

typora

ELK:

E:ElasticSearch:它是一个数据库，nosql中的一种，就是实现高亮，分词，

L:logstash:它是一个收集工具，收集各种各样的数据，把数据不仅仅存放到ES

K:kibana:它是一个界面化管理工具，自带好多功能，比如，查询，筛选查询，索引的周期管理

ELK在我们项目中的，日志收集分析占60%，40%是做订单管理

最牛逼最牛逼的是京东

nosql:

reids:内存数据库，8大数据结构。它能做筛选查询，统计，聚合查询，所有的数据都放在内存中

mongodb: 它是一个在redis和mysql之间的一种数据库，有些数据在内存，有些数据在硬盘

有一家非常牛逼的公司，用的它，存了十亿数据，它能做聚合查询，也能做筛选，而且只有

数据格式json就可以存储进去。最大的优点：快速开发，没有具体表结构，随便搞就可以

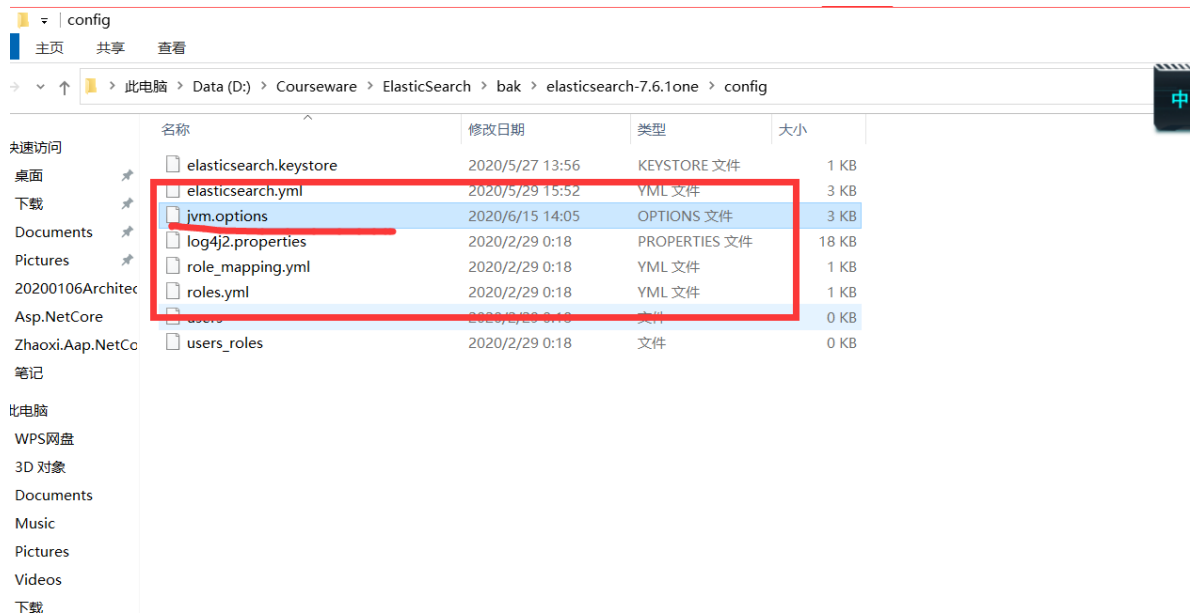
，最最牛逼是地图。只有牛逼的公司，尤其是做地理地图，

es: 可以存储各种各样的数据，json 必须可以存，它的数据上百亿

hbase: 大数据生态圈里面比较牛逼，列数据库

ElasticSearch和Lucene

```
1 #Lucene 它是java写的，分词加查询
2
3 ElasticSearch: 针对于Lucene封装了一次，使用效果比较好---我们的es集群中一个分片的实例，就是lucene实例：是跨语言，支持restful分隔
4
5 nosql:它的数据不需要严格规范，如果你需要也可以。每一种nosql产品他们都有不同的操作手法
6 百度--无作为
7
```



