

RESTful

介绍和使用教程

1. REST起源

表述性状态转变，基于HTTP、URI、XML、JSON等标准和协议，支持轻量级、跨平台、跨语言的架构设计，是Web服务的一种新的架构风格(一种思想)。

2. REST架构主要原则

- 对网络上所有资源都有一个资源标识符
- 对资源的操作不会改变标识符
- 同一资源有多种表现形式(xml、json)
- 所有操作都是无状态的(客户端和服务端不必保存对方详细信息)

符合上述原则的架构方式称为RESTful。RESTful是遵循REST风格的web服务，REST式的web服务是一种ROA(面向资源的架构)。

3. URL(统一资源定位符)和URI(统一资源标识符)

URL是URI的特例，包含了定位web资源的足够信息，而URI不包含任何访问资源的方法，唯一作用是解析。

URI一般由三部分组成：

- 访问资源的命名机制
- 存放资源的主机名
- 资源自身的名称，由路径表示，着重强调于资源

例如：`/ServletDemo/mydemo3`

URL也由三部分组成：

- 协议(或称为服务方式)
- 存有该资源的主机IP地址(有时包括端口号)
- 主机资源的具体地址，如目录和文件名等

例如：`http://localhost:8080/ServletDemo/mydemo3`

ES与Solr对比

1. Solr利用Zookeeper进行分布式处理，而ES自身带有分布式协调管理功能
2. Solr支持更多格式数据，而ES仅支持JSON格式
3. ES在处理实时搜索应用时效率更高，实时性更好，比如新往Github中创建一个工程，能马上搜索到，而Solr在传统的搜索应用中表现好于ES

安装配置(Linux)

1. jdk1.8及以上

2. ES图形化操作界面(chrome浏览器插件方式安装head)

[下载](#)并解压->chrome->更多工具->扩展程序->加载已解压扩展程序->选择head插件即可

3. 安装node.js

- 下载[linux版本64-bit的nodejs](#)并解压
- 重命名 `mv node-v12.14.1-linux-x64 nodejs-v12.14.1`

- 建立软连接

```
sudo ln -s nodejs-v12.14.1/bin/node /usr/local/bin
```

```
sudo ln -s nodejs-v12.14.1/bin/npm /usr/local/bin
```

- 查看nodejs版本

```
node -v
```

若提示:

```
The program 'node' can be found in the following packages:
```

```
*node
```

```
*nodejs-legacy
```

则执行命令: `sudo apt-get install nodejs-legacy`

4. 添加非root用户(若此时是root用户)

原因: Elastic默认不支持root用户运行

执行命令: `useradd 用户名`

5. 更改目标目录权限(若此时是root用户)

执行命令: `chown 用户名:用户名 目标目录/ -R`

切换用户: `su - 用户名`

6. 下载并解压ES7.5

解压路径: `/home/hadoop/Downloads/elasticsearch-7.5.1`

配置环境变量(可选: 以 `elasticsearch` 即可在终端启动服务)

```
vim ~/.bashrc
export ES=/home/hadoop/Downloads/elasticsearch-7.5.1
export PATH=${ES}/bin:

source ~/.bashrc #使配置立即生效
```

7. 修改配置

- 修改elasticsearch.yml中 `network.host:0.0.0.0` ->任意网络均可访问

- 修改jvm.options

`network.host` 地址不是 `localhost` 或 `127.0.0.1` 时, 即认为是生产环境, 要求配置更高

`-xms1g` ->初始堆内存, 默认1g, 需要根据虚拟机变化

`-xmx1g` ->最大堆内存, 默认1g

- root权限下修改内存映射最大数量, 可以是root用户下修改, 或者执行 `sudo` 命令修改

```
sudo vim /etc/sysctl.conf ,vm.max_map_count=65535
```

```
sudo sysctl -p ->配置生效
```

8. 访问

ES启动后, 默认绑定两个端口, **9200->http(浏览器访问); 9300->tcp(使用api对服务器管理的端口)**

启动: `elasticsearch`

后台启动: `elasticsearch -d`

停止:

```
jps -l #查看进程
kill elasticsearch进程ID
```

- 浏览器访问

输入: localhost:9200

结果:

```
{
  "name" : "node-1",
  "cluster_name" : "liu", //集群名字
  "cluster_uuid" : "dZOY4LIVSb-8Kv0Id8fn4Q",
  "version" : {
    "number" : "7.5.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "3ae9ac9a93c95bd0cdc054951cf95d88e1e18d96",
    "build_date" : "2019-12-16T22:57:37.835892Z",
    "build_snapshot" : false,
    "lucene_version" : "8.3.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

- 终端访问

输入: curl 'http://localhost:9200/?pretty'

结果:

```
hadoop@ubuntu:~$ curl 'http://localhost:9200/?pretty'
{
  "name" : "node-1",
  "cluster_name" : "liu",
  "cluster_uuid" : "dZOY4LIVSb-8Kv0Id8fn4Q",
  "version" : {
    "number" : "7.5.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "3ae9ac9a93c95bd0cdc054951cf95d88e1e18d96",
    "build_date" : "2019-12-16T22:57:37.835892Z",
    "build_snapshot" : false,
    "lucene_version" : "8.3.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

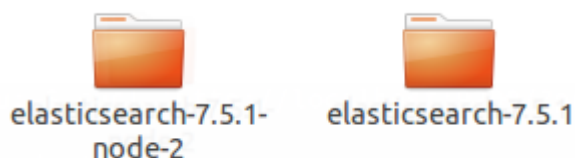
9. 集群与节点

节点: 运行着es的实例

集群: 一组拥有相同 cluster.name 的节点集合

构建集群(关闭防火墙):

- 解压ES安装包并重命名为 elasticsearch-7.5.1-node-2,此时目录如下:

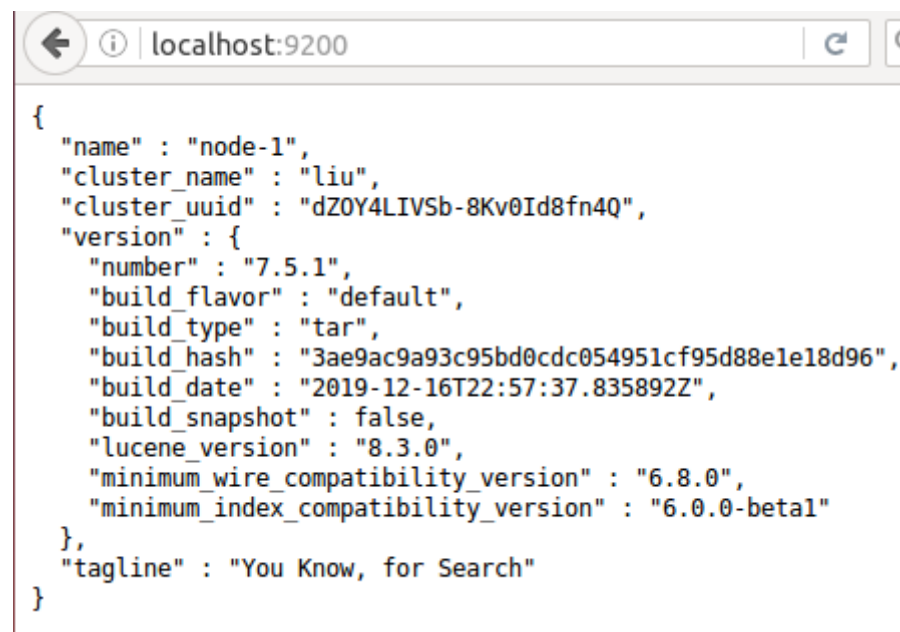


- 分别进入并打开两个文件夹的 `./config/elasticsearch.yml` 文件，修改属性 `cluster.name:liu`（属性值一致即可），及属性 `node.name`（属性不一样即可）
- 分别启动节点node-1(`http:9200,tcp:9300`)和节点node-2(`http:9201,tcp:9301`): `./bin/elasticsearch`
- 检查集群
查看进程: `jps -l`
结果:

```
hadoop@ubuntu:~$ jps -l
11510 org.elasticsearch.bootstrap.Elasticsearch
13339 org.elasticsearch.bootstrap.Elasticsearch
13549 sun.tools.jps.Jps
```

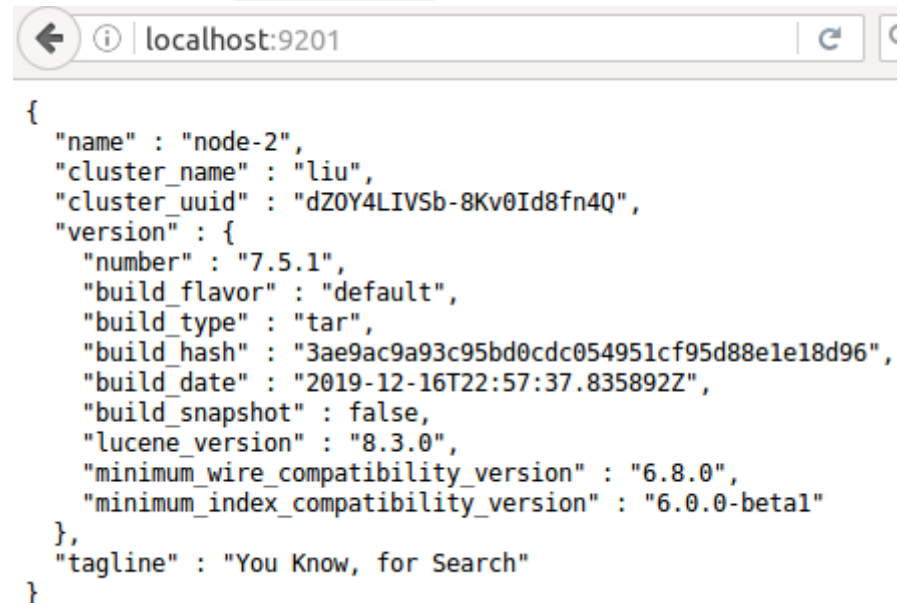
■ 浏览器访问

访问node-1节点: `localhost:9200`



```
{
  "name" : "node-1",
  "cluster_name" : "liu",
  "cluster_uuid" : "dZ0Y4LIVSb-8Kv0Id8fn4Q",
  "version" : {
    "number" : "7.5.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "3ae9ac9a93c95bd0cdc054951cf95d88e1e18d96",
    "build_date" : "2019-12-16T22:57:37.835892Z",
    "build_snapshot" : false,
    "lucene_version" : "8.3.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

访问node-2节点: `localhost:9201`



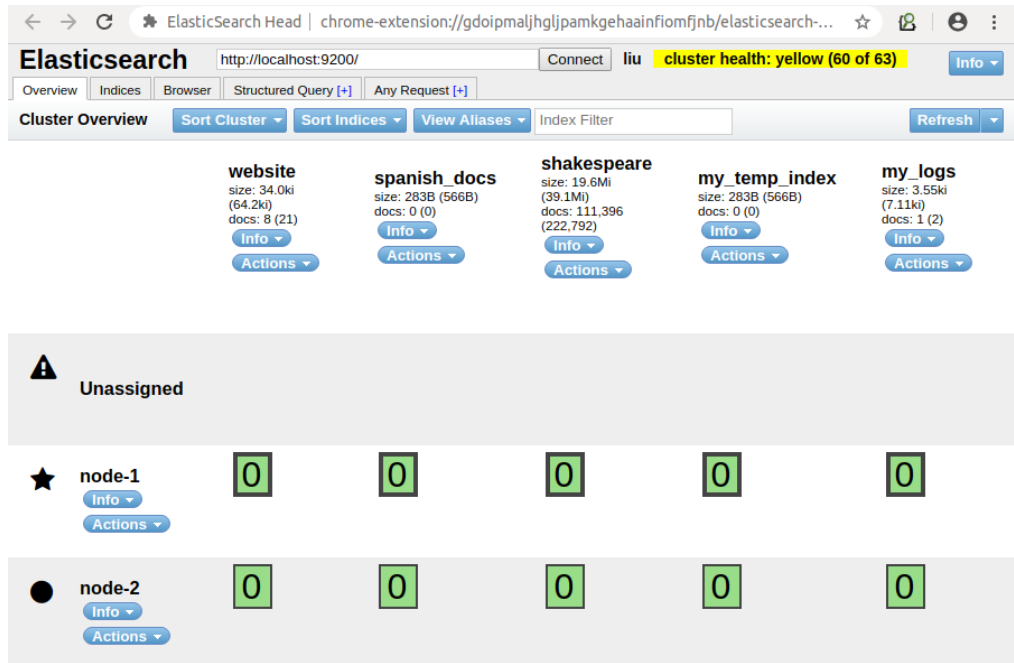
```
{
  "name" : "node-2",
  "cluster_name" : "liu",
  "cluster_uuid" : "dZ0Y4LIVSb-8Kv0Id8fn4Q",
  "version" : {
    "number" : "7.5.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "3ae9ac9a93c95bd0cdc054951cf95d88e1e18d96",
    "build_date" : "2019-12-16T22:57:37.835892Z",
    "build_snapshot" : false,
    "lucene_version" : "8.3.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

