尚品汇商城复习

版本：V 1.0

商品搜索模块

一、商品检索功能介绍

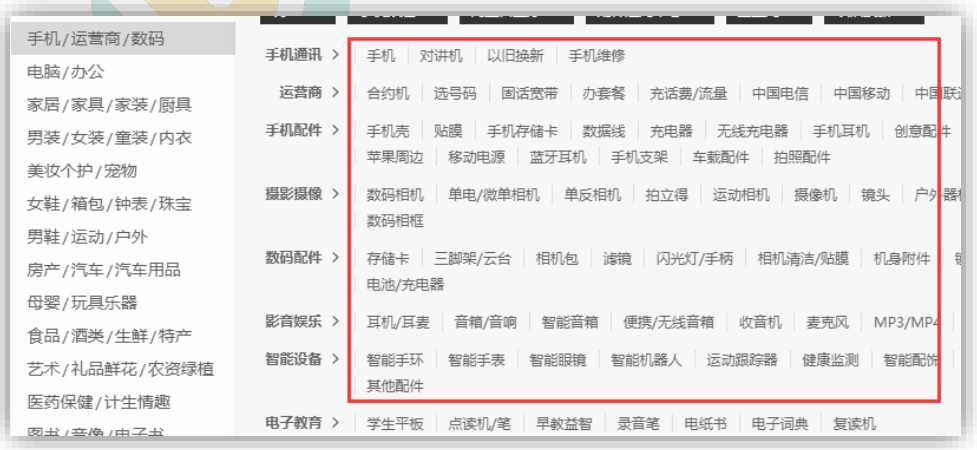
1、功能简介

什么是搜索，计算机根据用户输入的关键词进行匹配，从已有的数据库中摘录出相关的记录反馈给用户。

常见的全网搜索引擎，像百度、谷歌这样的。但是除此以外，搜索技术在垂直领域也有广泛的使用，比如淘宝、京东搜索商品，万芳、知网搜索期刊，csdn 中搜索问题贴。也都是基于海量数据的搜索。

