## ElasticSearch

### ElasticSearch介绍

在微服务中，随着业务的发展，数据量越来越庞大，传统的MySQL数据库渐渐地难以满足复杂业务的需求，所以在微服务架构系统下，一般都会用到分布式搜索的技术，其中ElasticSearch是最流行的。

ElasticSearch：一款非常强大的开源的搜索引擎，可以帮助我们从海量数据中快速找到需要的内容（关键字搜索、高亮显示、模糊搜索、距离）。

### ELK介绍（elasticsearch、kibana、Logstash、Beats）

ElasticSearch结合kibana、Logstash、Beats，就是我们所说的elestic stack（ELK）。被广泛应用在微服务日志数据收集、分析、实时监控等领域。

#### 日志数据分析：

项目在运行过程中会产生海量的日志信息，这些日志信息可以方便我们去定位项目运行出现的问题，比如系统报错了方便找出错误，因为线上运行的应用不可能打断点，可以将日志数据可视化展示；

#### 实时监控：

项目的运行过程中项目的运行状态也是数据，每一刻都有状态；比如CPU情况、内存情况、访问频率，这些也会被管理以可视化展示出来，能非常清楚知道项目运行情况。

ELK技术栈中，核心是ElasticSearch，负责存储、计算、搜索、分析数据；Logstash、Beats负责数据抓取（当然，数据抓取不一定用Logstash、Beats）；Kibana负责数据可视化（当然，数据可视化不一定用Kibana）

### ElasticSearch底层实现：Lucence

ElasticSearch的底层实现用到了Lucene。Lucene是一个Java语言的搜索引擎类库（简单来说就是jar包），是Apache公司的顶级项目。

#### Lucene优势：

1：易扩展，只是实现了搜索引擎相关核心api，可以基于此做二次开放、定制，实现更高级的功能；

2：高性能（基于倒排索引）。

Lucene的缺点：

1：只限于java语言开发；

2：学习曲线陡峭；

3：不支持水平扩展。

#### ElasticSearch就是基于Lunece做的二次开发，优点是：

1：天生就支持分布式，可水平扩展（就是支持集群，所以支持海量数据和高并发场景）；

2：提供Restful接口，跟语言无关，可被任何语言调用，使用场景更加广泛。

总结：

Lunece是1999年，ElasticSearch是2010年问世，中间相隔十几年，所以，基于Lunece的搜索引擎并不只有ElasticSearch，其中还有比较出名的Solr，只不过ElasticSearch比较优秀。

#### 小总结

什么是ElasticSearch?

一个开源的分布式搜索引擎，可以用来实现搜索、日志统计、分析、系统监控等功能

什么是elastic stack（ELGK）

是以ElasticSearch为核心的技术栈，包括beats、logstash、kibana、elasticsearch

什么是Lunece？

是Apache的开源搜索引擎类库，提供了搜索引擎的核心api

### 正向索引和倒排索引

ElasticSearch底层是基于Lunece来实现的，而Lunece中的核心技术就是倒排索引，倒排索引是与传统数据库，例如MySQL中正向索引去对比得出的名称，可见倒排索引一定和数据库的正向索引有比较大的区别。

#### 正向索引：

传统数据库（如mysql）采用的正向索引，例如常见的很多表中的id创建索引。形成一颗B+树，那么根据id检索的速度就会变得非常快。这种方式的索引就是正向索引，但是如果搜索的字段不是id，而是一个普通字段，因为普通字段一般不会加索引，即便加了索引，但是如果搜索的不是精确的字段值，搜索的是部分字段值，那么就得使用模糊匹配（like），而一但使用了模糊匹配，就算字段有索引也不生效，这种情况下（没有索引），就会采用逐条扫描的方式来判断表中每一行是否满足，不包含则丢弃，包含则把记录放入结果集。此时如果表的数据过多（如1000万条），那么就要扫描1000万次，全表扫描，性能可想而知地差。**正向索引做这种字段局部内容检索的时候效率比较差。**

#### 倒排索引：

文档document：每一条数据就是一个文档，参照数据库表中的一条记录；

词条term：文档按照语义分成的词语（重点是词语），如果是中文就按中文含义分词，如：华为小米充电器，可以分成（华为、小米、充电器、充电、电器），分出来得每个词语就是一个词条。