■ 终端访问

访问node-1节点: `curl 'http://localhost:9200/?pretty'`

访问node-2节点: `curl 'http://localhost:9201/?pretty'`

■ chrome-head插件访问(2个节点)



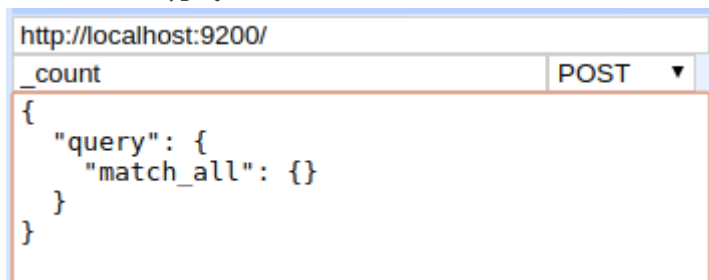
问题: 复制并启动node-2后, 执行集群健康监控命令, 显示节点数目仍为1

解决: 将node-2的data目录下内容清空, 重启node-2节点, 再查看集群节点数目, 显示为2。

10. 小试牛刀

计算集群中文档数量:

- chrome-head方式



- 终端方式

```
curl -H "Content-Type:application/json" -XPOST
'http://localhost:9200/_count?pretty' -d '
{
  "query":{
    "match_all":{}
  }
}'
```

倒排索引

[搜索引擎之倒排索引解读](#)

[倒排索引-搜索引擎入门](#)

倒排索引(反向索引): 基于信息主体的关键属性值进行构建的“关键词-文档”形式的一种映射结构, 实现了通过物品属性信息对物品进行映射时, 可以帮助用户快速定位到目标信息, 从而极大降低了信息获取难度。

倒排索引构建过程：

1. Doc2term词项构造(词项集，即单词词典)

该过程主要是利用分词系统将文档中的各项属性的文本信息拆分成一些表意较强且重要的词汇，便于用户查找。

- 文本词条化
主要任务是将一段连续的文本序列信息拆分成多个子序列；需要用到NLP相关技术对内容进行特征抽取，生成对应词典，再基于词典利用分词器进行分词。
- 停用词过滤
- 词条归一化
任务就是将一些看起来不完全一致的词条划分为一个等价类，比如英式单词colour和美式单词color归为一类，用户在查询时，只需要对等价类中的任意单词进行搜索
- 词干提取、词行还原
词干提取的主要思想是“缩减”，将词条转化为词干，如将复数形式转换为单数形式

2. 倒排记录表(倒排文件)的构建

面向的是海量的文档数据集合，无法完全存放在内存当中，需要写入磁盘，需要考虑内存使用。

在无法全内存的情况下，倒排记录表的主要构建思想是“分割”，亦即基于一定的处理逻辑对全量文档集合进行等份的批量处理。

基本构建方法如下：

1. 通过一系列的处理将文档集合转化为“词项ID—文档ID”对
2. 对词项ID、文档ID进行排序，将具有相同“词项ID-文档ID”归并到该词项所对应的倒排记录表中
3. 将上述步骤产生的倒排索引写入磁盘，生成中间文件
4. 将上述所有的中间文件合并成最终的倒排索引

从**业务应用场景**的角度出发，倒排记录表的构建方法主要有：**单遍扫描构建和多遍扫描构建**：

- 单遍扫描：指的是仅对文档集合进行一次遍历，即可完成倒排索引的构建。由于内存开销问题，会将全量文档集进行分割，转换成几个内存大小相同的文档集合，然后依次执行前文中提及到的构建方法。
- 多遍扫描：主要用于构建索引时获取关于文档的更多相关信息，搜索用户的需求并不止于关键字查询，像短语查询、模糊查询、精确筛选、模糊筛选、排序、聚合统计等等需求。

从**工程角度**出发，倒排记录表的构建方法主要有：**分布式构建和动态构建**：

- 分布式构建：对于一些大型搜索引擎如Web搜索引擎，单台机器已无法支撑其索引构建，需要多台机器组成集群对其进行分布式处理，将构建成的倒排索引进行分割，分布在多台机器上，每台机器各自形成独立的索引结构。
- 动态构建：该方法中的文档集合是变化的，这要求在对文档集进行索引构建时也要对文档的更新进行自适应。如商品的上下架，都会引起索引的动态更新问题。

常见策略：

- **周期性对文档进行全量重建索引**：最简单直接、且有效的索引更新策略，对于数量级较大的搜索引擎来说处理简单便捷，由于动态索引计算的复杂性，使用其它策略会使得索引难维护，甚至引发严重的性能问题。**适用于大型搜索引擎**，且不会涉及到索引热切换问题。
- **基于主索引的前提下，构建辅助索引**，用于储存新文档，维护于内存中，当辅助索引达到一定的内存占用时，写入磁盘与主索引进行合并：进行主辅索引合并时会遇到比较大的存储开销，该策略在生产环境中可用性并不高。

入门实践

注：实践代码均为终端命令下代码，但在chrome-head中进行实际操作
(索引==数据库，类型==表，文档==行，每个员工信息即一个文档)

1. 索引员工文档（目录中加入员工信息）

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/megacorp/employee/1?pretty' -d '{
  "first_name":"John",
  "last_name":"Smith",
  "age":25,
  "about":"I love suzhou",
  "interests":["sports","music"]
}'
```

响应结果：

```
{
  "_index" : "megacorp", //索引名
  "_type" : "employee", //类型
  "_id" : "1", //文档id
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

2. 检索单个文档(指明索引、类型、ID)

索引：megacorp

类型：employee

ID：1

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/1?pretty'
```

检索部分结果：

```
{
  "_index": "megacorp", //索引
  "_type": "employee", //类型
  "_id": "1", //ID
  "_version": 1,
  "_seq_no": 0,
  "_primary_term": 1,
  "found": true,
  "_source": { //文档内容
    "first_name": "John",
```

```
{
  "last_name": "Smith",
  "age": 25,
  "about": "I love suzhou",
  "interests": [
    "sports",
    "music"
  ]
}
```

GET：检索文档

DELETE：删除文档

HEAD：检查文档是否存在

PUT：更新已存在文档

简易搜索

search API有两种表单：一种简易版查询字符串，另一种DSL(使用JSON完整表示请求体)

查询字符串搜索类型"blog"，且字段"title"中包含"My"字符的文档：

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_all/blog/_search?q=title:My&pretty'
```

查询字符串搜索允许任意用户在索引中任何一个字段上运行潜在的慢查询语句，可能暴露私有信息甚至使集群瘫痪，故**生产环境中一般使用全功能的请求体搜索API**。

3. 搜索全部员工(指明索引、类型)

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty'
```

部分结果(默认返回前10条结果)

```
{
  "took": 56,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 3, //命中数量
      "relation": "eq"
    },
    "max_score": 1.0, //最大匹配分数
    "hits": [
      {
        "_index": "megacorp", //索引
        "_type": "employee", //类型
        "_id": "1", //文档ID
        "_score": 1.0, //匹配分数
        "_source": { //文档内容
```



```

        "first_name": "John",
        "last_name": "Smith",
        "age": 25,
        "about": "I love suzhou",
        "interests": [
            "sports",
            "music"
        ]
    },
    {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "2",
        "_score": 1.0,
        "_source": {
            "first_name": "Jane",
            "last_name": "Smith",
            "age": 32,
            "about": "I love qingdao",
            "interests": [
                "music"
            ]
        }
    },
    {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "3",
        "_score": 1.0,
        "_source": {
            "first_name": "Douglas",
            "last_name": "Fir",
            "age": 35,
            "about": "I like swimming",
            "interests": [
                "forestry"
            ]
        }
    }
]
}

```

4. 搜索姓氏中包含“Smith”的员工

查询参数之间使用 & 连接，例如要查询姓“Smith”，名“Jane”，则：

```
q=last_name:Smith&q=first_name:Jane
```

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty&q=last_name:Smith'
```

部分检索结果：

```

{
  "took": 118,
  "timed_out": false,

```

```

    "_shards": {
      "total": 1,
      "successful": 1,
      "skipped": 0,
      "failed": 0
    },
    "hits": {
      "total": {
        "value": 2,
        "relation": "eq"
      },
      "max_score": 0.47000363,
      "hits": [
        {
          "_index": "megacorp",
          "_type": "employee",
          "_id": "1",
          "_score": 0.47000363,
          "_source": {
            "first_name": "John",
            "last_name": "Smith",
            "age": 25,
            "about": "I love suzhou",
            "interests": [
              "sports",
              "music"
            ]
          }
        },
        {
          "_index": "megacorp",
          "_type": "employee",
          "_id": "2",
          "_score": 0.47000363,
          "_source": {
            "first_name": "Jane",
            "last_name": "Smith",
            "age": 32,
            "about": "I love qingdao",
            "interests": [
              "music"
            ]
          }
        }
      ]
    }
  }
}

```

特定领域语言(DSL)

DSL以JSON请求体形式替代查询字符串命令行参数，允许构建更复杂、更强大的查询。

上例4中查询姓氏为“Smith”的DSL形式为：

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d '
{
  "query": {
    "match": {
      "last_name": "Smith"
    }
  }
}'
```

5. 搜索年龄<=30,且姓氏为"Smith"的员工

/bool/filter/ must/ ->先过滤年龄, 再匹配形式; 其中 gt 为"greater than"缩写

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
  "query": {
    "bool": {
      "filter": {
        "range": {
          "age": {
            "gt": 30
          }
        }
      },
      "must": {
        "match": {
          "last_name": "Smith"
        }
      }
    }
  }
}'
```

检索结果:

```
{
  "took": 32,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 0.47000363,
    "hits": [ //命中数据集合
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "2",
        "_score": 0.47000363,
```

```

        "_source": {
          "first_name": "Jane",
          "last_name": "Smith",
          "age": 32,
          "about": "I love qingdao",
          "interests": [
            "music"
          ]
        }
      ]
    }
  }
}

```

6. 搜索 所有喜欢“swimming”的员工

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
  "query": {
    "match": {
      "about": "swimming"
    }
  }
}'

```

检索结果:

