



动手学习Elasticsearch中的Multi Match Query

📅 2016-12-23 | 📁 [Elasticsearch](#) | 📖 阅读次数 790

在Elasticsearch全文检索中，我们用的比较多的就是Multi Match Query，其支持对多个字段进行匹配。Elasticsearch支持5种类型的Multi Match，我们一起来深入学习下它们的区别。

5种类型的Multi Match Query

直接从官网的文档上摘抄一段来：

- `best_fields`: (default) Finds documents which match any field, but uses the `_score` from the best field.
- `most_fields`: Finds documents which match any field and combines the `_score` from each field.
- `cross_fields`: Treats fields with the same analyzer as though they were one big field. Looks for each word in any field.
- `phrase`: Runs a `match_phrase` query on each field and combines the `_score` from each field.
- `phrase_prefix`: Runs a `match_phrase_prefix` query on each field and combines the `_score` from each field.

这里我们只考虑前面三种，后两种可以另外单独研究，就先忽略了。

创建测试索引，预置测试数据

创建gino_product索引

```
PUT /gino_product
{
  "mappings": {
    "product": {
      "properties": {
        "productName": {
          "type": "string",
          "analyzer": "fulltext_analyzer",
          "copy_to": [
            "bigSearchField"
          ]
        },
      },
    },
  },
}
```



```
"brandName": {
  "type": "string",
  "analyzer": "fulltext_analyzer",
  "copy_to": [
    "bigSearchField"
  ],
  "fields": {
    "brandName_pinyin": {
      "type": "string",
      "analyzer": "pinyin_analyzer",
      "search_analyzer": "standard"
    },
    "brandName_keyword": {
      "type": "string",
      "analyzer": "keyword",
      "search_analyzer": "standard"
    }
  }
},
"sortName": {
  "type": "string",
  "analyzer": "fulltext_analyzer",
  "copy_to": [
    "bigSearchField"
  ],
  "fields": {
    "sortName_pinyin": {
      "type": "string",
      "analyzer": "pinyin_analyzer",
      "search_analyzer": "standard"
    }
  }
},
"productKeyword": {
  "type": "string",
  "analyzer": "fulltext_analyzer",
  "copy_to": [
    "bigSearchField"
  ]
},
"bigSearchField": {
  "type": "string",
  "analyzer": "fulltext_analyzer"
}
}
},
"settings": {
  "index": {
    "number_of_shards": 1,
    "number_of_replicas": 0
  },

```



```
"analysis": {
  "tokenizer": {
    "simple_pinyin": {
      "type": "pinyin",
      "first_letter": "none"
    }
  },
  "analyzer": {
    "fulltext_analyzer": {
      "type": "ik",
      "use_smart": true
    },
    "pinyin_analyzer": {
      "type": "custom",
      "tokenizer": "simple_pinyin",
      "filter": [
        "word_delimiter",
        "lowercase"
      ]
    }
  }
}
```

插入一些测试数据

```
POST /gino_product/product/1
{
  "productName": "耐克女生运动轻跑鞋",
  "brandName": "耐克",
  "sortName": "鞋子",
  "productKeyword": "耐克, 潮流, 运动, 轻跑鞋"
}

POST /gino_product/product/2
{
  "productName": "耐克女生休闲运动服",
  "brandName": "耐克",
  "sortName": "上衣",
  "productKeyword": "耐克, 休闲, 运动"
}

POST /gino_product/product/3
{
  "productName": "阿迪达斯女生冬季运动板鞋",
  "brandName": "阿迪达斯",
  "sortName": "鞋子",
  "productKeyword": "阿迪达斯, 冬季, 运动, 板鞋"
}
```



```
POST /gino_product/product/4
{
  "productName": "阿迪达斯女生冬季运动夹克外套",
  "brandName": "阿迪达斯",
  "sortName": "上衣",
  "productKeyword": "阿迪达斯, 冬季, 运动, 夹克, 外套"
}
```

测试数据总览

查询 1 个分片中用的 1 个. 4 命中. 耗时 0.002 秒

_index	_type	_id	_score ▼	productName	brandName	sortName	productKeyword
gino_product	product	1	1	耐克女生运动轻跑鞋	耐克	鞋子	耐克, 潮流, 运动, 轻跑鞋
gino_product	product	2	1	耐克女生休闲运动服	耐克	上衣	耐克, 休闲, 运动
gino_product	product	3	1	阿迪达斯女生冬季运动板鞋	阿迪达斯	鞋子	阿迪达斯, 冬季, 运动, 板鞋
gino_product	product	4	1	阿迪达斯女生冬季运动夹克外套	阿迪达斯	上衣	阿迪达斯, 冬季, 运动, 夹克, 外套

