基于Elasticsearch实现搜索推荐

2017-03-21 | Elasticsearch | 阅读次数 3472

在基于Elasticsearch实现搜索建议一文中我们曾经介绍过如何基于Elasticsearch来实现搜索建议，而本文是在此基础上进一步优化搜索体验，在当搜索无结果或结果过少时提供推荐搜索词给用户。

背景介绍

在根据用户输入和筛选条件进行搜索后，有时返回的是无结果或者结果很少的情况，为了提升用户搜索体验，需要能够给用户推荐一些相关的搜索词，比如用户搜索【迪奥】时没有找到相关的商品，可以推荐搜索【香水】、【眼镜】等关键词。

设计思路

首先需要分析搜索无结果或者结果过少可能的原因，我总结了一下，主要包括主要可能：

1. 搜索的关键词在本网不存在，比如【迪奥】；
2. 搜索的关键词在本网的商品很少，比如【科比】；
3. 搜索的关键词拼写有问题，比如把【阿迪达斯】写成了【阿迪大斯】；
4. 搜索的关键词过多，由于我们采用的是cross_fields，在一个商品内不可能包含所有的Term，导致无结果，比如【阿迪达斯 耐克 卫衣 运动鞋】；

那么针对以上情况，可以采用以下方式进行处理：

1. 搜索的关键词在本网不存在，可以通过爬虫的方式获取相关知识，然后根据**搜索建议词**去提取，比如去百度百科的迪奥词条里就能提取出【香水】、【香氛】和【眼镜】等关键词；当然基于爬虫的知识可能存在偏差，此时需要能够有人工审核或人工更正的部分；
2. 搜索的关键词在本网的商品很少，有两种解决思路，一种是通过方式1的爬虫去提取关键词，另外一种是通过返回商品的信息去聚合出关键词，如品牌、品类、风格、标签等，这里我们采用的是后者（在测试后发现后者效果更佳）；
3. 搜索的关键词拼写有问题，这就需要**拼写纠错**出场了，先纠错然后根据纠错后的词去提供搜索推荐；
4. 搜索的关键词过多，有两种解决思路，一种是识别关键词的类型，如是品牌、品类、风格还是性别，然后通过一定的组合策略来实现搜索推荐；另外一种则是根据用户的输入到搜索建议词里去匹配，设置最

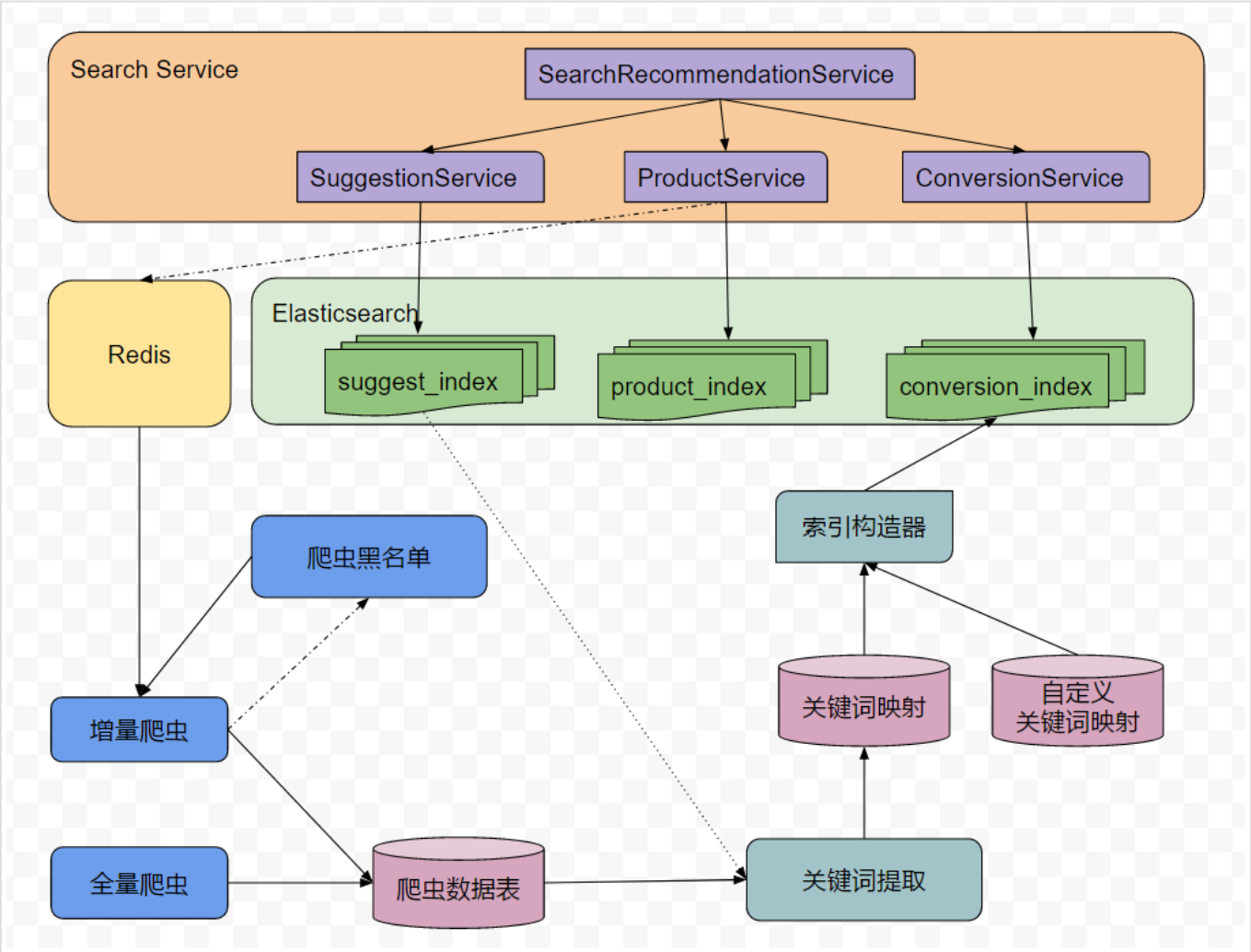
小匹配为一个匹配到一个Term即可，这种方式实现比较简单而且效果也不错，所以我们采用的是后者。