```

{
  "took": 3,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 0.9808292,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "3",
        "_score": 0.9808292,
        "_source": {
          "first_name": "Douglas",
          "last_name": "Fir",
          "age": 35,
          "about": "I like swimming",
          "interests": [
            "forestry"
          ]
        }
      }
    ]
  }
}

```

```

    }
  }
}

```

默认情况下，ES根据结果相关性(文档与检索条件的匹配程度)评分对检索结果排序，而传统数据库对记录的查询只有匹配或不匹配。

7. 短语搜索(确切的匹配若干个单词或者短语)

将"match"改为"match_phrase"，其实对于下面检索语句来讲，这两种检索结果是一样的

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d
'{
  "query": {
    "match_phrase": {
      "about": "like swimming"
    }
  }
}'

```

检索结果：

```

{
  "took": 64,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 1.3862944,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "3",
        "_score": 1.3862944,
        "_source": {
          "first_name": "Douglas",
          "last_name": "Fir",
          "age": 35,
          "about": "I like swimming",
          "interests": [
            "forestry"
          ]
        }
      }
    ]
  }
}

```

```
}
```

8. 高亮搜索(检索结果中高亮匹配关键字)

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d
'{
  "query": {
    "match_phrase": {
      "about": "like swimming"
    }
  },
  "highlight": {
    "fields": {
      "about": {}
    }
  }
}'
```

检索结果中有新部分"highlight":

```
"highlight" : {
  "about" : [
    "I <em>like</em> <em>swimming</em>"
  ]
}
```

分析统计

ES有聚合功能, 允许在数据上生成复杂的分析统计, 比GroupBy功能更强大

9. 统计各兴趣点对应的人数(等同于GroupBy(兴趣点))

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d
'{
  "aggs": {
    "all_interests": {
      "terms": {
        "field": "interests.keyword" #以"interests"为key统计
      }
    }
  }
}'
```

统计结果:

```
{
  "took": 44,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  }
}
```

```

},
"hits": {
  "total": {
    "value": 3,
    "relation": "eq"
  },
  "max_score": 1.0,
  "hits": [
    {
      "_index": "megacorp",
      "_type": "employee",
      "_id": "1",
      "_score": 1.0,
      "_source": {
        "first_name": "John",
        "last_name": "Smith",
        "age": 25,
        "about": "I love suzhou",
        "interests": [
          "sports",
          "music"
        ]
      }
    },
    {
      "_index": "megacorp",
      "_type": "employee",
      "_id": "2",
      "_score": 1.0,
      "_source": {
        "first_name": "Jane",
        "last_name": "Smith",
        "age": 32,
        "about": "I love qingdao",
        "interests": [
          "music"
        ]
      }
    },
    {
      "_index": "megacorp",
      "_type": "employee",
      "_id": "3",
      "_score": 1.0,
      "_source": {
        "first_name": "Douglas",
        "last_name": "Fir",
        "age": 35,
        "about": "I like swimming",
        "interests": [
          "forestry"
        ]
      }
    }
  ]
},
"aggregations": {
  "all_interests": {

```

```

    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 0,
    "buckets": [
      {
        "key": "music",
        "doc_count": 2 //喜欢"music"的人数
      },
      {
        "key": "forestry",
        "doc_count": 1 //喜欢"forestry"的人数
      },
      {
        "key": "sports",
        "doc_count": 1 //喜欢"sports"的人数
      }
    ]
  }
}

```

10. 分析姓氏“Smith”的兴趣爱好(interests)对应的人数

等同于 `where last_name=Smith group by interests`

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d '{
  "query": {
    "match": {
      "last_name": "Smith"
    }
  },
  "aggs": {
    "all_interests": {
      "terms": {
        "field": "interests.keyword" #以"interests"为key统计姓氏"Smith"人数
      }
    }
  }
}'

```

分析结果:

```

{
  "took": 6,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2,
      "relation": "eq"
    },

```



```

    "max_score": 0.47000363,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "1",
        "_score": 0.47000363,
        "_source": {
          "first_name": "John",
          "last_name": "Smith",
          "age": 25,
          "about": "I love suzhou",
          "interests": [
            "sports",
            "music"
          ]
        }
      },
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "2",
        "_score": 0.47000363,
        "_source": {
          "first_name": "Jane",
          "last_name": "Smith",
          "age": 32,
          "about": "I love qingdao",
          "interests": [
            "music"
          ]
        }
      }
    ]
  },
  "aggregations": {
    "all_interests": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "music",
          "doc_count": 2 //两个姓氏"Smith"的人喜欢"music"
        },
        {
          "key": "sports",
          "doc_count": 1 //两个姓氏"Smith"的人喜欢"sports"
        }
      ]
    }
  }
}

```

11. 聚合分级汇总：统计每种兴趣下员工的平均年龄
 等同于 `avg(age) group by interests`

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
  "aggs": {
    "all_interests": {
      "terms": {
        "field": "interests.keyword"
      },
      "aggs": {
        "avg_age": {
          "avg": {
            "field": "age"
          }
        }
      }
    }
  }
}'
```

分级汇总结果:

```
{
  "took": 7,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 3,
      "relation": "eq"
    },
    "max_score": 1.0,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "1",
        "_score": 1.0,
        "_source": {
          "first_name": "John",
          "last_name": "Smith",
          "age": 25,
          "about": "I love suzhou",
          "interests": [
            "sports",
            "music"
          ]
        }
      },
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "2",
        "_score": 1.0,
```

```

        "_source": {
            "first_name": "Jane",
            "last_name": "Smith",
            "age": 32,
            "about": "I love qingdao",
            "interests": [
                "music"
            ]
        }
    },
    {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "3",
        "_score": 1.0,
        "_source": {
            "first_name": "Douglas",
            "last_name": "Fir",
            "age": 35,
            "about": "I like swimming",
            "interests": [
                "forestry"
            ]
        }
    }
]
},
"aggregations": {
    "all_interests": {
        "doc_count_error_upper_bound": 0,
        "sum_other_doc_count": 0,
        "buckets": [
            {
                "key": "music",
                "doc_count": 2,
                "avg_age": {
                    "value": 28.5
                }
            },
            {
                "key": "forestry",
                "doc_count": 1,
                "avg_age": {
                    "value": 35.0
                }
            },
            {
                "key": "sports",
                "doc_count": 1,
                "avg_age": {
                    "value": 25.0
                }
            }
        ]
    }
}
}

```

分布式集群

ES用于构建高可用和可扩展的系统，致力于隐藏分布式系统的复杂性。

扩展方式： 纵向扩展(更好的服务器)和横向扩展(更多的服务器)。纵向扩展有其局限性，真正的扩展应该是横向的，通过添加节点来均摊负载和增加可靠性。

节点： 一个ES实例。

集群： 一个或多个节点组成，具有相同的cluster.name，集群内节点协同工作，分享数据和负载。

主节点： 集群内含有一个主节点，任何节点都可以成为主节点，负责临时管理集群级别的一些变更(如：索引和节点的创建或删除)，不参与文档级别的变更和搜索。

通信过程： 每个节点都知道要访问的文档存在哪个节点上，可以转发请求到相应的节点上，我们访问的节点负责收集各节点返回的数据，最后一期返回给客户端。

集群状态

三种状态： green、yellow、red

终端查看集群状态：`curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/_cluster/health/?pretty'`

green：所有主要分片和复制分片都可用。

yellow：所有主要分片可用，但不是所有复制分片都可用。

red：不是所有的主要分片都可用。

1. 创建索引"blogs"，并为其分配3个主分片和1个复制分片(默认情况下，一个索引分配5个主分片)

```
curl -H "Content-Type:application/json" -XPUT 'http://localhost:9200/blogs'
-d '{
  "settings": {
    "number_of_shards": 3,
    "number_of_replicas": 1
  }
}'
```

主分片： 主分片的数量决定索引最多能存储的容量，索引创建完成时，主分片的数量就确定了，索引中的每个文档属于一个单独的主分片。**应用程序直接与索引通信**，而不是分片。**复制分片的数量可以在运行的集群中动态变更**，如下是设置每个主分片有2个复制分片：

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/blogs/_settings?pretty' -d '{
  "number_of_replicas": 2
}'
```

横向扩展的节点的最大数目： 主分片+复制分片，每个分片独占一个节点，由于主分片数目在索引创建完成后已经确定，若想继续扩展，则修改复制分片数量。

数据

1. 文档
特指最顶层结构或者根对象(root object)序列化成的JSON数据(以唯一ID标识，并存储于Elasticsearch中)
2. 文档元数据(关于文档的信息)
三个必须元数据：

- `_index`: 文档存储的地方
- `_type`: 文档代表的对象的类(7.5已固定, 8.x会删掉), **从6.0以后不允许一个索引下创建多个类型**, 否则报错如下(表示该索引下包含了[blog, pageviews]两种类型):

```
"type" : "illegal_argument_exception",
"reason" : "Rejecting mapping update to [website] as the final mapping
would have more than 1 type: [blog, pageviews]"
```

- `_id`: 文档唯一标识

3. 索引一个文档

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/website/blog/123?pretty' -d ' {
  "title": "My first blog entry",
  "text": "Just trying this out...",
  "date": "2019/12/24"
}'
```

响应结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "123",
  "_version": 1, //每个文档都有一个版本, 每当文档变化(包括删除)都会使_version增加
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 0,
  "_primary_term": 1
}
```

4. 自增文档ID(不指定ID,请求结构: **POST**)

```
curl -H "Content-Type:application/json" -XPOST
'http://localhost:9200/website/blog/?pretty' -d ' {
  "title": "My second blog entry",
  "text": "Still trying this out...",
  "date": "2019/12/24"
}'
```

响应结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "oZvBNm8BgwLmYfoFvhlE",
  "_version": 1,
  "result": "created",
  "_shards": {
```

```
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 1,
  "_primary_term": 1
}
```

5. 检索文档ID为 123 的文档(请求结构方法: **GET**)

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/website/blog/123?pretty'
```

检索结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "123",
  "_version": 1,
  "_seq_no": 0,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "title": "My first blog entry",
    "text": "Just trying this out...",
    "date": "2019/12/24"
  }
}
```

6. 检索文档部分内容

只检索"title、text"字段: `curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/website/blog/123?_source=title,text '`

只检索_source 字段: `curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/website/blog/123/_source?pretty '`

检索结果:

```
{
  "title": "My first blog entry",
  "text": "Just trying this out...",
  "date": "2019/12/24"
}
```

7. 检索文档是否存在(请求方法: **HEAD**; 只有HTTP头, 不返回响应体)

```
curl -H "Content-Type:application/json" -i -XHEAD
'http://localhost:9200/website/blog/123/'
```

检索结果:

`HTTP/1.1 200 OK`: 代表文档存在

`HTTP/1.1 404 Not Found`: 代表在查询的那一刻文档不存在, 但并不表示几毫秒后依旧不存在。另一个进程在这期间可能创建新文档。

8. 更新文档

ES中文档不可变，更新文档，可用PUT重新索引该文档(覆盖旧文档)，"_version"会增加。

9. 创建一个新文档(而非覆盖一个已经存在的文档，当目标文档已经存在时，返回409)

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/website/blog/124/_create?pretty' -d ' {
  "title": "My third blog entry",
  "text": "Yet trying this out...",
  "date": "2019/12/24"
}'
```

响应结果：

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "124",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 2,
  "_primary_term": 1
}
```

10. 删除文档(请求方法：DELETE)

删除一个文档不会立即从磁盘上移除，它只是被标记成已删除。Elasticsearch 将会在之后添加更多索引的时候才会在后台进行删除内容的清理。

```
curl -H "Content-Type:application/json" -XDELETE
'http://localhost:9200/website/blog/124/?pretty'
```

