Elasticsearch查询语法

1. Searchtimeout：

   (1) 设置：默认没有timeout，如果设置了timeout，那么会执行timeout机制。

   (2) Timeout机制：假设用户查询结果有1W条数据，但是需要10″才能查询完毕，但是用户设置了1″的timeout，那么不管当前一共查询到了多少数据，都会在1″后ES讲停止查询，并返回当前数据。

   (3) 用法：GET /_search?timeout=1s/ms/m

2. ES常用查询：

   (1) Query_string：

      1. 查询所有：GET /product/_search
      2. 带参数：GET /product/_search?q=name:xiaomi
      3. 分页：GET /product/_search?from=0&size=2&sort=price:asc

   (2) Query DSL：

      1. **match_all**：匹配所有

```
GET /product/_search
{
  "query":{
    "match_all": {}
  }
}
```

      2. **match**：name中包含“nfc”

```
GET /product/_search
{
  "query": {
    "match": {
      "name": "nfc"
    }
  }
}
```

      3. **sort**：按照价格倒序排序

```
GET /product/_search
{
  "query": {
    "multi_match": {
      "query": "nfc",
      "fields": ["name","desc"]
    }
  },
  "sort": [
    {
      "price": "desc"
    }
  ]
}
```

      4. **multi_match**：根据多个字段查询一个关键词，name和desc中包含“nfc”的doc

```
GET /product/_search
{
  "query": {
    "multi_match": {
      "query": "nfc",
      "fields": ["name","desc"]
    }
  },
```

```
  "sort": [
    {
      "price": "desc"
    }
  ]
}
```

5. **_source 元数据：**想要查询多个字段，例子中为只查询“name”和“price”字段。

```
GET /product/_search
{
  "query":{
    "match": {
      "name": "nfc"
    }
  },
  "_source": ["name","price"]
}
```

6. **分页（deep-paging）**：查询第一页（每页两条数据）

```
GET /product/_search
{
  "query":{
    "match_all": {}
  },
  "sort": [
    {
      "price": "asc"
    }
  ],
  "from": 0,
  "size": 2
}
```

(3) **Full-text queries**：全文检索

1. **query-term**：不会被分词，

```
GET /product/_search
{
  "query": {
    "term": {
      "name": "nfc"
    }
  }
}
```

2. **match和term区别**：

```
GET /product/_search
{
  "query": {
    "term": {
      "name": "nfc phone" 这里因为没有分词，所以查询没有结果
    }
  }
}
GET /product/_search
{
  "query": {
    "bool": {
      "must": [
        {"term":{"name":"nfc"}},
        {"term":{"name":"phone"}}
      ]
    }
  }
}
GET /product/_search
{
```

```
  "query": {
    "terms": {
      "name":["nfc","phone"]
    }
  }
}
GET /product/_search
{
  "query": {
    "match": {
      "name": "nfc phone"
    }
  }
}
```

3. ☆全文检索：

```
GET /product/_search
{
  "query": {
    "match": {
      "name": "xiaomi nfc zhineng phone"
    }
  }
}
#验证分词
GET /_analyze
{
  "analyzer": "standard",
  "text":"xiaomi nfc zhineng phone"
}
```

(4) Phrase search：短语搜索，和全文检索相反，“nfc phone”会作为一个短语去检索

```
GET /product/_search
{
  "query": {
    "match_phrase": {
      "name": "nfc phone"
    }
  }
}
```

(5) Query and filter：查询和过滤

1. bool：可以组合多个查询条件，bool查询也是采用more_matches_is_better的机制，因此满足must和should子句的文档将会合并起来计算分值。

   1) must：必须满足

   子句（查询）必须出现在匹配的文档中，并将有助于得分。

   2) filter：过滤器 不计算相关度分数，cache☆

   子句（查询）必须出现在匹配的文档中。但是不像 must查询的分数将被忽略。Filter子句在filter上下文中执行，这意味着计分被忽略，并且子句被考虑用于缓存。

