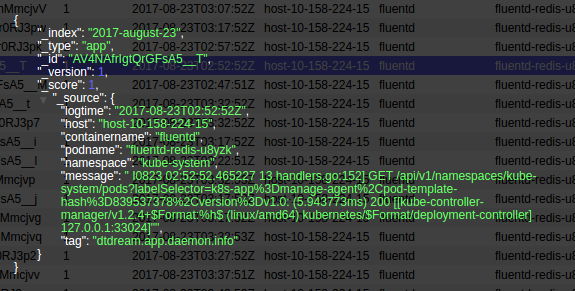
1. **搜索操作**

**curl -XGET <http://[ip]:[port]/[index名称]/_search?pretty>**

**curl -XPOST <http://[ip]:[port]/[index名称]/_search?pretty> ‘[json]’**

****

**1.简单查询（以head插件为例）：**

**模糊匹配：**

*{*

*"query":{*

*"match":{*

*"host":"10-158-224-15"*

*}*

*}*

*}*

***精确匹配：***

*{*

*"query":{*

*"match\_phrase":{*

*"host":"10-158-224-15"*

*}*

*}*

*}*

***全部匹配：***

*{*

*"query":{*

*"match\_all":{ }*

*}*

*}*

***过滤器匹配：***

**

***排序：***

*{*

*"query":{*

*"match\_all":{ }*

*},*

*"sort":{*

*"logtime": { "order": "desc" }*

*},*

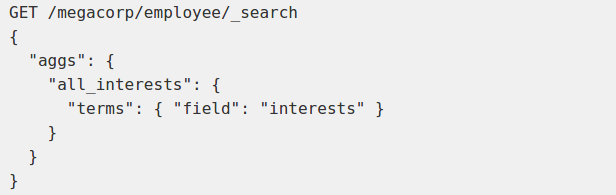
*"size":100*

*}*

*备用做排序的字段最好不要时text或string类型的，因为这两种类型需要设置fielddata，对性能有一定损耗。*

*2***.聚合**

**聚合：**

****

****

**Filter DSL:3-6**

**3.term 过滤**

term主要用于精确匹配哪些值，比如数字，日期，布尔值或 not\_analyzed 的字符串(未经分析的文本数据类型)：

{ "term": { "age":    26           }}   
{ "term": { "date":   "2014-09-01" }}   
{ "term": { "public": true         }}   
{ "term": { "tag":    "full\_text"  }}

完整的例子， hostname 字段完全匹配成 saaap.wangpos.com 的数据：

{   
  "query": {   
    "term": {   
      "hostname": "saaap.wangpos.com"   
    }   
  }   
}

**3.terms 过滤**

terms 跟 term 有点类似，但 terms 允许指定多个匹配条件。 如果某个字段指定了多个值，那么文档需要一起去做匹配：

{   
    "terms": {   
        "tag": [ "search", "full\_text", "nosql" ]   
        }   
}

完整的例子，所有http的状态是 302 、304 的， 由于ES中状态是数字类型的字段，所有这里我们可以直接这么写。：

{   
  "query": {   
    "terms": {   
      "status": [   
        304,   
        302   
      ]   
    }   
  }   
}

**4.range 过滤**

range过滤允许我们按照指定范围查找一批数据：

{   
    "range": {   
        "age": {   
            "gte":  20,   
            "lt":   30   
        }   
    }   
}

范围操作符包含：

* gt :: 大于
* gte:: 大于等于
* lt :: 小于
* lte:: 小于等于

一个完整的例子， 请求页面耗时大于1秒的数据，upstream\_response\_time 是 nginx 日志中的耗时，ES中是数字类型。

{   
  "query": {   
    "range": {   
      "upstream\_response\_time": {   
        "gt": 1   
      }   
    }   
  }   
}

## **5.exists 和 missing 过滤**

exists 和 missing 过滤可以用于查找文档中是否包含指定字段或没有某个字段，类似于SQL语句中的IS\_NULL条件.

{   
    "exists":   {   
        "field":    "title"   
    }   
}

这两个过滤只是针对已经查出一批数据来，但是想区分出某个字段是否存在的时候使用。

## **6.bool 过滤**

bool 过滤可以用来合并多个过滤条件查询结果的布尔逻辑，取代了以前的filtered它包含一下操作符：

* must :: 多个查询条件的完全匹配,相当于 and。
* must\_not :: 多个查询条件的相反匹配，相当于 not。
* should :: 至少有一个查询条件匹配, 相当于 or。

这些参数可以分别继承一个过滤条件或者一个过滤条件的数组：

{   
    "bool": {   
        "must":     { "term": { "folder": "inbox" }},   
        "must\_not": { "term": { "tag":    "spam"  }},   
        "should": [   
                    { "term": { "starred": true   }},   
                    { "term": { "unread":  true   }}   
        ]   
    }   
}

# **Query DSL:**

## **7.match\_all 查询**

可以查询到所有文档，是没有查询条件下的默认语句。

{   
    "match\_all": {}   
}

此查询常用于合并过滤条件。 比如说你需要检索所有的邮箱,所有的文档相关性都是相同的，所以得到的\_score为1.