es:使用之前，如果你是win10，必须安装java jdk

ps

- 1 ### 注意如果你是自己电脑
- 2 先安装java jdk 10+
- 3 再来安装node.js

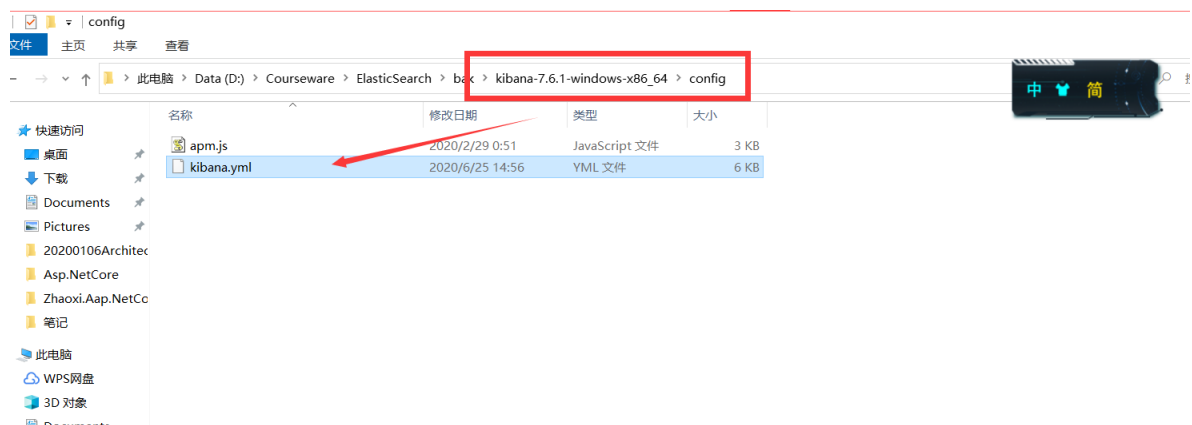
建议：使用elk时候，记住了，选择版本一定要一样

如果在docker安装，切记，

在阿里云上面的安全端口号，把9200,9300,5601 全加入进去

kibana 修改配置文件

- 1 i18n.locale: "zh-CN" --可以支持中文



修改给分配的内存 --最小的256m

```
15 ## for more information
16 ##
17 #####
18
19 # Xms represents the initial size of total heap space
20 # Xmx represents the maximum size of total heap space
21
22 -Xms256M
23 -Xmx256M
24
25 #####
26 ## Expert settings
27 #####
28 ##
29 ## All settings below this section are considered
30 ## expert settings. Don't tamper with them unless
31 ## you understand what you are doing
```