   3) should：可能满足 or

   子句（查询）应出现在匹配的文档中。

   4) must_not：必须不满足 不计算相关度分数 not

   子句（查询）不得出现在匹配的文档中。子句在过滤器上下文中执行，这意味着计分被忽略，并且子句被视为用于缓存。由于忽略计分，0因此将返回所有文档的分数。

   5) minimum_should_match

2. 案例：

1) demo案例

#首先筛选name包含“xiaomi phone”并且价格大于1999的数据（不排序），

#然后搜索name包含“xiaomi”and desc 包含“shouji”

```
GET /product/_search
{
  "query": {
    "bool":{
      "must": [
        {"match": { "name": "xiaomi"}},
        {"match": {"desc": "shouji"}}
      ],
      "filter": [
        {"match_phrase":{"name":"xiaomi phone"}},
        {"range": {
          "price": {
            "gt": 1999
          }
        }}
      ]
    }
  }
}
```

2) bool多条件 name包含xiaomi 不包含erji 描述里包不包含nfc都可以，价钱要大于等于4999

```
#
GET /product/_search
{
  "query": {
      "bool":{
          #name中必须不能包含“erji”
      "must": [
        {"match": { "name": "xiaomi"}}
      ],
          #name中必须包含“xiaomi”
      "must_not": [
        {"match": { "name": "erji"}}
      ],
        #should中至少满足0个条件，参见下面的minimum_should_match的解释
      "should": [
        {"match": {
          "desc": "nfc"
        }}
      ],
        #筛选价格大于4999的doc
      "filter": [
        {"range": {
          "price": {
            "gt": 4999
          }
        }}
      ]
    }
  }
}
```

3. 嵌套查询：

1) minimum_should_match：参数指定should返回的文档必须匹配的子句的数量或百分比。如果bool查询包含至少一个

should子句，而没有must或 filter子句，则默认值为1。否则，默认值为0

```
GET /product/_search
{
  "query": {
    "bool":{
      "must": [
        {"match": { "name": "nfc"}}
      ],
      "should": [
        {"range": {
          "price": {"gt":1999}
        }},
        {"range": {
          "price": {"gt":3999}
        }}
      ],
      "minimum_should_match": 1
    }
  }
}
```

2) 案例：

```
GET /product/_search
{
  "query": {
    "bool": {
      "filter": {
        "bool": {
          "should": [
            { "range": {"price": {"gt": 1999}}},
            { "range": {"price": {"gt": 3999}}}
          ],
          "must": [
            { "match": {"name": "nfc"}}
          ]
        }
      }
    }
  }
}
```

(6) Compound queries：组合查询

1. 想要一台带NFC功能的 或者 小米的手机 但是不要耳机

```
SELECT * from product
where (`name` like "%xiaomi%" or `name` like '%nfc%')
AND `name` not LIKE '%erji%'
```

```
GET /product/_search
{
  "query": {
    "constant_score":{
      "filter": {
        "bool": {
          "should":[
            {"term":{"name":"xiaomi"}},
            {"term":{"name":"nfc"}}
            ],
          "must_not":[
            {"term":{"name":"erji"}}
            ]
        }
      },
      "boost": 1.2
    }
```

```
 }
}
```

2. 搜索一台xiaomi nfc phone或者一台满足 是一台手机 并且 价格小于等于2999

```
SELECT * FROM product
WHERE NAME LIKE '%xiaomi nfc phone%'
OR (
    NAME LIKE '%erji%'
    AND price > 399
    AND price <=999);
GET /product/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "bool":{
          "should":[
            {"match_phrase":{"name":"xiaomi nfc phone"}},
            {
              "bool":{
                "must":[
                  {"term":{"name":"phone"}},
                  {"range":{"price":{"lte":"2999"}}}
                  ]
              }
            }
          ]
        }
      }
    }
  }
}
```

(7) Highlight search：

```
GET /product/_search
{
   "query" : {
      "match_phrase" : {
         "name" : "nfc phone"
      }
   },
   "highlight":{
     "fields":{
       "name":{}
     }
   }
}
```

3. Deep paging图解
   (1) 解释：当你的数据超过1W，不要使用
   (2) 返回结果不要超过1000个，500以下为宜
   (3) 解决办法：
      1. 尽量避免深度分页查询
      2. 使用Scroll search（只能下一页，没办法上一页，不适合实时查询）
4. Scroll search：图解
   (1) 解决 deep paging问题
5. filter缓存原理：图解