响应结果：

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "124",
  "_version": 2, //删除、自增
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 4,
  "_primary_term": 1
}
```

版本控制

幻读：两个操作同时对一个文档修改，导致只有最近一次的修改有效。

悲观并发控制：访问之前先对文档加锁，在访问过程中只允许自己修改数据。

乐观并发控制：所有对文档修改前要检查自己读取的文档的"_version"与ES中对应文档的"_version"是否一致，只有一致时才能修改；不一致时重新读取再修改。

11. 创建新的“124”文档

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/website/blog/124/_create/?pretty' -d '{
  "title": "My third blog entry",
  "text": "Yet trying this out...",
  "date": "2019/12/24"
}'
```

响应结果：

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "124",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 5, //用于乐观并发控制
  "_primary_term": 1 //用于乐观并发控制
}
```

重新索引文档保存修改时，可以指定"_version"参数，但仅适用老版本
新版本ES(实践7.5.1)需指明"if_seq_no"和"if_primary_term"的值：

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/website/blog/124?if_seq_no=5&if_primary_term=1' -d '{
  "title": "My four blog entry",
  "text": "Yet trying this out...",
  "date": "2019/12/24"
}'
```

响应结果：

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "124",
  "_version": 2, //自增了
  "result": "updated", //表示更新
  "_shards": {
    "total": 2,
```



```

        "successful": 2,
        "failed": 0
    },
    "_seq_no": 6, //自增了
    "_primary_term": 1
}

```

12. 外部版本号

其他数据库作为主数据库，使用ES搜索数据，则主数据库发生变化，就将其拷贝到ES中，主数据库中若有版本控制字段如时间戳，可在ES查询字符串后面添加"version_type=external"使用外部数据库的版本控制字段。在更新时，只要当前ES中文档的版本号小于指定的外部版本号，即可请求更新成功，外部版本号会存储到"_version"中。

创建包含外部版本号为5的新博客：

```

curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/website/blog/2?version=5&version_type=external' -d '
{
    "title": "My first external blog entry",
    "text": "Starting to get the hang of this..."
}'

```

响应结果：

```

{
    "_index": "website",
    "_type": "blog",
    "_id": "2",
    "_version": 5, //版本号
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 2,
        "failed": 0
    },
    "_seq_no": 7,
    "_primary_term": 1
}

```

使用外部版本号(version=10)更新博客：

```

curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/website/blog/2?version=10&version_type=external' -d '
{
    "title": "My second external blog entry",
    "text": "Starting to get the hang of this..."
}'

```

响应结果：

```

{
    "_index": "website",
    "_type": "blog",
    "_id": "2",
    "_version": 10, //外部版本号
}

```

```
"result": "updated", //更新操作
"_shards": {
  "total": 2,
  "successful": 2,
  "failed": 0
},
"_seq_no": 8, //又自增啦
"_primary_term": 1
}
```

局部更新

13. 文档局部更新

执行流程：

1. 从旧文档检索JSON
2. 修改JSON
3. 删除旧文档
4. 索引文档

为博客2添加“tags”字段和“view”字段(*doc*：局部文档参数，会合并到现有文档中，存在的标量字段被覆盖，新字段被添加)：

```
curl -H "Content-Type:application/json" -XPOST
'http://localhost:9200/website/blog/2/_update?pretty' -d '{
  "doc": {
    "tags": [
      "testing"
    ],
    "views": 0
  }
}'
```

响应结果：

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "2",
  "_version": 11, //自增啦
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 9, //自增啦
  "_primary_term": 1
}
```

检索博客2内容：

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/website/blog/2?pretty'
```

检索结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "2",
  "_version": 11,
  "_seq_no": 9,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "title": "My second external blog entry",
    "text": "Starting to get the hang of this...",
    "views": 0, //新添字段
    "tags": [ //新添字段
      "testing"
    ]
  }
}
```

使用**内置Groovy脚本**局部更新博客2的“views”字段值:

```
curl -H "Content-Type:application/json" -XPOST
'http://localhost:9200/website/blog/2/_update/?pretty' -d '{
  "script": "ctx._source.views+=1"
}'
```

检索博客2结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "2",
  "_version": 12,
  "_seq_no": 10,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "title": "My second external blog entry",
    "text": "Starting to get the hang of this...",
    "views": 1, //+1啦
    "tags": [
      "testing"
    ]
  }
}
```

外部脚本(定义变量并传参)更新博客2的“views”字段:

- 创建脚本(ID: “blog-views”)

```
curl -H "Content-Type:application/json" -XPOST
'http://localhost:9200/_scripts/blog-views/?pretty' -d ' {
  "script": {
    "lang": "painless",
    "source": "ctx._source.views+=params.new_views"
  }
}'
```

- 使用“ID=blog-views”的脚本，并传入参数“new_views=2”

```
curl -H "Content-Type:application/json" -XPOST
'http://localhost:9200/website/blog/2/_update/?pretty' -d ' {
  "script": {
    "id": "blog-views",
    "params": {
      "new_views": 2
    }
  }
}'
```

响应结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "2",
  "_version": 13, //自增啦
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 11, //自增啦
  "_primary_term": 1
}
```

检索博客2:

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/website/blog/2?pretty'
```

检索结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "2",
  "_version": 13,
  "_seq_no": 11,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "title": "My second external blog entry",
  }
}
```

```
    "text": "Starting to get the hang of this...",
    "views": 3, //"blog-views"外部脚本执行成功
    "tags": [
        "testing"
    ]
  }
}
```

14. 更新可能不存在的文档

"upsert"参数定义文档来使其不存在时被创建(文档不存在时使用"upsert"内容初始化)

```
curl -H "Content-Type:application/json" -XPOST
'http://localhost:9200/website/blog/10/_update/?pretty' -d ' {
  "script": "ctx._source.views+=1",
  "upsert": {
    "views": 1
  }
}'
```

响应结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "10",
  "_version": 1,
  "result": "created", //自动被创建
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 12,
  "_primary_term": 2
}
```

检索博客10:

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/website/blog/10/?pretty'
```

博客10检索结果:

```
{
  "_index": "website",
  "_type": "blog",
  "_id": "10",
  "_version": 1,
  "_seq_no": 12,
  "_primary_term": 2,
  "found": true,
  "_source": {
    "views": 1
  }
}
```

局部更新冲突处理(请求方法: update): 对于与顺序无关的操作, 冲突发生时, 重新尝试更新即可。参数 `retry_on_conflict` 设置报错前的重试次数。

15. 一次性检索多个文档(请求方法: `_mget`)

其参数是一个"docs"数组, 数组每个节点定义一个文档(`_index/_type/_id`元数据, 亦可指定 `_source`中字段)

一个检索两个文档(/website/blog/2和/website/blog/10/views):

```
curl -H "Content-Type:application/json" -XPOST 'http://localhost:9200/_mget?pretty' -d '{
  "docs": [
    {
      "_index": "website",
      "_type": "blog",
      "_id": 2
    },
    {
      "_index": "website",
      "_type": "blog",
      "_id": 10,
      "_source": "views"
    }
  ]
}'
```

检索结果(并且每个文档的检索和报告都是独立的, 一个文档的不存在并不影响其他文档的检索):

```
{
  "docs": [
    {
      "_index": "website",
      "_type": "blog",
      "_id": "2",
      "_version": 13,
      "_seq_no": 11,
      "_primary_term": 1,
      "found": true, //成功检索
      "_source": {
        "title": "My second external blog entry",
        "text": "Starting to get the hang of this...",
        "views": 3,
        "tags": [
          "testing"
        ]
      }
    }
  ]
}
```

```

    ]
  },
  {
    "_index": "website",
    "_type": "blog",
    "_id": "10",
    "_version": 2,
    "_seq_no": 13,
    "_primary_term": 2,
    "found": true, //成功检索
    "_source": {
      "views": 2
    }
  }
]
}

```

若检索的文档在同一“_index”中，可以在URL中定义默认的“/_index或/_index/_type”

```

curl -H "Content-Type:application/json" -XPOST 'http://localhost:9200/website/blog/_mget?pretty' -d ' {
  "docs": [
    {
      "_id": 2
    },
    {
      "_id": 10,
      "_source": "views"
    }
  ]
}'

```

若文档有相同的“/_index/_type”，则可用“ids”直接指定文档ID，如检索“ID=2、ID=10”的文档：

```

curl -H "Content-Type:application/json" -XPOST 'http://localhost:9200/website/blog/_mget?pretty' -d ' {
  "ids": [
    "2",
    "10"
  ]
}'

```

即使“mget”的所有文档都找不到，由于“mget”请求成功，所以HTTP请求状态码为“200”，“found”标志可表示每个文档是否检索成功

批量更新(bulk)

bulk请求不是原子操作，它们不能实现事务

两个重点：

1. 每行必以“\n”符号结尾，包括最后一行
2. 每行数据不能包含未被转义的换行符(Json不能用美观打印，即pretty)

请求体格式：

```
{ action:{ metadata }}\n
{ request body }\n
{ action:{ metadata }}\n
{ request body }\n
...
```

action/metadata: action指定文档行为, metadata指定具体哪个文档(/_index/_type/_id)
其中action可取:

1. create: 当文档不存在时创建, 必须有{ request body }
2. index: 创建新文档或者替换已有文档, 必须有{ request body }
3. update: 局部更新文档, 必须有{ request body }
4. delete: 删除一个文档

index与create的区别:

- create: 当文档已经存在时不会创建新文档, 插入失败。
- index: 当文档存在时, 仍然可以执行替换操作; 插入时如果没有指定"_version", 那对于已有的"doc", "_version"会递增, 并对文档覆盖。插入时如果指定"_version", 但与已有的文档"_version"不相等, 则插入失败; 如果与已有文档的"_version"相同则覆盖, "_version"递增。

bulk请求表单(每行必须回车):

```
curl -H "Content-Type:application/json" -XPOST 'http://localhost:9200/_bulk' -d
'
{"delete":{"_index":"website","_type":"blog","_id":"234"}}
{"create":{"_index":"website","_type":"blog","_id":"234"}}
{"title":"My second blog post"}
{"index":{"_index":"website","_type":"blog"}}
{"title":"My third blog post"}
{"update":{"_index":"website","_type":"blog","_id":"234"}}
{"doc":{"title":"My updated blog post"}}
'
```

响应结果:

```
{
  "took": 147,
  "errors": false,
  "items": [
    {
      "delete": {
        "_index": "website",
        "_type": "blog",
        "_id": "234",
        "_version": 2,
        "result": "deleted",
        "_shards": {
          "total": 2,
          "successful": 2,
          "failed": 0
        },
        "_seq_no": 15,
        "_primary_term": 2,
        "status": 200
      }
    }
  ]
}
```



```

    },
    {
      "create": {
        "_index": "website",
        "_type": "blog",
        "_id": "234",
        "_version": 3,
        "result": "created",
        "_shards": {
          "total": 2,
          "successful": 2,
          "failed": 0
        },
        "_seq_no": 16,
        "_primary_term": 2,
        "status": 201
      }
    },
    {
      "index": {
        "_index": "website",
        "_type": "blog",
        "_id": "C9LF028BohgRI0CU-DTy",
        "_version": 1,
        "result": "created",
        "_shards": {
          "total": 2,
          "successful": 2,
          "failed": 0
        },
        "_seq_no": 17,
        "_primary_term": 2,
        "status": 201
      }
    },
    {
      "update": {
        "_index": "website",
        "_type": "blog",
        "_id": "234",
        "_version": 4,
        "result": "updated",
        "_shards": {
          "total": 2,
          "successful": 2,
          "failed": 0
        },
        "_seq_no": 18,
        "_primary_term": 2,
        "status": 200
      }
    }
  ]
}