head 这是一个插件，类似于kibana，

## 环境搭建

```
1 环境搭建（windows）
2 ps: 切记这些软件包的版本必须一致
3
4 启动ES
5 找到 bin/elasticsearch.bat 双击就可以了
6 #如果需要通过head 连接，则需要修改config/elasticsearch.yml 新增下面两句话解决跨域问题
7 http.cors.enabled: true
8 http.cors.allow-origin: "*"
9 启动 head
10 #先修改es配置 config/elasticsearch.yml
11 http.cors.enabled: true
12 http.cors.allow-origin: "*"
13 #在head目录中 输入命令 记得一定要按照node js
14 npm install --node.js 安装相关的依赖包
15 npm run start --启动当前的插件
16
17 Elasticsearch集群不同颜色代表什么？
18 绿色—最健康的状态，代表所有的主分片和副本分片都可用；
19 黄色—所有的主分片可用，但是部分副本分片不可用；
20 红色—部分主分片不可用。（此时执行查询部分数据仍然可以查到，遇到这种情况，还是赶快解决比较好。
21 启动Kibana
22 #找到/bin/kibana.bat 双击就可以了
23 #如果需要修改连接的es的地址，则修改config/kibana.yml
24 集群搭建（多个节点运行windows）
25 # 复制出不同的三个es 程序，然后修改config/elasticsearch.yml 文件
26 # 切记 启动的时候，把目录下的data目录删除掉
27 启动三个节点
28
29 #第一个节点的配置
30 # ===== Elasticsearch Configuration =====
```

```
31 # 配置es的集群名称, es会自动发现在同一网段下的es,如果在同一网段下有多个集群,就可以用这个属性来区分不同的集群。
32 cluster.name: elasticsearch
33 # 节点名称
34 node.name: node-001
35 # 指定该节点是否有资格被选举成为node
36 node.master: true
37 #第一次启动的时候,指定的主节点,同样的,第一个主节点必须先启动
38 cluster.initial_master_nodes: ["127.0.0.1:9300"]
39 # 指定该节点是否存储索引数据,默认为true。
40 node.data: true
41 # 设置绑定的ip地址还有其它节点和该节点交互的ip地址,本机ip
42 network.host: 127.0.0.1
43 # 指定http端口,你使用head、kopf等相关插件使用的端口
44 http.port: 9200
45 # 设置节点间交互的tcp端口,默认是9300。
46 transport.tcp.port: 9300
47 #设置集群中master节点的初始列表,可以通过这些节点来自动发现新加入集群的节点。
48 #因为下两台elasticsearch的port端口会设置成9301 和 9302 所以写入两台#elasticsearch地址的完整路径
49 discovery.zen.ping.unicast.hosts:
50 ["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
51 #如果要使用head,那么需要解决跨域问题,使head插件可以访问es
52 http.cors.enabled: true
53 http.cors.allow-origin: "*"
54 #第二个节点的配置
55 # ===== Elasticsearch Configuration =====
56 # 配置es的集群名称, es会自动发现在同一网段下的es,如果在同一网段下有多个集群,就可以用这个属性来区分不同的集群。
57 cluster.name: elasticsearch
58 # 节点名称
59 node.name: node-002
60 # 指定该节点是否有资格被选举成为node
61 node.master: true
62 # 指定该节点是否存储索引数据,默认为true。
63 node.data: true
64 # 设置绑定的ip地址还有其它节点和该节点交互的ip地址,本机ip
65 network.host: 127.0.0.1
66 #第一次启动的时候,指定的主节点,同样的,第一个主节点必须先启动
67 cluster.initial_master_nodes: ["127.0.0.1:9300"]
68 # 指定http端口,你使用head、kopf等相关插件使用的端口
69 http.port: 9201
70 # 设置节点间交互的tcp端口,默认是9300。
71 transport.tcp.port: 9301
72 #设置集群中master节点的初始列表,可以通过这些节点来自动发现新加入集群的节点。
73 #因为下一台elasticsearch的port端口会设置成9301 所以写入两台#elasticsearch地址的完整路径
74 discovery.zen.ping.unicast.hosts:
75 ["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
76 #如果要使用head,那么需要增加新的参数,使head插件可以访问es
77 http.cors.enabled: true
78 http.cors.allow-origin: "*"
79 #第三个节点的配置
80 # ===== Elasticsearch Configuration =====
81 # 配置es的集群名称, es会自动发现在同一网段下的es,如果在同一网段下有多个集群,就可以用这个属性来区分不同的集群。
82 cluster.name: elasticsearch
83 # 节点名称
84 node.name: node-003
85 # 指定该节点是否有资格被选举成为node
86 node.master: true
```

```
85 #第一次启动的时候,指定的主节点,同样的,第一个主节点必须先启动
86 cluster.initial_master_nodes: ["127.0.0.1:9300"]
87 # 指定该节点是否存储索引数据,默认为true。
88 node.data: true
89 # 设置绑定的ip地址还有其它节点和该节点交互的ip地址,本机ip
90 network.host: 127.0.0.1
91 # 指定http端口,你使用head、kopf等相关插件使用的端口
92 http.port: 9202
93 # 设置节点间交互的tcp端口,默认是9300。
94 transport.tcp.port: 9302
95 #设置集群中master节点的初始列表,可以通过这些节点来自动发现新加入集群的节点。
96 #因为下一台elasticsearch的port端口会设置成9301 所以写入两台#elasticsearch地址的完整
    路径
97 discovery.zen.ping.unicast.hosts:
    ["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
98 #如果要使用head,那么需要增加新的参数,使head插件可以访问es
99 http.cors.enabled: true
100 http.cors.allow-origin: "*"
101 ###docker 安装
102
103 安装ES docker
104 docker network create --driver bridge --subnet 192.168.0.0/16 --gateway
    192.168.0.1 mynet
105
106 #下载es镜像
107 docker pull elasticsearch:7.2.0
108 #运行es 限定内存大小
109 docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -e
    "discovery.type=single-node" -e ES_JAVA_OPTS="-Xms100m -Xmx200m" --net mynet
    elasticsearch:7.2.0
110 安装kibana
111 #下载kibana镜像
112 docker pull kibana:7.2.0
113 #运行 如果地址没起作用,可以在容器里面的usr/share/kibana/config 中修改配置文件
114 docker run -p 5601:5601 -v -d -e ELASTICSEARCH_URL=http://阿里云:9200 --net mynet
    kibana:7.2.0
115 #查看日志
116 docker logs dockerid
117 #验证
118 curl http://localhost:5601
119 安装Logstash
120 1、下载Logstash镜像
121
122 docker pull logstash:7.2.0
123
124
125 2、编辑logstash.yml配置文件
126 logstash.yml配置文件放在宿主机/data/elk/logstash目录下,内容如下:
127
128 path.config: /usr/share/logstash/conf.d/*.conf
129 path.logs: /var/log/logstash
130
131
132 3、编辑test.conf文件
133 test.conf文件放在宿主机/data/elk/logstash/conf.d目录下,内容如下:
134
135 复制代码
136 input {
137     beats {
138         port => 5044
139         codec => "json"
```

```

140     }
141 }
142
143 output {
144     elasticsearch { hosts => ["39.96.34.52:9200"] }
145     stdout { codec => rubydebug }
146 }
147 复制代码
148
149
150 4、启动logstash
151
152 docker run -d -p 8001:8001 --net mynet --log-driver json-file -v
    /usr/share/logstash/config/logstash.conf:/usr/share/logstash/pipeline/logstash.c
    onf logstash:7.2.0
153
154
155
156
157 5、查看容器运行状态
158
159 docker ps
160
161 docker logs -f xinyar-logstash
162
163 #启动
164 logstash.bat -f logstash.conf
165
166
167

