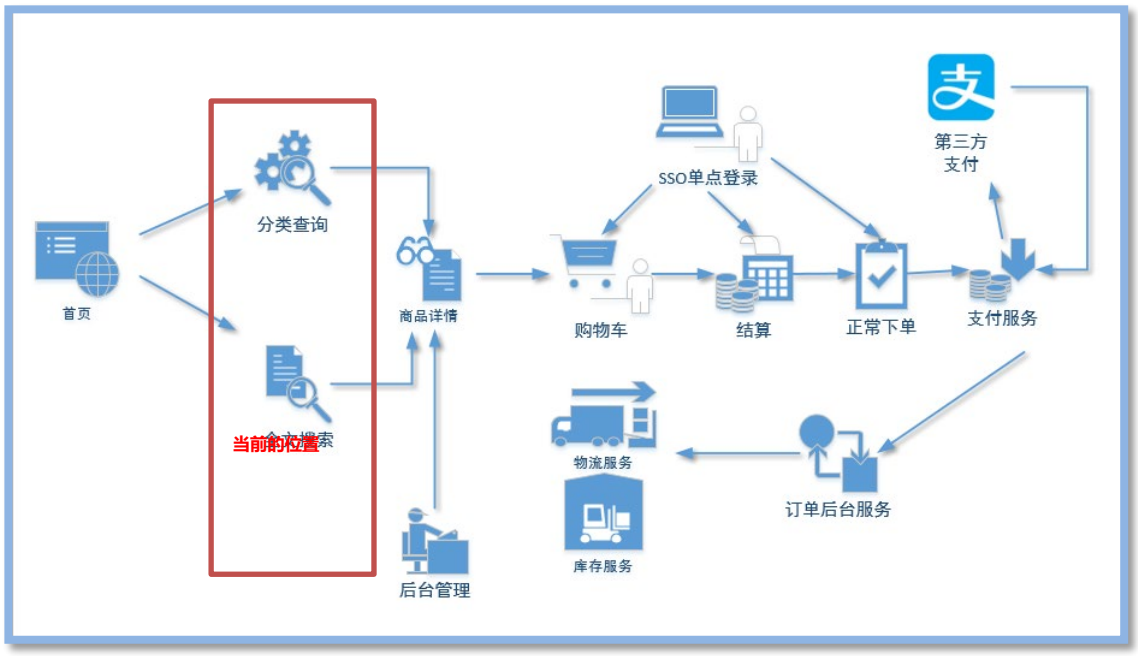
尚品汇商城



一、商品检索功能介绍

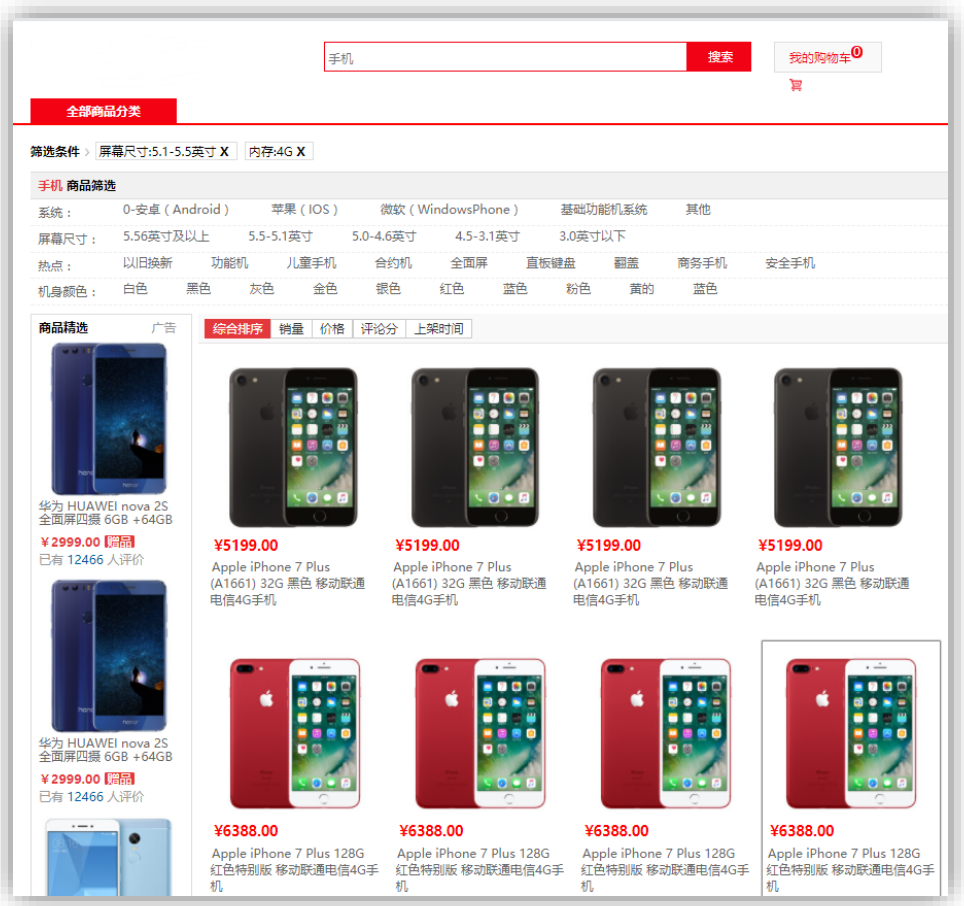
根据用户输入的检索条件，查询出对应的商品

1.1 检索两个入口

首页的分类

搜索栏

1.2 检索列表展示页面



1.3 根据业务搭建数据结构

1.3.1 建立 mapping!

这时我们要思考三个问题：

1、 哪些字段需要分词

a) 例如：商品名称

2、 我们用哪些字段进行过滤

a) 平台属性值

b) 分类 Id

3、 哪些字段我们需要通过搜索查询出来。

a) 商品名称,价格,图片等。

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

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

根据以上制定出如下结构：mappings

Index：goods

~~type：_doc~~

document: properties - rows

field: id,price,title…

Es 中 index 默认是 true。

注意：ik_max_word 中文词库必须有!

attrs：平台属性值的集合，主要用于平台属性值过滤。

1.3.2 nested 介绍

nested：类型是一种特殊的对象 object 数据类型(specialised version of the object datatype)，允许对象数组彼此独立地进行索引和查询。

demo： 建立一个普通的 index

如果 linux 中有这个 my_index 先删除！DELETE /my_index

步骤 1：建立一个 index

```
PUT my_index/_doc/1
{
  "group" : "fans",
  "user" : [
    {
      "first" : "John",
      "last" :  "Smith"
    },
    {
      "first" : "Alice",
      "last" :  "White"
    }
  ]
}
```

步骤 2：执行查询

```
GET my_index/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "user.first": "Alice" }},
```

```
      { "match": { "user.last":  "Smith" }}
    ]
   }
  }
}
```

查询结果：

```
"hits" : {
  "total" : 1,
  "max_score" : 0.5753642,
  "hits" : [
    {
      "_index" : "my_index",
      "_type" : "_doc",
      "_id" : "1",
      "_score" : 0.5753642,
      "_source" : {
        "group" : "fans",
        "user" : [
          {
            "first" : "John",
            "last" : "Smith"
          },
          {
            "first" : "Alice",
            "last" : "White"
          }
        ]
      }
```

能够查询出数据的原因是因为：建立 my_index 的时候，它默认的数据类型是 Object

{user.first:"John ,Alice"}

{user.last:"Smith,White"}

实际上：从业务的角度出发应该没有数据： 因为

User1 {John，Smith}

User2 {Alice，White}

是两个对象 而 {Alice,Smith} 在当前的 user 中不存在!