分别搜索【运动】

```
POST /gino_product/_search
{
  "query": {
    "multi_match": {
      "query": "运动",
      "fields": [
        "brandName^100",
        "brandName.brandName_pinyin^100",
        "brandName.brandName_keyword^100",
        "sortName^80",
        "sortName.sortName_pinyin^80",
        "productName^60",
        "productKeyword^20"
      ],
      "type": <multi-match-type>,
      "operator": "AND"
    }
  }
}
```

发现使用3种type都可以搜索出4条商品数据，而且排序也是一致的。

分别搜索【运动 上衣】

```
POST /gino_product/_search
{
  "query": {
```



```

"multi_match": {
  "query": "运动 上衣",
  "fields": [
    "brandName^100",
    "brandName.brandName_pinyin^100",
    "brandName.brandName_keyword^100",
    "sortName^80",
    "sortName.sortName_pinyin^80",
    "productName^60",
    "productKeyword^20"
  ],
  "type": <multi-match-type>,
  "operator": "AND"
}
}
}

```

这次搜索只有cross_field才能搜索出数据，而使用best_fields和most_fields不行，为什么？

使用**validate API**来比较区别

```

POST /gino_product/_validate/query?rewrite=true
{
  "query": {
    "multi_match": {
      "query": "运动 上衣",
      "fields": [
        "brandName^100",
        "brandName.brandName_pinyin^100",
        "brandName.brandName_keyword^100",
        "sortName^80",
        "sortName.sortName_pinyin^80",
        "productName^60",
        "productKeyword^20"
      ],
      "type": <multi-match-type>,
      "operator": "AND"
    }
  }
}

```

best_fields: 所有输入的Token必须在一个字段上全部匹配。


每个字段匹配时分别使用mapping上定义的*analyzer*和*search_analyzer*。

```

(+brandName:运动 +brandName:上衣)^100.0
| (+brandName.brandName_pinyin:运 +brandName.brandName_pinyin:动 +brandName.brandName_
| (+brandName.brandName_keyword:运 +brandName.brandName_keyword:动 +brandName.brandName_

```

```
| (+sortName:运动 +sortName:上衣)^80.0
| (+sortName.sortName_pinyin:运 +sortName.sortName_pinyin:动 +sortName.sortName_pinyin:上 +sortName.sortName_pinyin:衣)^80.0
| (+productName:运动 +productName:上衣)^60.0
| (+productKeyword:运动 +productKeyword:上衣)^20.0
```



most_fields: 所有输入的Token必须在一个字段上全部匹配。

与best_fields不同之处在于相关性评分，best_fields取最大匹配得分（max计算），而most_fields取所有匹配之和（sum计算）。

```
(
  (+brandName:运动 +brandName:上衣)^100.0
  (+brandName.brandName_pinyin:运 +brandName.brandName_pinyin:动 +brandName.brandName_pinyin:上 +brandName.brandName_pinyin:衣)^100.0
  (+brandName.brandName_keyword:运 +brandName.brandName_keyword:动 +brandName.brandName_keyword:上 +brandName.brandName_keyword:衣)^100.0
  (+sortName:运动 +sortName:上衣)^80.0
  (+sortName.sortName_pinyin:运 +sortName.sortName_pinyin:动 +sortName.sortName_pinyin:上 +sortName.sortName_pinyin:衣)^80.0
  (+productName:运动 +productName:上衣)^60.0
  (+productKeyword:运动 +productKeyword:上衣)^20.0
)
```

cross_fields: 所有输入的Token必须在同一组的字段上全部匹配。

首先ES会对cross_fields进行查询重写分组，分组的依据是search_analyzer。具体到我们的例子中

【brandName.brandName_pinyin、brandName.brandName_keyword、sortName.sortName_pinyin】这三个字段的search_analyzer是standard，而其余的字段是fulltext_analyzer，因此最终被分为了两组。

```
(
  (
    +(brandName.brandName_pinyin:运^100.0 | sortName.sortName_pinyin:运^80.0 | brandName.brandName_keyword:运^100.0 | sortName.sortName_pinyin:动^80.0 | brandName.brandName_keyword:动^100.0 | sortName.sortName_pinyin:上^80.0 | brandName.brandName_keyword:上^100.0 | sortName.sortName_pinyin:衣^80.0 | brandName.brandName_keyword:衣^100.0)^100.0
  )
  (
    +(productKeyword:运动^20.0 | brandName:运动^100.0 | sortName:运动^80.0 | productName:运动^100.0 | productKeyword:上衣^20.0 | brandName:上衣^100.0 | sortName:上衣^80.0 | productName:上衣^100.0)^100.0
  )
)
```

继续探索和思考

如何让best_fields和most_fields也可以匹配出商品？

最常见的做法就是使用_all字段或者copyTo字段来实现，比如我们mapping里面的bigSearchField字段。

如何改进cross_fields的搜索结果?