```

## 基本操作

针对于es里面的任何操作指令，都是大写

```

1  #查看所有的索引库
2  GET _cat/indices
3  #查询当前库所有数据
4  GET dbclay/_doc/_search
5  #插入数据
6  PUT dbclay/_doc/1
7  {
8      "name":"clay"
9  }

```

这个sql查询，是一个插件，其他人搞的；

如果你做的数据是，全链路，订单系统，你都可以用sql --我们自己生产环境上，就sql

记得到网上--

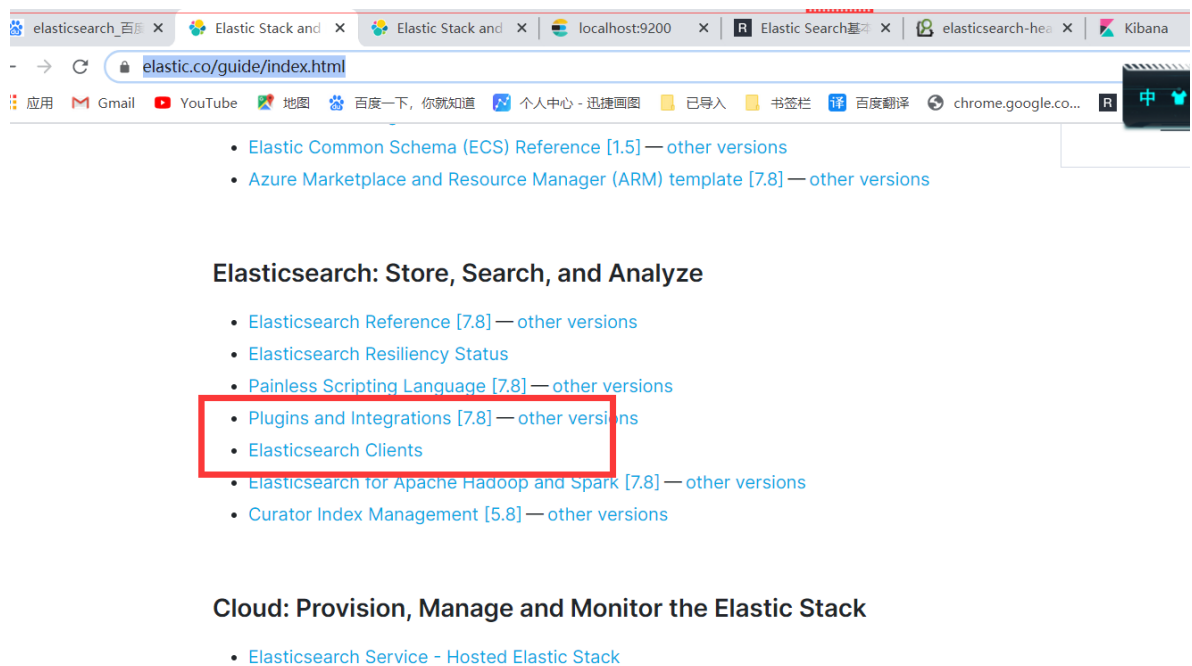
记得查数据的时候用csv，可以大量节省流量

性能上没有半毛钱的生气

如果你操作是各种各样的数据，字段不规范

*.netcore*

```
1 https://www.elastic.co/guide/index.html
2
```



# Elastic Search基本操作

基本操作

## 创建文档

```
1 # userinfo 代表的是类型 后续则就取消了 索引名称/类型/id 默认类型是_doc PUT
  dbindex/_doc/1
2 PUT dbindex/userinfo/1
3 {
4   "name":"clay",
5   "age":18,
6   "actiontime":"2020-05-02 23:22:22.333",
7   "id":123
8 }
9 #可以不需要指定id
10 POST dbindex/userinfo
11 {
12   "name":"clay",
13   "age":18,
14   "actiontime":"2020-05-02 23:22:22.333",
15   "id":123
16 }
```

## 查看文档信息

```
1 #1.代表文档的id
2 GET dbindex/userinfo/1
3 #2.查看所有的数据
```

## 查看所有索引和索引中字段类型

```
1 #如果自己的文档字段没有指定，那么es 就会给我们默认配置字段类型
2 GET dbindex
3 #查看所有索引情况！
4 GET _cat/indices?v
```

## 字段类型

字符串: text, keyword

数值: long, integer, short, byte, double, float, half float, scaled float

日期类型: date

布尔类型: boolean

二进制类型: binary

等等....