所以，我们在实现搜索推荐的核心是之前讲到的搜索建议词，它提供了本网主要的关键词，另外一个很重要的是它本身包含了**关联商品数的属性**，这样就可以保证推荐给用户的关键词是可以搜索出结果的。

实现细节

整体设计

整体设计框架如下图所示：



搜索建议词索引

在基于Elasticsearch实现搜索建议一文已有说明，请移步阅读。此次增加了一个keyword.keyword_lowercase的字段用于拼写纠错，这里列取相关字段的索引：

```
PUT /suggest_index
{
  "mappings": {
    "suggest": {
```

```
        "properties": {
          "keyword": {
            "fields": {
              "keyword": {
                "type": "string",
                "index": "not_analyzed"
              },
              "keyword_lowercase": {
                "type": "string",
                "analyzer": "lowercase_keyword"
              },
              "keyword_ik": {
                "type": "string",
                "analyzer": "ik_smart"
              },
              "keyword_pinyin": {
                "type": "string",
                "analyzer": "pinyin_analyzer"
              },
              "keyword_first_py": {
                "type": "string",
                "analyzer": "pinyin_first_letter_keyword_analyzer"
              }
            },
            "type": "multi_field"
          },
          "type": {
            "type": "long"
          },
          "weight": {
            "type": "long"
          },
          "count": {
            "type": "long"
          }
        }
      }
    }
  }
```

商品数据索引

这里只列取相关字段的mapping：

```
PUT /product_index
{
  "mappings": {
    "product": {
      "properties": {
        "productSkn": {
```

```
          "type": "long"
        },
        "productName": {
          "type": "string",
          "analyzer": "ik_smart"
        },
        "brandName": {
          "type": "string",
          "analyzer": "ik_smart"
        },
        "sortName": {
          "type": "string",
          "analyzer": "ik_smart"
        },
        "style": {
          "type": "string",
          "analyzer": "ik_smart"
        }
      }
    }
  }
}
```

关键词映射索引

主要就是source和dest直接的映射关系。

```
PUT /conversion_index
{
  "mappings": {
    "conversion": {
      "properties": {
        "source": {
          "type": "string",
          "analyzer": "lowercase_keyword"
        },
        "dest": {
          "type": "string",
          "index": "not_analyzed"
        }
      }
    }
  }
}
```

爬虫数据爬取

在实现的时候，我们主要是爬取了百度百科上面的词条，在实际的实现中又分为了全量爬虫和增加爬虫。

全量爬虫

全量爬虫我这边是从网上下载了一份他人汇总的词条URL资源，里面根据一级分类包含多个目录，每个目录又根据二级分类包含多个词条，每一行的内容的格式如下：

```
李宁!http://baike.baidu.com/view/10670.html?fromTaglist
diesel!http://baike.baidu.com/view/394305.html?fromTaglist
ONLY!http://baike.baidu.com/view/92541.html?fromTaglist
lotto!http://baike.baidu.com/view/907709.html?fromTaglist
```