倒排索引创建的时候会形成一张新的表（倒排索引表），这张表里面有两个字段，一个字段是词条term，一个字段是文档id。也就是说倒排索引在存储是会先把文档中的内容分成词条去存，比如说一个文档中包含标题title（华为小米充电器），要对标题title创建倒排索引，那么就要把标题内容做分词，得到五个词语：华为、小米、充电器、充电、电器，这个时候就把分词得到的每个词条存到表中的term，然后记录对应的文档id。有多少个词语就存多个词条；如果新的文档分词后有想相同的词语，因为词条不能重复，那么就在对应词条的再加上文档的id，以此类推。所以不管是有四个文档，还是有四千万个文档，将来一定都可以分成一个一个的词条，并且这些词条肯定会有大量的重复，但是我们不用重复记录相同词条，只需要记录一个，同时记录重复文档id即可，这样可以确保倒排索引当中词条term字段是绝对不会出现重复的，因此，因为词条的唯一性，我们就可以给词条term创建索引了，将来根据词条查找文档的速度就相当快了。

比如我们搜索“华为手机”，第一步会把“华为手机”进行分词，得到“华为”、“手机”两个词条，然后去倒排索引表中进行查询，因为倒排索引表中根据词条建立了索引，所以根据“华为”、“手机”这两个词条去查询的速度是很快的，然后查找这两个词条对应的文档id，于是就会得到两组文档id，这个时候就会知道包含“华为”，“手机”词条的所有文档了，将来还可以根据文档id出现的频率进行排序，然后根据文档id就可以去正向索引表查询文档，因为正向索引表根据id建立了索引，那么就可以快速地定位到对应的文档。

综上，可以发现，倒排索引的过程其实是经过两次检索，第一次是根据用户输入内容的词条去词条列表中找到对应的文档id；第二次是根据文档id去找文档，虽然是两次，但每一次的检索都是通过索引去找的，所以搜索效率是很高的，是比全表扫描的效率要高得多得多的。同时也能看出来了，倒排索引之所以叫倒排索引，因为在正向索引中是逐行去查找，是先找到文档，看文档是否符合词条要求；而倒排索引是反过来的，是基于词条去创建索引，然后去关联到对应的文档，查找的时候是先去找到对应的词条，再根据词条找到对应的文档，即倒排索引是根据词找文档，而正向是是根据文档匹配词。倒排索引适合、擅长基于文档的部分内容进行搜索，即更复杂的搜索需求，比如浏览器关键字搜索，商城商品搜索，都是复杂的搜索场景。

#### 小总结：

什么是文档和词条：

每一条数据就是一个文档；

对文档中的内容分词，得到的词语就是词条

什么是正向索引：

基于文档id创建索引，查询词条时必须先找到文档，而后判断是否包含词条

什么是倒排索引：

对文档内容分词，对词条创建索引，并记录词条所在文档的信息，查询时先根据词条查询到文档id，而后获取到文档

### ElasticSearch概念：

#### 文档Document：

ElasticSearch是面向文档存储的，可以是数据库中的一条商品数据，一个订单信息。文档数据会被序列化为JSON格式后存储在elasticsearch中。类似数据库中表的行（Row），即一条记录。

#### 字段Field：

就是Json文档中的字段，类型数据库中表的列（Column）

#### 索引Index：

相同类型的文档的集合（可视为数据库中的一张表Table）。

#### 隐射Mapping：

索引中文档的字段约束信息（即字段名字叫什么、数据类型是什么。。），类似数据库中表的结构约束Schema。

#### DSL：

dsl是elasticsearch提供的**JSON风格的请求语句**，用来操作elasticsearch，实现CRUD，所以dsl和sql是一样的，就是一种语法，这种语法就是用来描述要干的事，在Mysql中，sql写好了通过Connection把语句发给MySQL，MySQL就能进行相应的处理，在ElasticSearch中，写好DSL后，要怎么发出去呢，是通过基于HTTP请求发出去的，因为ES对外暴露的Restful的接口。