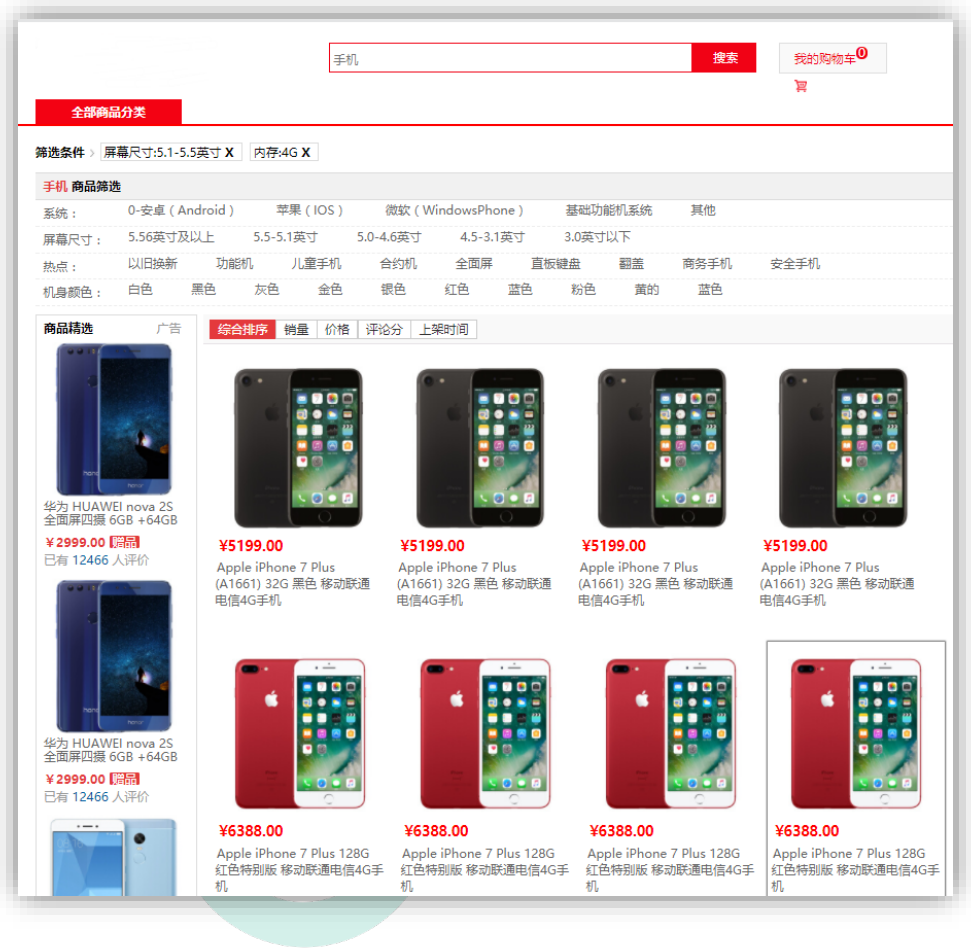
1.1、入口： 两个

首页的分类



搜索栏

1.2、列表展示页面



2、全文检索工具 elasticsearch

2.1、lucene 与 elasticsearch（solr）

lucene 只是一个提供全文搜索功能类库的核心工具包，而真正使用它还需要一个完善的服务框架搭建起来的应用。

好比 lucene 是类似于 jdk，而搜索引擎软件就是 tomcat 的。

目前市面上流行的搜索引擎软件，主流的就两款，elasticsearch 和 solr,这两款都是基于 lucene 的搭建的，可以独立部署启动的搜索引擎服务软件。由于内核相同，所以两者除了服务器安装、部署、管理、集群以外，对于数据的操作，修改、添加、保存、查询等等都十分类似。就好像都是支持 sql 语言的两种数据库软件。只要学会其中一个另一个很容易上手。

从实际企业使用情况来看，elasticSearch 的市场份额逐步在取代 solr，国内百度、京东、新浪都是基于 elasticSearch 实现的搜索功能。国外就更多了 像维基百科、GitHub、Stack Overflow 等等也都是基于 ES 的。

2.2、elasticSearch 的使用场景

1、为用户提供按关键字查询的全文搜索功能。

2、著名的 ELK 框架(ElasticSearch,Logstash,Kibana)，实现企业海量日志的处理分析的解决方案。大数据领域的重要一份子。

2.3、elasticsearch 的基本概念

cluster	整个 elasticsearch 默认就是集群状态，整个集群是一份完整、互备的数据。
node	集群中的一个节点，一般只一个进程就是一个 node
shard	分片，即使是一个节点中的数据也会通过 hash 算法，分成多个片存放，默认是 5 片。
index	相当于 rdbms 的 database, 对于用户来说是一个逻辑数据库，虽然物理上会被分多个 shard 存放，也可能存放在多个 node 中。
type	类似于 rdbms 的 table，但是与其说像 table，其实更像面向对象中的 class，同一 Json 的格式的数据集合。
document	类似于 rdbms 的 row、面向对象里的 object
field	相当于字段、属性

2.4、中文分词

elasticsearch本身自带的中文分词，就是单纯把中文一个字一个字的分开，根本没有词汇的概念。但是实际应用中，用户都是以词汇为条件，进行查询匹配的，如果能够把文章以词汇为单位切分开，那么与用户的查询条件能够更贴切的匹配上，查询速度也更加快速。