步骤 3：删除当前索引

DELETE /my_index

步骤 4：建立一个 nested 类型的

```
PUT my_index
{
  "mappings": {
     "properties": {
       "user": {
         "type": "nested"
       }
    }
  }
}
```

user 字段映射为 nested 类型，而不是默认的 object 类型

重新执行步骤 1，使用 nested 查询

```
GET /my_index/_search
{
  "query": {
    "nested": {
      "path": "user",
      "query": {
        "bool": {
          "must": [
            {"match": {"user.first": "Alice"}},
            {"match": {"user.last": "Smith"}}
          ]
        }
      }
    }
  }
}
```

此查询得不到匹配，是因为 Alice 和 Smith 不在同一个嵌套对象中。

{"match": {"user.last": "White"}} 此时就会有数据：

1.4 搭建 service-list 服务

在 service 模块下搭建，搭建方式如 service-item

1.4.1 修改配置 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>service</artifactId>
        <version>1.0</version>
    </parent>

    <artifactId>service-list</artifactId>
    <version>1.0</version>

    <packaging>jar</packaging>
    <name>service-list</name>
    <description>service-list</description>

    <dependencies>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>service-product-client</artifactId>
            <version>1.0</version>
        </dependency>


        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-
elasticsearch</artifactId>
        </dependency>

    </dependencies>
```

```
</project>
```

说明：

1，引入 service-product-client 模块

2，引入 spring-boot-starter-data-elasticsearch 依赖

1.4.2 添加配置文件

bootstrap.properties

```
spring.application.name=service-list
spring.profiles.active=dev
spring.cloud.nacos.discovery.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.prefix=${spring.application.name}
spring.cloud.nacos.config.file-extension=yaml
spring.cloud.nacos.config.shared-configs[0].data-id=common.yaml
```

说明：添加 es 配置

添加主启动类

```
@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
@ComponentScan({"com.atguigu.gmall"})
@EnableDiscoveryClient
@EnableFeignClients(basePackages= {"com.atguigu.gmall"})
public class ServiceListApplication {
    public static void main(String[] args) {

        SpringApplication.run(ServiceListApplication.class,args);
    }
}
```

1.4.3 构建实体与 es mapping 建立映射关系

```
package com.atguigu.gmall.model.list;


@Document(indexName = "goods", shards = 3, replicas = 1)
```

```
@Data
public class Goods {

    @Id
    private Long id;

    @Field(type = FieldType.Keyword, index = false)
    private String defaultImg;

    @Field(type = FieldType.Text, analyzer = "ik_max_word")
    private String title;

    @Field(type = FieldType.Double)
    private Double price;

    @Field(type = FieldType.Date,format = DateFormat.custom, pattern
= "yyyy-MM-dd HH:mm:ss")
    private Date createTime; // 新品

    @Field(type = FieldType.Long)
    private Long tmId;

    @Field(type = FieldType.Keyword)
    private String tmName;


    @Field(type = FieldType.Keyword)
    private String tmLogoUrl;


    @Field(type = FieldType.Long)
    private Long category1Id;

    @Field(type = FieldType.Keyword)
    private String category1Name;

    @Field(type = FieldType.Long)
    private Long category2Id;

    @Field(type = FieldType.Keyword)
    private String category2Name;

    @Field(type = FieldType.Long)
    private Long category3Id;

    @Field(type = FieldType.Keyword)
    private String category3Name;

    @Field(type = FieldType.Long)
    private Long hotScore = 0L;

