Elastic查询

Search

根据id查询

JSON 复制代码

```
GET heshen_test_v1/_doc/1
```

MGET查询

- 可以查询不同索引的数据一同返回

JSON 复制代码

```
GET _mget
{
  "docs":[
    {
      "_index":"heshen_test_v1",
      "_id":1
    },
    {
      "_index":"heshen_test_v2",
      "_id":1
    }
  ]
}
```

- 可以返回不同字段的数据，

JSON | 复制代码

```
GET _mget
{
  "docs":[
    {
      "_index":"heshen_test_v1",
      "_id":1,
      "_source": ["itemName","id"]
    },
    {
      "_index":"heshen_test_v2",
      "_id":1,
      "_source": ["weight","price"]
    }
  ]
}
```

Search

语法	范围
_search	全部索引
_all/_search	全部索引
heshen_test_v1/_search	heshen_test_v1
heshen_test_v1,heshen_test_v2/_search	heshen_test_v1,heshen_test_v2
heshen_test*/_search	heshen_test开头的索引

Url查询(用的不多)

- 使用q表示查询字符串
- size 返回条数
- doc

JSON | 复制代码

```
GET heshen_test_v1/_search?q=weight:13&&size=10
```

Request Body 查询

doc

格式以及返回

JSON | 复制代码

```
req:
GET heshen_test_v1/_search?search_type=query_then_fetch
{
  "from": 1,
  "size": 10,
  "timeout": "1s",
  "query": {
    "term": {
      "_id": {
        "value": 1
      }
    }
  }
}

resp:
{
  "took" : 1, //花费的时间
  "timed_out" : false, //是否超时
  "_shards" : {//分片
    "total" : 3,
    "successful" : 3,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,//总文档数
    "max_score" : 1.0,
    "hits" : [//结果集
      {
        "_index" : "heshen_test_v1",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          ...
        }
      }
    ]
  }
}
```

Query DSL

Query 和 Filter

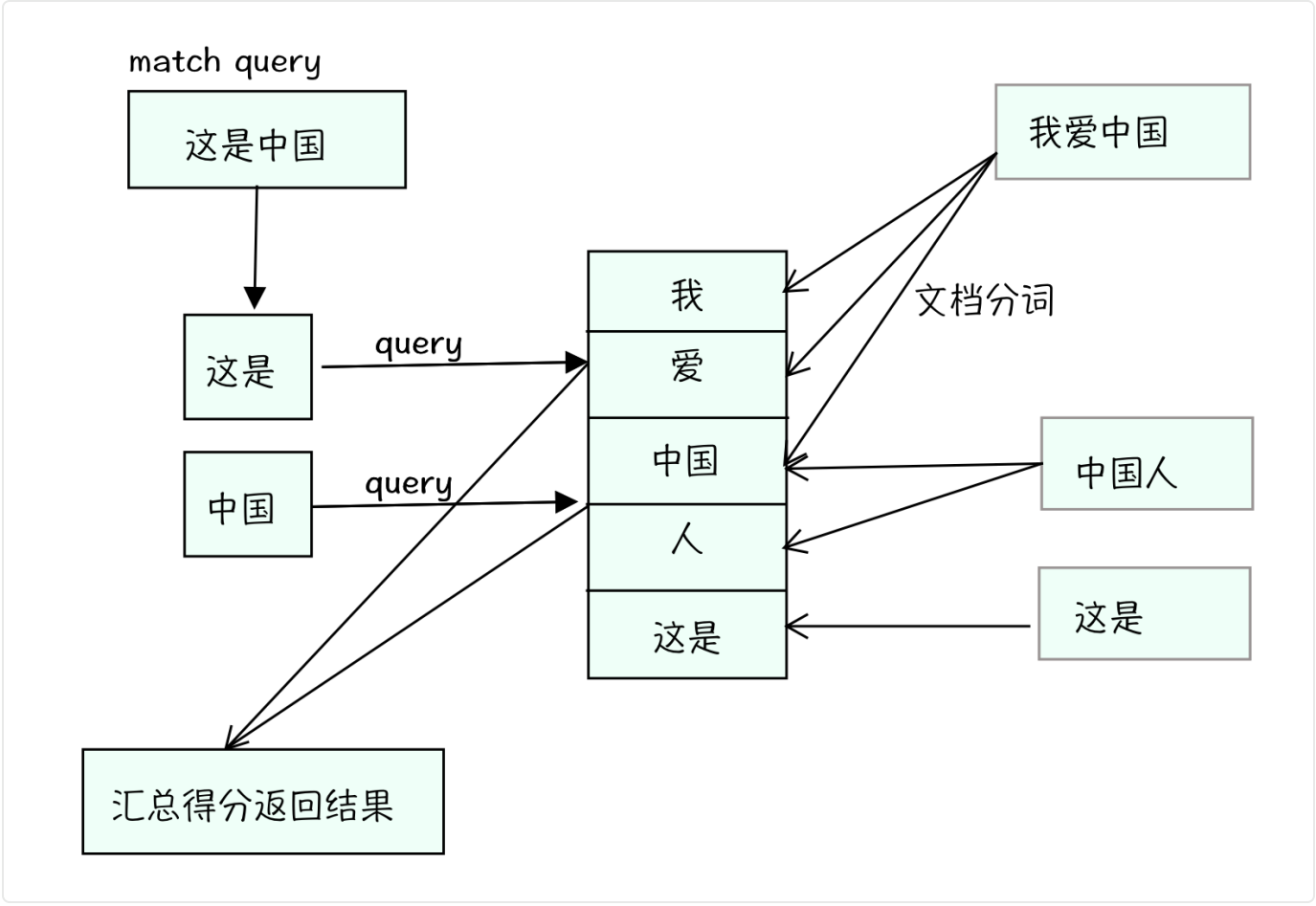
- 区别在于 filter 不计算相关性，只关心是否命中条件
- 计算相关性需要计算匹配分值，消耗性能。而且匹配的分值都是实时计算，没有缓存
- es会自动缓存 filter ，以提高性能，所以应该尽量使用过滤查询以减少性能消耗

