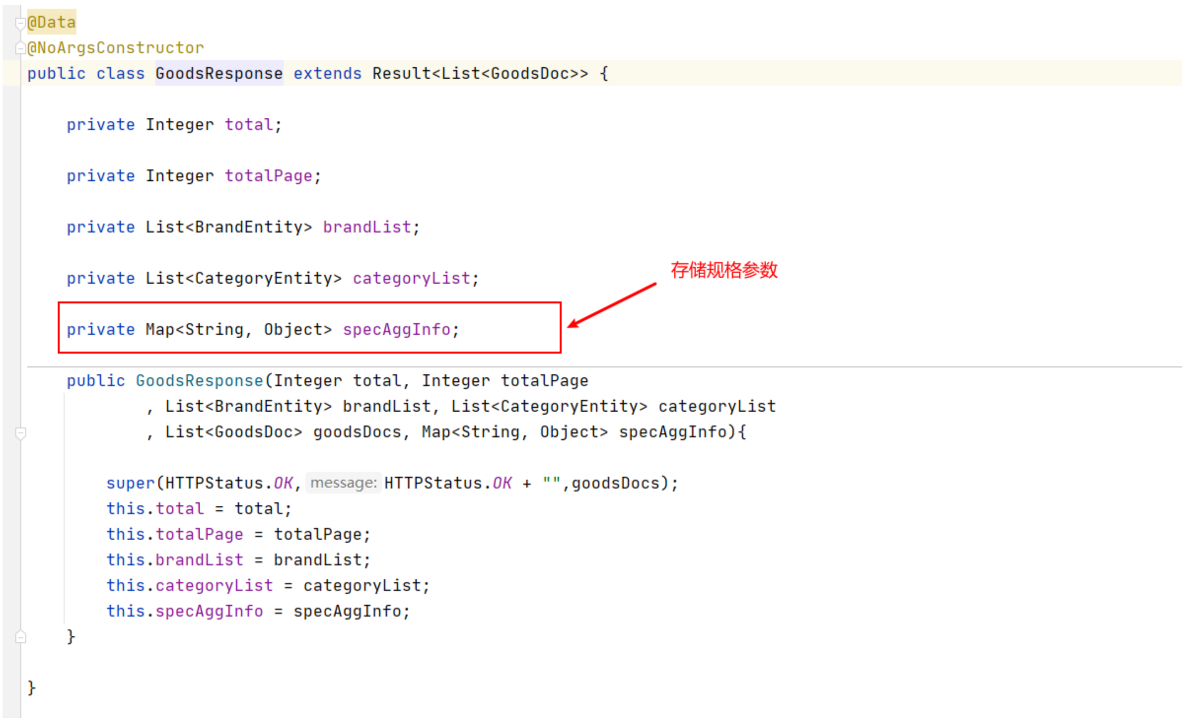
修改此函数
定义List集合
lambda表达式不能修改外部变量
list集合的set方法设置指定下标的值

热度最高的id为key

值为所有分类的数据

```
    private Map<Integer, List<CategoryEntity>>
getCategoryEntityList(Aggregations aggregations){

        Map<Integer, List<CategoryEntity>> map = new HashMap<>();

        Terms cate_agg = aggregations.get("cate_agg");

        List<? extends Terms.Bucket> cateBuckets = cate_agg.getBuckets();

        List<Integer> hotCidList = Arrays.asList(0); //热度最高的分类id
        List<Integer> maxCountList = Arrays.asList(0);

        List<String> cateIdList = cateBuckets.stream().map(cateBucket -> {
            Number keyAsNumber = cateBucket.getKeyAsNumber();
            Integer cateId = Integer.valueOf(keyAsNumber.intValue());

            if(maxCountList.get(0) < cateBucket.getDocCount()){

 maxCountList.set(0,Long.valueOf(cateBucket.getDocCount()).intValue());
                hotCidList.set(0,keyAsNumber.intValue());
            }

            return cateId + "";
        }).collect(Collectors.toList());
        Result<List<CategoryEntity>> cateResult =
categoryFeign.getCateByIds(String.join(",",cateIdList));

        map.put(hotCidList.get(0),cateResult.getData());//key为热度最高的cid value
为cid集合对应的数据

        return map;
    }
```

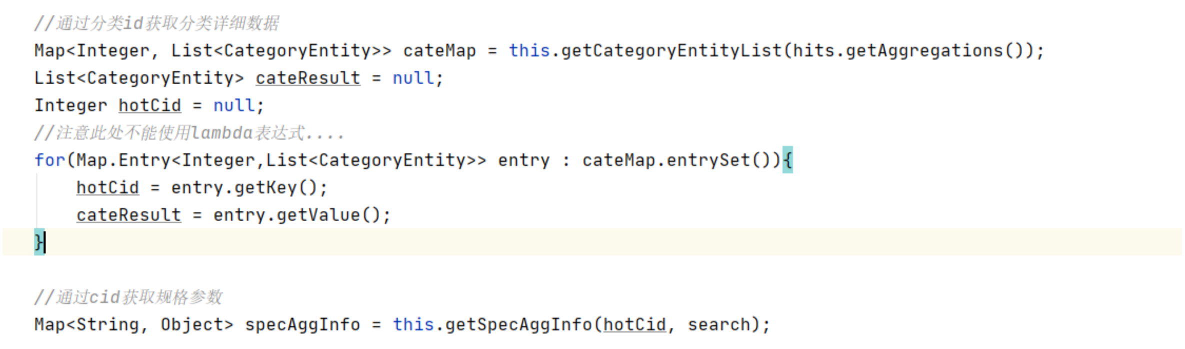
4.2 response中新增规格参数属性



```
@Data
@NoArgsConstructor
public class GoodsResponse extends Result<List<GoodsDoc>> {

    private Integer total;

    private Integer totalPage;

    private List<BrandEntity> brandList;

    private List<CategoryEntity> categoryList;

    private Map<String, Object> specAggInfo;

    public GoodsResponse(Integer total, Integer totalPage
            , List<BrandEntity> brandList, List<CategoryEntity> categoryList
            , List<GoodsDoc> goodsDocs, Map<String, Object> specAggInfo){

        super(HTTPStatus.OK, message: HTTPStatus.OK + "",goodsDocs);
        this.total = total;
        this.totalPage = totalPage;
        this.brandList = brandList;
        this.categoryList = categoryList;
        this.specAggInfo = specAggInfo;
    }

}
```

```
import com.baidu.shop.base.Result;
import com.baidu.shop.document.GoodsDoc;
import com.baidu.shop.entity.BrandEntity;
import com.baidu.shop.entity.CategoryEntity;
import com.baidu.shop.status.HTTPStatus;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;
import java.util.Map;

/**
 * @ClassName GoodsDTO
 * @Description: TODO
 * @Author shenyaqi
 * @Date 2020/9/8
 * @Version V1.0
 **/
@Data
@NoArgsConstructor
public class GoodsResponse extends Result<List<GoodsDoc>> {

    private Integer total;
    private Integer totalPage;

    private List<BrandEntity> brandList;

    private List<CategoryEntity> categoryList;

    private Map<String, Object> specAggInfo;

    public GoodsResponse(Integer total, Integer totalPage
            , List<BrandEntity> brandList, List<CategoryEntity> categoryList
            , List<GoodsDoc> goodsDocs, Map<String, Object> specAggInfo){

        super(HTTPStatus.OK,HTTPStatus.OK + "",goodsDocs);
        this.total = total;
        this.totalPage = totalPage;
        this.brandList = brandList;
        this.categoryList = categoryList;
        this.specAggInfo = specAggInfo;
    }

}
```

