# ELK学习笔记（来自龙果学院，内容非原创。）

## 结构化搜索\_在案例中实战使用term filter来搜索数据

（1）插入一些测试帖子数据

POST /forum/article/\_bulk

{ "index": { "\_id": 1 }}

{ "articleID" : "XHDK-A-1293-#fJ3", "userID" : 1, "hidden": false, "postDate": "2017-01-01" }

{ "index": { "\_id": 2 }}

{ "articleID" : "KDKE-B-9947-#kL5", "userID" : 1, "hidden": false, "postDate": "2017-01-02" }

{ "index": { "\_id": 3 }}

{ "articleID" : "JODL-X-1937-#pV7", "userID" : 2, "hidden": false, "postDate": "2017-01-01" }

{ "index": { "\_id": 4 }}

{ "articleID" : "QQPX-R-3956-#aD8", "userID" : 2, "hidden": true, "postDate": "2017-01-02" }

（2）查看mapping

GET /forum/\_mapping/article

{

"forum": {

"mappings": {

"article": {

"properties": {

"articleID": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

},

"hidden": {

"type": "boolean"

},

"postDate": {

"type": "date"

},

"userID": {

"type": "long"

}

}

}

}

}

}

type=text，默认会设置两个field，一个是field本身，比如articleID，就是分词的；还有一个的话，就是field.keyword，articleID.keyword，默认不分词，会最多保留256个字符

（3）根据用户ID搜索帖子

GET /forum/article/\_search

{

"query" : {

"constant\_score" : {

"filter" : {

"term" : {

"userID" : 1

}

}

}

}

}

term filter/query：对搜索文本不分词，直接拿去倒排索引中匹配，你输入的是什么，就去匹配什么

比如说，如果对搜索文本进行分词的话，“helle world” --> “hello”和“world”，两个词分别去倒排索引中匹配

term，“hello world” --> “hello world”，直接去倒排索引中匹配“hello world”

（4）根据发帖日期搜索帖子

GET /forum/article/\_search

{

"query" : {

"constant\_score" : {

"filter" : {

"term" : {

"postDate" : "2017-01-01"

}

}

}

}

}

（5）根据帖子ID搜索帖子

GET /forum/article/\_search

{

"query" : {

"constant\_score" : {

"filter" : {

"term" : {

"articleID" : "XHDK-A-1293-#fJ3"

}

}

}

}

}

查不到数据，已分词

GET /forum/article/\_search

{

"query" : {

"constant\_score" : {

"filter" : {

"term" : {

"articleID.keyword" : "XHDK-A-1293-#fJ3"

}

}

}

}

}

articleID.keyword，是es内置建立的field，就是不分词的。所以一个articleID过来的时候，会建立两次索引，一次是自己本身，是要分词的，分词后放入倒排索引；另外一次是基于articleID.keyword，不分词，保留256个字符最多，直接一个字符串放入倒排索引中。

所以term filter，对text过滤，可以考虑使用内置的field.keyword来进行匹配。但是有个问题，默认就保留256个字符。所以尽可能还是自己去手动建立索引，指定not\_analyzed吧。不指定not\_analyzed也可以，将type=keyword即可。

（6）查看分词

GET /forum/\_analyze

{

"field": "articleID",

"text": "XHDK-A-1293-#fJ3"

}

默认是analyzed的text类型的field，建立倒排索引的时候，就会对所有的articleID分词，分词以后，原本的articleID就没有了，只有分词后的各个word存在于倒排索引中。

## \_结构化搜索\_filter执行原理深度剖析（bitset机制与caching机制）

（1）在倒排索引中查找搜索串，获取document list,以date来举例：

word doc1 doc2 doc3

2017-01-01 \* \*

2017-02-02 \* \*

2017-03-03 \* \* \*

比如过滤出2017-02-02，到倒排索引中一找，发现2017-02-02对应的document list是doc2,doc3

（2）为每个在倒排索引中搜索到的结果，构建一个bitset，[0, 0, 0, 1, 0, 1]。

使用找到的doc list，构建一个bitset，就是一个二进制的数组，数组每个元素都是0或1，用来标识一个doc对一个filter条件是否匹配，如果匹配就是1，不匹配就是0，比如： [0, 1, 1]

doc1：不匹配这个filter的

doc2和do3：是匹配这个filter的

尽可能用简单的数据结构去实现复杂的功能，可以节省内存空间，提升性能

（3）遍历每个过滤条件对应的bitset，优先从最稀疏的开始搜索，查找满足所有条件的document

后面会讲解，一次性其实可以在一个search请求中，发出多个filter条件，每个filter条件都会对应一个bitset

遍历每个filter条件对应的bitset，先从最稀疏的开始遍历

[0, 0, 0, 1, 0, 0]：比较稀疏

[0, 1, 0, 1, 0, 1]