```

分布式CRUD

- 路由文档到分片

创建文档时，确定其分片；根据文档"_id"也可自定义，生成hash值，然后除以**主切片**的数量得到一个余数(remainder)，余数即其所在分片。故**主分片的数量只能在创建索引时定义且不能修改，否则其他路由值失效。**

- 新建、索引和删除文档

新建、索引和删除请求都是写(write)操作，它们必须在主分片上成功完成后才能复制到相关的复制分片上，同一集群的每个节点均知道文档存在哪个节点上，并均可转发请求。

默认情况下，写(create、index、delete)请求在主分片上执行成功后，会将同样的操作同步到其他复制分片上，待所有复制分片报告成功后，主分片所在的节点报告成功到请求节点(接收写操作的节点)，请求节点再报告给客户端。但可以通过设置"**replication**"参数的值改变过程：

其他参数：

- consistency：默认主分片在尝试写入时需要规定数量(quorum)或过半的分片(可以是主分片或复制分片)可用，防止数据被错误写入其他网络分区。
- timeout：分片副本不足时，ES会等待更多的分片出现。默认等待一分钟。

搜索

ES真正的强大之处

ES不仅存储JSON文档，也会索引文档字段使得文档的每个字段都能被查询。

16. 空搜索(结果说明)

"hits"：包含"total"字段表示匹配到的文档总数，hits数组还包含匹配到的**前10条数据**(依据相关性分数"_score"降序排列)

"_score"：与搜索关键字的相关性分数

"took"：搜索请求的毫秒数

"_shards"：参与查询的分片

"timeout"：查询是否超时，若超时，ES返回请求超时前收集到的结果，timeout不会停止执行查询，它 仅仅告诉你目前顺利返回结果的节点然后关闭连接。但在后台，其他分片可能依旧执行查询，尽管结果已经被发送，所以**timeout实际不能中断长时间的查询**

17. 分页

空搜索默认返回前10条匹配的数据(默认返回第一页，每页默认10条数据)，想看到其他数据，可以指定"size"和"from"：

- size：每页结果数，默认为10
- from：跳过开始的结果数，默认0

例：从结果第5条开始，返回一页(包含5条结果数据)

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?size=5&from=5'
```

集群中深度分页：不可取，因为分布式系统中，一个搜索请求常常涉及多个主分片，每个分片生成自己排好序的结果，接着需要集中起来排序以确保整体排序正确。

映射和分析

映射： 确认每个字段的数据类型，如text、number等(5.x以上没有string类型，**需要分词用text类型，不需要分词用keyword类型**)

分析： 全文文本的分词(包括标记化和标准化)、为标准化后的词典中每个词建立反向索引

ES搜索引擎与其他数据库根本差异在于确切值(如keyword类型)及全文文本之间：

- 确切值：要么匹配要么不匹配，指明字段的检索发生的是确切值的搜索
- 全文文本：不指明字段，默认查询"_all"字段，返回的是与查询字符串相关性得分高(降序)的文档

索引文档时，全文字段被分析(利用**分析器**，包含字符过滤器、分词器、标记过滤器)为单独的词来创建**倒排索引**：

- 当你查询全文字段(如"_all")，查询将使用相同的分析器来分析查询字符串，以产生正确的词列表
- 当你查询一个确切值(如指明查询字段)字段，查询将不分析查询字符串，但是可以自己制定

映射

索引中每个文档都有一个类型(type)，每个类型拥有自己的映射(mapping)或者模式定义(schema definition)。**一个映射定义了字段、字段对应的数据类型及字段被ES处理的方式。**

当索引一个含新字段的文档时，ES会动态映射猜测该新字段的数据类型。猜测依据规则为JSON的基本数据类型。

索引新文档(/gb/tweet/1):

```
curl -H "Content-Type:application/json" -XPUT 'http://localhost:9200/gb/tweet/1' -d '{
  "date": "2019-12-26",
  "name": "damon",
  "user_id": 1001
}'
```

查看映射(ES7中剔除了映射类型):

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/gb/_mapping/?pretty'
```

映射结果:

```
{
  "gb": {
    "mappings": {
      "properties": {
        "date": {
          "type": "date"
        },
        "name": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        }
      }
    }
  }
}
```

```

        },
        "user_id": {
            "type": "long"
        }
    }
}
}
}

```

自定义字段映射(尤对字符串字段类型)

- 优点一：区分全文字符串字段(需要分词)和确切值(不需要分词)，如"中国"可以通过设置"**index**"属性值(指定被索引的方式)确保其不被分词
- 优点二：指定自定义日期格式，如英文日期与中文日期格式不同

映射字段参数：

- type：指定字段数据类型，如text、number、object(内部对象类型)等
- index：指定字段被索引的方式。**analyzed**：被分析(会被分词)再被索引；**not_analyzed**：只被索引，不被分析，和确切值一样。**no**：不会被索引，就不会被搜索到
- analyzer：指定使用的分析器

注：空字段""、null、[]、[null]不会被索引

复合对象映射及索引

文档内容：

```
```json
```

```

{
 "tweet": "Elasticsearch is very flexible",
 "user": {
 "id": "@johnsmith",
 "gender": "male",
 "age": 26,
 "name": {
 "full": "John Smith",
 "first": "John",
 "last": "Smith"
 }
 }
}
...

```

其对应的扁平表单(Lucene文件包含一个键-值对应的扁平表单)：

```
{
 "tweet": [elasticsearch, flexible, very],
 "user.id": [@johnsmith],
 "user.gender": [male],
 "user.age": [26],
 "user.name.full": [john, smith],
 "user.name.first": [john],
 "user.name.last": [smith]
}
```

## 结构化查询(DSL)

**优点：** 查询更灵活、精准、易于阅读并且debug

空查询：

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/_search?pretty' -d ' {}'
```

空查询对应的结构化查询：

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "match_all": {}
 }
}'
```

结构化查询需要"query关键字"，查询子句需要"match关键字"

查找"title"字段包含"My"的文档：

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "match": {
 "title": "My"
 }
 }
}'
```

合并多子句(如"bool"复合子句合并其他子句)：

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "bool": {
 "must": { #必须
 "match": {
 "title": "My"
 }
 },
 "must_not": { #必须不
 "match": {
 "title": "second"
 }
 }
 }
 }
}'
```

```
 }
 },
 "should": { # "title"字段含有"updated", 文档相关性分数更高, 排名更靠前
 "match": {
 "title": "updated"
 }
 }
}
}
```

## 结构化查询与结构化过滤

- 结构化查询语句  
一条查询语句会计算每个文档与查询语句的相关性, 会给出一个相关性评分"\_score", 并且按照相关性对匹配到的文档进行排序。结果不可缓存, 但倒排索引能提高检索性能。
- 结构化过滤语句  
一条过滤语句会询问每个文档的字段是否包含特定值。**过滤文档结果可缓存, 无需计算相关性分数**

**使用场景:** 使用查询语句做全文本搜索或在其他需要进行相关性评分的时候, 剩下的全部使用过滤语句。

## 过滤关键字

### 最外层关键字"query"

- term过滤  
主要用于精准匹配哪些值, 不会匹配短语的其他变形
- terms过滤  
允许指定多个匹配条件(数组), 需要一起匹配
- range过滤  
允许按照指定范围查找一批数据, 操作符: "gt"->大于; "gte"->大于等于; "lt"->小于; "lte"->小于等于
- exists过滤和missing过滤  
用于查找文档中是否包含指定字段或没有某个字段
- bool过滤  
合并多个过滤条件查询结果的布尔逻辑

## 查询关键字

### 最外层关键字"query"

- match\_all查询  
可以查询到所有文档, 没有查询条件下的默认语句
- match查询  
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"下查询字符; 在"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, 它将为你搜索确切值
- multi\_match查询  
允许做"match"查询的基础上同时搜索多个字段
- bool查询  
多用于合并多个查询子句。**与bool过滤相似, 区别在于, bool查询有"match"关键字表示查询以及需要计算相关性分数。**可以组合"must"、"must\_not"、"should"、"filter"子句

# 查询过滤组合

*更详细的过滤应该放在其他过滤器之前*

## 18. 单一查询语句

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "match": {
 "title": "My"
 }
 }
}'
```

单一过滤语句:

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "term": {
 "title": "updated"
 }
 }
}'
```

等价于:

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "bool": {
 "must": {
 "match_all": {}
 },
 "filter": {
 "term": {
 "title": "updated"
 }
 }
 }
 }
}'
```

亦等价于:

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "bool": {
 "filter": {
 "term": {
 "title": "updated"
 }
 }
 }
 }
}'
```

合并([先查询再对查询结果过滤](#) /bool/ must/ filter/):

```
curl -H "Content-Type:application/json" -XGET 'http:
//localhost:9200/_search?pretty' -d ' { "query": {
 "bool": { #bool复合查询
 "must": { #查询子句
 "match": {
 "title": "My"
 }
 },
 "filter": { #过滤子句
 "term": {
 "title": "updated"
 }
 }
 }
}'
```

## 复合查询

**最外层关键字均是"query"**

- bool查询  
组合多子句的默认查询(如: "must"、"must\_not"、"filter"、"should"), 返回**与所有子查询都匹配的文档**
- boosting查询  
返回与"positive"查询匹配的文档, 并减少与"negative"查询匹配的文档得分, 包含: "positive"查询、"negative"查询、"negative\_boost"关键字("negative"查询相关性得分的降低比率)
- constant\_score查询  
包装其他查询的查询, 在"filter"上下文中执行, 所有过滤匹配的文档被给予不变相关性分数(通过参数"boost"指定)
- dis\_max查询  
包含多组查询子句的查询, 返回与任一子查询匹配的文档, 相关性分数为子查询中最好得分分数
- function\_score查询  
使用函数修改查询文档的相关性分数



## 验证查询

**"\_validate"**

19. 验证查询语句的有效性(**explain**: 输出语句非法的具体信息)

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/website/blog/_validate/query?pretty&explain' -d ' {
 "query": {
 "term": {
 "title": "updated"
 }
 }
}'
```

## 排序

"\_score": 只与查询语句有关, 过滤语句不影响"\_score"值

20. 检索结果按"\_score"升序输出

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "bool": {
 "must": {
 "match": {
 "title": "My"
 }
 }
 }
 },
 "sort": { #排序
 "_score": {
 "order": "asc"
 }
 }
}'
```

等价的查询字符串自定义排序:

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?sort=_score:asc&pretty' -d ' {
 "query": {
 "bool": {
 "must": {
 "match": {
 "title": "My"
 }
 }
 }
 }
}'
```

## 相关性

---

查询语句会为文档添加"\_score"字段，默认情况下，返回结果按照相关性分数降序排列。

**"fuzzy"查询：** 计算与关键词的拼写相似程度

**"terms"查询：** 计算找到的内容与关键字组成部分匹配的百分比

**全文本搜索：** 一般只计算内容与关键词的类似程度

ES计算相似度算法：**TF/IDF(检索词频率/反向文档频率)、地理位置相近程度、模糊相似度等**

- **TF(检索词频率)**  
检索词在单个文档出现的频率。TF越高，相关性越高
- **IDF(反向文档频率)**  
出现检索词的文档的频率。IDF越高，相关性越低
- **字段长度准则**  
检索词出现在一个短的title要比出现在一个长的title对相关性分值的贡献高

TF和IDF是在每个分片中计算出来的，而不是每个索引中(考虑性能原因)。

**使用分片本地IDF可能出现的问题：** 一个分片可能包含多份文档，若目标词在不同分片的分布不均匀，会导致在目标词出现频率大的分片上重要性小，而在出现频率小的分片上重要性大，影响IDF，进而可能导致相关性较低的结果排在相关性较高的结果前面(称为**关联失效**)