4.3 search方法

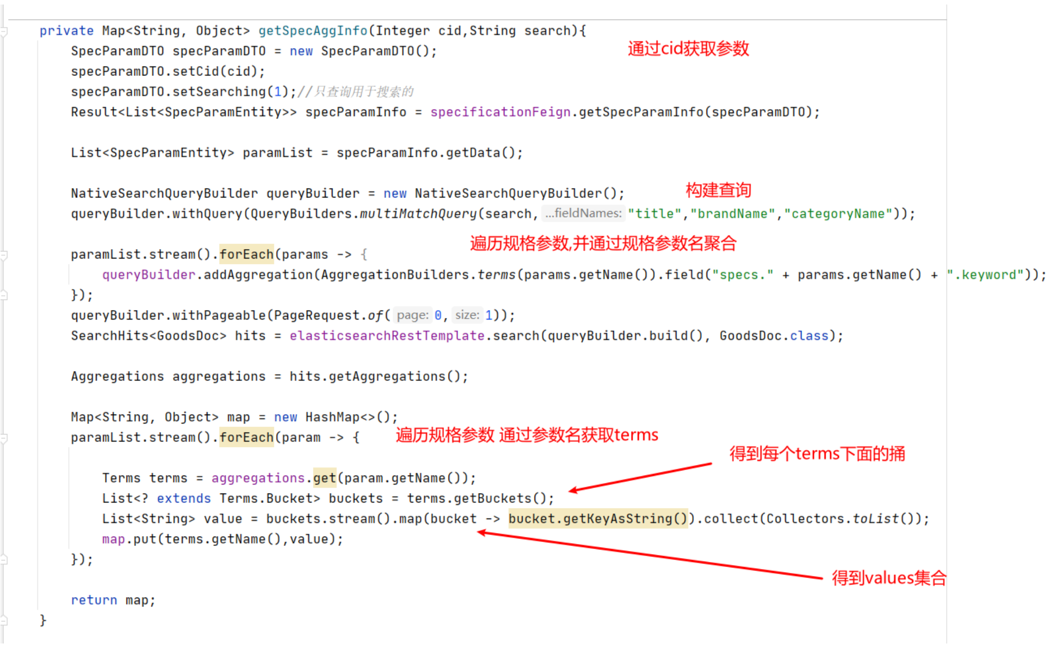


```
//通过分类id获取分类详细数据
Map<Integer, List<CategoryEntity>> cateMap = this.getCategoryEntityList(hits.getAggregations());
List<CategoryEntity> cateResult = null;
Integer hotCid = null;
//注意此处不能使用lambda表达式....
for(Map.Entry<Integer,List<CategoryEntity>> entry : cateMap.entrySet()){
    hotCid = entry.getKey();
    cateResult = entry.getValue();
}

//通过cid获取规格参数
Map<String, Object> specAggInfo = this.getSpecAggInfo(hotCid, search);
```

```
    @Override
    public GoodsResponse search(String search, Integer page) {

        NativeSearchQueryBuilder queryBuilder = this.getQueryBuilder(search,
page);
        SearchHits<GoodsDoc> hits =
elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);

        List<SearchHit<GoodsDoc>> highLightHit =
ESHighLightUtil.getHighLightHit(hits.getSearchHits());
        List<GoodsDoc> goodsDocs = highLightHit.stream().map(searchHit ->
searchHit.getContent()).collect(Collectors.toList());

        //通过品牌id获取品牌详细数据
        List<BrandEntity> brandResult =
getBrandEntityList(hits.getAggregations());

        //通过分类id获取分类详细数据
        Map<Integer, List<CategoryEntity>> cateMap =
this.getCategoryEntityList(hits.getAggregations());
        List<CategoryEntity> cateResult = null;
```

```java
        Integer hotCid = null;
        //注意此处不能使用lambda表达式....
        for(Map.Entry<Integer,List<CategoryEntity>> entry : cateMap.entrySet()){
            hotCid = entry.getKey();
            cateResult = entry.getValue();
        }

        //通过cid获取规格参数
        Map<String, Object> specAggInfo = this.getSpecAggInfo(hotCid, search);

        GoodsResponse goodsResponse = new
GoodsResponse(Long.valueOf(hits.getTotalHits()).intValue()
                ,
Double.valueOf(Math.ceil(Long.valueOf(hits.getTotalHits()).doubleValue() /
10)).intValue()
                , brandResult, cateResult, goodsDocs,specAggInfo);

        return goodsResponse;
    }
```

4.4 抽取出来的获取规格参数方法

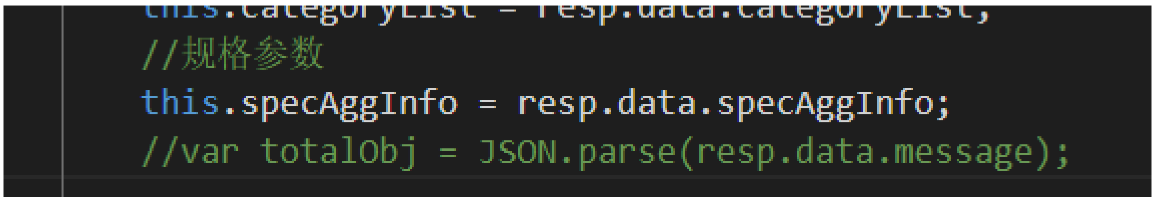

通过cid获取参数
构建查询
遍历规格参数,并通过规格参数名聚合
遍历规格参数 通过参数名获取terms
得到每个terms下面的捅
得到values集合

```java
private Map<String, Object> getSpecAggInfo(Integer cid,String search){
    SpecParamDTO specParamDTO = new SpecParamDTO();
    specParamDTO.setCid(cid);
    specParamDTO.setSearching(1);//只查询用于搜索的
    Result<List<SpecParamEntity>> specParamInfo = specificationFeign.getSpecParamInfo(specParamDTO);

    List<SpecParamEntity> paramList = specParamInfo.getData();

    NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();
    queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search, ...fieldNames: "title","brandName","categoryName"));

    paramList.stream().forEach(params -> {
        queryBuilder.addAggregation(AggregationBuilders.terms(params.getName()).field("specs." + params.getName() + ".keyword"));
    });
    queryBuilder.withPageable(PageRequest.of( page: 0, size: 1));
    SearchHits<GoodsDoc> hits = elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);

    Aggregations aggregations = hits.getAggregations();

    Map<String, Object> map = new HashMap<>();
    paramList.stream().forEach(param -> {

        Terms terms = aggregations.get(param.getName());
        List<? extends Terms.Bucket> buckets = terms.getBuckets();
        List<String> value = buckets.stream().map(bucket -> bucket.getKeyAsString()).collect(Collectors.toList());
        map.put(terms.getName(),value);
    });

    return map;
}
```

```java
    private Map<String, Object> getSpecAggInfo(Integer cid,String search){
        SpecParamDTO specParamDTO = new SpecParamDTO();
        specParamDTO.setCid(cid);
        specParamDTO.setSearching(1);//只查询用于搜索的
        Result<List<SpecParamEntity>> specParamInfo =
specificationFeign.getSpecParamInfo(specParamDTO);

        List<SpecParamEntity> paramList = specParamInfo.getData();

        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();

 queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search,"title","brandName"
,"categoryName"));

        paramList.stream().forEach(params -> {
```

```
 queryBuilder.addAggregation(AggregationBuilders.terms(params.getName()).field("
specs." + params.getName() + ".keyword"));
        });
        queryBuilder.withPageable(PageRequest.of(0,1));
        SearchHits<GoodsDoc> hits =
elasticsearchRestTemplate.search(queryBuilder.build(), GoodsDoc.class);

        Aggregations aggregations = hits.getAggregations();

        Map<String, Object> map = new HashMap<>();
        paramList.stream().forEach(param -> {

            Terms terms = aggregations.get(param.getName());
            List<? extends Terms.Bucket> buckets = terms.getBuckets();
            List<String> value = buckets.stream().map(bucket ->
bucket.getKeyAsString()).collect(Collectors.toList());
            map.put(terms.getName(),value);
        });

        return map;
    }
```

4.5 search.html

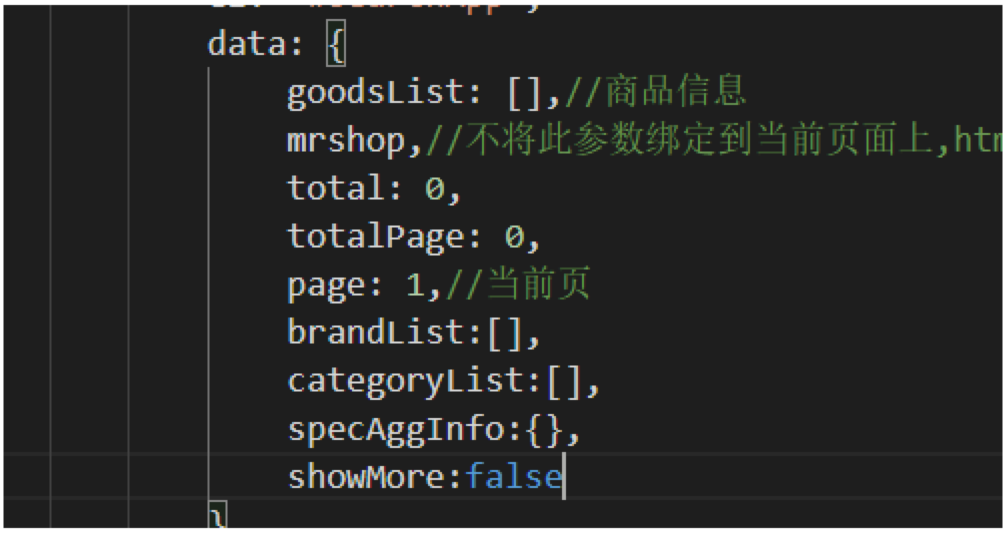


```
    //规格参数
    this.specAggInfo = resp.data.specAggInfo;
    //var totalObj = JSON.parse(resp.data.message);
```