这样在启动的时候我们就可以使用多线程甚至分布式的方式爬虫自己感兴趣的词条内容作为初始化数据保持到爬虫数据表。为了保证幂等性，如果再次全量爬取时就需要排除掉数据库里已有的词条。

增量爬虫

1. 在商品搜索接口中，如果搜索某个关键词关联的商品数为0或小于一定的阈值（如20条），就通过Redis的ZSet进行按天统计；
2. 统计的时候是区分搜索无结果和结果过少两个Key的，因为两种情况实际上是有所区别的，而且后续在搜索推荐查询时也有用到这个统计结果；
3. 增量爬虫是每天凌晨运行，根据前一天统计的关键词进行爬取，爬取前需要排除掉已经爬过的关键词和黑名单中的关键词；
4. 所谓黑名单的数据包含两种：一种是每天增量爬虫失败的关键字（一般会重试几次，确保失败后加入黑名单），一种是人工维护的确定不需要爬虫的关键词；

爬虫数据关键词提取

1. 首先需要明确关键词的范围，这里我们采用的是suggest中类型为品牌、品类、风格、款式的词作为关键词；
2. 关键词提取的核心步骤就是对爬虫内容和关键词分别分词，然后进行分词匹配，看该爬虫数据是否包含关键词的所有Term（如果就是一个Term就直接判断包含就好了）；在处理的时候还可以对匹配到关键词的次数进行排序，最终的结果就是一个key-value的映射，如{迪奥 -> [香水,香氛,时装,眼镜], 纪梵希 -> [香水,时装,彩妆,配饰,礼服]}；

管理关键词映射

1. 由于爬虫数据提取的关键词是和词条的内容相关联的，因此很有可能提取的关键词效果不大好，因此就需要人工管理；
2. 管理动作主要是包括添加、修改和置失效关键词映射，然后增量地更新到conversion_index索引中；

搜索推荐服务的实现

1. 首先如果对搜索推荐的入口进行判断，一些非法的情况不进行推荐（比如关键词太短或太长），另外由于搜索推荐并非核心功能，可以增加一个全局动态参数来控制是否进行搜索推荐；
2. 在**设计思路**里面我们分析过可能有4中场景需要搜索推荐，如何高效、快速地找到具体的场景从而减少不必要的查询判断是推荐服务实现的关键；这个在设计的时候就需要综合权衡，我们通过一段时间的观察后，目前采用的逻辑的伪代码如下：

```
public JSONObject recommend(SearchResult searchResult, String queryWord) {
    try {
        String keywordsToSearch = queryWord;

        // 搜索推荐分两部分
        // 1) 第一部分是最常见的情况，包括有结果、根据SKN搜索、关键词未出现在空结果Redis ZS
        if (containsProductInSearchResult(searchResult)) {
            // 1.1)  搜索有结果的 优先从搜索结果聚合出品牌等关键词进行查询
            String aggKeywords = aggKeywordsByProductList(searchResult);
            keywordsToSearch = queryWord + " " + aggKeywords;
        } else if (isQuerySkn(queryWord)) {
            // 1.2)  如果是查询SKN 没有查询到的 后续的逻辑也无法推荐 所以直接到ES里去获取关
            keywordsToSearch = aggKeywordsBySkns(queryWord);
            if (StringUtils.isEmpty(keywordsToSearch)) {
                return defaultSuggestRecommendation();
            }
        }

        Double count = searchKeyWordService.getKeywordCount(RedisKeys.SEARCH_KEY
        if (count == null || queryWord.length() >= 5) {
            // 1.3) 如果该关键词一次都没有出现在空结果列表或者长度大于5 则该词很有可能是可以
            //      因此优先取suggest_index去搜索一把 减少后面的查询动作
            JSONObject recommendResult = recommendBySuggestIndex(queryWord, keyw
            if (isNotEmptyResult(recommendResult)) {
                return recommendResult;
            }
        }

        // 2) 第二部分是通过Conversion和拼写纠错去获取关键词 由于很多品牌的拼写可能比较相近
        String spellingCorrentWord = null, dest = null;
        if (allowGetingDest(queryWord) && StringUtils.isNotEmpty((dest = getSugg
            // 2.1) 爬虫和自定义的Conversion处理
            keywordsToSearch = dest;
        } else if (allowSpellingCorrent(queryWord)
                        && StringUtils.isNotEmpty((spellingCorrentWord = sugges
            // 2.2) 执行拼写检查 由于在搜索建议的时候会进行拼写检查 所以缓存命中率高
            keywordsToSearch = spellingCorrentWord;
        } else {
            // 2.3) 如果两者都没有 则直接返回
            return defaultSuggestRecommendation();
```

```
41          }
42
43          JSONObject recommendResult = recommendBySuggestIndex(queryWord, keywords
44          return isNotEmptyResult(recommendResult) ? recommendResult : defaultSugg
45      } catch (Exception e) {
46          logger.error("[func=recommend][queryWord=" + queryWord + "]", e);
47          return defaultSuggestRecommendation();
48      }
49  }
```