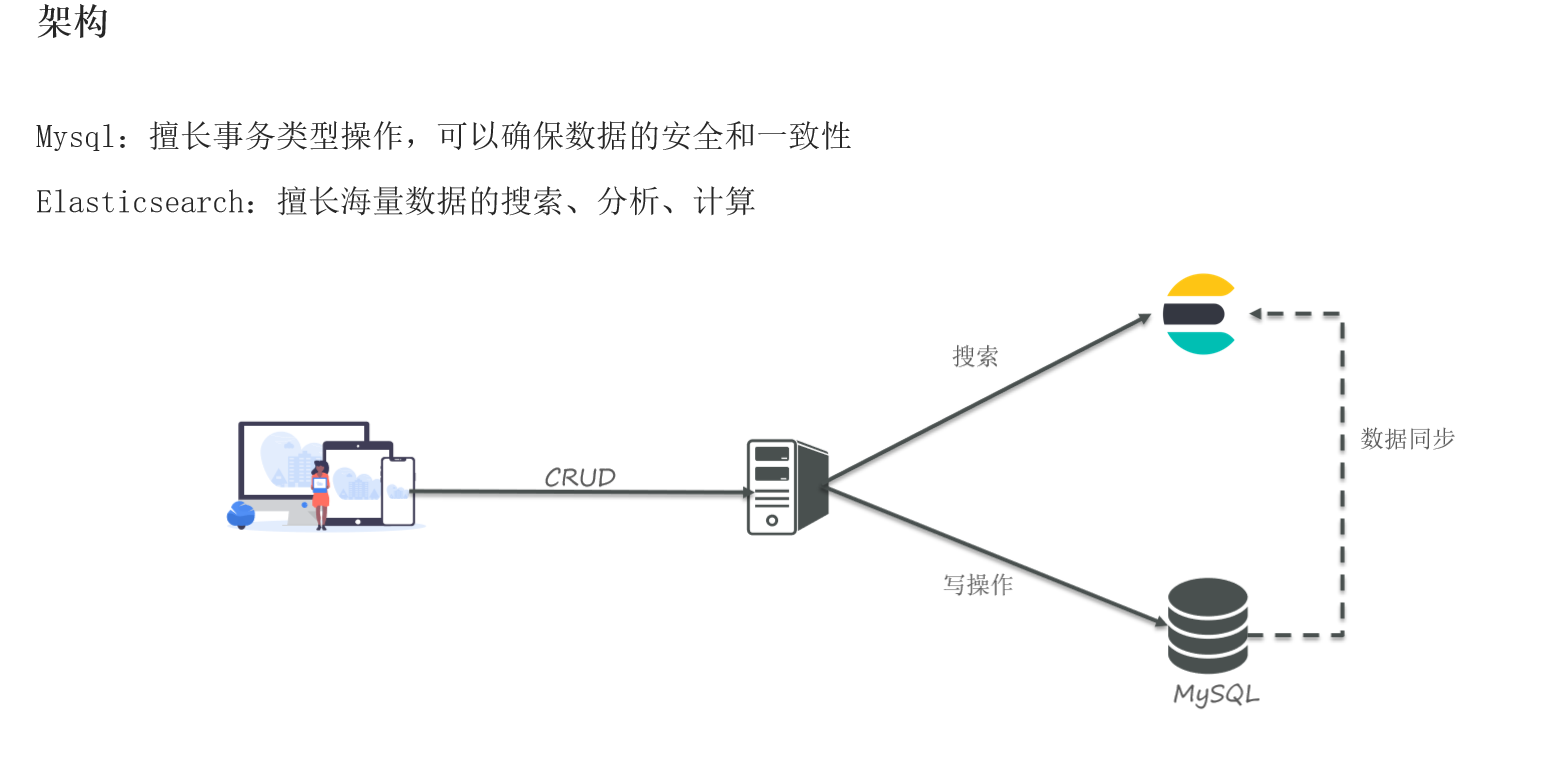
所以将来操作ES的时候，和操作数据库相同的道理，第一件事是先去创建索引库，即索引Index，因为创建索引，那么就要指定约束映射Mapping，用来约束索引中的字段“长什么样子”，然后添加文档Document。然后是文档的CRUD。在ES中，写好DSL通过HTTP的请求发送过去给ES。