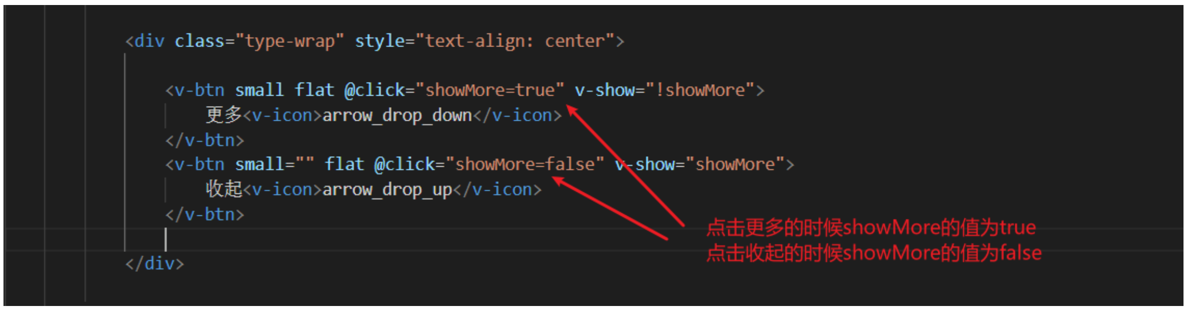
```
    <script type="text/javascript">
        var vm = new Vue({
            el: "#searchApp",
            data: {
                goodsList: [],//商品信息
                mrshop,//不将此参数绑定到当前页面上,html代码不能使用mrshop.***函数
                total: 0,
                totalPage: 0,
                page: 1,//当前页
                brandList:[],
                categoryList:[],
                specAggInfo:{}
            },
            components: {
                b2cTop: () => import("./js/pages/top.js")
            },
            watch: {
                page() {
                    this.searchEsData();
                }
            },
            created() {
                this.searchEsData();
            },
            methods: {
                searchEsData() {
                    const search = mrshop.parse(location.search.substring(1));//
将key=value转为key:value
                    //查询请求
                    mrshop.http.get('es/search', {
                        params: {
                            search: search.key,//将查询的内容传递到后台
                            page: this.page//当前页
                        }
                    }).then(resp => {

                        console.log(resp)
                        //处理sku
                        const goodsList = resp.data.data.map(goods => {

                            goods.skus = JSON.parse(goods.skus);//将字符串转为json
                            goods.selected = goods.skus[0];//设置当前选中
                            return goods;
                        })

                        this.total = resp.data.total;//总条数
                        this.totalPage = resp.data.totalPage//总页数

                        this.brandList = resp.data.brandList;
                        this.categoryList = resp.data.categoryList;
                        //规格参数
                        this.specAggInfo = resp.data.specAggInfo;
                        //var totalObj = JSON.parse(resp.data.message);

                        //this.total = totalObj.obj;//总条数
                        //this.totalPage = totalObj.totalPage//总页数
                        this.goodsList = goodsList;

                    }).catch(error => console.log(error));

                }
            }
        });
    </script>
```

```
                    <!--遍历规格参数数据-->
                    <div class="type-wrap" v-for="(val,key,index) in
specAggInfo">

                        <div class="fl key">{{ key }}</div>
                        <div class="fl value">
                            <ul class="type-list">
                                <!--遍历参数值-->
                                <li v-for="(item,index) in val">
                                    <a>{{ item }}</a>
                                </li>
                            </ul>
                        </div>
                        <div class="fl ext"></div>
                    </div>
```

4.5 更多 收起 功能实现

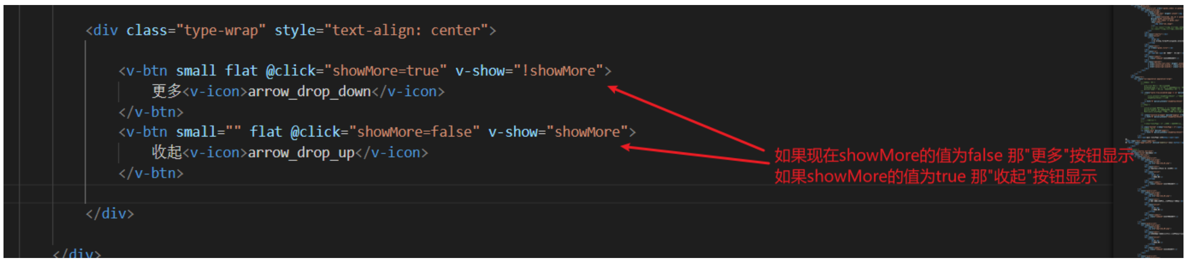


```
data: {
    goodsList: [],//商品信息
    mrshop,//不将此参数绑定到当前页面上,htm
    total: 0,
    totalPage: 0,
    page: 1,//当前页
    brandList:[],
    categoryList:[],
    specAggInfo:{},
    showMore:false
}
```



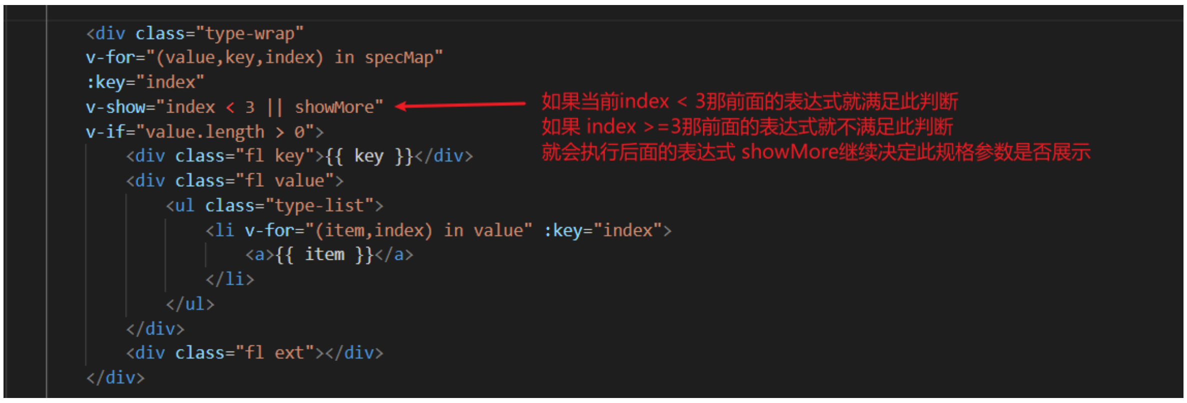
```
<div class="type-wrap" style="text-align: center">

    <v-btn small flat @click="showMore=true" v-show="!showMore">
        更多<v-icon>arrow_drop_down</v-icon>
    </v-btn>
    <v-btn small="" flat @click="showMore=false" v-show="showMore">
        收起<v-icon>arrow_drop_up</v-icon>
    </v-btn>

</div>
```

点击更多的时候showMore的值为true
点击收起的时候showMore的值为false



```
<div class="type-wrap" style="text-align: center">

    <v-btn small flat @click="showMore=true" v-show="!showMore">
        更多<v-icon>arrow_drop_down</v-icon>
    </v-btn>
    <v-btn small="" flat @click="showMore=false" v-show="showMore">
        收起<v-icon>arrow_drop_up</v-icon>
    </v-btn>

</div>

</div>
```

如果现在showMore的值为false 那"更多"按钮显示
如果showMore的值为true 那"收起"按钮显示



```
<div class="type-wrap"
v-for="(value,key,index) in specMap"
:key="index"
v-show="index < 3 || showMore"
v-if="value.length > 0">
    <div class="fl key">{{ key }}</div>
    <div class="fl value">
        <ul class="type-list">
            <li v-for="(item,index) in value" :key="index">
                <a>{{ item }}</a>
            </li>
        </ul>
    </div>
    <div class="fl ext"></div>
</div>
```

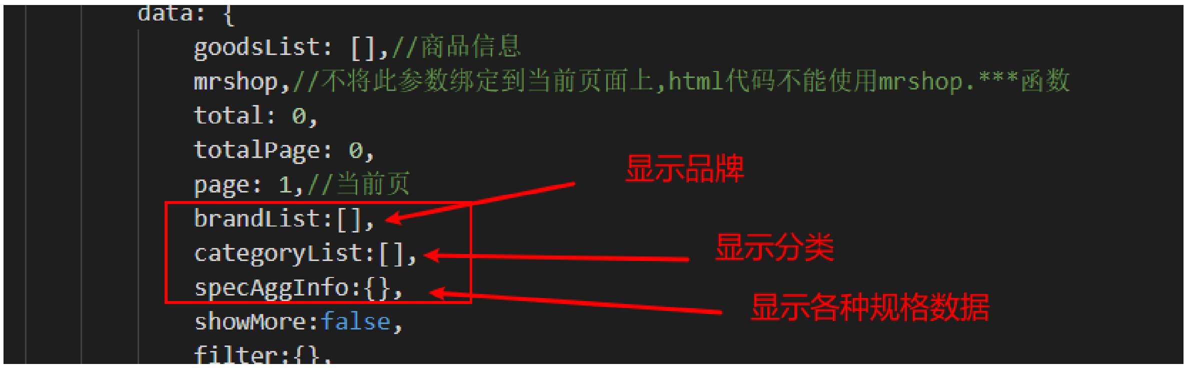
如果当前index < 3那前面的表达式就满足此判断
如果 index >=3那前面的表达式就不满足此判断
就会执行后面的表达式 showMore继续决定此规格参数是否展示

```
<!--遍历规格参数数据
index < 3 || showMore 默认只显示三条数据
如果showMore的状态为true则显示全部数据-->
<div class="type-wrap"
v-for="(val,key,index) in specAggInfo"
v-show="index < 3 || showMore">
    <div class="fl key">{{ key }}</div>
    <div class="fl value">
        <ul class="type-list">
            <!--遍历参数值-->
            <li v-for="(item,index) in val">
                <a>{{ item }}</a>
            </li>
        </ul>
```

```
                    </div>
                    <div class="fl ext"></div>
                </div>
                <div class="type-wrap" style="text-align: center">
                    <!--点击改变showMore的状态
                    showMore如果当前是现实全部则不显示更多选项-->
                    <v-btn small flat @click="showMore = true" v-
show="!showMore">
                        更多<v-icon>arrow_drop_down</v-icon>
                    </v-btn>
                    <v-btn small="" flat @click="showMore = false" v-
show="showMore">
                        收起<v-icon>arrow_drop_up</v-icon>
                    </v-btn>
                </div>
```