其中涉及到的几个函数简单说明下：

- aggKeywordsByProductList方法用商品列表的结果，聚合出出现次数最多的几个品牌和品类（比如各2个），这样我们就可以得到4个关键词，和原先用户的输入拼接后调用recommendBySuggestIndex获取推荐词；
- aggKeywordsBySkns方法是根据用户输入的SKN先到product_index索引获取商品列表，然后再调用aggKeywordsByProductList去获取品牌和品类的关键词列表；
- getSuggestConversionDestBySource方法是查询conversion_index索引去获取关键词提取的结果，这里在调用recommendBySuggestIndex时有个参数，该参数主要是用于处理是否限制只能是输入的关键词；
- getSpellingCorrectKeyword方法为拼写检查，在调用suggest_index处理时有个地方需要注意一下，拼写检查是基于编辑距离的，大小写不一致的情况会导致Elasticsearch Suggester无法得到正确的拼写建议，因此在处理时需要两边都转换为小写后进行拼写检查；
- 最终都需要调用recommendBySuggestIndex方法获取搜索推荐，因为通过suggest_index索引可以确保推荐出去的词是有意义的且关联到商品的。该方法核心逻辑的伪代码如下：

```
1   private JSONObject recommendBySuggestIndex(String srcQueryWord, String keywords
2       // 1) 先对keywordsToSearch进行分词
3       List<String> terms = null;
4       if (isLimitKeywords) {
5           terms = Arrays.stream(keywordsToSearch.split(",")).filter(term -> term
6                       .distinct().collect(Collectors.toList());
7       } else {
8           terms = searchAnalyzeService.getAnalyzeTerms(keywordsToSearch, "ik_smar
9       }
10
11      if (CollectionUtils.isEmpty(terms)) {
12          return new JSONObject();
13      }
14
15      // 2) 根据terms搜索构造搜索请求
16      SearchParam searchParam = new SearchParam();
17      searchParam.setPage(1);
18      searchParam.setSize(3);
19
```

```
20      // 2.1) 构建FunctionScoreQueryBuilder
21      QueryBuilder queryBuilder = isLimitKeywords ? buildQueryBuilderByLimit(term
22                                      : buildQueryBuilder(keywordsToSearch, terms);
23      searchParam.setQuery(queryBuilder);
24
25      // 2.2) 设置过滤条件
26      BoolQueryBuilder boolFilter = QueryBuilders.boolQuery();
27      boolFilter.must(QueryBuilders.rangeQuery("count").gte(20));
28      boolFilter.mustNot(QueryBuilders.termQuery("keyword.keyword_lowercase", src
29      if (isLimitKeywords) {
30          boolFilter.must(QueryBuilders.termsQuery("keyword.keyword_lowercase", t
31              .map(String::toLowerCase).collect(Collectors.toList())));
32      }
33      searchParam.setFiter(boolFilter);
34
35      // 2.3) 按照得分、权重、数量的规则降序排序
36      List<SortBuilder> sortBuilders = new ArrayList<>(3);
37      sortBuilders.add(SortBuilders.fieldSort("_score").order(SortOrder.DESC));
38      sortBuilders.add(SortBuilders.fieldSort("weight").order(SortOrder.DESC));
39      sortBuilders.add(SortBuilders.fieldSort("count").order(SortOrder.DESC));
40      searchParam.setSortBuilders(sortBuilders);
41
42      // 4) 先从缓存中获取
43      final String indexName = SearchConstants.INDEX_NAME_SUGGEST;
44      JSONObject suggestResult = searchCacheService.getJSONObjectFromCache(indexN
45      if (suggestResult != null) {
46          return suggestResult;
47      }
48
49      // 5) 调用ES执行搜索
50      SearchResult searchResult = searchCommonService.doSearch(indexName, searchF
51
52      // 6) 构建结果加入缓存
53      suggestResult = new JSONObject();
54      List<String> resultTerms = searchResult.getResultList().stream()
55              .map(map -> (String) map.get("keyword")).collect(Collectors.toList(
56      suggestResult.put("search_recommendation", resultTerms);
57      searchCacheService.addJSONObjectToCache(indexName, searchParam, suggestResu
58      return suggestResult;
59  }
60
61  private QueryBuilder buildQueryBuilderByLimit(List<String> terms) {
62      FunctionScoreQueryBuilder functionScoreQueryBuilder
63          = new FunctionScoreQueryBuilder(QueryBuilders.matchAllQuery());
64
65      // 给品类类型的关键词加分
66      functionScoreQueryBuilder.add(QueryBuilders.termQuery("type", Integer.value
67          ScoreFunctionBuilders.weightFactorFunction(3));
68
69      // 按词出现的顺序加分
70      for (int i = 0; i < terms.size(); i++) {
71          functionScoreQueryBuilder.add(QueryBuilders.termQuery("keyword.keyword_
```

```
    terms.get(i).toLowerCase()),
            ScoreFunctionBuilders.weightFactorFunction(terms.size() - i));
    }

    functionScoreQueryBuilder.boostMode(CombineFunction.SUM);
    return functionScoreQueryBuilder;
}

private QueryBuilder buildQueryBuilder(String keywordsToSearch, Set<String> ter
    // 1) 对于suggest的multi-fields至少要有一个字段匹配到 匹配得分为常量1
    MultiMatchQueryBuilder queryBuilder = QueryBuilders.multiMatchQuery(keyworc
            "keyword.keyword_ik", "keyword.keyword_pinyin",
            "keyword.keyword_first_py", "keyword.keyword_lowercase")
        .analyzer("ik_smart")
        .type(MultiMatchQueryBuilder.Type.BEST_FIELDS)
        .operator(MatchQueryBuilder.Operator.OR)
        .minimumShouldMatch("1");

    FunctionScoreQueryBuilder functionScoreQueryBuilder
        = new FunctionScoreQueryBuilder(QueryBuilders.constantScoreQuery(queryE

    for (String term : termSet) {
        // 2) 对于完全匹配Term的加1分
        functionScoreQueryBuilder.add(QueryBuilders.termQuery("keyword.keyword_
            ScoreFunctionBuilders.weightFactorFunction(1));

        // 3) 对于匹配到一个Term的加2分
        functionScoreQueryBuilder.add(QueryBuilders.termQuery("keyword.keyword_
            ScoreFunctionBuilders.weightFactorFunction(2));
    }

    functionScoreQueryBuilder.boostMode(CombineFunction.SUM);
    return functionScoreQueryBuilder;
}
```