**关联失效：** 是个无需考虑的问题，因为数据太少，只要索引更多的文档，本地IDF和全局IDF的区别就会越小，索引在实际工作的数据量下，本地IDF也能很好工作。

## 数据字段

---

为了提高排序效率，ES会将所有字段的值(索引下所有文档中的值，并不仅是匹配到的那部分数据)加载到内存中，这就叫做“数据字段”

**问题：** 消耗很多内存

**解决：** 横向扩展，添加更多节点到集群

## 分布式搜索

---

### 与CRUD区别

---

一个CRUD操作指定了唯一单独的文档(由/\_index/\_type/\_id确定，经过hash即可知道文档所在分片编号)。而分布式搜索通过关键字检索匹配的文档，未指定具体文档，所以必须查询索引下所有的主分片及副本的所有文档看是否匹配。

### 两个阶段

---

- **查询阶段**  
搜索被请求节点(协调节点)向索引中的每个分片及副本广播，每个分片在本地执行搜索并建立匹配文档的优先队列(数目由"from"及"size"决定，并按照"\_score"降序排列)，每个分片返回搜索到的文档"ID"及对应的"\_score"(规模：from+size)到请求节点(协调节点)，由请求节点归并到自身的优先队列中产生全局排序结果。
- **取回阶段**  
查询阶段结束后，请求节点知道了搜索关键字所在的文档"ID"，会hash计算所在分片，并向目标分片发送文档"GET"请求，每个分片加载文档并丰富文档(如：高亮关键字等)，再将文档返回给请求节点，请求节点响应给客户端。

## 搜索选项

- preference(搜索偏好)  
允许你控制使用哪个分片或节点来处理搜索请求，取值  
如: "\_primary"、"\_primary\_first"、"\_local"、"\_only\_node:xyz"、"\_prefer\_node:xyz"、"\_shards:2,3"  
**结果震荡：** 由于搜索请求是在所有有效的分片副本间轮询的，两个拥有相同值排序字段的文档在原始分片和复制分片中的顺序不一致，导致每次刷新页面，文档顺序不一致。  
**解决：** 通过随机字符串(如用户ID)作为preference参数值，确保同一用户总是使用同一分片。
- timeout  
协调节点等待其他节点回答的时间，到达时间时协调节点将已收到的回答返回客户端。
- routing(路由选择)  
指定routing值限制只搜索部分分片，在大规模搜索系统中很有用。
- search\_type(查询类型)  
"query\_then\_fetch": 查询然后取回，默认的搜索类型；"dfs\_query\_then\_fetch": dfs搜索类型有一个预查询的阶段，它会从全部相关的分片里取回频数来计算全局的项目频数(与相关性被破坏有关)

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?pretty&search_type=count' -d '{
 "query": {
 "bool": {
 "must": {
 "match_all": {}
 },
 "filter": {
 "term": {
 "title": "updated"
 }
 }
 }
 }
}
```

## 索引管理

创建索引，并通过"settings"、"mappings"参数设置配置信息(如主分片数目、复制分片数目)和映射类型。可以通过修改config/elasticsearch.yml中添加下面的配置来防止自动创建索引：

```
action.auto_create_index: false
```

## 删除索引

- 删除单个索引

```
curl -H "Content-Type:application/json" -XDELETE
'http://localhost:9200/my_temp_index?pretty'
```

- 删除多个索引

```
curl -H "Content-Type:application/json" -XDELETE
'http://localhost:9200/index_one,index_two?pretty'
```

```
curl -H "Content-Type:application/json" -XDELETE
'http://localhost:9200/index_?*?pretty'
```

- 删除所有索引

```
curl -H "Content-Type:application/json" -XDELETE
'http://localhost:9200/_all?pretty'
```

## 索引设置

**"number\_of\_shards"**: 索引主分片数目，默认为1(6.x默认为5)，索引创建后不可更改，每个索引的主分片数目限制为1024。

**"number\_of\_replicas"**: 每个主分片的复制分片数，默认为1，可以动态修改，用于横向扩展。

21. 创建索引，设置主分片为1，复制分片为0

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/my_temp_index?pretty' -d ' {
 "settings": {
 "number_of_shards": 1,
 "number_of_replicas": 0
 }
}'
```

22. 动态更新复制分片数目

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/my_temp_index/_settings?pretty' -d ' {
 "number_of_replicas": 1
}'
```

23. 创建分析器es\_std(在索引"spanish\_docs"下)

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/spanish_docs?pretty' -d ' {
 "settings": {
 "analysis": {
 "analyzer": {
 "es_std": {
 "type": "standard",
 "stopwords": "_spanish_" #设置使用的西班牙语的停用词过滤器(删除所有可能搜索歧义的停用词如a、the等)
 }
 }
 }
 }
}'
```

# 自定义分析器

**分析器组成：** 字符过滤器、分词器、标记过滤器顺序组成

- 字符过滤器  
0或者多个，在字符串分词前更整洁，剔除所有html标签(等)
- 分词器  
1个，将字符串分割成单独的词或者标记，也可剔除标点符号，不同分词器不同行为特征
- 标记过滤器  
修改、添加或删除标记，可以进行单词标准化；例如：**"lowercase": 将大写全部变为小写**，**"stopwords": 删除停止词(a、the等)**，**"ascii\_folding": 删除变音符号**

24. 测试分析器(空格分析器)

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_analyze?pretty' -d '{
 "analyzer": "whitespace", #空格分析器
 "text": "The quick brown fox." #被分析的文本
}'
```

```
{
 "tokens": [
 {
 "token": "The",
 "start_offset": 0,
 "end_offset": 3,
 "type": "word",
 "position": 0
 },
 {
 "token": "quick",
 "start_offset": 4,
 "end_offset": 9,
 "type": "word",
 "position": 1
 },
 {
 "token": "brown",
 "start_offset": 10,
 "end_offset": 15,
 "type": "word",
 "position": 2
 },
 {
 "token": "fox.",
 "start_offset": 16,
 "end_offset": 20,
 "type": "word",
 "position": 3
 }
]
}
```

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_analyze?pretty' -d ' {
 "tokenizer": "standard", #分词器
 "filter": [#标记过滤器
 "lowercase",
 "asciifolding"
],
 "text": "Is this Dog?" #分析文本
}'
```

## 25. 指定并使用索引中自定义分析器

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/my_index?pretty' -d ' {
 "settings": {
 "analysis": {
 "analyzer": {
 "std_folded": { #自定义分析器
 "type": "custom",
 "tokenizer": "standard", #分词器
 "filter": [#标记过滤器(小写、去掉音标)
 "lowercase",
 "asciifolding"
]
 }
 }
 }
 },
 "mappings": {
 "properties": {
 "my_text": { #my_text映射设置
 "type": "text", #文本类型
 "analyzer": "std_folded" #使用自定义的分析器
 }
 }
 }
}'
```

### 指定文本分析使用的分析器

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/my_index/_analyze?pretty' -d ' {
 "analyzer": "std_folded", #直接指定自定义分析器
 "text": "Is this dog?" #被分析文本
}'
```

等价于：

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/my_index/_analyze?pretty' -d ' {
 "field": "my_text", #使用字段属性my_text的分析器
 "text": "Is this dog?"
}'
```

## 类型和映射

"Lucene": 用于全文检索和搜寻的开源程序库, 提供简单而强大的应用程序接口, 不是现成的搜索引擎产品, 可以用来制作搜索引擎产品。

全文检索: 计算机索引程序通过扫描文档中每个词, 对每个词建立一个索引, 指明该词在文章中出现的次数及位置, 当用户查询时, 检索程序就根据事先建立的索引进行文档的查询, 并将查找结果反馈给用户。总结: **Lucene全文检索就是对文档中全部内容进行分词, 并为每个词建立倒排索引的过程**

一个索引包含一种类型, **类型是具有相同字段属性文档的集合**, 每个文档的类型名被储存在一个叫"**\_type**"的元数据字段上。当我们搜索一种特殊类型的文档时, Elasticsearch 简单的通过"**\_type**"字段来过滤出这些文档。**映射是Elasticsearch将复杂JSON文档映射成Lucene需要的扁平化数据的方式**。"\_source"字段, 另有"\_all"、"\_id"等; ES用JSON字符串表示**文档主体保存在"\_source"字段中**, 写入磁盘前需要**压缩**, 可以通过映射设置禁用"\_source"字段。

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/_search?pretty' -d '{
 "query": {
 "match_all": {}
 },
 "_source": [
 "title",
 "created"
]
}'
```

## 动态映射

对于ES的新字段, "dynamic"设置是否自动添加到映射中, 取值如下:

"true": 自动添加新字段(默认);

"false": 忽略字段

"strict": 遇到未知字段, 抛出异常

## 动态字段匹配

- 日期检测

"date\_detection": 参数决定能否进行日期检测, "dynamic\_date\_formats": 规定日期匹配格式, 默认格式: "yyyy/MM/dd HH:mm:ss Z|yyyy/MM/dd Z"

26. 禁用日期检测

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index_six?pretty' -d '{
 "mappings": {
 "date_detection": false
 }
}'
```

索引文档:

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index_six/_doc/1?pretty' -d '{
 "create": "2019/12/31"
}'
```

查看映射：

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/index_six/_mapping?pretty'
```

映射结果：

```
{
 "index_six": {
 "mappings": {
 "date_detection": false, //禁用日期检测
 "properties": {
 "create": {
 "type": "text", //日期被当做text类型
 "fields": {
 "keyword": {
 "type": "keyword",
 "ignore_above": 256
 }
 }
 }
 }
 }
 }
}
```

## 27. 自定义日期映射格式

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index_seven?pretty' -d '{
 "mappings": {
 "dynamic_date_formats": [
 "MM/dd/yyyy"
]
 }
}'
```

索引文档：

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index_seven/_doc/1?pretty' -d '{
 "create": "12/31/2019"
}'
```

## 动态模版

*为动态添加的字段自定义匹配规则*

动态模版被指定为命名对象数组：



```

"dynamic_templates": [
 {
 "my_template_name": { //模版名字
 ... match conditions ... //匹配条件(即匹配文档字段)
 "mapping": { ...
 } //将匹配到字段映射为
 },
 ...
]

```

#### 匹配条件分

为: "match\_mapping\_type"、"match"、"match\_pattern"、"unmatch"、"path\_match"、"path\_unmatch"。

- match  
使用一种模式匹配字段名，匹配到的使用映射
- unmatch  
排除匹配到的字段名，不进行映射
- match\_pattern  
可以调整该参数使参数"match"使用正则表达式匹配字段名
- path\_match  
匹配的是字段路径，满足匹配的使用映射
- path\_unmatch  
排除匹配到的路径的字段

以match\_mapping\_type为例(使用json解析的格式，json中会以long替代integer，double替代float，使用更宽的数据类型)：

```

curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index_eight?pretty' -d ' {
 "mappings": {
 "dynamic_templates": [
 {
 "integers": { #模版名字(作用：将json标准检测的文档的long型新字段映射为
integer型)
 "match_mapping_type": "long", #匹配条件
 "mapping": {
 "type": "integer" #映射类型
 }
 },
 {
 "strings": { #模版名字(作用：将json标准检测的文档的string型新字段映射为可
分词的text型、确切值的keyword)
 "match_mapping_type": "string",
 "mapping": {
 "type": "text",
 "fields": {
 "raw": {
 "type": "keyword",
 "ignore_above": 256
 }
 }
 }
 }
 }
]
 }
}

```

```
 }
]
}
}
```

索引文档:

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index_eight/_doc/1?pretty' -d ' {
 "my_integer": 5,
 "my_string": "Some string"
}'
```