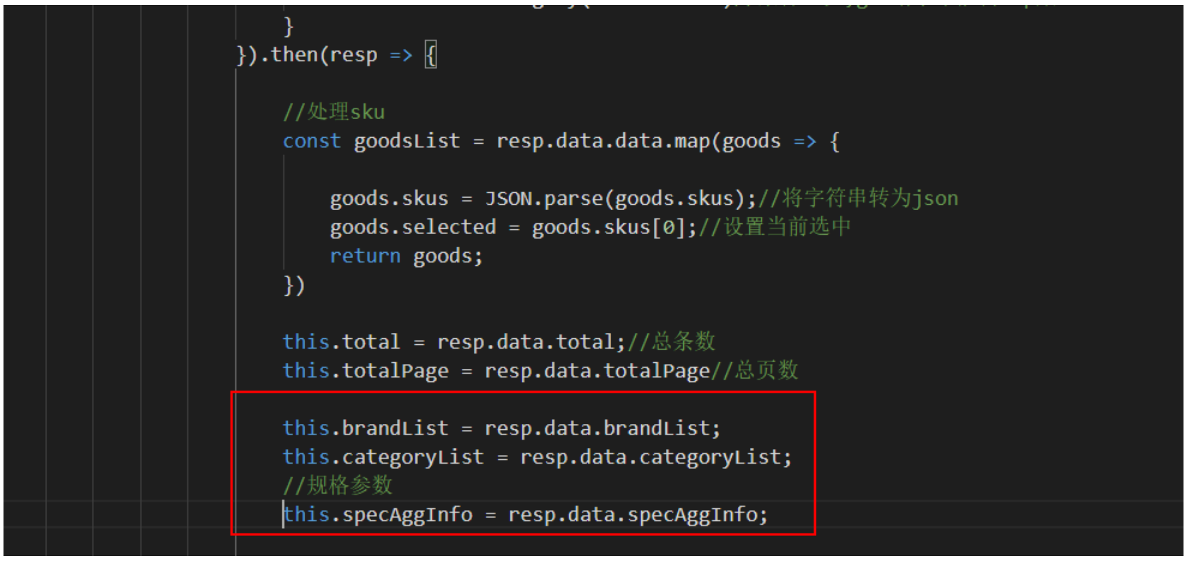
4.6 优化代码

上述代码中我们定义了一系列显示条件过滤的属性



```
        data: {
            goodsList: [],//商品信息
            mrshop,//不将此参数绑定到当前页面上,html代码不能使用mrshop.***函数
            total: 0,
            totalPage: 0,
            page: 1,//当前页
            brandList:[],
            categoryList:[],
            specAggInfo:{},
            showMore:false,
            filter:{},
```

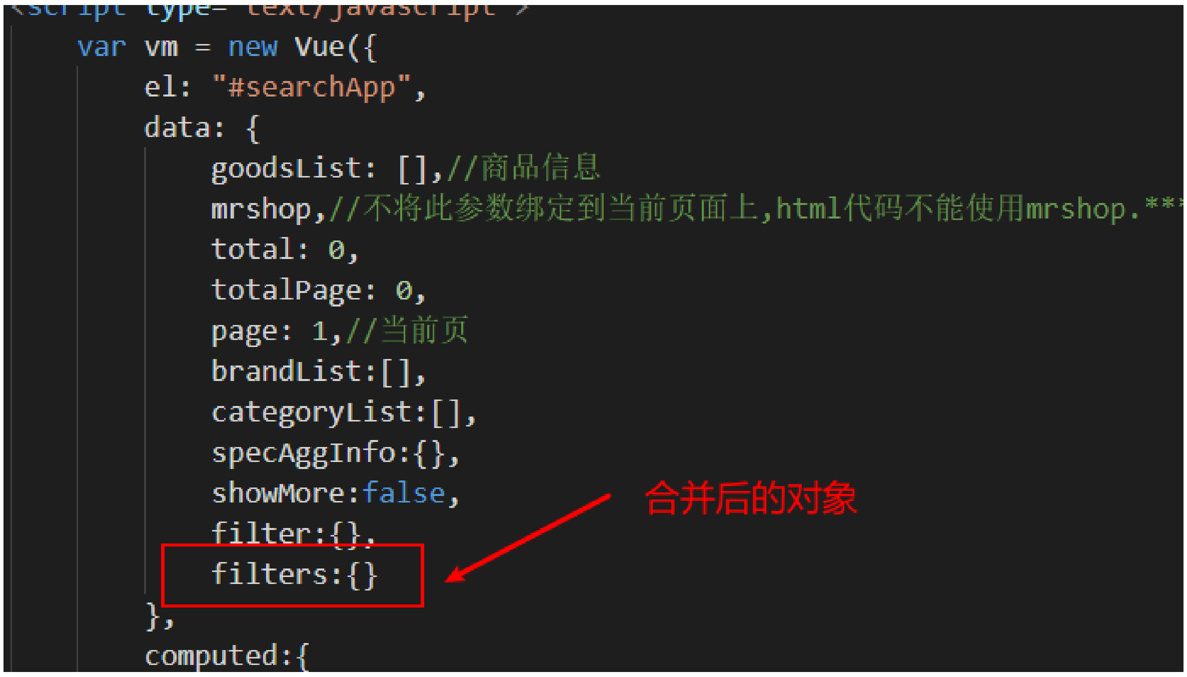
还有一些赋值操作



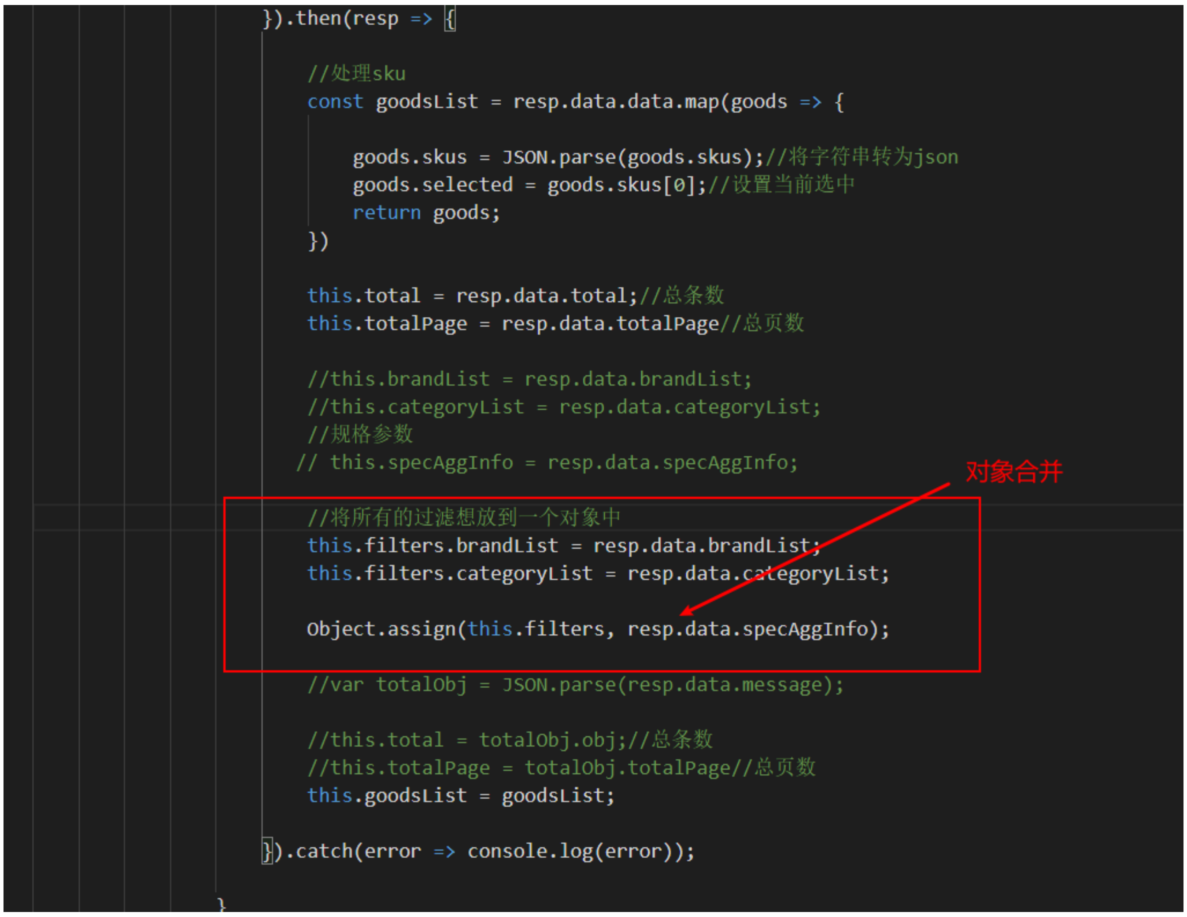
```
                }
            }).then(resp => {

                //处理sku
                const goodsList = resp.data.data.map(goods => {

                    goods.skus = JSON.parse(goods.skus);//将字符串转为json
                    goods.selected = goods.skus[0];//设置当前选中
                    return goods;
                })

                this.total = resp.data.total;//总条数
                this.totalPage = resp.data.totalPage//总页数

                this.brandList = resp.data.brandList;
                this.categoryList = resp.data.categoryList;
                //规格参数
                this.specAggInfo = resp.data.specAggInfo;
```