JSON 复制代码

```
GET /_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title":   "Search"        }},
        { "match": { "content": "Elasticsearch" }}
      ],
      "filter": [
        { "term":  { "status": "published" }},
        { "range": { "publish_date": { "gte": "2015-01-01" }}}
      ]
    }
  }
}
```

Full text query 全文搜索

- 全文检索，被查询的字段需要被分析
- 查询条件也会被分析



Match Query

- 最基本的全文索引查询，支持单词查询，模糊查询，短语查询，近义词查询

JSON 复制代码

```
GET heshen_text/_search
{
  "query": {
    "match": {
      "title": "中国"
    }
  }
}
```

Match Pharse Query

- 类似match，专门查询短语，可以指定短语的间隔

JSON | 复制代码

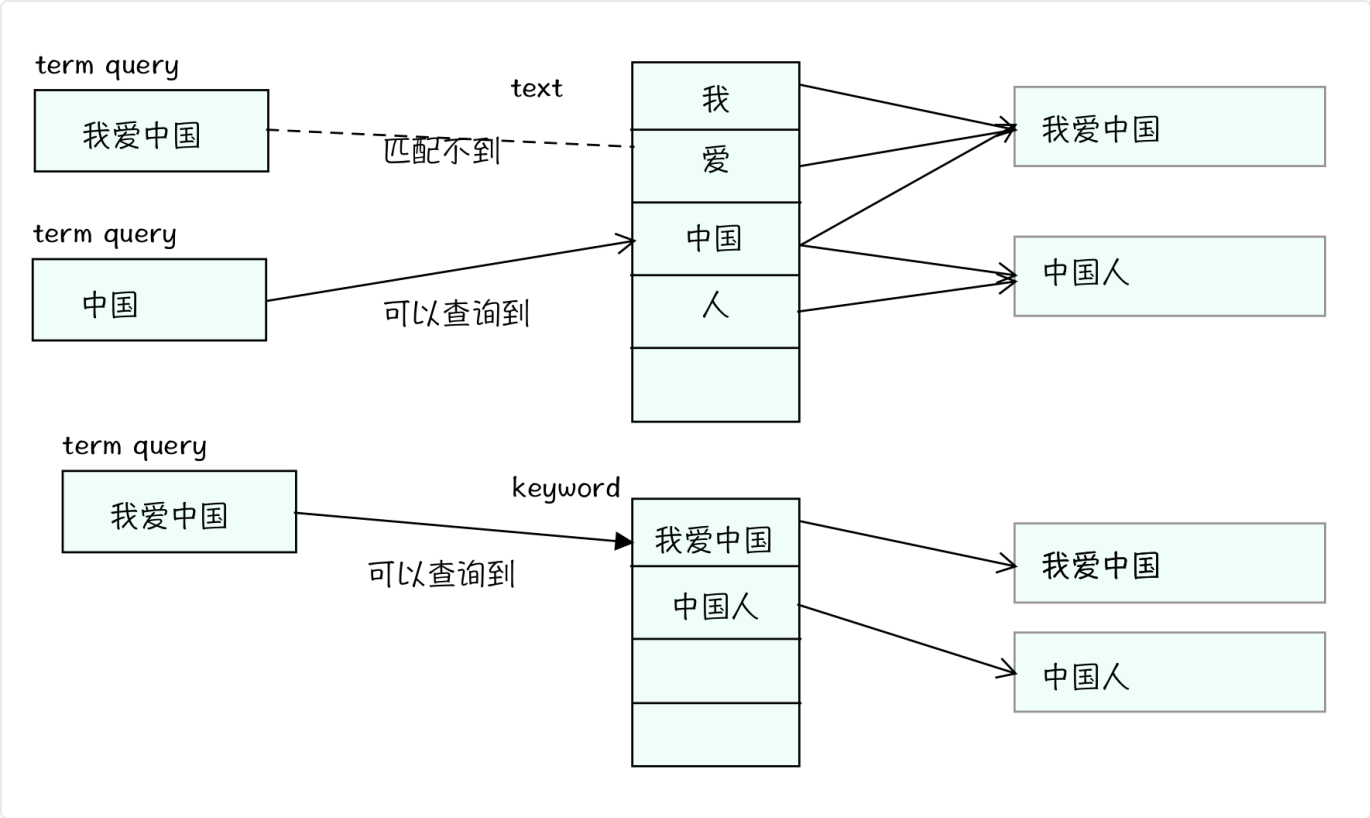
```
GET heshen_text/_search
{
  "query": {
    "match_phrase": {
      "title": {
        "query": "这中国",
        "slop": 1
      }
    }
  }
}
```

Match Phrase Prefix Query

- 类似于短语的查询，但是最后一个单词是前缀匹配
- 不如 is t 可以命中 this is a test

Term query 结构化搜索

- 精确匹配查询，查询条件不会被分析
- 通过用于结构化的数据，
- 对于被分析过的字段也可以使用，不过匹配的是分词之后的单词



Term (最常用)

- 单值等于匹配
- 类似于 =

JSON 复制代码

```
GET _search
{
  "query": {
    "term" : { "user" : "Kimchy" }
  }
}
```

Terms (最常用)

- 多值匹配
- 类似于 IN

JSON 复制代码

```
GET /_search
{
    "query": {
        "terms" : { "user" : ["kimchy", "elasticsearch"]}
    }
}
```

Terms Set

- 可以指定最小匹配数量

Range (常用)

- 可以按照区间查询日期，数字，以及区间类型
- 类似于 between

JSON 复制代码

```
GET _search
{
    "query": {
        "range" : {
            "age" : {
                "gte" : 10,
                "lte" : 20
            }
        }
    }
}
```

Exists (常用)

- 存在查询，返回字段不是 null 或者 [] 空数组的文档

```
GET /_search
{
    "query": {
        "exists": {
            "field": "user"
        }
    }
}
```