先遍历比较稀疏的bitset，就可以先过滤掉尽可能多的数据

遍历所有的bitset，找到匹配所有filter条件的doc

请求：filter，postDate=2017-01-01，userID=1

postDate: [0, 0, 1, 1, 0, 0]

userID: [0, 1, 0, 1, 0, 1]

遍历完两个bitset之后，找到的匹配所有条件的doc，就是doc4

就可以将document作为结果返回给client了

（4）caching bitset，跟踪query，在最近256个query中超过一定次数的过滤条件，缓存其bitset。对于小segment（<1000，或<3%），不缓存bitset。

比如postDate=2017-01-01，[0, 0, 1, 1, 0, 0]，可以缓存在内存中，这样下次如果再有这个条件过来的时候，就不用重新扫描倒排索引，反复生成bitset，可以大幅度提升性能。

在最近的256个filter中，有某个filter超过了一定的次数，次数不固定，就会自动缓存这个filter对应的bitset

filter针对小segment获取到的结果，可以不缓存，segment记录数<1000，或者segment大小<index总大小的3%

segment数据量很小，此时哪怕是扫描也很快；segment会在后台自动合并，小segment很快就会跟其他小segment合并成大segment，此时就缓存也没有什么意义，segment很快就消失了

针对一个小segment的bitset，[0, 0, 1, 0]

filter比query的好处就在于会caching，但是之前不知道caching的是什么东西，实际上并不是一个filter返回的完整的doc list数据结果。而是filter bitset缓存起来。下次不用扫描倒排索引了。

（5）filter大部分情况下来说，在query之前执行，先尽量过滤掉尽可能多的数据

query：是会计算doc对搜索条件的relevance score，还会根据这个score去排序

filter：只是简单过滤出想要的数据，不计算relevance score，也不排序

（6）如果document有新增或修改，那么cached bitset会被自动更新

比如，针对postDate=2017-01-01，[0, 0, 1, 0]

document，id=5，postDate=2017-01-01，会自动更新到postDate=2017-01-01这个filter的bitset中，全自动，缓存会自动更新。postDate=2017-01-01的bitset，[0, 0, 1, 0, 1]

document，id=1，postDate=2016-12-30，修改为postDate-2017-01-01，此时也会自动更新bitset，[1, 0, 1, 0, 1]

（7）以后只要是有相同的filter条件的，会直接来使用这个过滤条件对应的cached bitset

## 结构化搜索\_在案例中实战基于bool组合多个filter条件来搜索数据

1、搜索发帖日期为2017-01-01，或者帖子ID为XHDK-A-1293-#fJ3的帖子，同时要求帖子的发帖日期绝对不为2017-01-02

select \*

from forum.article

where (post\_date='2017-01-01' or article\_id='XHDK-A-1293-#fJ3')

and post\_date!='2017-01-02'

GET /forum/article/\_search

{

"query": {

"constant\_score": {

"filter": {

"bool": {

"should": [

{"term": { "postDate": "2017-01-01" }},

{"term": {"articleID": "XHDK-A-1293-#fJ3"}}

],

"must\_not": {

"term": {

"postDate": "2017-01-02"

}

}

}

}

}

}

}

must，should，must\_not，filter：必须匹配，可以匹配其中任意一个即可，必须不匹配

2、搜索帖子ID为XHDK-A-1293-#fJ3，或者是帖子ID为JODL-X-1937-#pV7而且发帖日期为2017-01-01的帖子

select \*

from forum.article

where article\_id='XHDK-A-1293-#fJ3'

or (article\_id='JODL-X-1937-#pV7' and post\_date='2017-01-01')

GET /forum/article/\_search

{

"query": {

"constant\_score": {

"filter": {

"bool": {

"should": [

{

"term": {

"articleID": "XHDK-A-1293-#fJ3"

}

},

{

"bool": {

"must": [

{

"term":{

"articleID": "JODL-X-1937-#pV7"

}

},

{

"term": {

"postDate": "2017-01-01"

}

}

]

}

}

]

}

}

}

}

}

3、梳理学到的知识点

（1）bool：must，must\_not，should，组合多个过滤条件

（2）bool可以嵌套

（3）相当于SQL中的多个and条件：当你把搜索语法学好了以后，基本可以实现部分常用的sql语法对应的功能

## 结构化搜索\_在案例中实战使用terms搜索多个值以及多值搜索结果优化

term: {"field": "value"}

terms: {"field": ["value1", "value2"]}

sql中的in

select \* from tbl where col in ("value1", "value2")