这些其实都是过滤条件数据

所以我们将这些数据合并成一个对象



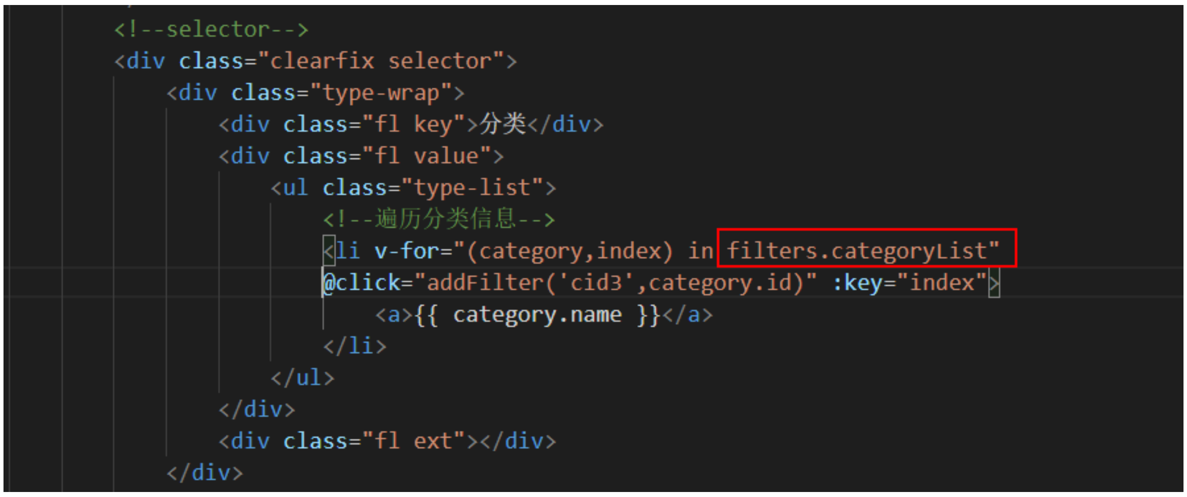
```
<script type="text/javascript">
    var vm = new Vue({
        el: "#searchApp",
        data: {
            goodsList: [],//商品信息
            mrshop,//不将此参数绑定到当前页面上,html代码不能使用mrshop.**
            total: 0,
            totalPage: 0,
            page: 1,//当前页
            brandList:[],
            categoryList:[],
            specAggInfo:{},
            showMore:false,
            filter:{},
            filters:{}
        },
        computed:{
```



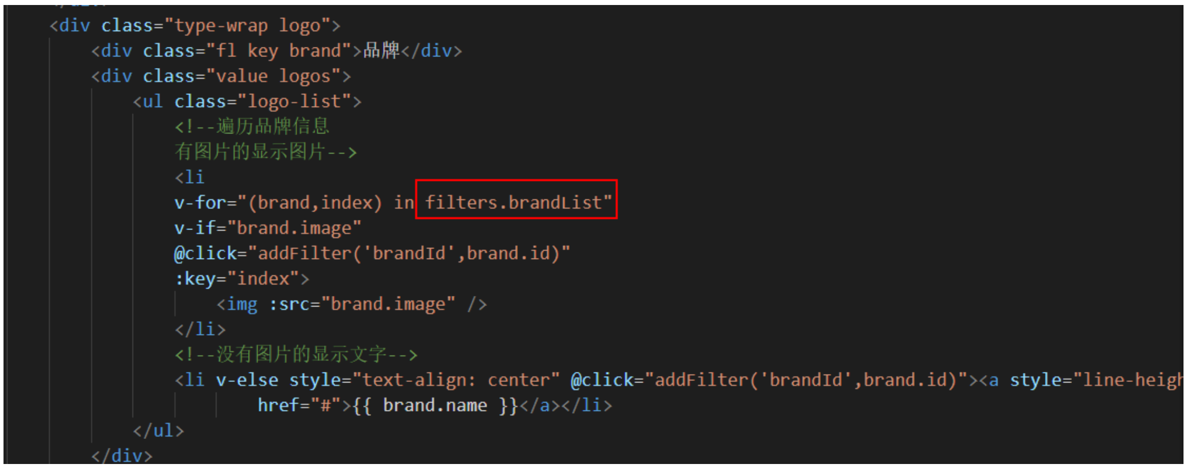
```
                }).then(resp => {

                    //处理sku
                    const goodsList = resp.data.data.map(goods => {

                        goods.skus = JSON.parse(goods.skus);//将字符串转为json
                        goods.selected = goods.skus[0];//设置当前选中
                        return goods;
                    })

                    this.total = resp.data.total;//总条数
                    this.totalPage = resp.data.totalPage//总页数

                    //this.brandList = resp.data.brandList;
                    //this.categoryList = resp.data.categoryList;
                    //规格参数
                   // this.specAggInfo = resp.data.specAggInfo;

                    //将所有的过滤想放到一个对象中
                    this.filters.brandList = resp.data.brandList;
                    this.filters.categoryList = resp.data.categoryList;

                    Object.assign(this.filters, resp.data.specAggInfo);

                    //var totalObj = JSON.parse(resp.data.message);

                    //this.total = totalObj.obj;//总条数
                    //this.totalPage = totalObj.totalPage//总页数
                    this.goodsList = goodsList;

                }).catch(error => console.log(error));

            }
```

```
    var vm = new Vue({
        el: "#searchApp",
        data: {
            goodsList: [],//商品信息
            mrshop,//不将此参数绑定到当前页面上,html代码不能使用mrshop.***函数
            total: 0,
            totalPage: 0,
            page: 1,//当前页
            brandList:[],
            categoryList:[],
            specAggInfo:{},
            showMore:false,
                filter:{},
                filters:{}
            },
            computed:{
                showFilterList(){

                }
            },
            components: {
                b2cTop: () => import("./js/pages/top.js")
            },
            watch: {
                page() {
                    this.searchEsData();
                }
            },
            created() {
                this.searchEsData();
            },
            methods: {
                getFilterValue (key,value) {

                    if(key == 'cid3'){
                        this.filters.categoryList.forEach(category => {
                            if(category.id == value){
                                value = category.name;
                            }
                        })
                        return value;
                    }else if(key == 'brandId'){
                        this.filters.brandList.forEach(brand => {
                            if(brand.id == value){
                                value = brand.name;
                            }
                        })
                        return value;
                    }

                    return value;
                },
                //新增条件过滤
                addFilter(key,value){
                    this.filter[key] = value;
                    this.searchEsData();
                },
                searchEsData() {

                    const search = mrshop.parse(location.search.substring(1));//
将key=value转为key:value
                    if(this.filter)
                    //查询请求
                    mrshop.http.get('es/search', {
                        params: {
                            search: search.key,//将查询的内容传递到后台
                            page: this.page,//当前页
                            filter:JSON.stringify(this.filter)//条件过滤,get请求不
能传map集合
                        }
                    }).then(resp => {

                        //处理sku
                        const goodsList = resp.data.data.map(goods => {

                            goods.skus = JSON.parse(goods.skus);//将字符串转为json
                            goods.selected = goods.skus[0];//设置当前选中
                            return goods;
                        })

                        this.total = resp.data.total;//总条数
                        this.totalPage = resp.data.totalPage//总页数

                        //将所有的过滤想放到一个对象中
                        this.filters.brandList = resp.data.brandList;
                        this.filters.categoryList = resp.data.categoryList;
                        Object.assign(this.filters, resp.data.specAggInfo);

                        this.goodsList = goodsList;

                    }).catch(error => console.log(error));

                }
            }
        });
```

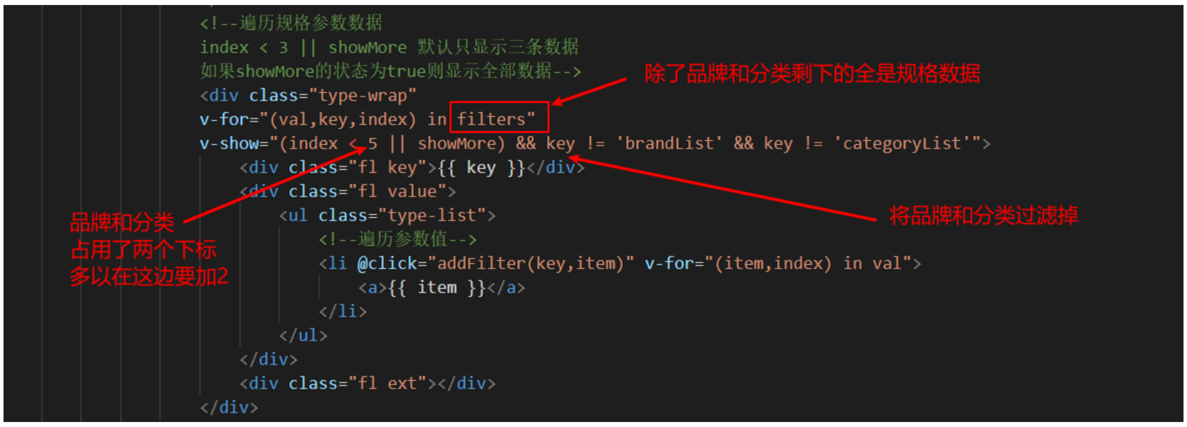
那现在模板就必须得改了



```
<!--selector-->
<div class="clearfix selector">
    <div class="type-wrap">
        <div class="fl key">分类</div>
        <div class="fl value">
            <ul class="type-list">
                <!--遍历分类信息-->
                <li v-for="(category,index) in filters.categoryList"
                @click="addFilter('cid3',category.id)" :key="index">
                    <a>{{ category.name }}</a>
                </li>
            </ul>
        </div>
        <div class="fl ext"></div>
    </div>
```



```
<div class="type-wrap logo">
    <div class="fl key brand">品牌</div>
    <div class="value logos">
        <ul class="logo-list">
            <!--遍历品牌信息
            有图片的显示图片-->
            <li
            v-for="(brand,index) in filters.brandList"
            v-if="brand.image"
            @click="addFilter('brandId',brand.id)"
            :key="index">
                <img :src="brand.image" />
            </li>
            <!--没有图片的显示文字-->
            <li v-else style="text-align: center" @click="addFilter('brandId',brand.id)"><a style="line-heigh
                    href="#">{{ brand.name }}</a></li>
        </ul>
    </div>
```



```
<!--遍历规格参数数据
index < 3 || showMore 默认只显示三条数据
如果showMore的状态为true则显示全部数据-->
<div class="type-wrap"
v-for="(val,key,index) in filters"
v-show="(index < 5 || showMore) && key != 'brandList' && key != 'categoryList'">
    <div class="fl key">{{ key }}</div>
    <div class="fl value">
        <ul class="type-list">
            <!--遍历参数值-->
            <li @click="addFilter(key,item)" v-for="(item,index) in val">
                <a>{{ item }}</a>
            </li>
        </ul>
    </div>
    <div class="fl ext"></div>
</div>
```