## 千万注意

```
1 keyword: 存储数据时候，不会分词建立索引
2 text: 存储数据时候，会自动分词，并生成索引（这是很智能的，但在有些字段里面是没用的，所以对于有些字段使用text则浪费了空间）。
```

## 创建索引约束类型

```
1 # 默认是_doc 完整版: PUT /Test2/_doc 默认的就是_doc
2 PUT /test2
```



```

3  {
4    "mappings": {
5
6      "properties": {
7        "name":{
8
9          "type": "text"
10       },
11       "age":{
12
13         "type": "long"
14
15       },
16       "actiontime":
17       {
18         "type": "date",
19         "format": "yyyy-MM-dd HH:mm:ss"
20
21       },
22       "describe":
23       {
24         "type": "keyword"
25       }
26
27     }
28   }
29
30 }
31
32 }

```

## 根据约束写入数据

```

1  PUT /test2/_doc/1
2  {
3
4    "name": "name2",
5    "age": 11,
6    "actiontime": "2018-09-24 19:23:45",
7    "describe": 12
8  }

```

## 查看约束情况

```

1  #语法: GET 索引名称
2  #查看数据库类型约束情况
3  GET /test2
4  #根据id=1查询
5  GET /test2/_doc/1

```

## 修改

```

1  #之前的修改 就是覆盖 增加版本号
2  POST /test2/_doc/1
3  {
4
5    "name": "name3",
6    "age": 11

```

```
7 }
8 #现在的修改 可以修改制定的值 不做替换
9 POST /test2/_doc/1/_update
10 {
11     "doc":{
12
13     "name":"name55"
14     }
15 }
```

## 删除

```
1 #删除当前文档
2 DELETE /test2/_doc/1
3 #删除索引
4 DELETE /test2
```

## 高级查询进阶

### text和keyword 的类型比较

```
1 Elasticsearch 5.0以后，string字段被拆分成两种新的数据类型：
2 Text：会分词，然后根据分词后的内容建立倒排索引（反向索引）
3 不支持聚合
4 keyword：不进行分词，直接直接根据字符串内容建立倒排索引（反向索引）
5 支持聚合
6 #keyword 不会在分词了，而text 会继续分词
7 PUT /clayindex
8 {
9     "mappings": {
10         "properties": {
11             "name":{
12                 "type": "keyword"
13             },
14             "address":{
15                 "type": "text"
16             }
17         }
18     }
19 }
20
21 GET /clayindex
22
23 GET clayindex/_analyze
24 {
25     "field": "name",
26     "text": "中华人民共和国"
27 }
28
29 GET clayindex/_analyze
30 {
31     "field": "address",
32     "text": "中华人民共和国"
33 }
34
35 #准备数据
36 DELETE /clayindex
37 PUT /clayindex
```

```
38 {
39   "mappings": {
40     "properties": {
41       "name": {
42         "type": "keyword"
43       },
44       "address": {
45         "type": "text"
46       },
47       "age": {
48         "type": "integer"
49       }
50     }
51   }
52 }
53 #插入数据 可以自己生成id //
54 POST /clayindex/_doc
55 {
56   "name": "曹操",
57   "address": "魏国",
58   "age": 18
59 }
60 }
61 POST /clayindex/_doc
62 {
63   "name": "贾诩",
64   "address": "魏国",
65   "age": 19
66 }
67 }
68 POST /clayindex/_doc
69 {
70   "name": "诸葛亮",
71   "address": "蜀国",
72   "age": 37
73 }
74 }
75 POST /clayindex/_doc
76 {
77   "name": "关羽",
78   "address": "蜀国",
79   "age": 35
80 }
81 }
82 POST /clayindex/_doc
83 {
84   "name": "周瑜",
85   "address": ["吴国", "蜀国"],
86   "age": 25
87 }
```

## 简单查询

```

1 #查询所有的数据
2 GET clayindex/_doc/_search
3 #查询name是曹操的数据
4 GET clayindex/_doc/_search?q=name:曹操
5 #查询地址是魏国的，注意查询会把 魏国拆分，会查出包含 魏国，魏，国的所有数据，因为类型是text
  会被分词
6 GET clayindex/_doc/_search?q=address:魏国