Mysql和ES怎么选？

Mysql：擅长事务类型操作，因为事务有ACID原则，可以确保数据的安全和一致性；

ES：擅长海量数据的搜索、分析、计算。

两者是互补、不是替代关系！



#### 小总结：

文档：一条数据就是一个文档，es中是Json格式

字段：json文档中的字段

索引：同类文档的集合

银色：索引中文档的约束，比如字段名称、类型

### 安装ElasticSearch、kibana

为什么要安装kibana，是因为kibana中提供了dev tools工具，可以让我们非常方便去编写ElasticSearch中的DSL语句，从而去操作ElasticSearch，这就是我们要安装kibana的原因

#### 单点安装ElasticSearch

##### 1：因为我们还需要部署kibana容器，因此需要让es和kibana容器互联。这里先创建一个网络：

docker network create es-net

##### 2：加载镜像，上传elasticsearch镜像的tar包到虚拟机中，然后运行命令加载即可

docker load -i es.tar

##### 3：运行docker命令，部署单点es：

docker run -d \

--name es \

-e "ES\_JAVA\_OPTS=-Xms512m -Xmx512m" \

-e "discovery.type=single-node" \

-v es-data:/usr/share/elasticsearch/data \

-v es-plugins:/usr/share/elasticsearch/plugins \

--privileged \

--network es-net \

-p 9200:9200 \

-p 9300:9300 \

elasticsearch:7.12.1

命令解释：

-e "cluster.name=es-docker-cluster"：设置集群名称

-e "http.host=0.0.0.0"：监听的地址，可以外网访问

-e "ES\_JAVA\_OPTS=-Xms512m -Xmx512m"：内存大小，默认1G，不能低于512M

-e "discovery.type=single-node"：非集群模式，即es是单点模式运行

-v es-data:/usr/share/elasticsearch/data：挂载逻辑卷，绑定es的数据目录

-v es-logs:/usr/share/elasticsearch/logs：挂载逻辑卷，绑定es的日志目录

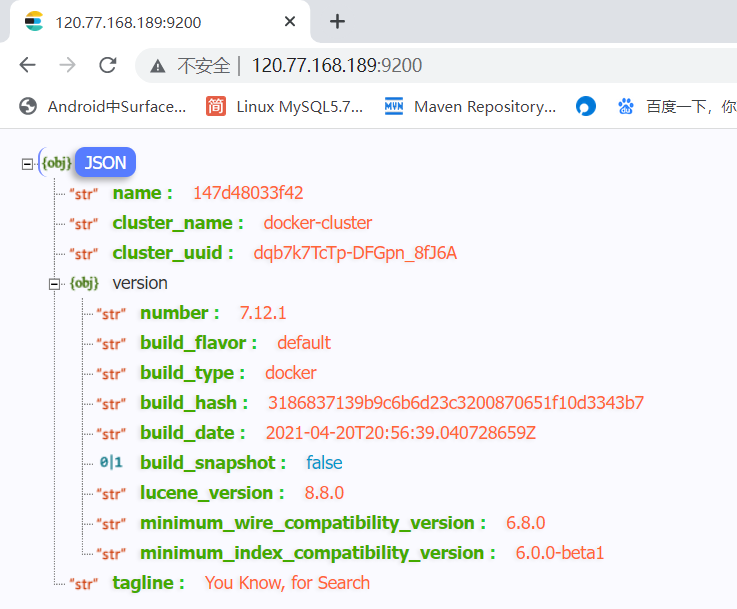
-v es-plugins:/usr/share/elasticsearch/plugins：挂载逻辑卷，绑定es的插件目录

--privileged：授予逻辑卷访问权

--network es-net ：加入一个名为es-net的网络中

-p 9200:9200：端口映射配置

##### 4：在浏览器中输入：http://120.77.168.189:9200 即可看到elasticsearch的响应结果：



#### 部署kibana

kibana的版本一定要与elasticSearch的版本保持一致！

##### 1：加载镜像，上传kibana镜像的tar包到虚拟机中，然后运行命令加载即可

Docker load -i kibana.tar

##### 2：运行docker命令，部署kibana

docker run -d \

--name kibana \

-e ELASTICSEARCH\_HOSTS=http://es:9200 \

--network=es-net \

-p 5601:5601 \

kibana:7.12.1

命令解释：

--network es-net ：

加入一个名为es-net的网络中，与elasticsearch在同一个网络中，所以kibana容器和elasticsearch可以用容器名互联，kibana可以帮助我们去操作es