由于cross_fields需要根据search_analyzer进行分组，因此像搜索【运动 shangyi】这样的输入时是无法匹配到商品的，因此应该尽可能地减少分组既尽量使用统一的search_analyzer，或者在search时强制指定search_analyzer覆盖mapping里定义的search_analyzer。

把operator改成OR会如何?

在上面的例子中，我们设置的operator均为AND，意味着所有搜索的Token都必须被匹配。那设置成OR会怎么样以及什么场景下该使用OR呢?

在使用OR的时候要特别注意，因为只要有一个Token匹配就会把商品搜索出来，比如上面的搜索【运动 上衣】的时候，会把鞋子的商品也匹配出来，这样搜索的准确度会远远降低。

在一些特殊的搜索中，比如我们搜索【耐克 阿迪达斯 上衣】，如果使用operator为AND，则无论使用哪种multi-search-type都无法匹配出商品（想想为什么？），此时我们可以设置operator为OR并且设置minimum_should_match为60%，这样就可以搜索出属于耐克和阿迪达斯的上衣了，这种情况相当于一种智能的搜索降级了。

```
/gino_product/_search
{
  "query": {
    "multi_match": {
      "query": "耐克 阿迪达斯 上衣",
      "fields": [
        "brandName^100",
        "brandName.brandName_pinyin^100",
        "brandName.brandName_keyword^100",
        "sortName^80",
        "sortName.sortName_pinyin^80",
        "productName^60",
        "productKeyword^20"
      ],
      "type": "cross_fields",
      "operator": "OR",
      "minimum_should_match": "60%"
    }
  }
}
```

再谈相关性评分

在Elasticsearch相关性打分机制学习一文中我们曾经探讨过best_fields和cross_fields相关性评分的机制，其中的例子使用的相同的search_analyzer。那对于分组情况下，cross_fields评分又是如何计算的呢?

我们还是用上面的例子，增加explain参数来看一下。



```
POST /gino_product/_search
{
  "explain": true,
  "query": {
    "multi_match": {
      "query": "运动 上衣",
      "fields": [
        "brandName^100",
        "brandName.brandName_pinyin^100",
        "brandName.brandName_keyword^100",
        "sortName^80",
        "sortName.sortName_pinyin^80",
        "productName^60",
        "productKeyword^20"
      ],
      "type": "cross_fields",
      "operator": "AND"
    }
  }
}
```

详细ES响应报文：[cross_fields_scoring.json](#)

通过上述validate API得到的分组信息和explain得到的评分详情信息，可以总结出一个cross_fields评分公式：

$$\text{score}(q, d) = \text{coord}(q, d) * \sum(\sum(\max(\text{score}(t, f))))$$

- $\text{coord}(q, d)$: 分组匹配因子，比如上面我们只有一个分组匹配，coord就是0.5（两个分组中匹配了一个分组）；
- $\text{score}(t, f)$: 搜索的一个Token和一个特定的字段的相关性评分（使用TFIDF）计算；
- \max : 搜索的一个Token在所有字段评分中取最大值；
- 分组内求和：一个分组内搜索的所有Token的最大值进行求和；
- 分组间求和：所有分组的得分最终进行求和计算；

小结

- best_fields对搜索为单个Token的情况下效果更好，比如搜索【耐克】的时候品牌为耐克和商品关键字包含耐克的时候前者相关性得分更高；但是对于都是为多个Token需要跨字段匹配时，只能引进大字段来匹配，这样权重的设置就失去意义了；
- most_fields和best_fields类似，其优点在于能够尽可能多地匹配，相关性评分机制更合理；

- `cross_fields`最大的优点在于能够跨字段匹配，而且充分利用到了各个字段的权重设置。但是需要注意的是匹配时是根据`search_analyzer`进行分组，不同分组直接的匹配无法跨字段。



参考材料

- [ElasticSearch Reference > Multi Match Query](#)



扫一扫，关注我的微信公众号

Elasticsearch # Multi Match Query # best_fields # cross_fields # most_fields

◀ 阅读随手记 201612

一个程序员的小结：2016年的进步、收获与成长 ▶

0

© 2016 - 2017 ♥ Gino Zhang

由 [Hexo](#) 强力驱动 | 主题 - [NexT.Mist](#)

👤 访问人数 33464 | 👁 总访问量 73042