查看映射结果:

```
{
 "index_eight": {
 "mappings": {
 "dynamic_templates": [
 {
 "integers": {
 "match_mapping_type": "long",
 "mapping": {
 "type": "integer"
 }
 }
 },
 {
 "strings": {
 "match_mapping_type": "string",
 "mapping": {
 "fields": {
 "raw": {
 "ignore_above": 256,
 "type": "keyword"
 }
 },
 "type": "text"
 }
 }
 }
],
 "properties": {
 "my_integer": {
 "type": "integer"
 },
 "my_string": {
 "type": "text",
 "fields": {
 "raw": {
 "type": "keyword",
 "ignore_above": 256
 }
 }
 }
 }
 }
 }
}
```

```
}
}
}
```

## 类型映射变动

**7.0版本:** `"include_type_name=false`(默认), 即不包含类型名, 使用默认的`"_doc"`。若设为`true`, 需指定类型名

28. 创建索引"index",写入属性字段"foo"

```
curl -H "Content-Type:application/json" -XPUT 'http://localhost:9200/index?include_type_name=false&pretty' -d '{
 "mappings": {
 "properties": { #映射直接在mapping下, 没有"_doc"
 "foo": {
 "type": "keyword"
 }
 }
 }
}'
```

更新索引"index"的映射, 添加新映射字段"bar":

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index/_mappings?include_type_name=false&pretty' -d '{
 "properties": {
 "bar": {
 "type": "text"
 }
 }
}'
```

获得索引"index"的映射信息:

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/index/_mappings?include_type_name=false&pretty'
```

向"index"中索引文档(类型"\_doc"):

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/index/_doc/1?pretty' -d '{
 "foo": "baz"
}'
```

检索文档1(指明{index}/\_doc/{id}):

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/index/_doc/1?pretty'
```

**ES的refresh:** 默认情况下，ES会每秒refresh一次，每次操作都会把内存缓冲区的内容拷贝到新创建的segment中去，这一步是在内存中操作的，这个时候新的文档就会被搜索了。也就是说**ES是近实时性的搜索，差不多1s钟，才能让数据可以被搜索到。**

**ES的flush:** Flush操作意味着，所有在内存缓冲区的文档被写到新的Lucene Segment中，也就是所有在内存中的segment被提交到了磁盘，同时清除translog。

**总之，ES的refresh操作是为了让最新的数据可以立即被搜索到。而flush操作则是为了让数据持久化到磁盘中，另外ES的搜索是在内存中处理的，因此Flush操作不影响数据能否被搜索到。**

## 深入分片

**倒排索引：**支持一个字段(词)多个值(词所在的文档及其位置)的最佳数据结构。倒排索引可能存储包含每个term 的文档数量，一个term出现在指定文档中的频次，每个文档中term的顺序，每个文档的长度，所有文档的平均长度。**不可变性：**写入磁盘的倒排索引不可变(也就不需要锁)

## 动态索引

*不重写整个倒排索引，增加额外的索引反应最近的变化*

ES底层依赖Lucene，引入段(segment)，一个段即是拥有完整功能的倒排索引。**Lucene中索引(也是指ES中的分片)指的是段的集合+提交点(记录上次同步提交到磁盘的所有段)**。Lucene引入了"Pre-segment search"，其工作流程(必须同步到磁盘即被提交才能检索到新索引的文档-->新文档从索引到被检索到需要几分钟):

1. 新索引文档在写入磁盘的段之前首先被写入内存区的索引缓存
2. 索引缓存中内容被提交到磁盘的时机：
  - 新段(倒排索引)写入磁盘
  - 新提交点写入磁盘
  - 发生文件同步(fsycn)到磁盘
3. 磁盘中新段被打开，包含的文档可以被检索
4. 内存区的索引缓存被清除，等待接收新的文档

通过这种方式(分段)，**新文档以较小代价加入索引(无需重写覆盖整个索引，只需添加新段到磁盘即可)，但这样，新文档从索引到被检索到仍然需要几分钟，因为必须同步到磁盘并且磁盘中段被打开才能被检索到。**

## 删除和更新

*段不可变，文档不能从旧段中移除，旧段也不能更新来反应文档的新版本*

**删除：**每个提交点包括.del文件，包含段上已经被删除的文档，当一个文档被删除，它实际上只是在.del文件中被标记为删除，依然可以匹配查询，但是最终 返回之前会被从结果中删除。

**更新：**当一个文档被更新，旧版本的文档被标记为删除，新版本的文档在新的段中索引，也许该文档的不同版本都会匹配一个查询，但是更老版本会从结果中删除。

## 近实时搜索

"pre-segment search"机制需要新索引的文档同步(fsycn)到磁盘才能被检索到(需要几分钟，同步到磁盘慢)。

**新文档同步到磁盘段的整体流向：**新文档被索引->被写入内存索引缓存->被写入新的段->新的段被写入文件系统缓存(代价低，且此时其内容可以被打开和读取)->文件系统缓存内容被同步(被提交)到磁盘

Refresh: :默认情况下，每个分片每秒自动刷新一次，通过修改配置项"refresh\_interval"可以改变刷新频率。

```
curl -H "Content-Type:application/json" -XPUT 'http://localhost:9200/my_logs/?pretty' -d '{
 "settings": {
 "refresh_interval": -1
 }
}'
```

## 持久化变更

**问题：**在近实时搜索中新文档的每次索引不再同步(fsync)到磁盘(代价太大)，导致发生故障时(如断电)可能会丢失从上次提交(fsycn)到此刻的对文档的操作

**解决：**增加事务日志(translog)，记录自上次提交点到此刻的每次操作，但其也需要定期同步到磁盘。事务日志记录了没有flush到硬盘的所有操作。当故障重启后，ES会用最近一次提交点从硬盘恢复所有已知的段，并且从日志里恢复所有的操作。

**新文档索引流程：**

1. 新文档加入内存索引缓存，同时写入事务日志
2. 分片进行refresh(默认每秒一次)
  - 内存缓冲区文档写入段中，不进行fsync(同步到磁盘)
  - 文件系统缓存中的段被打开，其中的新文档可以被检索
  - 内存缓冲区文档被清除，但事务日志不会
3. 随着新文档写入，事务日志逐渐增大，但不宜过大(过大会导致恢复过程太多工作要做)，事务日志也会不时同步到磁盘：文件系统缓存中段被提交到磁盘后，事务日志被清除。

## 合并段

每次refresh就创建新的段，但每次搜索请求都会一次检查每个段，段越多，查询越慢。ES后台合并段，小段合并成大段，大段合并成更大的段，但旧段不参与合并。

## 全文检索

最重要的两个方面：**相关度、分析**；一旦提到相关度和分析，指的都是查询(queries)而非过滤器(filters)。

**相关度：**根据文档与查询的相关程度对结果集进行排序的能力。相关度可以使用TF/IDF、地理位置相近程度、模糊相似度或其他算法计算。

**分析：**将一段文本转换为一组唯一的、标准化了的标记(token)，用以**(a)**创建倒排索引，**(b)**查询倒排索引。

所有的查询都会进行相关度计算，但不是所有的查询都有分析阶段。**文本查询可分为两类：**

- 基于短语的查询  
低级查询，只会计算相关度，没有分析阶段，只能精确匹配特定短语，如term查询等。
- 全文检索  
像match等的高级查询，即会计算相关度，也会对字段进行分析，会对查询语句分析产生要查询的短语列表，再使用低级查询(term查询并合并查询结果)进行查询。**match步骤：**检查被查询字段类型->分析查询字符串->寻找匹配的文档->为每个文档计算相关度分数

## 多词查询

match查询进行多词查询，提高查询精度。**默认("operator":"or")**返回能匹配至少一词的文档。但可以通过设置"operator"参数为"and"，要求所有查询关键字都匹配(位置不一定相连)的文档才能被返回。

29. 查询about字段为"love qingdao"的文档：

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d '{
 "query": {
 "match": {
 "about": {
 "query": "love qingdao"
 }
 }
 }
}'
```

等同于(会返回含有"love"、"qingdao"至少其中之一的文档)：

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d '{
 "query": {
 "match": {
 "about": "love qingdao"
 }
 }
}'
```

检索结果：

```
{
 "took": 2,
 "timed_out": false,
 "_shards": {
 "total": 1,
 "successful": 1,
 "skipped": 0,
 "failed": 0
 },
 "hits": {
 "total": {
 "value": 2,
 "relation": "eq"
 },
 "max_score": 1.4508328,
 "hits": [
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "2",
 "_score": 1.4508328,
 "_source": {
 "first_name": "Jane",
 "last_name": "Smith",

```

```

 "age": 32,
 "about": "I love qingdao",
 "interests": [
 "music"
]
 },
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "1",
 "_score": 0.47000363,
 "_source": {
 "first_name": "John",
 "last_name": "Smith",
 "age": 25,
 "about": "I love suzhou", #含有"love"字段
 "interests": [
 "sports",
 "music"
]
 }
 }
]
}

```

设置"operator"=and的检索语句(必须同时含有"love"和"qingdao", 但位置可以不定):

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
 "query": {
 "match": {
 "about": {
 "query": "love qingdao",
 "operator": "and"
 }
 }
 }
}'

```

检索结果(没了"love suzhou"):

```

{
 "took": 2,
 "timed_out": false,
 "_shards": {
 "total": 1,
 "successful": 1,
 "skipped": 0,
 "failed": 0
 },
 "hits": {
 "total": {
 "value": 1,
 "relation": "eq"
 }
 }
}

```

```

 },
 "max_score": 1.4508328,
 "hits": [
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "2",
 "_score": 1.4508328,
 "_source": {
 "first_name": "Jane",
 "last_name": "Smith",
 "age": 32,
 "about": "I love qingdao",
 "interests": [
 "music"
]
 }
 }
]
 }
}

```

## 控制查询精度

**"minimum\_should\_match"**: 参数值为整数时表示被视为相关的文档必须匹配的关键词个数，也可以为百分比(更合理)。同样适用于上例(例29)。

30. 查询字段"about"满足75%关键字(即至少满足两个关键字)的文档

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
 "query": {
 "match": {
 "about": {
 "query": "I love qingdao",
 "minimum_should_match": "75%"
 }
 }
 }
}'

```

检索结果:

```

{
 "took": 5,
 "timed_out": false,
 "_shards": {
 "total": 1,
 "successful": 1,
 "skipped": 0,
 "failed": 0
 },
 "hits": {
 "total": {
 "value": 2,
 "relation": "eq"
 }
 }
}

```



```

 },
 "max_score": 1.5843642,
 "hits": [
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "2",
 "_score": 1.5843642,
 "_source": {
 "first_name": "Jane",
 "last_name": "Smith",
 "age": 32,
 "about": "I love qingdao", // "I"、"love"、"qingdao"都匹配
 "interests": [
 "music"
]
 }
 },
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "1",
 "_score": 0.60353506,
 "_source": {
 "first_name": "John",
 "last_name": "Smith",
 "age": 25,
 "about": "I love suzhou", // "I"和"love"匹配
 "interests": [
 "sports",
 "music"
]
 }
 }
]
 }
}

```

## 组合查询

**"bool"查询可以接受"must"、"must\_not"、"should"查询子句**

例如以下查询语句：

```

curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/_search?pretty' -d ' {
 "query": {
 "bool": {
 "must": { #“title”字段必含有"My"
 "match": {
 "title": "My"
 }
 },
 "must_not": { #“title”字段必不含有"second"
 "match": {
 "title": "second"
 }
 }
 }
 }
}
'

```

```
},
"should": { #“title”字段含有“updated”，文档相关性分数越高，排名更靠前
 "match": {
 "title": "updated"
 }
}
}
```