-e ELASTICSEARCH\_HOSTS=http://es:9200"：

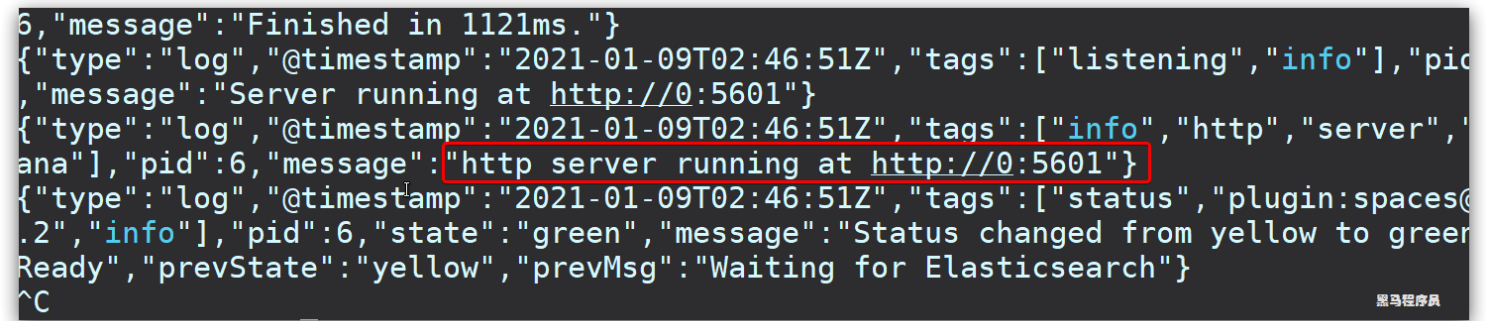
设置elasticsearch的地址，因为kibana已经与elasticsearch在一个网络，因此可以用容器名直接访问elasticsearch