    @Field(type = FieldType.Nested)
    private List<SearchAttr> attrs;
```

```
}
```

平台属性-平台属性值

```
@Data
public class SearchAttr {

    @Field(type = FieldType.Long)
    private Long attrId;
    @Field(type = FieldType.Keyword)
    private String attrName;
    @Field(type = FieldType.Keyword)
    private String attrValue;
}
```

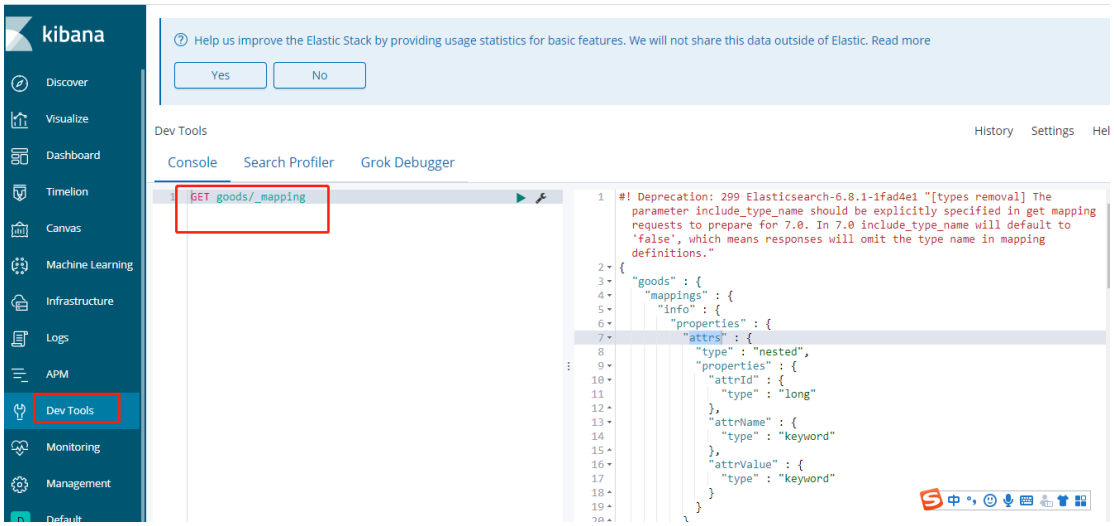
1.4.4 初始化 mapping 结构到 es 中

```
package com.atguigu.gmall.list.controller;

@RestController
@RequestMapping("api/list")
public class ListApiController {

    @Autowired
    private ElasticsearchRestTemplate restTemplate;

    /**
     * @return
     */
    @GetMapping("inner/createIndex")
    public Result createIndex() {
        restTemplate.createIndex(Goods.class);
        restTemplate.putMapping(Goods.class);
        return Result.ok();
    }

}
```

在浏览器运行：

http://localhost:8203/api/list/inner/createIndex

通过 kibana 查看 mapping



重点看：attrs 数据类型必须是 nested！

二、商品上架，下架

构建 goods 数据模型分析

Sku 基本信息（详情业务已封装了接口）

Sku 分类信息（详情业务已封装了接口）

Sku 的品牌信息（无）

Sku 对应的平台属性（详情业务已封装了接口）

2.1 在 service-product 封装接口

2.1.1 Sku 的品牌接口

ManageService

```
/**
 * 通过品牌 Id 来查询数据
 * @param tmId
 * @return
 */
BaseTrademark getTrademarkByTmId(Long tmId);
```

实现类

```
@Autowired
private BaseTrademarkMapper baseTrademarkMapper;


@Override
public BaseTrademark getTrademarkByTmId(Long tmId) {
    return baseTrademarkMapper.selectById(tmId);
}
```

ProductApiController

```
/**
 * 通过品牌 Id 集合来查询数据
 * @param tmId
 * @return
 */
@GetMapping("inner/getTrademark/{tmId}")
public BaseTrademark getTrademark(@PathVariable("tmId")Long tmId){
    return manageService.getTrademarkByTmId(tmId);
}
```

2.2 在 service-product-client 添加接口

ProductFeignClient

```
/**
 * 通过品牌 Id 集合来查询数据
 * @param tmId
```

```
 * @return
 */
@GetMapping("/api/product/inner/getTrademark/{tmId}")
BaseTrademark getTrademark(@PathVariable("tmId")Long tmId);
```

ProductDegradeFeignClient

```
@Override
public BaseTrademark getTrademark(Long tmId) {
    return null;
}
```

2.3 实现商品上架，下架功能

2.3.1 封装接口

```
package com.atguigu.gmall.list.service;


public interface SearchService {

    /**
     * 上架商品列表
   * @param skuId
     */
    void upperGoods(Long skuId);

    /**
     * 下架商品列表
   * @param skuId
     */
    void lowerGoods(Long skuId);