其中should中默认可以不匹配任何关键字，但匹配可以提高相关性分数；并且若没有"must"子句，"should"子句必须至少匹配一个关键字。

**相关性得分：**布尔查询通过把所有符合"must"和"should"的子句得分加起来，然后除以"must"和"should"子句的总数为每个文档计算相关性得分。

## 精度控制

"should"子句通过"minimum\_should\_match"参数控制"should"子句至少应该匹配的关键字数(默认为0), 值可以是百分数或整数。

31. 例如

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d '{
 "query": {
 "bool": {
 "should": [
 {
 "match": {
 "about": "I"
 }
 },
 {
 "match": {
 "about": "love"
 }
 },
 {
 "match": {
 "about": "suzhou"
 }
 }
],
 "minimum_should_match": 2 #至少含"should"中两个关键字的文档
 }
 }
}'
```

检索结果:

```
{
 "took": 7,
 "timed_out": false,
 "_shards": {
 "total": 1,
```

```

 "successful": 1,
 "skipped": 0,
 "failed": 0
 },
 "hits": {
 "total": {
 "value": 2,
 "relation": "eq"
 },
 "max_score": 1.5843642,
 "hits": [
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "1",
 "_score": 1.5843642,
 "_source": {
 "first_name": "John",
 "last_name": "Smith",
 "age": 25,
 "about": "I love suzhou",
 "interests": [
 "sports",
 "music"
]
 }
 },
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "2",
 "_score": 0.60353506,
 "_source": {
 "first_name": "Jane",
 "last_name": "Smith",
 "age": 32,
 "about": "I love qingdao",
 "interests": [
 "music"
]
 }
 }
]
 }
}

```

将"minimum\_should\_match"设为3后，检索结果只有一个文档，不再含"qingdao"。

## match与bool查询

高级查询(如match)都是在分析后对各个关键字进行低级查询(如term精确匹配)的结果合并而成。

32. 一个"match"等同于"should"下的多个"term"("operator"="or")

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
 "query": {
 "match": {
 "about": "love qingdao"
 }
 }
}'
```

等同的bool查询:

```
curl -H "Content-Type:application/json" -XGET 'http:
//localhost:9200/megacorp/employee/_search?pretty' -d ' {
 "query": {
 "bool": {
 "should": [#满足一个精确匹配就能返回
 {
 "term": {
 "about": "love"
 }
 },
 {
 "term": {
 "about": "qingdao"
 }
 }
]
 }
 }
}'
```

33. 一个"match"等同于"must"下的多个"term"("operator"="and")

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
 "query": {
 "match": {
 "about": {
 "query": "love qingdao",
 "operator": "and"
 }
 }
 }
}'
```

等同于:

```
curl -H "Content-Type:application/json" -XGET 'http:
//localhost:9200/megacorp/employee/_search?pretty' -d ' {
 "query": {
 "bool": {
 "must": [#必须都满足
 {
 "term": {
 "about": "love"
 }
 }
]
 }
 }
}'
```

```

 }
 },
 {
 "term": {
 "about": "qingdao"
 }
 }
]
}
}
}'

```

#### 34. 指定"minimum\_should\_match"参数

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d '
{
 "query": {
 "match": {
 "about": {
 "query": "I love qingdao",
 "minimum_should_match": "75%"
 }
 }
 }
}
}'

```

等同于:

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d '
{
 "query": {
 "bool": {
 "should": [#满足一个精确匹配就能返回
 {
 "term": {
 "about": "I"
 }
 },
 {
 "term": {
 "about": "love"
 }
 },
 {
 "term": {
 "about": "qingdao"
 }
 }
],
 "minimum_should_match": "75%"
 }
 }
}
}'

```

## 查询词权重

在任何查询子句中指定一个 "boost" 值来控制相对权重，默认值为1。一个大于1的 "boost" 值可以提高查询子句的相对权重。

35. 提高 "swimming" 权重，其他值 "boost" 默认为1 (匹配字段为 "about")

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/megacorp/employee/_search?pretty' -d ' {
 "query": {
 "bool": {
 "should": [
 {
 "match": {
 "about": {
 "query": "I",
 "boost": 1
 }
 }
 },
 {
 "match": {
 "about": {
 "query": "love",
 "boost": 1
 }
 }
 },
 {
 "match": {
 "about": {
 "query": "swimming",
 "boost": 5 #更高权重
 }
 }
 }
]
 }
 }
}'
```

检索结果:

```
{
 "took": 2,
 "timed_out": false,
 "_shards": {
 "total": 1,
 "successful": 1,
 "skipped": 0,
 "failed": 0
 },
 "hits": {
 "total": {
 "value": 3,
 "relation": "eq"
 },
 },
}
```

```
"max_score": 5.037678,
"hits": [
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "3",
 "_score": 5.037678,
 "_source": {
 "first_name": "Douglas",
 "last_name": "Fir",
 "age": 35,
 "about": "I like swimming",
 "interests": [
 "forestry"
]
 }
 },
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "1",
 "_score": 0.60353506,
 "_source": {
 "first_name": "John",
 "last_name": "Smith",
 "age": 25,
 "about": "I love suzhou",
 "interests": [
 "sports",
 "music"
]
 }
 },
 {
 "_index": "megacorp",
 "_type": "employee",
 "_id": "2",
 "_score": 0.60353506,
 "_source": {
 "first_name": "Jane",
 "last_name": "Smith",
 "age": 32,
 "about": "I love qingdao",
 "interests": [
 "music"
]
 }
 }
]
}
```

# 多字段搜索

## 多重查询字符串

### 36. 索引两个文档

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/books/book/1?pretty' -d '{
 "title": "War and Peace",
 "author": "Leo Tolstoy",
 "translator": "Constance Garnett"
}'
```

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/books/book/1?pretty' -d '{
 "title": "War and Peace",
 "author": "Leo Tolstoy",
 "translator": "Louise Maude"
}'
```

多重查询("bool"查询下嵌套"bool"查询):

```
curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/_search?pretty' -d '{
 {
 "query": {
 "bool": {
 "should": [
 {
 "match": {
 "title": "War and Peace"
 }
 },
 {
 "match": {
 "author": "Leo Tolstoy"
 }
 }
],
 "bool": {
 "should": [
 {
 "match": {
 "translator": "Constance Garnett"
 }
 },
 {
 "match": {
 "translator": "Louise Maude"
 }
 }
]
 }
 }
 }
 }
}
```



```
}
 }
]
}
}
```

使得"title"、"author"及嵌套的"bool"子句权重各占1/3。每个同级的子句权重默认相同，但可以通过"boost"参数修改权重(即子句优先级)。

## 最佳字段

bool查询相关性分数计算(合并来自每个字段的分值):

1. 运行其下的子查询
2. 相加查询返回的分值
3. 将相加得到的分值x匹配的查询子句的数量
4. 除以总的查询子句的数量

而"**dis\_max**"查询会使用**最佳匹配的查询的分值**(其中一个子查询语句的查询相关性分数)作为整个查询的整体分值。

**ite\_breaker**(取值[0,1])参数在"**dis\_max**"下，也会将其他匹配的查询子句考虑进来，相关性分数计算方式:

1. 取得最佳匹配查询子句的"\_score"
  2. 将其他每个匹配的子句的分值乘以"ite\_breaker"
  3. 将以上得到的分数进行累加并规范化
  4. 添加两个博客，允许多字段搜索(multi\_match)及最佳字段查询(dis\_max查询)
- blog\_1:

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/my_blogs/blog/1?pretty' -d '{
 "title": "Quick brown rabbits",
 "body": "Brown rabbits are commonly seen."
}'
```

blog\_2:

```
curl -H "Content-Type:application/json" -XPUT
'http://localhost:9200/my_blogs/blog/2?pretty' -d '{
 "title": "Keeping pets healthy",
 "body": "My quick brown fox eats rabbits on a regular basis."
}'
```

检索结果(blog\_1在前):

```
{
 "_index": "my_blogs",
 "_type": "blog",
 "_id": "1",
 "_score": 0.90425634, //blog_1的相关性分值
```

```

 "_source": {
 "title": "Quick brown rabbits",
 "body": "Brown rabbits are commonly seen."
 }
 },
 {
 "_index": "my_blogs",
 "_type": "blog",
 "_id": "2",
 "_score": 0.77041256, //blog_2相关性分值
 "_source": {
 "title": "Keeping pets healthy",
 "body": "My quick brown fox eats rabbits on a regular basis."
 }
 }
}

```

"dis\_max"查询使用最佳匹配的子查询的相关性分值:

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/my_blogs/_search?pretty' -d '
{
 "query": {
 "dis_max": {
 "queries": [
 {
 "match": {
 "title": "Brown fox"
 }
 },
 {
 "match": {
 "body": "Brown fox"
 }
 }
]
 }
 }
}'

```

检索结果(blog\_2在前):

```

{
 "_index": "my_blogs",
 "_type": "blog",
 "_id": "2",
 "_score": 0.77041256, //blog_2的相关性分值
 "_source": {
 "title": "Keeping pets healthy",
 "body": "My quick brown fox eats rabbits on a regular basis."
 }
},
{
 "_index": "my_blogs",
 "_type": "blog",
 "_id": "1",
 "_score": 0.6931472,

```

```

 "_source": {
 "title": "Quick brown rabbits",
 "body": "Brown rabbits are commonly seen."
 }
 }
}

```

"ite\_breaker"参数考虑其他匹配子查询分值(最佳字段查询调优):

```

curl -H "Content-Type:application/json" -XGET
'http://localhost:9200/my_blogs/_search?pretty' -d '
{
 "query": {
 "dis_max": {
 "queries": [
 {
 "match": {
 "title": "Brown fox"
 }
 },
 {
 "match": {
 "body": "Brown fox"
 }
 }
],
 "tie_breaker": 0.3
 }
 }
}'

```

检索结果:

```

{
 "_index": "my_blogs",
 "_type": "blog",
 "_id": "2",
 "_score": 0.77041256, //相关性分值不变，因为除了最佳查询子句，另外的子句不匹配
 "_source": {
 "title": "Keeping pets healthy",
 "body": "My quick brown fox eats rabbits on a regular basis."
 }
},
{
 "_index": "my_blogs",
 "_type": "blog",
 "_id": "1",
 "_score": 0.7564799, //相关性分值增大，增加了另外一个子句分值*0.3
 "_source": {
 "title": "Quick brown rabbits",
 "body": "Brown rabbits are commonly seen."
 }
}

```

# multi\_match查询

## 对多个字段执行相同的查询

"fields": ["\_title", "body^2"] -> "\_title": 通配符字段, "body^2": 增加"body"字段权重

37. 对字段"title"、"body"执行最佳字段查询, 查询字符串"Quick brown fox", 最少匹配30%:

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/my_blogs/_search?pretty' -d '{
 "query": {
 "multi_match": {
 "query": "Quick brown fox", #查询字符串
 "type": "best_fields", #最佳字段查询
 "fields": [#查询字段
 "title",
 "body"
],
 "tie_breaker": 0.3, #最佳字段查询调优
 "minimum_should_match": "30%" #最少匹配的关键词百分比
 }
 }
}'
```

等价于:

```
curl -H "Content-Type:application/json" -XGET 'http://localhost:9200/my_blogs/_search?pretty' -d '{
 "query": {
 "dis_max": {
 "queries": [
 {
 "match": {
 "title": { #查询字段
 "query": "Quick brown fox",
 "minimum_should_match": "30%"
 }
 }
 },
 {
 "match": {
 "body": { #查询字段
 "query": "Quick brown fox",
 "minimum_should_match": "30%"
 }
 }
 }
],
 "tie_breaker": 0.3
 }
 }
}'
```