1、为帖子数据增加tag字段

POST /forum/article/\_bulk

{ "update": { "\_id": "1"} }

{ "doc" : {"tag" : ["java", "hadoop"]} }

{ "update": { "\_id": "2"} }

{ "doc" : {"tag" : ["java"]} }

{ "update": { "\_id": "3"} }

{ "doc" : {"tag" : ["hadoop"]} }

{ "update": { "\_id": "4"} }

{ "doc" : {"tag" : ["java", "elasticsearch"]} }

2、搜索articleID为KDKE-B-9947-#kL5或QQPX-R-3956-#aD8的帖子，搜索tag中包含java的帖子

GET /forum/article/\_search

{

"query": {

"constant\_score": {

"filter": {

"terms": {

"articleID": [

"KDKE-B-9947-#kL5",

"QQPX-R-3956-#aD8"

]

}

}

}

}

}

GET /forum/article/\_search

{

"query" : {

"constant\_score" : {

"filter" : {

"terms" : {

"tag" : ["java"]

}

}

}

}

}

"took": 2,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 3,

"max\_score": 1,

"hits": [

{

"\_index": "forum",

"\_type": "article",

"\_id": "2",

"\_score": 1,

"\_source": {

"articleID": "KDKE-B-9947-#kL5",

"userID": 1,

"hidden": false,

"postDate": "2017-01-02",

"tag": [

"java"

]

}

},

{

"\_index": "forum",

"\_type": "article",

"\_id": "4",

"\_score": 1,

"\_source": {

"articleID": "QQPX-R-3956-#aD8",

"userID": 2,

"hidden": true,

"postDate": "2017-01-02",

"tag": [

"java",

"elasticsearch"

]

}

},

{

"\_index": "forum",

"\_type": "article",

"\_id": "1",

"\_score": 1,

"\_source": {

"articleID": "XHDK-A-1293-#fJ3",

"userID": 1,

"hidden": false,

"postDate": "2017-01-01",

"tag": [

"java",

"hadoop"

]

}

}

]

}

}

3、优化搜索结果，仅仅搜索tag\_cnt 为1，且tag只包含java的帖子

POST /forum/article/\_bulk

{ "update": { "\_id": "1"} }

{ "doc" : {"tag\_cnt" : 2} }

{ "update": { "\_id": "2"} }

{ "doc" : {"tag\_cnt" : 1} }

{ "update": { "\_id": "3"} }

{ "doc" : {"tag\_cnt" : 1} }

{ "update": { "\_id": "4"} }

{ "doc" : {"tag\_cnt" : 2} }

GET /forum/article/\_search

{

"query": {

"constant\_score": {

"filter": {

"bool": {

"must": [

{

"term": {

"tag\_cnt": 1

}

},

{

"terms": {

"tag": ["java"]

}

}

]

}

}

}

}

}

["java", "hadoop", "elasticsearch"]

4、学到的知识点梳理

（1）terms多值搜索

（2）优化terms多值搜索的结果

（3）相当于SQL中的in语句

## 结构化搜索\_在案例中实战基于range filter来进行范围过滤

1、为帖子数据增加浏览量的字段

**POST /forum/article/\_bulk**

{ "update": { "\_id": "1"} }

{ "doc" : {"view\_cnt" : 30} }

{ "update": { "\_id": "2"} }

{ "doc" : {"view\_cnt" : 50} }

{ "update": { "\_id": "3"} }

{ "doc" : {"view\_cnt" : 100} }

{ "update": { "\_id": "4"} }

{ "doc" : {"view\_cnt" : 80} }

2、搜索浏览量在30~60之间的帖子

**GET /forum/article/\_search**

{

"query": {

"constant\_score": {

"filter": {

"range": {

"view\_cnt": {

"gt": 30,

"lt": 60

}

}

}

}

}

}

3、搜索发帖日期在最近1个月的帖子

POST /forum/article/\_bulk

{ "index": { "\_id": 5 }}

{ "articleID" : "DHJK-B-1395-#Ky5", "userID" : 3, "hidden": false, "postDate": "2017-03-01", "tag": ["elasticsearch"], "tag\_cnt": 1, "view\_cnt": 10 }

某个日期向前推进多少天

GET /forum/article/\_search

{

"query": {

"constant\_score": {

"filter": {

"range": {

"postDate": {

"gt": "2017-03-10||-30d"

}

}

}

}

}

}

GET /forum/article/\_search

{

"query": {

"constant\_score": {

"filter": {

"range": {

"postDate": {

"gt": "now-30d"