-p 5601:5601：kibana端口映射配置

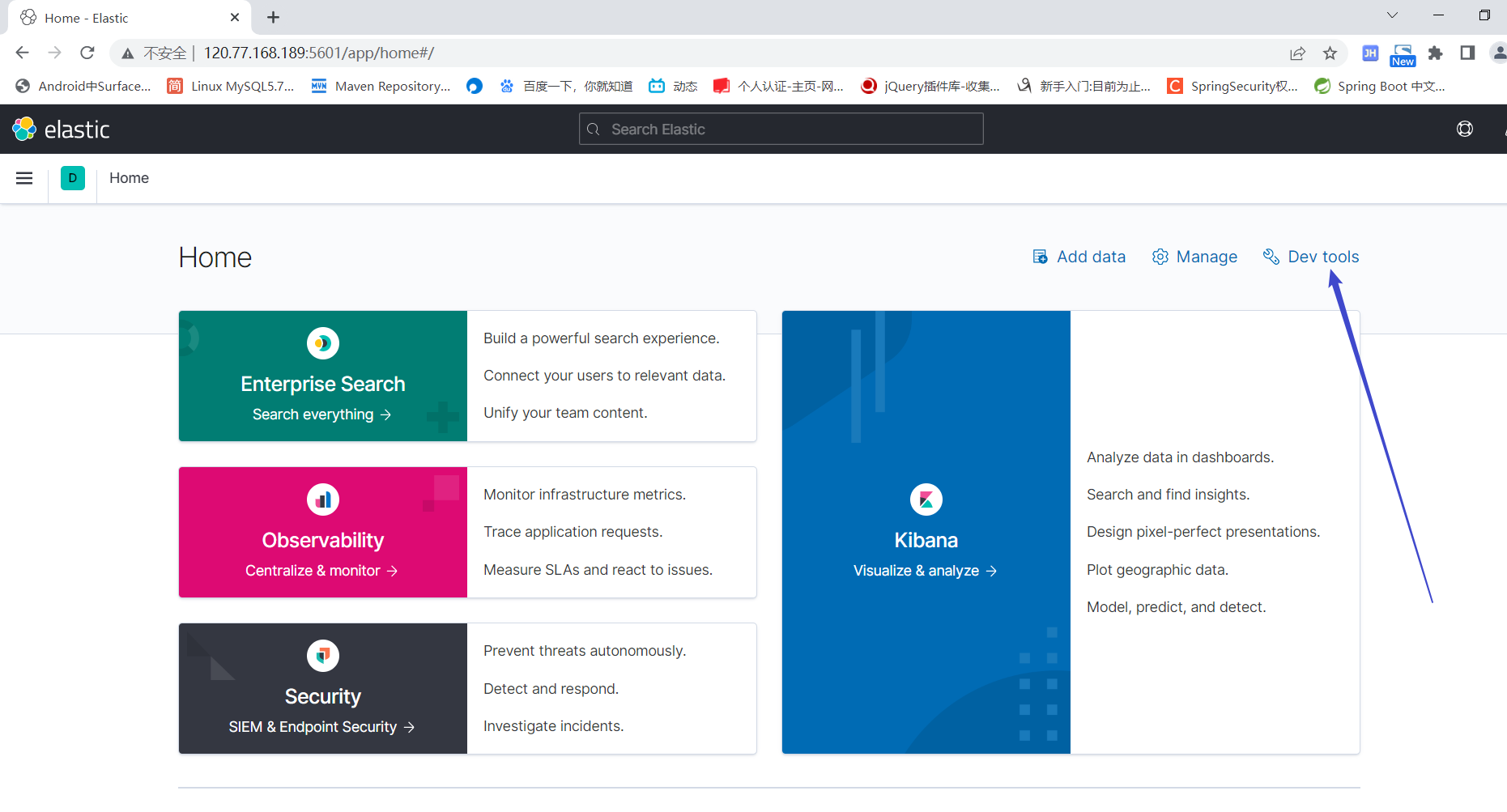
##### 3：kibana启动一般比较慢，需要多等待一会，可以通过命令：

docker logs -f kibana

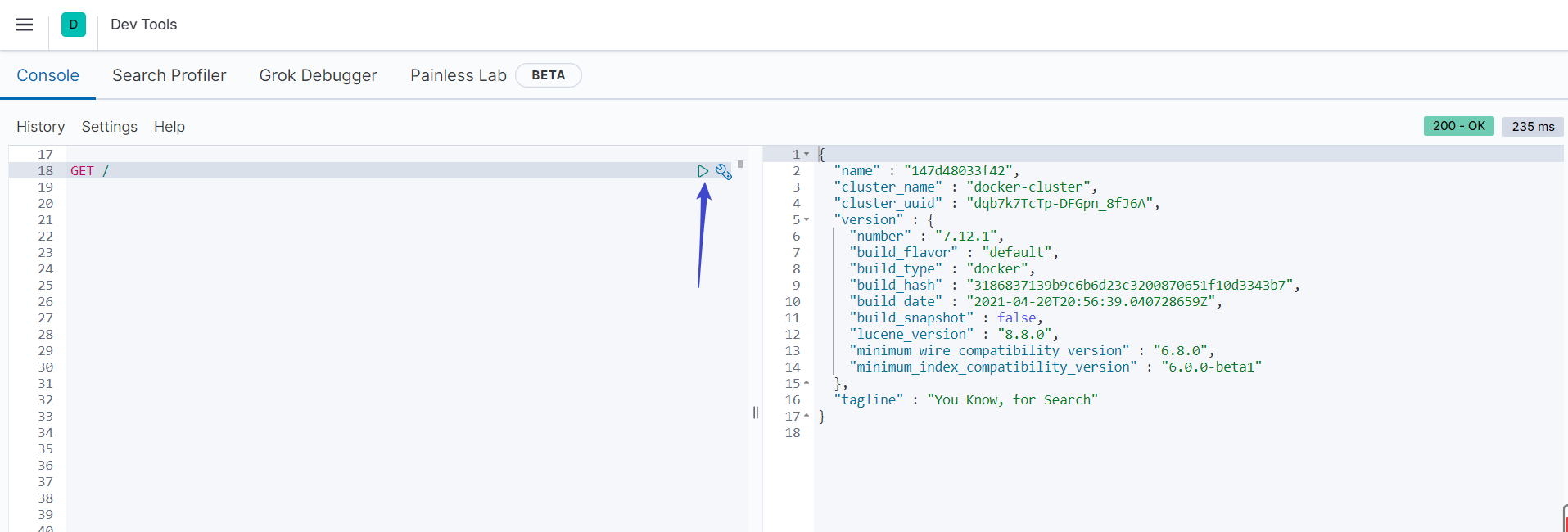
查看日志，看到有类似一下，说明运行kibana成功



##### 4：访问kibana，在浏览器输入地址访问：http://120.77.168.189:5601，即可看到结果