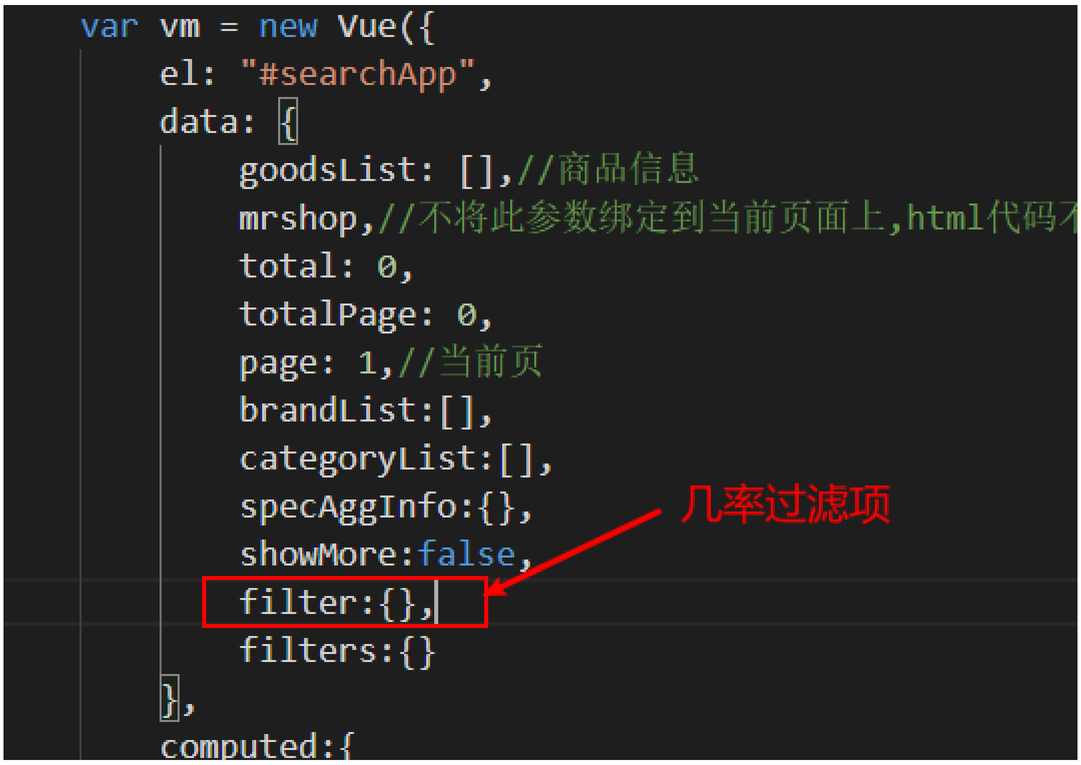
```
                    <div class="type-wrap">
                        <div class="fl key">分类</div>
                        <div class="fl value">
                            <ul class="type-list">
                                <!--遍历分类信息-->
                                <li v-for="(category,index) in
filters.categoryList"
                                @click="addFilter('cid3',category.id)"
:key="index">
                                    <a>{{ category.name }}</a>
                                </li>
                            </ul>
                        </div>
                        <div class="fl ext"></div>
                    </div>
```

```
                    <div class="type-wrap logo">
                        <div class="fl key brand">品牌</div>
                        <div class="value logos">
                            <ul class="logo-list">
                                <!--遍历品牌信息
                                有图片的显示图片-->
                                <li
                                v-for="(brand,index) in filters.brandList"
                                v-if="brand.image"
                                @click="addFilter('brandId',brand.id)"
                                :key="index">
                                    <img :src="brand.image" />
                                </li>
                                <!--没有图片的显示文字-->
                                <li v-else style="text-align: center"
@click="addFilter('brandId',brand.id)"><a style="line-height: 30px; font-size:
12px"
                                        href="#">{{ brand.name }}</a></li>
                            </ul>
                        </div>
                        <div class="fl ext">
                            <a href="javascript:void(0);" class="sui-btn">多选
</a>
                        </div>
                    </div>
```

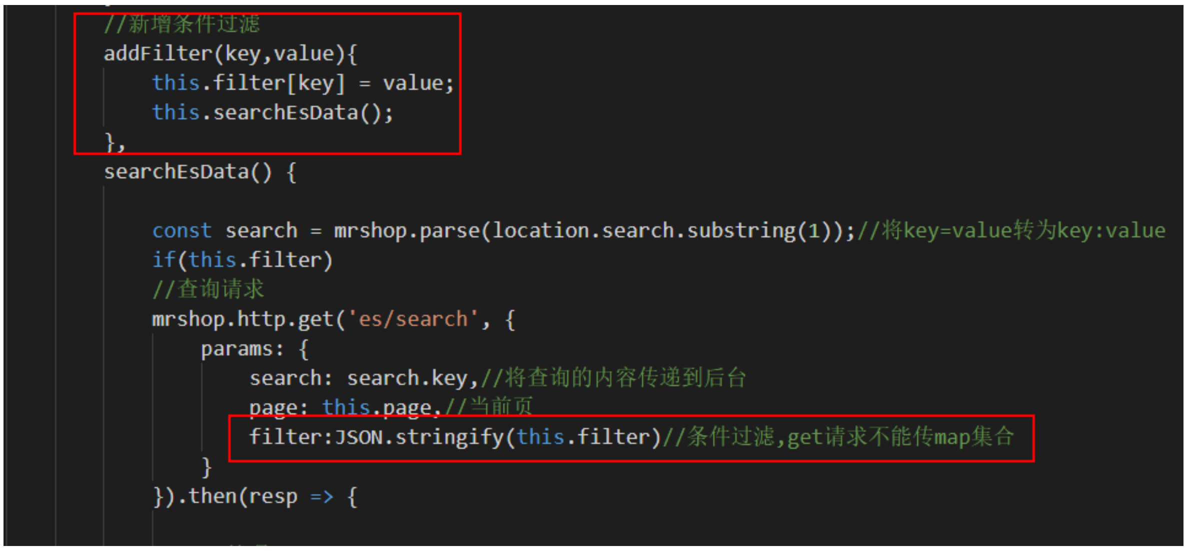
```
                    <div class="type-wrap"
                    v-for="(val,key,index) in filters"
                    v-show="(index < 5 || showMore) && key != 'brandList' && key
!= 'categoryList'">
                        <div class="fl key">{{ key }}</div>
                        <div class="fl value">
                            <ul class="type-list">
                                <!--遍历参数值-->
                                <li @click="addFilter(key,item)" v-for="
(item,index) in val">
                                    <a>{{ item }}</a>
                                </li>
                            </ul>
                        </div>
                        <div class="fl ext"></div>
                    </div>
```

5 搜索过滤

5.1 search.html



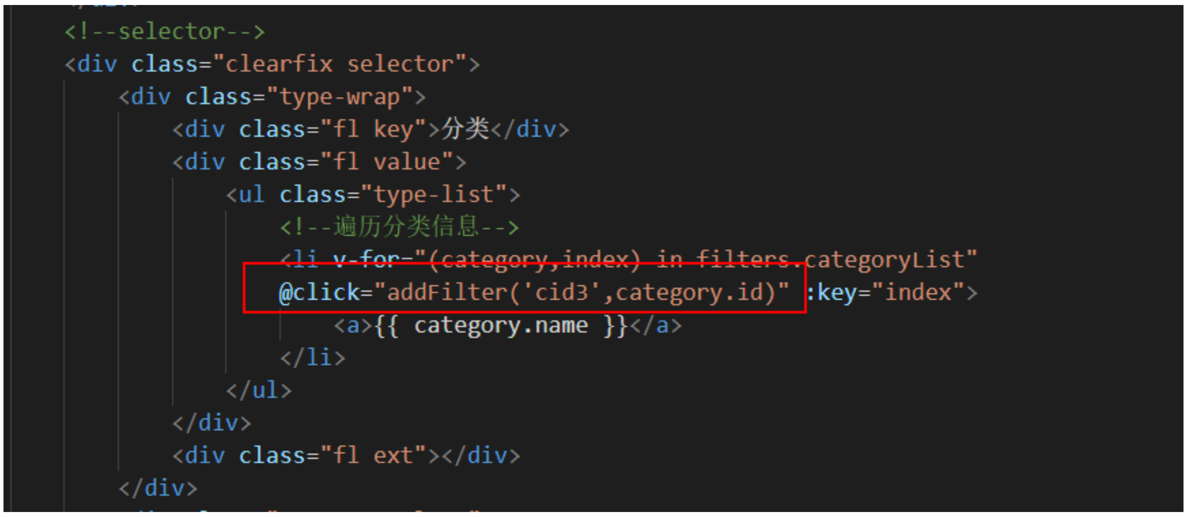
```
    var vm = new Vue({
        el: "#searchApp",
        data: {
            goodsList: [],//商品信息
            mrshop,//不将此参数绑定到当前页面上,html代码不
            total: 0,
            totalPage: 0,
            page: 1,//当前页
            brandList:[],
            categoryList:[],
            specAggInfo:{},
            showMore:false,
            filter:{},
            filters:{}
        },
        computed:{
```