```

## 复杂查询 参数查询

```

1 # name 在创建类型的时候是keyword 所以没有分词查询
2 GET clayindex/_doc/_search
3 {
4
5   "query":{
6
7     "match": {
8       "name": "曹操"
9     }
10  }
11 }
12 #address 创建的时候是text ，所以查询的时候被分词查询，比如 查出所有包含国字的数据
13 GET clayindex/_doc/_search
14 {
15
16   "query":{
17
18     "match": {
19       "address": "魏国"
20     }
21   }
22 }
23 #_source 可以设置需要返回需要的字段  select name from ==
24 GET clayindex/_doc/_search
25 {
26
27   "query":{
28
29     "match": {
30       "address": "魏国"
31     }
32   },
33   "_source":["name"]
34 }
35
36

```

## 排序

```

1 GET clayindex/_doc/_search
2 {
3
4   "query":{
5
6     "match": {
7       "address": "魏国"
8     }
9   },
10  "_source":["name"],

```

```
11  "sort":[
12    {"age":{"order":"desc"}},
13    {"name":{"order":"asc"}}
14  ]
15 }
```

## 分页查询

```
1  GET clayindex/_doc/_search
2  {
3
4    "query":{
5
6      "match": {
7        "address": "魏国"
8      }
9    },
10   "_source":["name"],
11   "sort":[
12     {"age":{"order":"desc"}},
13     {"name":{"order":"asc"}}
14   ],
15   "from":0,
16   "size":10
17 }
```

## 布尔值查询

条件1 and 条件2

must =>and

```
1  GET clayindex/_doc/_search
2  {
3
4    "query":{
5      "bool": {
6        "must": [
7          {
8            "match": {
9              "address": "魏国"
10            }
11          },{
12            "match": {
13              "name": "曹操"
14            }
15          }
16        ]
17      }
18    },
19   "_source":["name"],
20   "sort":[
21     {"age":{"order":"desc"}},
22     {"name":{"order":"asc"}}
23   ],
24   "from":0,
25   "size":2
26 }
```

条件1 or 条件2

should => or

```

1 GET clayindex/_doc/_search
2 {
3
4   "query":{
5     "bool": {
6       "should": [
7         {
8           "match": {
9             "address": "魏国"
10          }
11
12          },{
13            "match": {
14              "name": "曹操"
15            }
16
17          }
18        ]
19
20      }
21    },
22    "_source":["name"],
23    "sort":[
24      {"age":{"order":"desc"}},
25      {"name":{"order":"asc"}}
26    ],
27    "from":0,
28    "size":10
29  }
```

排除 must\_not

```

1 #查询名字不是曹操的信息
2 GET clayindex/_doc/_search
3 {
4
5   "query":{
6
7     "bool": {
8       "must_not": [
9         {
10           "match": {
11             "name": "曹操"
12           }
13         }
14       ]
15     }
16   }
17
18 }
19 }
```

数据过滤filter

```

1  gt   大于
2  gte  大于等于
3  lt   小于
4  lte  小于等于!
5  # 查询名字是曹操，然后年龄大于等于10 小于等于30
6  GET clayindex/_doc/_search
7  {
8
9      "query":{
10
11          "bool": {
12              "must": [
13                  {
14                      "match": {
15                          "name": "曹操"
16                      }
17                  }
18              ],
19              "filter": {
20                  "range": {
21                      "age": {
22                          "gte": 10,
23                          "lte": 20
24                      }
25                  }
26              }
27          }
28      }
29  }
30  }
31  }

```

## 配备多个条件查询

```

1  # 使用空格隔开，查询是吴国又是魏国的，只要满足一个条件就可以了
2  GET clayindex/_doc/_search
3  {
4
5      "query":{
6          "match":{"address":"吴国 魏国 "}
7      }
8  }
9  }

```

## 精确查询

### 直接根据倒序索引精确查询

```

1  match和term的区别
2  match，会使用分词器解析！注意：先分词然后再去查询
3  term，直接查询精确的
4  #在定义的时候可以指定查询时候的分词规则，注意只有text类型才是支持的，而且分词类型和搜索分词类型必须都要设置
5  PUT /clayindex3
6  {
7      "mappings": {
8          "properties": {
9              "name":{
10                 "type": "keyword"

