typora

ELK:

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

L:logstach: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

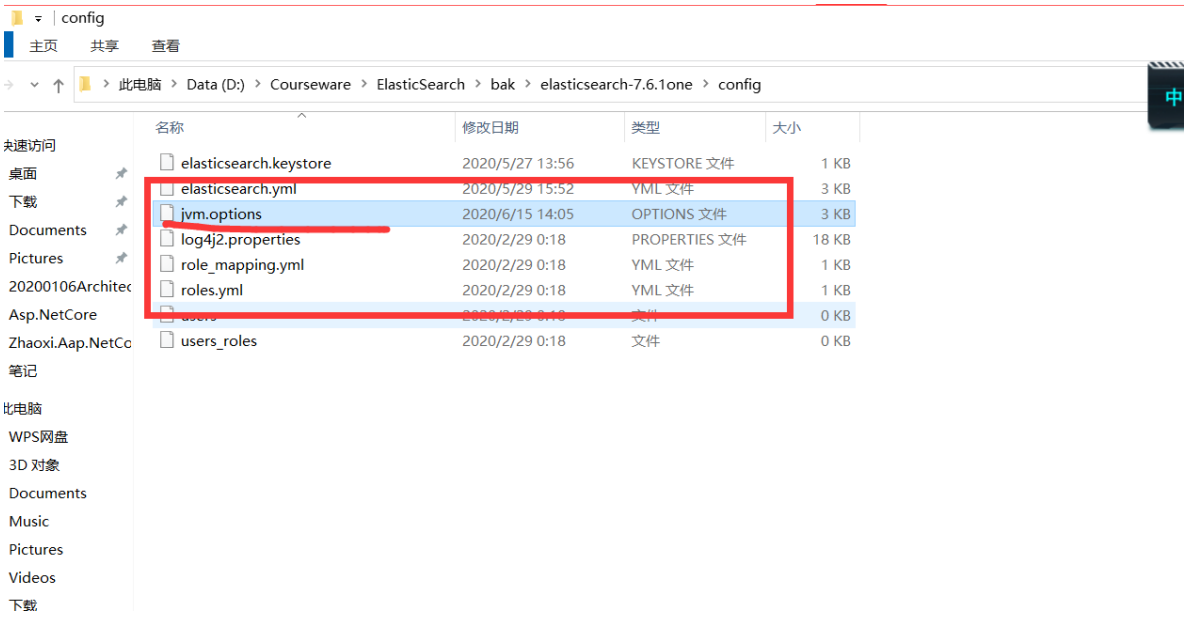
ElasticSearch和Lucene

```
#Lucene  它是java写的，分词加查询

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

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
百度--无作为