- 配合 bool 查询的must_not可以实现 不存在 条件

```
GET /_search
{
    "query": {
        "bool": {
            "must_not": {
                "exists": {
                    "field": "user"
                }
            }
        }
    }
}
```

模糊搜索

wildcard	根据占位符的内容进行匹配	
prefix	前缀匹配	
fuzzy	相似度匹配	
regexp		

wildcard 通配符查询

- 通配符 * ,类似于 wildcard like "X%"

- 匹配0个或者多个字符

```JSON
GET /_search
{
    "query": {
        "wildcard": {
            "user": {
                "value": "X*"
            }
        }
    }
}
```

- 占位符 ？,,类似于 wildcard like "X_"
- 匹配单个字符

```JSON
GET /_search
{
    "query": {
        "wildcard": {
            "user": {
                "value": "X?"
            }
        }
    }
}
```

prefix 前缀匹配

- 查找指定字段包含以指定确切前缀开头的术语的文档
- 类似于 wildcard like "X%"

JSON 复制代码

```
GET /_search
{ "query": {
    "wildcard" : { "author": "X" }
  }
}
```

fuzzy 误拼写模糊匹配

- 使用基于Levenshtein编辑距离实现的相似搜索
- Levenshtein编辑距离是什么：

Plain Text 复制代码

```
Levenshtein 距离，又称编辑距离，指的是两个字符串之间，由一个转换成另一个所需的最少编辑操作次数。
许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。
编辑距离的算法是首先由俄国科学家Levenshtein提出的，故又叫Levenshtein Distance
```

Plain Text | 复制代码

```
GET /_search
{
    "query": {
        "fuzzy" : {
            "user" : {
                "value": "ki",
                "boost": 1.0,
                "fuzziness": 2,
                "prefix_length": 0,
                "max_expansions": 100
            }
        }
    }
}
```

fuzziness：最大编辑距离【一个字符串要与另一个字符串相同必须更改的一个字符数】。默认为AUTO。

prefix_length：不会被“模糊化”的初始字符数。这有助于减少必须检查的术语数量。默认为0。

max_expansions：fuzzy查询将扩展到的最大术语数。默认为50。

transpositions：是否支持模糊转置（ab→ ba）。默认值为false。

Compound 复合查询

Bool (最常用)

- 最常用的组合查询，包含 must ， should ， must_not ， filter ,可以嵌套
- bool.filter的分值计算, 在filter子句查询中，分值将会都返回0。分值会受特定的查询影响。

复合类型

类型	解释
must	返回的文档必须满足must子句的条件，并且参与计算分值
should	1. 返回的文档可能满足should子句的条件。在一个Bool查询中，如果没有must或者filter，有一个或者多个should子句，那么只要满足一个就可以返回。minimum_should_match参数定义了至少满足几个子句 2. 如果一个查询既有filter又有should，那么至少包含一个should子句。
must_not	返回的文档必须不满足must_not定义的条件。
filter	返回的文档必须满足filter子句的条件。但是不会像Must一样，参与计算分值

使用

JSON | 复制代码

```
POST _search
{
  "query": {
    "bool" : {
      "must" : {
        "term" : { "user" : "kimchy" }
      },
      "filter": {
        "term" : { "tag" : "tech" }
      },
      "must_not" : {
        "range" : {
          "age" : { "gte" : 10, "lte" : 20 }
        }
      },
      "should" : [
        { "term" : { "tag" : "wow" } },
        { "term" : { "tag" : "elasticsearch" } }
      ],
      "minimum_should_match" : 1
    }
  }
```

其他不重要的复合查询

constant_score query

- 不计算得分，可以指定一个指定的常量分值

dis_max query

- 对多个自查询取最高的得分
- 如果自查询分值想近，还有加成的选项

function_score query

- 可以自定义评分的查询

boosting query

- 可以对子查询进行加分，和减分的操作

排序

排序是针对字段的原始内容进行的，倒排索引在排序中是没有用处的。
排序需要使用正排索引，通过文档id和字段快速得到原始的文档内容。
ES对于排序，有两种方式：
– Field Data 支持
– Doc Values (列式存储，对Text无效)