```

```

11     },
12     "address": {
13         "type": "text",
14         "analyzer": "standard",
15         "search_analyzer": "standard"
16     },
17     "age": {
18         "type": "integer"
19     }
20 }
21 }
22 }
23
24 #分词和搜索分词的类型可以不同
25 PUT /clayindex4
26 {
27     "mappings": {
28         "properties": {
29             "name": {
30                 "type": "keyword"
31             },
32             "address": {
33                 "type": "text",
34                 "analyzer": "ik_max_word",
35                 "search_analyzer": "standard"
36             },
37             "age": {
38                 "type": "integer"
39             }
40         }
41     }
42 }
43 #注意, 针对 text 类型区分精确查询和模糊查询, 如果是keyword类型的话不区分, 因为没有给
keyword类型做分词
44 #根据魏国模糊查询, 包含魏和国的都可以查询
45 GET /clayindex/_doc/_search
46 {
47     "query": {
48         "match": {
49             "address": "魏国"
50         }
51     }
52 }
53 }
54
55 #精准查询 查不出数据, 因为分词默认分成了 魏和国, 所以 可以试试查查魏或者国
56 GET /clayindex/_doc/_search
57 {
58     "query": {
59         "term": {
60             "address": "魏国"
61         }
62     }
63 }
64 }
65 #keyword 类型不会被分词的
66 GET _analyze
67 {
68     "analyzer": "keyword",
69     "text": "魏国"
70 }

```



```

71
72 #这是默认情况下的分词情况
73 GET _analyze
74 {
75   "analyzer": "standard",
76   "text": "魏国"
77 }

```

多个值得精确查询-》直接根据倒序索引来查的

```

1  #根据名字是曹操，而address 是国的索引
2  GET clayindex/_doc/_search
3  {
4
5    "query":{
6      "bool": {
7        "must": [
8          {
9            "term": {
10             "address": "国"
11           }
12
13         },{
14           "term": {
15             "name": "曹操"
16           }
17
18         }
19       ]
20
21     }
22   },
23   "_source":["name"],
24   "sort":[
25     {"age":{"order":"desc"}},
26     {"name":{"order":"asc"}}
27   ],
28   "from":0,
29   "size":2
30 }

```

高亮查询

```

1  #查询name高亮，给包含曹操的高亮，注意，keyword 不做分词
2  GET clayindex/_doc/_search
3  {
4
5    "query":{
6      "match": {
7        "name": "曹操"
8      }
9    },
10   "highlight":{
11
12     "fields": {
13       "name": {}
14     }
15   }
16 }

```

```

17 #查询魏国，并且高亮，给魏和国都高亮了，因为是text 分词了
18 GET clayindex/_doc/_search
19 {
20
21     "query":{
22         "match": {
23             "address": "魏国"
24         }
25     },
26     "highlight":{
27
28         "fields": {
29             "address": {}
30         }
31     }
32 }

```

## 自定义高亮的风格

```

1 GET clayindex/_doc/_search
2 {
3
4     "query":{
5         "match": {
6             "name": "曹操"
7         }
8     },
9     "highlight":{
10
11         "pre_tags": "<p class='gaoliang'>",
12         "post_tags": "</p>",
13         "fields": {
14             "name": {}
15         }
16     }
17 }
18
19 匹配
20 按照条件匹配
21 精确匹配
22 区间范围匹配
23 匹配字段过滤
24 多条件查询
25 高亮查询

```

## SQL 查询

```

1 # 通过sql查询的话，必须不能存在text的类型
2 POST /sqldbindex/_doc
3 {
4     "name":"a",
5     "age":12,
6     "address":"中华人民共和国",
7     "birth":"2019-01-0"
8
9 }
10 GET /sqldbindex
11
12 POST /_xpack/sql?format=txt

```

```

13 {
14   "query":"select * from sqldbindex"
15 }
16 # txt,json,csv 等等多种格式表名称就是索引的名称
17 POST /_xpack/sql?format=txt
18 {
19   "query":"select address,count(distinct name),sum(age) from sqldbindex group by
20   address "
21 }
22 #分页
23 POST /_xpack/sql?format=json
24 {
25   "query":"select address,count(distinct name),sum(age) from sqldbindex group by
26   address limit 2 ",
27   "fetch_size":5
28 }

```

## 滚动查询

```

1  #当搜索请求返回一个结果页面时，滚动API可用于从单个搜索请求检索大量结果(甚至所有结果)，这
2  与在传统数据库上使用游标的方式非常相似。数据量太大 不像一次性查出来
3  #写入数据
4  POST /scrolldb/_doc
5  {
6    "title":"abc"
7  }
8
9  #根据滚动查询，查询2条，不能超过1m内存
10 POST /scrolldb/_search?scroll=1m
11 {
12   "size": 2,
13   "query": {
14     "match" : {
15       "title" : "abc"
16     }
17   }
18 }
19 # 使用返回来的scroll_id 继续查询，直到查不出为止 每次返回的size 不会改变
20 POST /_search/scroll
21 {
22   "scroll" : "2m",
23   "scroll_id" :
24   "DXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAAA0AWT1F2d0ppdC1TV3EtUUJaNUh4dEJHZw=="
25 }