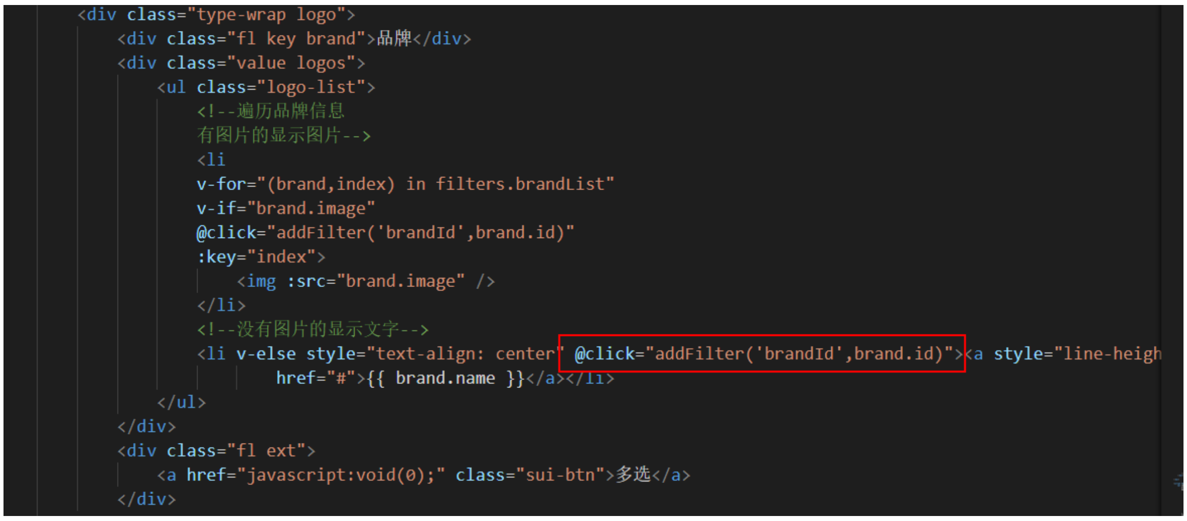
```
//新增条件过滤
addFilter(key,value){
    this.filter[key] = value;
    this.searchEsData();
},
searchEsData() {

    const search = mrshop.parse(location.search.substring(1));//将key=value转为key:value
    if(this.filter)
    //查询请求
    mrshop.http.get('es/search', {
        params: {
            search: search.key,//将查询的内容传递到后台
            page: this.page,//当前页
            filter:JSON.stringify(this.filter)//条件过滤,get请求不能传map集合
        }
    }).then(resp => {
```

```
methods: {
    //新增条件过滤
    addFilter(key,value){
        this.filter[key] = value;
        this.searchEsData();
    },
    searchEsData() {

        const search = mrshop.parse(location.search.substring(1));//将key=value转为key:value
        if(this.filter)
        //查询请求
        mrshop.http.get('es/search', {
            params: {
                search: search.key,//将查询的内容传递到后台
                page: this.page,//当前页
                filter:JSON.stringify(this.filter)//条件过滤,get请求不能传map集合
            }
        }).then(resp => {

            //处理sku
            const goodsList = resp.data.data.map(goods => {

                goods.skus = JSON.parse(goods.skus);//将字符串转为json
                goods.selected = goods.skus[0];//设置当前选中
                return goods;
            })

            this.total = resp.data.total;//总条数
            this.totalPage = resp.data.totalPage//总页数

            //将所有的过滤想放到一个对象中
            this.filters.brandList = resp.data.brandList;
            this.filters.categoryList = resp.data.categoryList;
            Object.assign(this.filters, resp.data.specAggInfo);

            this.goodsList = goodsList;

        }).catch(error => console.log(error));
```

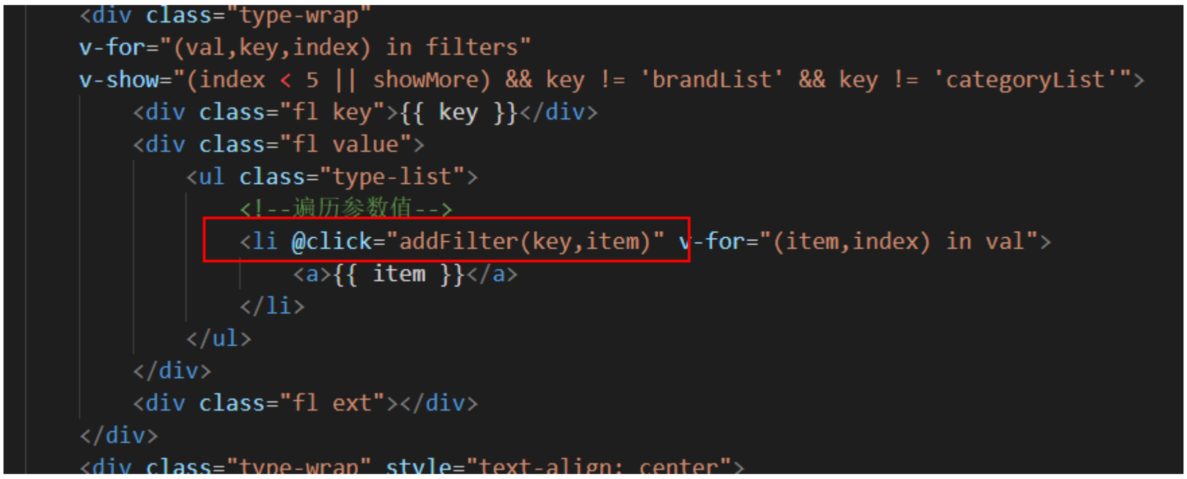
```
            }
        }
```



```
<!--selector-->
<div class="clearfix selector">
    <div class="type-wrap">
        <div class="fl key">分类</div>
        <div class="fl value">
            <ul class="type-list">
                <!--遍历分类信息-->
                <li v-for="(category,index) in filters.categoryList"
                @click="addFilter('cid3',category.id)" :key="index">
                    <a>{{ category.name }}</a>
                </li>
            </ul>
        </div>
        <div class="fl ext"></div>
    </div>
```



```
<div class="type-wrap logo">
    <div class="fl key brand">品牌</div>
    <div class="value logos">
        <ul class="logo-list">
            <!--遍历品牌信息
            有图片的显示图片-->
            <li
            v-for="(brand,index) in filters.brandList"
            v-if="brand.image"
            @click="addFilter('brandId',brand.id)"
            :key="index">
                <img :src="brand.image" />
            </li>
            <!--没有图片的显示文字-->
            <li v-else style="text-align: center" @click="addFilter('brandId',brand.id)"><a style="line-heigh
                    href="#">{{ brand.name }}</a></li>
        </ul>
    </div>
    <div class="fl ext">
        <a href="javascript:void(0);" class="sui-btn">多选</a>
    </div>
```

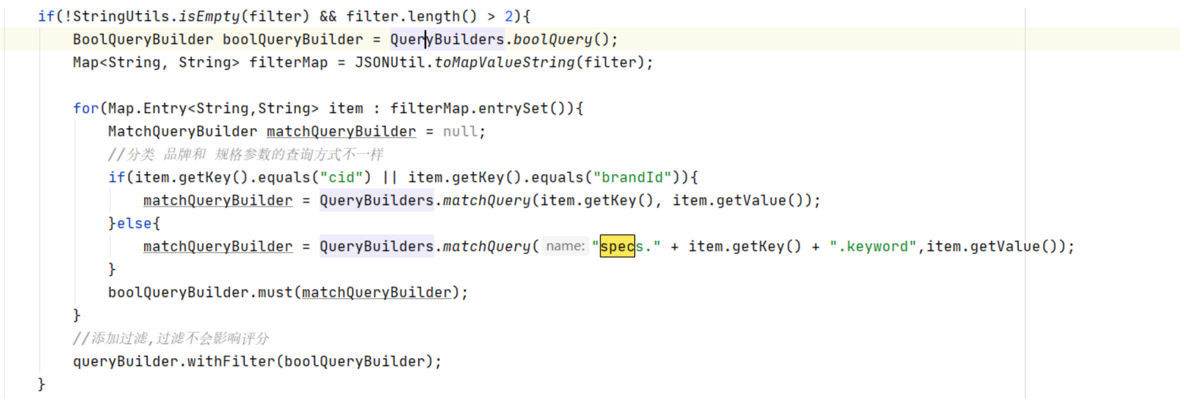


```
<div class="type-wrap"
v-for="(val,key,index) in filters"
v-show="(index < 5 || showMore) && key != 'brandList' && key != 'categoryList'">
    <div class="fl key">{{ key }}</div>
    <div class="fl value">
        <ul class="type-list">
            <!--遍历参数值-->
            <li @click="addFilter(key,item)" v-for="(item,index) in val">
                <a>{{ item }}</a>
            </li>
        </ul>
    </div>
    <div class="fl ext"></div>
</div>
```

5.2 ShopElasticsearchService

```
    @ApiOperation(value = "搜索")
    @GetMapping(value = "es/search")
    GoodsResponse search(@RequestParam String search, @RequestParam Integer page
            , @RequestParam String filter);
```