```



es:使用之前，如果你是win10 ，必须安装java jdk

ps

```
### 注意如果你是用自己电脑
先安装java jdk 10+
再来安装node.js
```

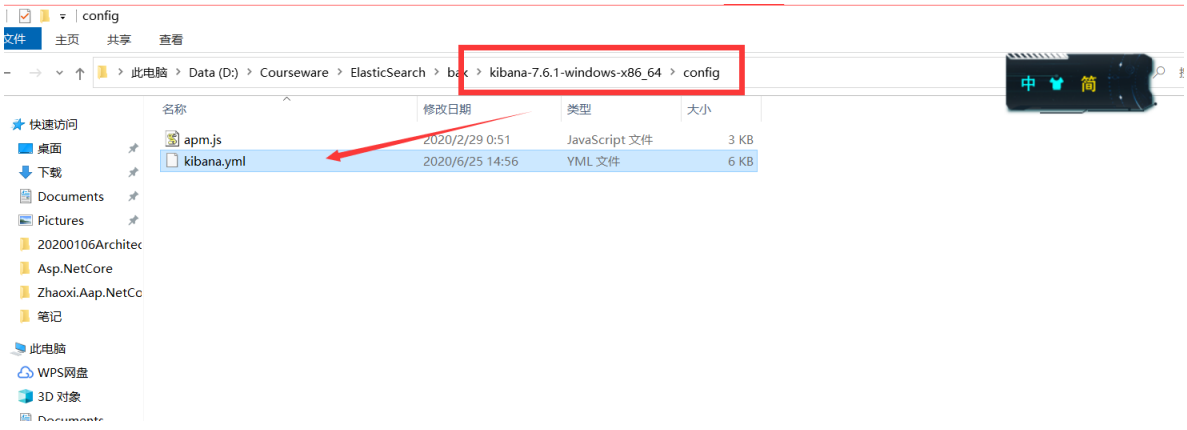
建议：使用elk时候，记住了，选择版本一定要一样

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

```
i18n.locale: "zh-CN" --可以支持中文
```



修改给分配的内存 --最小的256m

```
## for more information
##
################################################################

# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space

-Xms256M
-Xmx256M

################################################################
## Expert settings
################################################################
##
## All settings below this section are considered
## expert settings. Don't tamper with them unless
## you understand what you are doing
```

head 这是一个插件，类似于kibana，

环境搭建

```
环境搭建（windows）
ps:切记这些软件包的版本必须一致

启动ES
找到 bin/elasticsearch.bat 双击就可以了
#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
http.cors.enabled: true
http.cors.allow-origin: "*"
启动 head
#先修改es配置 config/elasticsearch.yml
http.cors.enabled: true
http.cors.allow-origin: "*"
#在head目录中 输入命令  记得一定要按照node js
npm install  --node.js 安装相关的依赖包
npm run start  --启动当前的插件

Elasticsearch集群不同颜色代表什么？
绿色—最健康的状态，代表所有的主分片和副本分片都可用；
黄色—所有的主分片可用，但是部分副本分片不可用；
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。
启动Kibana
#找到/bin/kibana.bat 双击就可以了
#如果需要修改连接的es的地址，则修改config/kibana.yml
集群搭建（多个节点运行windows）
# 复制出不同的三个es 程序，然后修改config/elasticsearch.yml 文件
# 切记 启动的时候，把目录下的data目录删除掉
启动三个节点

#第一个节点的配置
# ================= Elasticsearch Configuration ===================
```

```
# 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。
cluster.name: elasticsearch
# 节点名称
node.name: node-001
# 指定该节点是否有资格被选举成为node
node.master: true
#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
cluster.initial_master_nodes: ["127.0.0.1:9300"]
# 指定该节点是否存储索引数据,默认为true。
node.data: true
# 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
network.host: 127.0.0.1
# 指定http端口,你使用head、kopf等相关插件使用的端口
http.port: 9200
# 设置节点间交互的tcp端口,默认是9300。
transport.tcp.port: 9300
#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。
#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
discovery.zen.ping.unicast.hosts:
["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
#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
http.cors.enabled: true
http.cors.allow-origin: "*"
#第二个节点的配置
# ================= Elasticsearch Configuration ===================
# 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。
cluster.name: elasticsearch
# 节点名称
node.name: node-002
# 指定该节点是否有资格被选举成为node
node.master: true
# 指定该节点是否存储索引数据,默认为true。
node.data: true