##### 在kibana模拟http://120.77.168.189:9200/：**GET /**



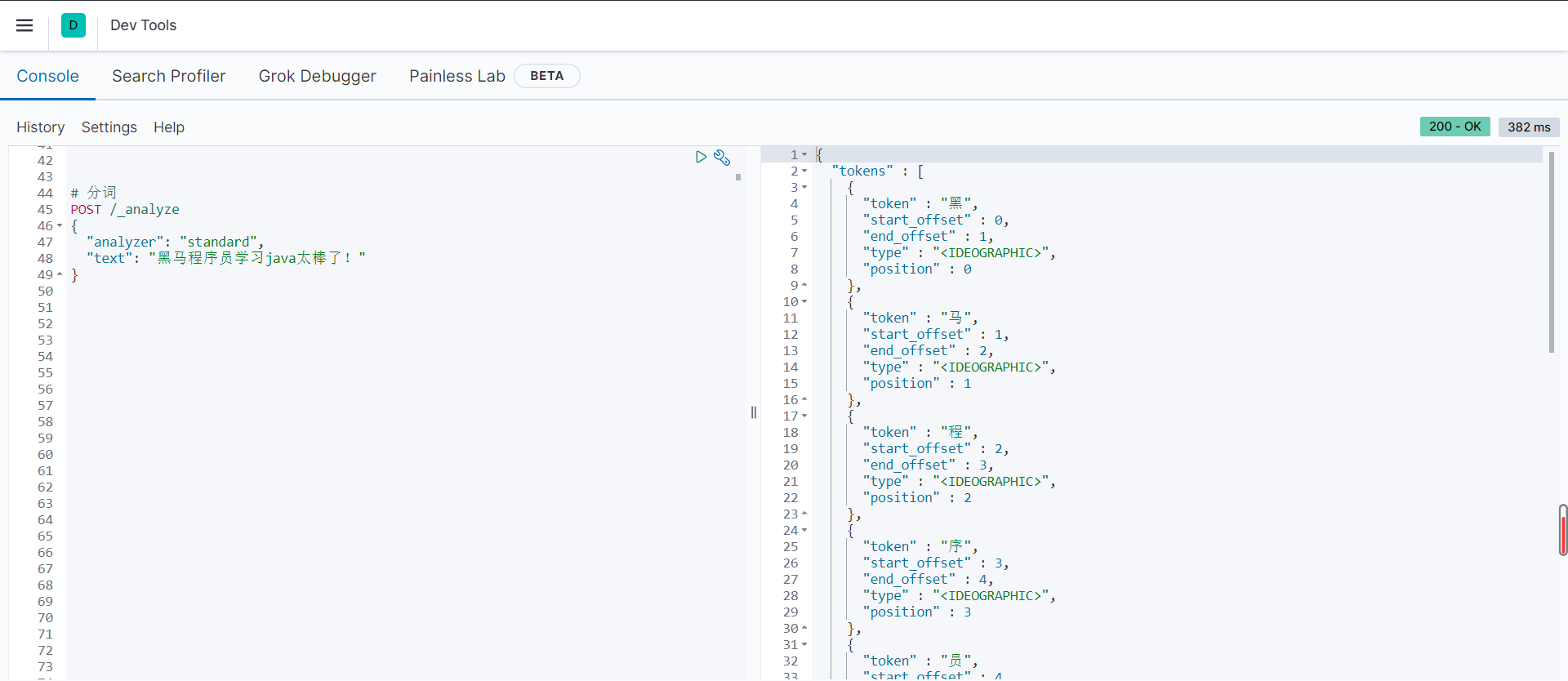
#### 安装分词器

ElasticSearch在创建倒排索引时需要对文档分词；在搜索时，需要对用户输入内容分词。分词就需要对中文、英文的语义进行分析，按照词语含义进行拆分，这种拆分是比较复杂的，需要特殊的算法去支持，在ES中但默认的分词规则对中文处理并不友好。