分词器下载网址：https://github.com/medcl/elasticsearch-analysis-ik

3、根据业务搭建数据结构

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 红米 手机 K30Pro

2、我们用哪些字段进行过滤（当做查询的条件）

a) 平台属性值

b) 分类 Id

c) 品牌、价格区间、热度、评论、销量

3、哪些字段我们需要通过搜索查询出来。

a) Id(隐藏)，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：info

document: properties - rows

field: id,price,title…

Es 中 index 默认是 true。

info= Type

对应的 mapping 结构：

```
Put
{
  "goods" : {
    "mappings" : {
      "skuInfo" : {
        "properties" : {
          "attrs" : {
            "type" : "nested",
            "properties" : {
              "attrId" : {
                "type" : "long"
              },
              "attrName" : {
                "type" : "keyword"
              },
              "attrValue" : {
                "type" : "keyword"
              }
            }
          },
          "category1Id" : {
            "type" : "long"
          },
          "category1Name" : {
            "type" : "keyword"
```

```
      },
      "category2Id" : {
        "type" : "long"
      },
      "category2Name" : {
        "type" : "keyword"
      },
      "category3Id" : {
        "type" : "long"
      },
      "category3Name" : {
        "type" : "keyword"
      },
      "createTime" : {
        "type" : "date"
      },
      "defaultImg" : {
        "type" : "keyword",
        "index" : false
      },
      "hotScore" : {
        "type" : "long"
      },
      "id" : {
        "type" : "long"
      },
      "price" : {
        "type" : "double"
      },
```

```
        "title" : {
          "type" : "text",
          "analyzer" : "ik_max_word"//中华人民共和国人民大会堂
中华 华人 人民 共和 共和国  国人 大会 大会堂 会堂
          ik _word  中华人民共和国 人民大会堂
        },
        "tmId" : {
          "type" : "long"
        },
        "tmLogoUrl" : {
          "type" : "keyword"
        },
        "tmName" : {
          "type" : "keyword"
        }
      }
    }
  }
}
```

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。

3.2 构建实体与 es mapping 建立映射关系

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

@Document(indexName = "goods", type = "info", shards = 3, replicas = 2)
```

```
@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)
    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;

}
```

```
@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;
}
```

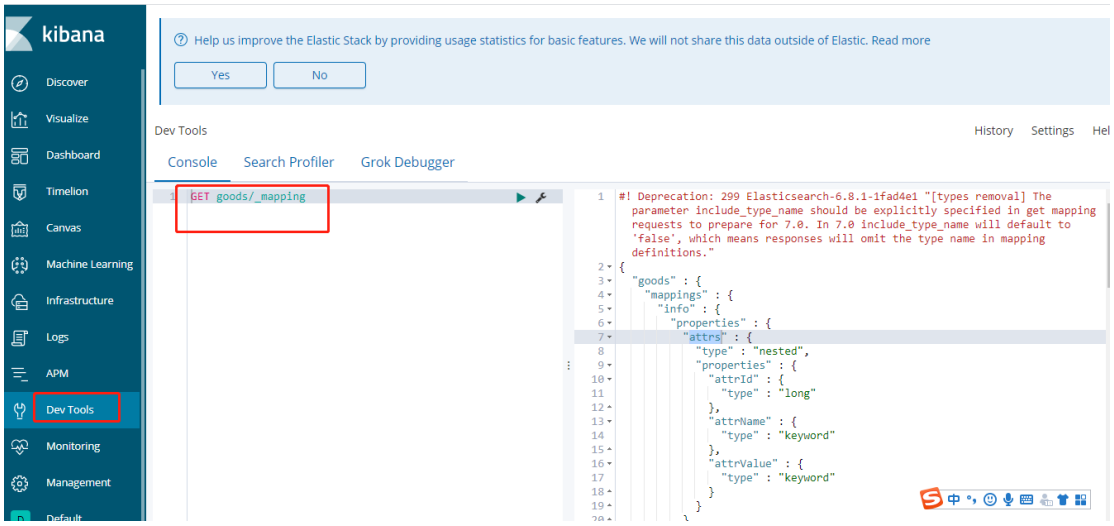
3.2 初始化 mapping 结构到 es 中

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

@RestController
@RequestMapping("api/list")
public class ListApiController {Jest
    @Autowired
    private ElasticsearchRestTemplate restTemplate;
    /**
     *
     * @return
     */
    @GetMapping("inner/createIndex")
    public Result createIndex() {
        restTemplate.createIndex(Goods.class);
        restTemplate.putMapping(Goods.class);
        return Result.ok();
    }

}
```