```

我们可以在规范的数据库下面，对有些字段，做一些限制，有些字段，只能int，有些字段不分词

使用es的时候，它默认自带分词器，是老外，一个单词一个单词

，如果我们的项目中有大量的中文，你就需要自己在安装一个插件

好多人的姓名，好多的药品

订单数据，或者全链路的日志，或者业务数据

比较规范的数据

es 连接

公司所有

90% 如果是做日志收集的话,kafka json

你给我提供一个 topic,小型的公司，或者数据量不大的公司

只会往kafka里面写json--

很少对es数据库做删除，做修改

稍微包装--，如果能力允许，按照我的思路。。硬气

好好学习，多写，多练，多思考

把视频和代码变成库存--

结束了刷个C 神

如果业务中整个链路不是web调用 guid

redis,mysql ,sqlserver

mysql(gudi)

sqlserver(gudi)

redis(guid)

这些东西都在我当前的进程里面

```
static rid=guid
```

```
rid=1 mysql redis //oracle rid=2
```

```
rid=2
```

```
//数据槽 //elk
```

## text, keyword类型区别

```
1 | text:如果一个字段的值你需要分词，你就用它
2 | keyword: 如果这个字段的值，你不需要分词，则需要用这个类型，比如姓名，一些专业名称，药品，
   | 商品名称
3 |
4 | string
5 |
6 | 我之前用的es 中如果有一个字段类型是text，不支持sql;
7 |
8 | 注重细节细节
9 |
10 | clay给你们提个意见，从今天，开始，给自己立一个小的目标从细节
```

---

```
1 | 倒排索引=反向索引
2 | 如果要存储一段内容
3 | 先把内容中的句子分词，然后每一个分词就是一个关键词
```

## fst：索引压缩法

```
1 | #通过一种手段，在内存中共享了key的前缀和后缀，
2 |
3 | 用内存中很少的数据，表示出大量信息
4 |
5 | 1 2 3 4 5
6 | 1 3 6 10 15
7 |
8 | int = 4个字节
```

```

9
10 int =3 4
11
12 bit 8个数字标识一个字节
13
14 0001 =3 0.5个字节
15
16 01010100 1个字节
17 1 3 5
18
19
20 【1 汉字】
21 【2 汉字】
22
23 1 int
24
25 最小单位, bit
26 手毛孔
27
28
29
30 01010100 0表示没有, 1表示有。
31 1 3 5
32
33 010101000101010001010100
34
35
36 1 2 5 7 9 11
37
38 1 4 6 5 8 9
39
40 1 5 11 16 24 33
41
42
43 8:73 227 2 5:30 11 29
44
45 1001001 11100011 10 11110 1011 11101
46 7 8 2 5 4 5
47
48 bit[8] bit[5]
49
50
51 你的思路不在线-- 专一
52
53
54
55 int16[]
56 int32 []
57 int64[]
58
59 int16[]
60
61
62 int64[]
63
64 我会给你好多数据
65 int[64] 8个字节
66
67 1,2,4,8 ...10000 2的63次方
68
69 int64[]

```

70  
71  
72 压缩value  
73 为什么是以65535为界限 ,死值  
74 跟2 --  
75  
76 为什么做成65535呢，也是为节省空间  
77  
78 30位  
79  
80 100000  
81  
82 0 65535 65536 131071 131071 ===  
83  
84  
85 如果你面试永远遇不到，OK，今天东西没必要，你不是架构师  
86  
87  
88 如果你是一个架构师，这些东西，哪怕面试官，我们也需要懂，为什么  
89  
90  
91 如果你仅仅是为了面试的话，你这层次到不了（）--你永远做不了架构师  
92  
93 通过很少量的信息，来表示更多的数据--  
94  
95  
96 不说话  
97  
98  
99 65535 ---  
100  
101  
102  
103  
104 小一层  
105  
106 《《《《《-6699-6701 6601-6699-》》》》  
107  
108 6601-6699  
109  
110 <<< 6601-6630 6631-6699>>>>>>>  
111  
112 CCCCCC  
113  
114  
115  
116  
117  
118 买阿里云服务的，或者自己用linux，不要用国外的仓库，  
119  
120 用阿里云的仓库  
121  
122 让你在配置文件中。配置一下，当选择新节点的时候，需要有大于4分之3的节点任务之前的主节点，才会选择新的节点  
123  
124 100%  
125  
126 大数据高并发里面，  
127  
128  
129 这段时间，可能短暂就不能用了==

```
130
131 discovery.zen.minimum_master_nodes:4
132
133
134
135 我们的es集群会出现各种不可知的问题===
136
137 分片就是解决负载均衡的问题
138
139 一个分片就是lucene 实例
140 1.实现负载均衡
141 2.实现数据的高可用，数据的备份
142
143
144
145
146
147
148
149
150
151
```