}
```

2.3.2 制作操作 es 的接口

```
package com.atguigu.gmall.list.repository;
```

```
import com.atguigu.gmall.model.list.Goods;
import
org.springframework.data.elasticsearch.repository.ElasticsearchReposito
ry;

public interface GoodsRepository extends
ElasticsearchRepository<Goods,Long> {

}
```

2.3.3 接口实现类

实现类

```
package com.atguigu.gmall.list.service.impl;

@Service
public class SearchServiceImpl implements SearchService {


@Autowired
private ProductFeignClient productFeignClient;


@Autowired
private GoodsRepository goodsRepository;


 /**
 * 上架商品列表
 * @param skuId
 */
@Override
public void upperGoods(Long skuId) {
    Goods goods = new Goods();
    //查询sku对应的平台属性
    List<BaseAttrInfo> baseAttrInfoList =
productFeignClient.getAttrList(skuId);
    if(null != baseAttrInfoList) {
        List<SearchAttr> searchAttrList =
baseAttrInfoList.stream().map(baseAttrInfo -> {
            SearchAttr searchAttr = new SearchAttr();
            searchAttr.setAttrId(baseAttrInfo.getId());
            searchAttr.setAttrName(baseAttrInfo.getAttrName());
            //一个sku只对应一个属性值
            List<BaseAttrValue> baseAttrValueList =
baseAttrInfo.getAttrValueList();

searchAttr.setAttrValue(baseAttrValueList.get(0).getValueName());
```

```
            return searchAttr;
        }).collect(Collectors.toList());

        goods.setAttrs(searchAttrList);
    }

    //查询sku信息
    SkuInfo skuInfo = productFeignClient.getSkuInfo(skuId);
    // 查询品牌
    BaseTrademark baseTrademark = productFeignClient.getTrademark(skuInfo.getTmId());
    if (baseTrademark != null){
        goods.setTmId(skuInfo.getTmId());
        goods.setTmName(baseTrademark.getTmName());

         goods.setTmLogoUrl(trademark.getLogoUrl());

    }

    // 查询分类
    BaseCategoryView baseCategoryView = productFeignClient.getCategoryView(skuInfo.getCategory3Id());
    if (baseCategoryView != null) {
        goods.setCategory1Id(baseCategoryView.getCategory1Id());
        goods.setCategory1Name(baseCategoryView.getCategory1Name());
        goods.setCategory2Id(baseCategoryView.getCategory2Id());
        goods.setCategory2Name(baseCategoryView.getCategory2Name());
        goods.setCategory3Id(baseCategoryView.getCategory3Id());
        goods.setCategory3Name(baseCategoryView.getCategory3Name());
    }

    goods.setDefaultImg(skuInfo.getSkuDefaultImg());
    goods.setPrice(skuInfo.getPrice().doubleValue());
    goods.setId(skuInfo.getId());
    goods.setTitle(skuInfo.getSkuName());
    goods.setCreateTime(new Date());

    this.goodsRepository.save(goods);
}

/**
 * 下架商品列表
 * @param skuId
 */
@Override
public void lowerGoods(Long skuId) {
    this.goodsRepository.deleteById(skuId);
}

}
```

2.3.4 控制器

```
package com.atguigu.gmall.list.controller;


/**
 * <p>
 * 商品搜索列表接口
 * </p>
 *
 */
@RestController
@RequestMapping("api/list")
public class ListApiController {

    @Autowired
    private SearchService searchService;

    @Autowired
    private ElasticsearchRestTemplate restTemplate;


    /**
     * 上架商品
   * @param skuId
     * @return
     */
    @GetMapping("inner/upperGoods/{skuId}")
    public Result upperGoods(@PathVariable("skuId") Long skuId) {
        searchService.upperGoods(skuId);
        return Result.ok();
    }

    /**
     * 下架商品
   * @param skuId
     * @return
     */
    @GetMapping("inner/lowerGoods/{skuId}")
    public Result lowerGoods(@PathVariable("skuId") Long skuId) {
        searchService.lowerGoods(skuId);
        return Result.ok();
    }