最后，从实际运行的统计来看，有90%以上的查询都能在1.3)的情况下返回推荐词，而这一部分还没有进行拼写纠错和conversion_index索引的查询，因此还是比较高效的；剩下的10%在最坏的情况且缓存都没有命中的情况下，最多还需要进行三次ES的查询，性能是比较差的，但是由于有缓存而且大部分的无结果的关键词都比较集中，因此也在可接受的范围，这一块可以考虑再增加一个动态参数，在大促的时候进行关闭处理。

小结与后续改进

- 通过以上的设计和实现，我们实现了一个效果不错的搜索推荐功能，线上使用效果如下：

```
//搜索【迪奥】，本站无该品牌商品
没有找到 "迪奥" 相关的商品，  为您推荐 "香水" 的搜索结果。或者试试 "香氛"  "眼镜"

//搜索【puma 运动鞋 上衣】，关键词太多无法匹配
```

```
没有找到 "puma 运动鞋 上衣" 相关的商品， 为您推荐 "PUMA 运动鞋" 的搜索结果。或者试试 "PUMA 运动鞋

//搜索【puma 上衣】，结果太少
"puma 上衣" 搜索结果太少了，试试 "上衣"  "PUMA"  "PUMA 休闲" 关键词搜索

//搜索【51489312】特定的SKN，结果太少
"51489312" 搜索结果太少了，试试 "夹克"  "PUMA"  "户外" 关键词搜索

//搜索【blackjauk】，拼写错误
没有找到 "blackjauk" 相关的商品， 为您推荐 "BLACKJACK" 的搜索结果。或者试试 "BLACKJACK T恤"  "
```

- 后续考虑的改进包括：1.继续统计各种无结果或结果太少场景出现的频率和对应推荐词的实现，优化搜索推荐服务的效率；2.爬取更多的语料资源，提升Conversion的准确性；3.考虑增加个性化的功能，给用户推荐Ta最感兴趣的内容。

扫一扫，关注我的微信公众号

Elasticsearch # 搜索 # 推荐 # 拼写纠错

‹ 一个简易的Elasticsearch动态同义词插件

基于word2vec和Elasticsearch实现个性化搜索 ›

5