5.3 ShopElasticsearchServiceImpl



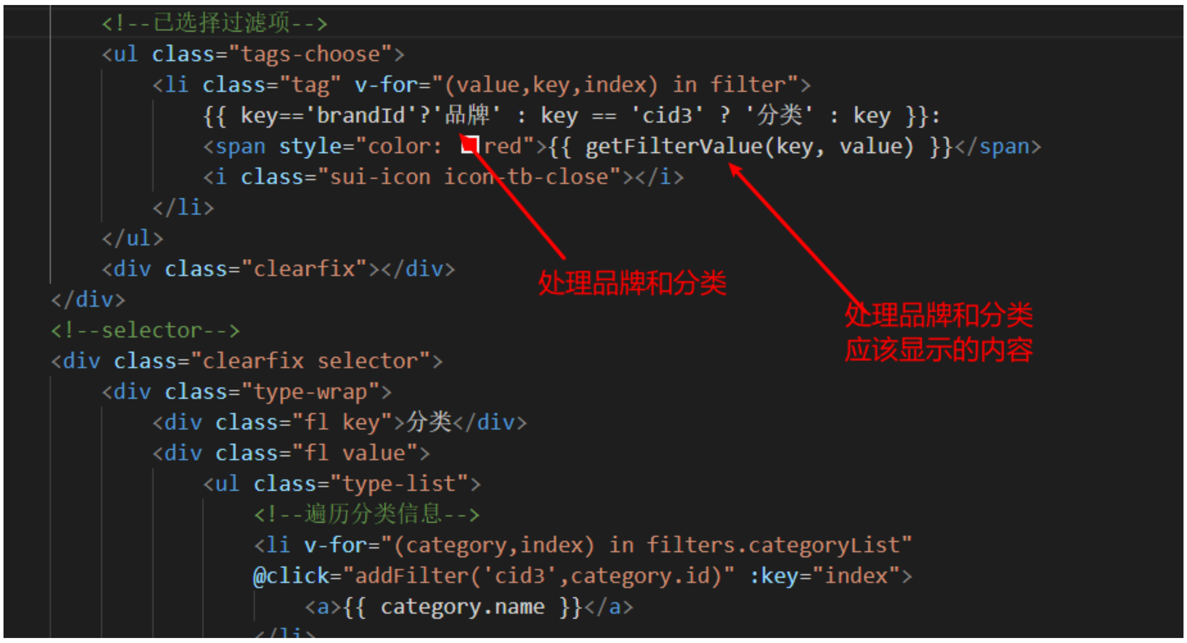
```
if(!StringUtils.isEmpty(filter) && filter.length() > 2){
    BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
    Map<String, String> filterMap = JSONUtil.toMapValueString(filter);

    for(Map.Entry<String,String> item : filterMap.entrySet()){
        MatchQueryBuilder matchQueryBuilder = null;
        //分类 品牌和 规格参数的查询方式不一样
        if(item.getKey().equals("cid") || item.getKey().equals("brandId")){
            matchQueryBuilder = QueryBuilders.matchQuery(item.getKey(), item.getValue());
        }else{
            matchQueryBuilder = QueryBuilders.matchQuery( name: "specs." + item.getKey() + ".keyword",item.getValue());
        }
        boolQueryBuilder.must(matchQueryBuilder);
    }
    //添加过滤,过滤不会影响评分
    queryBuilder.withFilter(boolQueryBuilder);
}
```

```
    private NativeSearchQueryBuilder getQueryBuilder(String search, Integer
page,String filter){
        NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();

        if (!StringUtils.isEmpty(search)) {
            //多字段同时查询

 queryBuilder.withQuery(QueryBuilders.multiMatchQuery(search,"title","brandName"
,"categoryName"));
        }

        if(!StringUtils.isEmpty(filter) && filter.length() > 2){
            BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
            Map<String, String> filterMap = JSONUtil.toMapValueString(filter);

            for(Map.Entry<String,String> item : filterMap.entrySet()){
                MatchQueryBuilder matchQueryBuilder = null;
                //分类 品牌和 规格参数的查询方式不一样
                if(item.getKey().equals("cid3") ||
item.getKey().equals("brandId")){
                    matchQueryBuilder = QueryBuilders.matchQuery(item.getKey(),
item.getValue());
                }else{
                    matchQueryBuilder = QueryBuilders.matchQuery("specs." +
item.getKey() + ".keyword",item.getValue());
                }
                boolQueryBuilder.must(matchQueryBuilder);
            }
            //添加过滤,过滤不会影响评分
            queryBuilder.withFilter(boolQueryBuilder);
        }

        queryBuilder.withPageable(PageRequest.of(page-1,10));
        //设置查询出来的内容,页面上做多只需要id,title,skus
        queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]
{"id","title","skus"}, null));
        //设置高亮字段

 queryBuilder.withHighlightBuilder(ESHighLightUtil.getHighlightBuilder("title"))
;

        //聚合

 queryBuilder.addAggregation(AggregationBuilders.terms("cate_agg").field("cid3")
);
```

```
 queryBuilder.addAggregation(AggregationBuilders.terms("brand_agg").field("brandId"));

        return queryBuilder;
    }
```

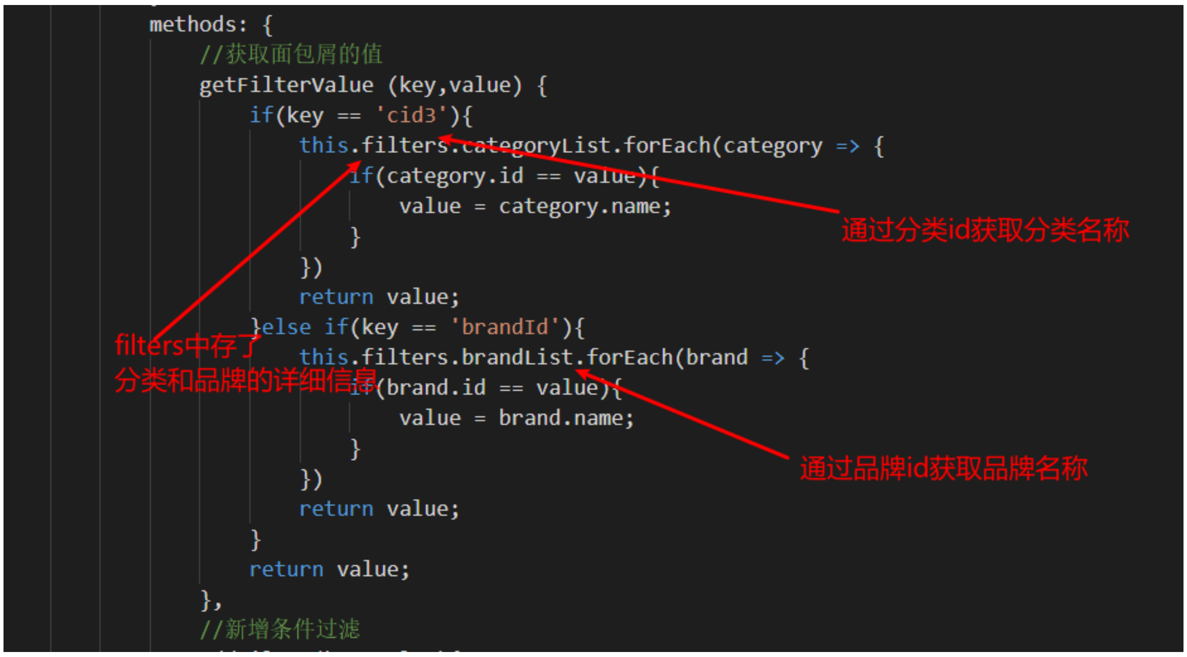
6 面包屑

6.1 面包屑展示



```
    <!--已选择过滤项-->
    <ul class="tags-choose">
        <li class="tag" v-for="(value,key,index) in filter">
            {{ key=='brandId'?'品牌' : key == 'cid3' ? '分类' : key }}:
            <span style="color: red">{{ getFilterValue(key, value) }}</span>
            <i class="sui-icon icon-tb-close"></i>
        </li>
    </ul>
    <div class="clearfix"></div>
</div>
<!--selector-->
<div class="clearfix selector">
    <div class="type-wrap">
        <div class="fl key">分类</div>
        <div class="fl value">
            <ul class="type-list">
                <!--遍历分类信息-->
                <li v-for="(category,index) in filters.categoryList"
                @click="addFilter('cid3',category.id)" :key="index">
                    <a>{{ category.name }}</a>
```

```
                    <!--已选择过滤项-->
                    <ul class="tags-choose">
                        <li class="tag" v-for="(value,key,index) in filter">
                            {{ key=='brandId'?'品牌' : key == 'cid3' ? '分类' : key }}:
                            <span style="color: red">{{ getFilterValue(key, value) }}</span>
                            <i class="sui-icon icon-tb-close"></i>
                        </li>
                    </ul>
```



```
methods: {
    //获取面包屑的值
    getFilterValue (key,value) {
        if(key == 'cid3'){
            this.filters.categoryList.forEach(category => {
                if(category.id == value){
                    value = category.name;
                }
            })
            return value;
        }else if(key == 'brandId'){
            this.filters.brandList.forEach(brand => {
                if(brand.id == value){
                    value = brand.name;
                }
            })
            return value;
        }
        return value;
    },
    //新增条件过滤
```

```
            //获取面包屑的值
            getFilterValue (key,value) {
                if(key == 'cid3'){
                    this.filters.categoryList.forEach(category => {
                        if(category.id == value){
                            value = category.name;
                        }
                    })
                    return value;
                }else if(key == 'brandId'){
                    this.filters.brandList.forEach(brand => {
                        if(brand.id == value){
                            value = brand.name;
                        }
                    })
                    return value;
                }
                return value;
            },
```

6.2 点击×号取消过滤

```
                <ul class="tags-choose">
                    <li class="tag" v-for="(value,key,index) in filter">
                        {{ key=='brandId'?'品牌' : key == 'cid3' ? '分类' :
key }}:
                        <span style="color: red">{{ getFilterValue(key,
value) }}</span>
                        <i @click="removeFilterItem(key)" class="sui-icon
icon-tb-close"></i>
                    </li>
                </ul>
            removeFilterItem(key){
                delete this.filter[key];
                this.searchEsData();
            },
```