# 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
network.host: 127.0.0.1
#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
cluster.initial_master_nodes: ["127.0.0.1:9300"]
# 指定http端口,你使用head、kopf等相关插件使用的端口
http.port: 9201
# 设置节点间交互的tcp端口,默认是9300。
transport.tcp.port: 9301
#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。
#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
discovery.zen.ping.unicast.hosts:
["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
#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
http.cors.enabled: true
http.cors.allow-origin: "*"
#第三个节点的配置
# ================= Elasticsearch Configuration ===================
# 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。
cluster.name: elasticsearch
# 节点名称
node.name: node-003
# 指定该节点是否有资格被选举成为node
node.master: true
```

```
#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
cluster.initial_master_nodes: ["127.0.0.1:9300"]
# 指定该节点是否存储索引数据,默认为true。
node.data: true
# 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
network.host: 127.0.0.1
# 指定http端口,你使用head、kopf等相关插件使用的端口
http.port: 9202
# 设置节点间交互的tcp端口,默认是9300。
transport.tcp.port: 9302
#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。
#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
discovery.zen.ping.unicast.hosts:
["127.0.0.1:9300","127.0.0.1:9301","127.0.0.1:9302"]
#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
http.cors.enabled: true
http.cors.allow-origin: "*"
###docker 安装

安装ES docker
docker network create --driver bridge --subnet 192.168.0.0/16 --gateway
192.168.0.1  mynet

#下载es镜像
docker pull elasticsearch:7.2.0
#运行es 限定内存大小
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
安装kibana
#下载kibana镜像
docker pull kibana:7.2.0
#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
docker run -p 5601:5601 -v -d -e ELASTICSEARCH_URL=http://阿里云:9200 --net mynet
kibana:7.2.0
#查看日志
docker logs dockerid
#验证
curl http://localhost:5601
安装Logstash
1、下载Logstash镜像

docker pull logstash:7.2.0


2、编辑logstash.yml配置文件
logstash.yml配置文件放在宿主机/data/elk/logstash目录下，内容如下：

path.config: /usr/share/logstash/conf.d/*.conf
path.logs: /var/log/logstash


3、编辑test.conf文件
test.conf文件放在宿主机/data/elk/logstash/conf.d目录下，内容如下：

复制代码
input {
    beats {
        port => 5044
        codec => "json"
```

```
    }
}

output {
  elasticsearch { hosts => ["39.96.34.52:9200"] }
  stdout { codec => rubydebug }
}
复制代码


4、启动logstash

docker run -d -p 8001:8001 --net mynet --log-driver json-file  -v /usr/share/logstach/config/logstash.conf:/usr/share/logstash/pipeline/logstash.conf logstash:7.2.0




5、查看容器运行状态

docker ps

docker logs -f xinyar-logstash

#启动
logstash.bat -f logstash.conf
```