}
```

添加数据

通过 kibana 查看数据

说明：后期学习了 mq，我们可以根据后台系统添加和修改等操作，发送 mq 消息自动上下架商品

http://localhost:8203/api/list/inner/upperGoods/10

http://localhost:8203/api/list/inner/lowerGoods/10

三、商品热度排名

搜索商品时，后面我们会根据热点排序，何时更新热点？我们在获取商品详情时调用更新

3.1 封装接口与实现类与控制器

SearchService

```
/**
 * 更新热点
 * @param skuId
 */
void incrHotScore(Long skuId);
```

实现类

```
@Autowired
private RedisTemplate redisTemplate;


@Override
public void incrHotScore(Long skuId) {
    // 定义key
    String hotKey = "hotScore";
    // 保存数据
  Double hotScore = redisTemplate.opsForZSet().incrementScore(hotKey,
"skuId:" + skuId, 1);
    if (hotScore%10==0){
        // 更新es
        Optional<Goods> optional = goodsRepository.findById(skuId);
```

```
        Goods goods = optional.get();
        goods.setHotScore(Math.round(hotScore));
        goodsRepository.save(goods);
    }
}
```

控制器

```
ListApiController

/**
 * 更新商品 incrHotScore
 *
 * @param skuId
 * @return
 */
@GetMapping("inner/incrHotScore/{skuId}")
public Result incrHotScore(@PathVariable("skuId") Long skuId) {
    // 调用服务层
  searchService.incrHotScore(skuId);
    return Result.ok();
}
```

3.2 在 service-list-client 封装接口

3.2.1 搭建 service-list-client

搭建方式如 service-item-client

3.2.2 修改 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.atguigu.gmall</groupId>
        <artifactId>service-client</artifactId>
        <version>1.0</version>
    </parent>
```

```
    <artifactId>service-list-client</artifactId>
    <version>1.0</version>

    <packaging>jar</packaging>
    <name>service-list-client</name>
    <description>service-list-client</description>

</project>
```

3.2.3 添加接口

```
package com.atguigu.gmall.list.client;


@FeignClient(value = "service-list", fallback =
ListDegradeFeignClient.class)
public interface ListFeignClient {

   /**
 * 更新商品incrHotScore
 * @param skuId
 * @return
 */
@GetMapping("/api/list/inner/incrHotScore/{skuId}")
Result incrHotScore(@PathVariable("skuId") Long skuId);

}
```

```
package com.atguigu.gmall.list.client.impl;


@Component
public class ListDegradeFeignClient implements ListFeignClient {

    @Override
public Result incrHotScore(Long skuId) {
    return null;
}
}
```

3.3 在 service-item 模块调用接口

引入依赖

```
<dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>service-list-client</artifactId>
    <version>1.0</version>
</dependency>
```

接口调用

```
@Service
public class ItemServiceImpl implements ItemService {

    @Autowired
    private ProductFeignClient productFeignClient;

    @Autowired
    private ListFeignClient listFeignClient;

    @Autowired
    private ThreadPoolExecutor threadPoolExecutor;

    @Override
    public Map<String, Object> getBySkuId(Long skuId) {


        Map<String, Object> result = new HashMap<>();

       ...


        //获取分类信息
    CompletableFuture<Void>      categoryViewCompletableFuture      =
skuCompletableFuture.thenAcceptAsync(skuInfo -> {
            BaseCategoryView               categoryView               =
productFeignClient.getCategoryView(skuInfo.getCategory3Id());

            //分类信息
      result.put("categoryView", categoryView);
        }, threadPoolExecutor);

        //更新商品 incrHotScore
        CompletableFuture<Void>    incrHotScoreCompletableFuture    =
CompletableFuture.runAsync(() -> {
            listFeignClient.incrHotScore(skuId);
        }, threadPoolExecutor);

        CompletableFuture.allOf(skuCompletableFuture,
spuSaleAttrCompletableFuture,
skuValueIdsMapCompletableFuture,skuPriceCompletableFuture,
```

```
categoryViewCompletableFuture,
incrHotScoreCompletableFuture).join();
        return result;
    }
}
```