通过 kibana 查看 mapping



二、商品上架，下架

封装商品上下架接口

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);
}

}
```

```
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();
    }

}
```

三、商品热度排名

es 查询的 dsl 语句中我们是用了 hotScore 来进行排序的。

但是 hotScore 从何而来，根据业务去定义，也可以扩展更多类型的评分，让用户去选择如何排序。

这里的 hotScore 我们假定以点击量来决定热度。

那么我们每次用户点击，将这个评分+1。

1、问题

1、 es 大量的写操作会影响 es 性能，因为 es 需要更新索引，而且 es 不是内存数据库，会做相应的 io 操作。

2、而且修改某一个值，在高并发情况下会有冲突，造成更新丢失，需要加锁，而 es 的乐观锁会恶化性能问题。

从业务角度出发，其实我们为商品进行排序所需要的热度评分，并不需要非常精确，大致能比出个高下就可以了。

利用这个特点我们可以稀释掉大量写操作。

2、解决思路

用 redis 做精确计数器，redis 是内存数据库读写性能都非常快，利用 redis 的原子性的自增可以解决并发写操作。

redis 每计 10 或 100 次数（可以被 10 或 100 整除）我们就更新一次 es ，这样写操作就被稀释了 10-100 倍，这个倍数可以根据业务情况灵活设定。

3、搜索封装更新热度排名接口

SearchService 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%100==0){
        // 更新 es
        Optional<Goods> optional = goodsRepository.findById(skuId);
        Goods goods = optional.get();
        goods.setHotScore(Math.round(hotScore));
        goodsRepository.save(goods);
    }
}
```

```
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();
}
```

4、在 service-item 模块调用接口

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);

        //用户每次访问商品详情时，更新商品热度 incrHotScore
```

```
        CompletableFuture<Void>    incrHotScoreCompletableFuture    =
CompletableFuture.runAsync(() -> {
            listFeignClient.incrHotScore(skuId);
        }, threadPoolExecutor);

        CompletableFuture.allOf(skuCompletableFuture,
spuSaleAttrCompletableFuture,
skuValueIdsMapCompletableFuture,skuPriceCompletableFuture,
categoryViewCompletableFuture,
incrHotScoreCompletableFuture).join();
        return result;
    }
}
```

四、常见面试问题

1、你们项目在 es 里存了哪些数据?

Goods 类中装的属性:skuid\名称\价格\默认图片\分类信息 1-3 级\品牌信息(id name logo)\平台属性(聚合)\热度

商品名称: 要分词 要和关键字匹配

搜索条件:除了 skuid 和图片 剩下的都是搜索的条件

入口: 1\首页的分类 2\搜索框的关键字

执行搜索的过程:

1、构建查询的语句，使用 rest 客户端进行查询条件的拼装，指定要查的文档库，指定要展示的结果。

2、执行查询

3、解析结果，解析出品牌的聚合和平台属性的，这俩是展示给用户作为筛选条件，让用户选择。

2、ES 的倒排索引? （重要）

正排索引：根据 id 去找到相对应的词

Id（索引）	词
1	红海行动
2	红海事件
3	红海行动事件
4	湄公河行动

倒排索引：根据词，看这个词都在哪个 id（索引）里出现了

词（商品名称）	Id（索引）
红海 行动	红海 1、2、3；行动 1、3、4
红海 事件	红海 1、2、3；事件 2、3
红海 行动 事件	红海 1、2、3；行动 1、3、4；事件 2、3
湄公河 行动	湄公河 4；行动 1、3、4

3、 ES 数据同步问题怎么处理?

数据库中 商品数据 ES 中也有商品数据。他俩一样的。

后台管理系统操作数据库 进行商品数据修改了，ES 中还是原来的数据。

Skuid =21 的商品 1999

有一天 价格改了 1999 2199

分布式事务的体现。

解决:

1、 可不可以直接在商品服务中 写操作 ES 代码

2、 远程调用搜索服务

消息队列

消息最终一致性。