基本操作

针对于es里面的任何操作指令，都是大写

```
#查看所有的索引库
GET _cat/indices
#查询当前库所有数据
GET dbclay/_doc/_search
 #插入数据
PUT  dbclay/_doc/1
{
  "name":"clay"
}
```

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

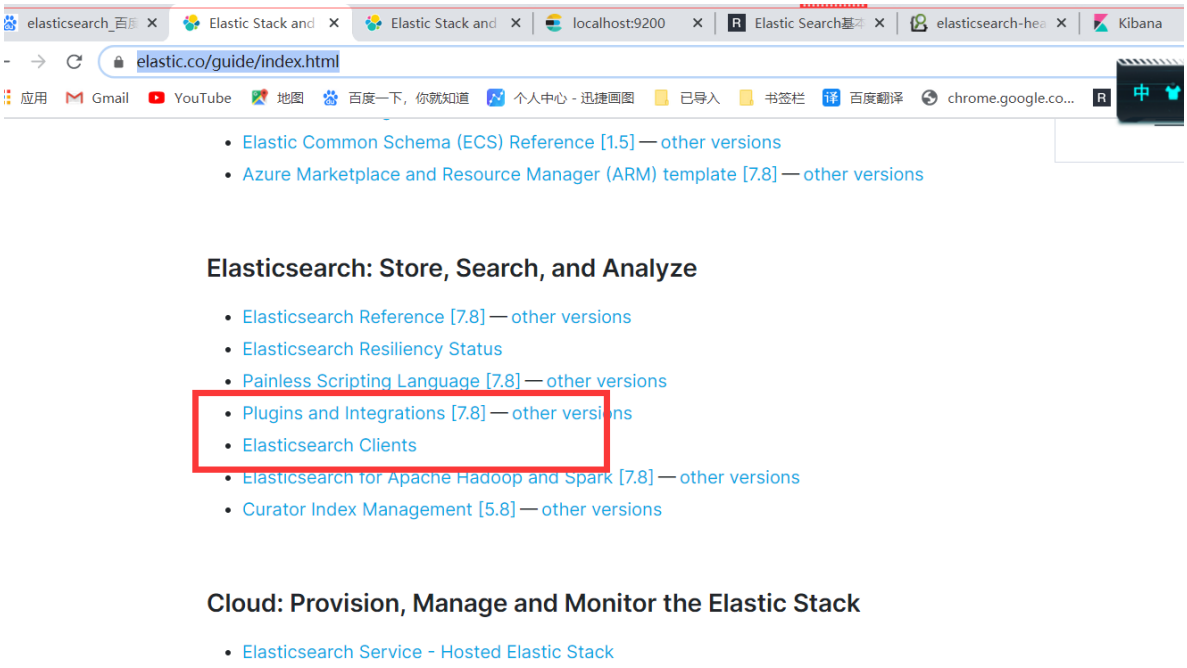
性能上没有半毛钱的生气

如果你操作是各种各样的数据，字段不规范

.netcore

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

```



Elastic Search基本操作

基本操作

创建文档

```
# 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
PUT  dbindex/userinfo/1
{
  "name":"clay",
  "age":18,
  "actiontime":"2020-05-02 23:22:22.333",
  "id":123
}
#可以不需要指定id
POST dbindex/userinfo
{
 "name":"clay",
  "age":18,
  "actiontime":"2020-05-02 23:22:22.333",
  "id":123
}
```

查看文档信息

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

查看所有索引和索引中字段类型

```
#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
GET dbindex
#查看所有索引情况!
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

```
keyword：存储数据时候，不会分词建立索引
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

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

```
{
  "mappings": {

     "properties": {
         "name":{

            "type": "text"
         },
         "age":{

           "type": "long"

         },
          "actiontime":
          {
            "type": "date",
            "format":"yyyy-MM-dd HH:mm:ss"

          },
          "describe":
          {
            "type": "keyword"
          }


     }

  }

}
```

根据约束写入数据

```
PUT /test2/_doc/1
{

  "name":"name2",
  "age":11,
  "actiontime":"2018-09-24 19:23:45",
  "describe":12
}
```

查看约束情况

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

修改

```
#之前的修改 就是覆盖 增加版本号
POST /test2/_doc/1
{

  "name":"name3",
  "age":11
```

```
}
#现在的修改   可以修改制定的值 不做替换
POST /test2/_doc/1/_update
{
   "doc":{

  "name":"name55"
   }
}
```

删除

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

高级查询进阶

text和keyword 的类型比较

```
ElasticSearch 5.0以后，string字段被拆分成两种新的数据类型：
Text：会分词，然后根据分词后的内容建立倒排索引（反向索引）
不支持聚合
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）
支持聚合
#keyword  不会在分词了，而text 会继续分词
PUT /clayindex
{
   "mappings": {
     "properties": {
     "name":{
       "type": "keyword"
     },
     "address":{
        "type": "text"
     }
     }
   }
}

GET /clayindex

GET clayindex/_analyze
{
  "field": "name",
  "text": "中华人民共和国"

}
GET clayindex/_analyze
{
  "field": "address",
  "text": "中华人民共和国"

}
#准备数据
DELETE /clayindex
PUT /clayindex
```

```
{
   "mappings": {
     "properties": {
     "name":{
       "type": "keyword"
     },
     "address":{
        "type": "text"
     },
     "age":{
        "type": "integer"
     }
     }
   }
}
#插入数据 可以自己生成id //
POST  /clayindex/_doc
{
   "name":"曹操",
   "address":"魏国",
   "age":18

}
POST  /clayindex/_doc
{
   "name":"贾诩",
   "address":"魏国",
   "age":19

}
POST  /clayindex/_doc
{
   "name":"诸葛亮",
   "address":"蜀国",
   "age":37

}
POST  /clayindex/_doc
{
   "name":"关羽",
   "address":"蜀国",
   "age":35

}
POST  /clayindex/_doc
{
   "name":"周瑜",
   "address":["吴国","蜀国"],
   "age":25
}
```

简单查询

```
#查询所有的数据
GET clayindex/_doc/_search
#查询name是曹操的数据
GET clayindex/_doc/_search?q=name:曹操
#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
GET clayindex/_doc/_search?q=address:魏国
```

复杂查询 参数查询