}

}

}

}

}

}

4、梳理一下学到的知识点

range做范围过滤，相当与sql中的between，或者是>=1，<=1

## 如何手动控制全文检索结果的精准度

1、为帖子数据增加标题字段

POST /forum/article/\_bulk

{ "update": { "\_id": "1"} }

{ "doc" : {"title" : "this is java and elasticsearch blog"} }

{ "update": { "\_id": "2"} }

{ "doc" : {"title" : "this is java blog"} }

{ "update": { "\_id": "3"} }

{ "doc" : {"title" : "this is elasticsearch blog"} }

{ "update": { "\_id": "4"} }

{ "doc" : {"title" : "this is java, elasticsearch, hadoop blog"} }

{ "update": { "\_id": "5"} }

{ "doc" : {"title" : "this is spark blog"} }

2、搜索标题中包含java或elasticsearch的blog

这个，就跟之前的那个term query，不一样了。不是搜索exact value，是进行full text全文检索。

match query，是负责进行全文检索的。当然，如果要检索的field，是not\_analyzed类型的，那么match query也相当于term query。

GET /forum/article/\_search

{

"query": {

"match": {

"title": "java elasticsearch"

}

}

}

3、搜索标题中包含java和elasticsearch的blog

搜索结果精准控制的第一步：灵活使用and关键字，如果你是希望所有的搜索关键字都要匹配的，那么就用and，可以实现单纯match query无法实现的效果

GET /forum/article/\_search

{

"query": {

"match": {

"title": {

"query": "java elasticsearch",

"operator": "and"

}

}

}

}

4、搜索包含java，elasticsearch，spark，hadoop，4个关键字中，至少3个的blog

控制搜索结果的精准度的第二步：指定一些关键字中，必须至少匹配其中的多少个关键字，才能作为结果返回

GET /forum/article/\_search

{

"query": {

"match": {

"title": {

"query": "java elasticsearch spark hadoop",

"minimum\_should\_match": "75%"

}

}

}

}

5、用bool组合多个搜索条件，来搜索title

GET /forum/article/\_search

{

"query": {

"bool": {

"must": { "match": { "title": "java" }},

"must\_not": { "match": { "title": "spark" }},

"should": [

{ "match": { "title": "hadoop" }},

{ "match": { "title": "elasticsearch" }}

]

}

}

}

6、bool组合多个搜索条件，如何计算relevance score

must和should搜索对应的分数，加起来，除以must和should的总数

排名第一：java，同时包含should中所有的关键字，hadoop，elasticsearch

排名第二：java，同时包含should中的elasticsearch

排名第三：java，不包含should中的任何关键字

should是可以影响相关度分数的

must是确保说，谁必须有这个关键字，同时会根据这个must的条件去计算出document对这个搜索条件的relevance score

在满足must的基础之上，should中的条件，不匹配也可以，但是如果匹配的更多，那么document的relevance score就会更高

{

"took": 6,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 3,

"max\_score": 1.3375794,

"hits": [

{

"\_index": "forum",

"\_type": "article",

"\_id": "4",

"\_score": 1.3375794,

"\_source": {

"articleID": "QQPX-R-3956-#aD8",

"userID": 2,

"hidden": true,

"postDate": "2017-01-02",

"tag": [

"java",

"elasticsearch"

],

"tag\_cnt": 2,

"view\_cnt": 80,

"title": "this is java, elasticsearch, hadoop blog"

}

},

{

"\_index": "forum",

"\_type": "article",

"\_id": "1",

"\_score": 0.53484553,

"\_source": {

"articleID": "XHDK-A-1293-#fJ3",

"userID": 1,

"hidden": false,

"postDate": "2017-01-01",

"tag": [

"java",

"hadoop"

],

"tag\_cnt": 2,

"view\_cnt": 30,

"title": "this is java and elasticsearch blog"

}

},

{

"\_index": "forum",

"\_type": "article",

"\_id": "2",

"\_score": 0.19856805,

"\_source": {

"articleID": "KDKE-B-9947-#kL5",

"userID": 1,

"hidden": false,

"postDate": "2017-01-02",

"tag": [

"java"

],

"tag\_cnt": 1,

"view\_cnt": 50,

"title": "this is java blog"

}

}

]

}

}

7、搜索java，hadoop，spark，elasticsearch，至少包含其中3个关键字

默认情况下，should是可以不匹配任何一个的，比如上面的搜索中，this is java blog，就不匹配任何一个should条件

但是有个例外的情况，如果没有must的话，那么should中必须至少匹配一个才可以

比如下面的搜索，should中有4个条件，默认情况下，只要满足其中一个条件，就可以匹配作为结果返回

但是可以精准控制，should的4个条件中，至少匹配几个才能作为结果返回

GET /forum/article/\_search

{

"query": {

"bool": {

"should": [

{ "match": { "title": "java" }},

{ "match": { "title": "elasticsearch" }},

{ "match": { "title": "hadoop" }},

{ "match": { "title": "spark" }}

],

"minimum\_should\_match": 3

}

}

}

梳理一下学到的知识点

1、全文检索的时候，进行多个值的检索，有两种做法，match query；should

2、控制搜索结果精准度：and operator，minimum\_should\_match