## **8.match 查询**

match查询是一个标准查询，不管你需要全文本查询还是精确查询基本上都要用到它。

如果你使用 match 查询一个全文本字段，它会在真正查询之前用分析器先分析match一下查询字符：

{   
    "match": {   
        "tweet": "About Search"   
    }   
}

如果用match下指定了一个确切值，在遇到数字，日期，布尔值或者not\_analyzed 的字符串时，它将为你搜索你给定的值：

{ "match": { "age":    26           }}   
{ "match": { "date":   "2014-09-01" }}   
{ "match": { "public": true         }}   
{ "match": { "tag":    "full\_text"  }}

提示： 做精确匹配搜索时，你最好用过滤语句，因为过滤语句可以缓存数据。

match查询只能就指定某个确切字段某个确切的值进行搜索，而你要做的就是为它指定正确的字段名以避免语法错误。

## **9.multi\_match 查询**

multi\_match查询允许你做match查询的基础上同时搜索多个字段，在多个字段中同时查一个：

{   
    "multi\_match": {   
        "query":    "full text search",   
        "fields":   [ "title", "body" ]   
    }   
}

## **10.bool 查询**

bool 查询与 bool 过滤相似，用于合并多个查询子句。不同的是，bool 过滤可以直接给出是否匹配成功， 而bool 查询要计算每一个查询子句的 \_score （相关性分值）。

* must:: 查询指定文档一定要被包含。
* must\_not:: 查询指定文档一定不要被包含。
* should:: 查询指定文档，有则可以为文档相关性加分。

以下查询将会找到 title 字段中包含 "how to make millions"，并且 "tag" 字段没有被标为 spam。 如果有标识为 "starred" 或者发布日期为2014年之前，那么这些匹配的文档将比同类网站等级高：

{   
    "bool": {   
        "must":     { "match": { "title": "how to make millions" }},   
        "must\_not": { "match": { "tag":   "spam" }},   
        "should": [   
            { "match": { "tag": "starred" }},   
            { "range": { "date": { "gte": "2014-01-01" }}}   
        ]   
    }   
}

提示： 如果bool 查询下没有must子句，那至少应该有一个should子句。但是 如果有must子句，那么没有should子句也可以进行查询。

上面内容来自： [http://es.xiaoleilu.com/054\_Query\_DSL/70\_Important\_clauses.html](http://es.xiaoleilu.com/054_Query_DSL/70_Important_clauses.html" \o "http://es.xiaoleilu.com/054_Query_DSL/70_Important_clauses.html)

ElasticSearch 查询（match和term）   
<http://www.cnblogs.com/yjf512/p/4897294.html>

## **11.wildcards 查询**

使用标准的shell通配符查询

参考：

[https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-wildcard-query.html](https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-wildcard-query.html" \o "https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-wildcard-query.html)

以下查询能够匹配包含W1F 7HW和W2F 8HW的文档：

GET /my\_index/address/\_search   
{   
    "query": {   
        "wildcard": {   
            "postcode": "W?F\*HW"   
        }   
    }   
}

又比如下面查询 hostname 匹配下面shell通配符的：

{   
  "query": {   
    "wildcard": {   
      "hostname": "wxopen\*"   
    }   
  }   
}

## **12.regexp 查询**

假设您只想匹配以W开头，紧跟着数字的邮政编码。使用regexp查询能够让你写下更复杂的模式：

GET /my\_index/address/\_search   
{   
    "query": {   
        "regexp": {   
            "postcode": "W[0-9].+"   
        }   
    }   
}

这个正则表达式的规定了词条需要以W开头，紧跟着一个0到9的数字，然后是一个或者多个其它字符。

下面例子是所有以 wxopen 开头的正则

{   
  "query": {   
    "regexp": {   
      "hostname": "wxopen.\*"   
    }   
  }   
}

参考：

[https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-regexp-query.html](https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-regexp-query.html" \o "https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-regexp-query.html)

## **13.prefix 查询**

以什么字符开头的，可以更简单地用 prefix，如下面的例子：

{   
  "query": {   
    "prefix": {   
      "hostname": "wxopen"   
    }   
  }   
}

参考 ：

[https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-prefix-query.html](https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-prefix-query.html" \o "https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-prefix-query.html)

更多的查询命令，可以看：

[https://www.elastic.co/guide/en/elasticsearch/reference/current/term-level-queries.html#term-level-queries](https://www.elastic.co/guide/en/elasticsearch/reference/current/term-level-queries.html" \l "term-level-queries" \o "https://www.elastic.co/guide/en/elasticsearch/reference/current/term-level-queries.html#term-level-queries)

## **14.短语匹配(Phrase Matching)**

当你需要寻找邻近的几个单词时，你会使用match\_phrase查询：

GET /my\_index/my\_type/\_search

{

"query": {

"match\_phrase": {

"title": "quick brown fox"

}

}

}

和match查询类似，match\_phrase查询首先解析查询字符串来产生一个词条列表。然后会搜索所有的词条，  
但只保留含有了所有搜索词条的文档，并且词条的位置要邻接。一个针对短语quick fox的查询不会匹配  
我们的任何文档，因为没有文档含有邻接在一起的quick和box词条。

match\_phrase查询也可以写成类型为phrase的match查询：

"match": {

"title": {

"query": "quick brown fox",

"type": "phrase"

}

}

**二、示例**

1.

{"query":{"bool":{"must":[

{"range":{

"log\_time":{

"from":"2017-09-21T08:00:00Z",

"include\_lower":true,

"include\_upper":true,

"to":"2017-09-22T08:59:00Z"}

}},

{

"term":{"\_type":"system"}

}

]}},

"from":600000,

"size":100

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**“query”:**{

"bool":{

"filter":[{

"match":{

"namespace":{

"query":"d48ec51c7-f3d1-4344-a1b",

"type":"phrase"

}

}

},{

"match":{

"tag":{

"query":"dtdream",

"type":"phrase\_prefix"

}

}

}],

"must":[{

"range":{

"logtime"{

"from":"2017-09-26T00:00:00Z",

"include\_lower":true,

"include\_upper":true,

"to":"2017-09-28T23:59:00Z"

}

}

},{

"term":{

"\_type":"app"

}

}]

}

}

}