```
# name 在创建类型的时候是keywork  所以没有分词查询
GET clayindex/_doc/_search
{

  "query":{

     "match": {
       "name": "曹操"

     }

  }
}
#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
GET clayindex/_doc/_search
{

  "query":{

     "match": {
       "address": "魏国"
     }
  }
}
#_source 可以设置需要返回需要的字段  select name from ==
GET clayindex/_doc/_search
{

  "query":{

     "match": {
       "address": "魏国"
     }
  },
  "_source":["name"]
}
```

排序

```
GET clayindex/_doc/_search
{

  "query":{

     "match": {
       "address": "魏国"
     }
  },
  "_source":["name"],
```

```
  "sort":[
    {"age":{"order":"desc"}},
    {"name":{"order":"asc"}}
    ]
}
```

分页查询

```
GET clayindex/_doc/_search
{

  "query":{

     "match": {
       "address": "魏国"
     }
  },
  "_source":["name"],
  "sort":[
    {"age":{"order":"desc"}},
    {"name":{"order":"asc"}}
    ],
   "from":0,
   "size":10
}
```

布尔值查询

条件1 and 条件2

must =>and

```
GET clayindex/_doc/_search
{

  "query":{
      "bool": {
        "must": [
          {
     "match": {
       "address": "魏国"
     }

          },{
            "match": {
       "name": "曹操"
     }

          }
        ]

      }
  },
  "_source":["name"],
  "sort":[
    {"age":{"order":"desc"}},
    {"name":{"order":"asc"}}
    ],
   "from":0,
   "size":2
```

```
}
```

条件1 or 条件2

should => or

```
GET clayindex/_doc/_search
{

  "query":{
      "bool": {
        "should": [
          {
     "match": {
       "address": "魏国"
     }

          },{
            "match": {
       "name": "曹操"
     }

          }
        ]

      }
  },
  "_source":["name"],
  "sort":[
    {"age":{"order":"desc"}},
    {"name":{"order":"asc"}}
    ],
   "from":0,
   "size":10
}
```

排除 must_not

```
#查询名字不是曹操的信息
GET clayindex/_doc/_search
{

  "query":{

    "bool": {
      "must_not": [
        {
           "match": {
       "name": "曹操"

     }
        }
      ]
    }

  }
}
```

数据过滤filter

```
gt   大于
gte  大于等于
lt  小于
lte  小于等于!
# 查询名字是曹操，然后年龄大于等于10 小于等于30
GET clayindex/_doc/_search
{

  "query":{

    "bool": {
      "must": [
        {
           "match": {
       "name": "曹操"

     }
        }
      ],
      "filter": {
        "range": {
          "age": {
            "gte": 10,
            "lte": 20
          }
        }
      }
    }

  }
}
```

配备多个条件查询

```
# 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
GET clayindex/_doc/_search
{

  "query":{
    "match":{"address":"吴国 魏国 "}

  }
}
```

精确查询

直接根据倒序索引精确查询

```
match和term的区别
match，会使用分词器解析！注意：先分词然后再去查询
term ，直接查询精确的
#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
PUT /clayindex3
{
   "mappings": {
     "properties": {
     "name":{
       "type": "keyword"
```

```
    },
    "address":{
       "type": "text",
       "analyzer": "standard",
        "search_analyzer": "standard"
    },
     "age":{
       "type": "integer"
    }
    }
  }
}

#分词和搜索分词的类型可以不同
PUT /clayindex4
{
   "mappings": {
     "properties": {
     "name":{
       "type": "keyword"
     },
     "address":{
       "type": "text",
       "analyzer": "ik_max_word",
        "search_analyzer": "standard"
     },
      "age":{
       "type": "integer"
     }
     }
   }
}
#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
#根据魏国模糊查询，包含魏和国的都可以查询
GET /clayindex/_doc/_search
{
  "query":{
    "match": {
      "address":"魏国"
    }
  }

}

#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
GET /clayindex/_doc/_search
{
  "query":{
    "term": {
      "address":"魏国"
    }
  }

}
#keyword 类型不会被分词的
GET _analyze
{
  "analyzer": "keyword",
  "text": "魏国"
}
```

```
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 }
```

```
#查询魏国，并且高亮，给魏和国都高亮了，因为是text 分词了
GET clayindex/_doc/_search
{

  "query":{
  "match": {
    "address": "魏国"
  }
  },
  "highlight":{

    "fields": {
      "address": {}
    }
  }
}
```

自定义高亮的风格

```
GET clayindex/_doc/_search
{

  "query":{
  "match": {
    "name": "曹操"

  }
  },
  "highlight":{

    "pre_tags": "<p class='gaoliang'>",
    "post_tags": "</p>",
    "fields": {
      "name": {}
    }
  }
}
匹配
按照条件匹配
精确匹配
区间范围匹配
匹配字段过滤
多条件查询
高亮查询
```

SQL 查询

```
# 通过sql查询的话，必须不能存在text的类型
POST /sqldbindex/_doc
{
  "name":"a",
  "age":12,
  "address":"中国人民共和国",
  "birth":"2019-01-0"

}
GET /sqldbindex

POST /_xpack/sql?format=txt
```

```
{
  "query":"select * from sqldbindex"

}
# txt,json,csv 等等多种格式表名称就是索引的名称
POST /_xpack/sql?format=txt
{
  "query":"select address,count(distinct name),sum(age) from sqldbindex group by address "
}
#分页
POST /_xpack/sql?format=json
{
  "query":"select address,count(distinct name),sum(age) from sqldbindex group by address limit 2 ",
  "fetch_size":5
}
```

滚动查询

```
#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
#写入数据
POST /scrolldb/_doc
{
  "title":"abc"

}

#根据滚动查询，查询2条，不能超过1m内存
POST /scrolldb/_search?scroll=1m
{
    "size": 2,
    "query": {
        "match" : {
            "title" : "abc"
        }
    }
}
# 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
 POST /_search/scroll
{
    "scroll" : "2m",
    "scroll_id" :
"DXF1ZXJ5QW5kRmV0Y2gBAAAAAAAAA0AWT1F2d0ppdC1TV3EtUUJaNUh4dEJHZw=="
}
```

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

比较的规范的数据

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 myql redis //oracle rid=2

rid=2

//数据槽 //elk

text，keyword类型区别

```
text:如果一个字段的值你需要分词，你就用它
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

string

我之前用的es 中如果有一个字段类型是text，不支持sql;

注重细节细节

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
```

```
倒排索引=反向索引
如果要存储一段内容
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

```
#通过一种手段，在内存中共享了key的前缀和后缀，

用内存中很少的数据，表示出大量信息

1 2 3 4 5
1 3 6 10 15

int  = 4个字节
```

int =3 4

bit 8个数字标识一个字节

0001 =3   0.5个字节

01010100 1个字节
1 3 5

【1 汉字】
【2 汉字】

1 int

最小单位，bit
手毛孔

01010100   0表示没有，1表示有。
1 3 5

01010100010101000101010 0

1 2 5 7 9 11

1 4 6   5 8 9

1 5 11   16 24 33

8:73 227 2   5:30 11 29

1001001   11100011 10   11110 1011 11101
7   8   2   5   4   5

bit[8]   bit[5]

你的思路不在线-- 专一

int16[]
int32 []
int64[]

int16[]

int64[]

我会给你好多数据
int[64] 8个字节

1,2,4,8 ...10000   2的63次方

int64[]

压缩value
为什么是以65535为界限 ,死值
跟2 --

为什么做成65535呢，也是为节省空间

30位

100000

0 65535   65536 131071   131071 ===

如果你面试永远遇不到，OK，今天东西没必要，你不是架构师

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

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

通过很少量的信息，来表示更多的数据--

不说话

65535 ---

小一层

《《《《《--6699-6701   6601-6699--》》》》》

6601-6699

<<< 6601-6630   6631-6699>>>>>>>>

CCCCCC

买阿里云服务的，或者自己用linux，不要用国外的仓库，

用阿里云的仓库

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

100%

大数据高并发里面，

这段时间，可能短暂就不能用了==

```
130
131 iscovery.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
```