可以在kibana的DevTools中测试：



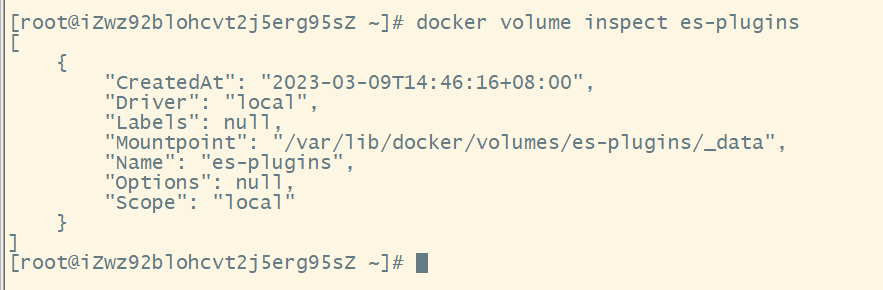
从下图结果可以看到，ES的默认分词器对中文会逐字分词！显然不符合，所以要对中文分词，必须替换默认的分词器：IK分词器。



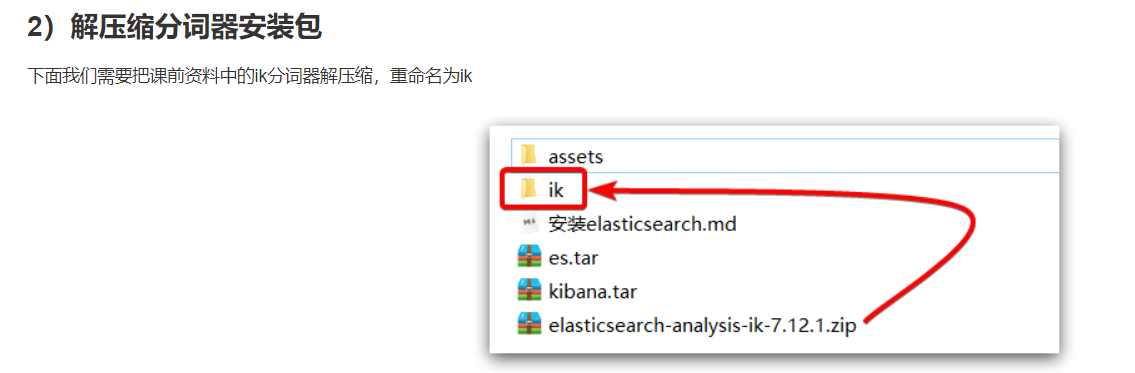
##### 离线安装IK分词器

##### 1：查看插件数据卷目录：docker volume inspect es-plugins

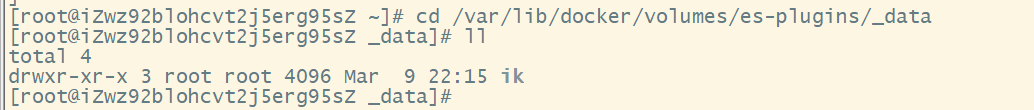
装插件需要知道elasticsearch的plugins目录位置，而我们用了数据卷挂载，因此需要查看elasticsearch的数据卷目录，通过下面命令查看:



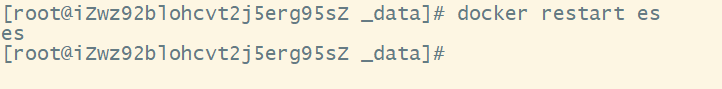
##### 2：解压IK分词器安装包



##### 3：用xftp将解压后的ik分词器文件夹上传到es-plugins插件数据卷中



##### 4：重启es容器：docker restart es



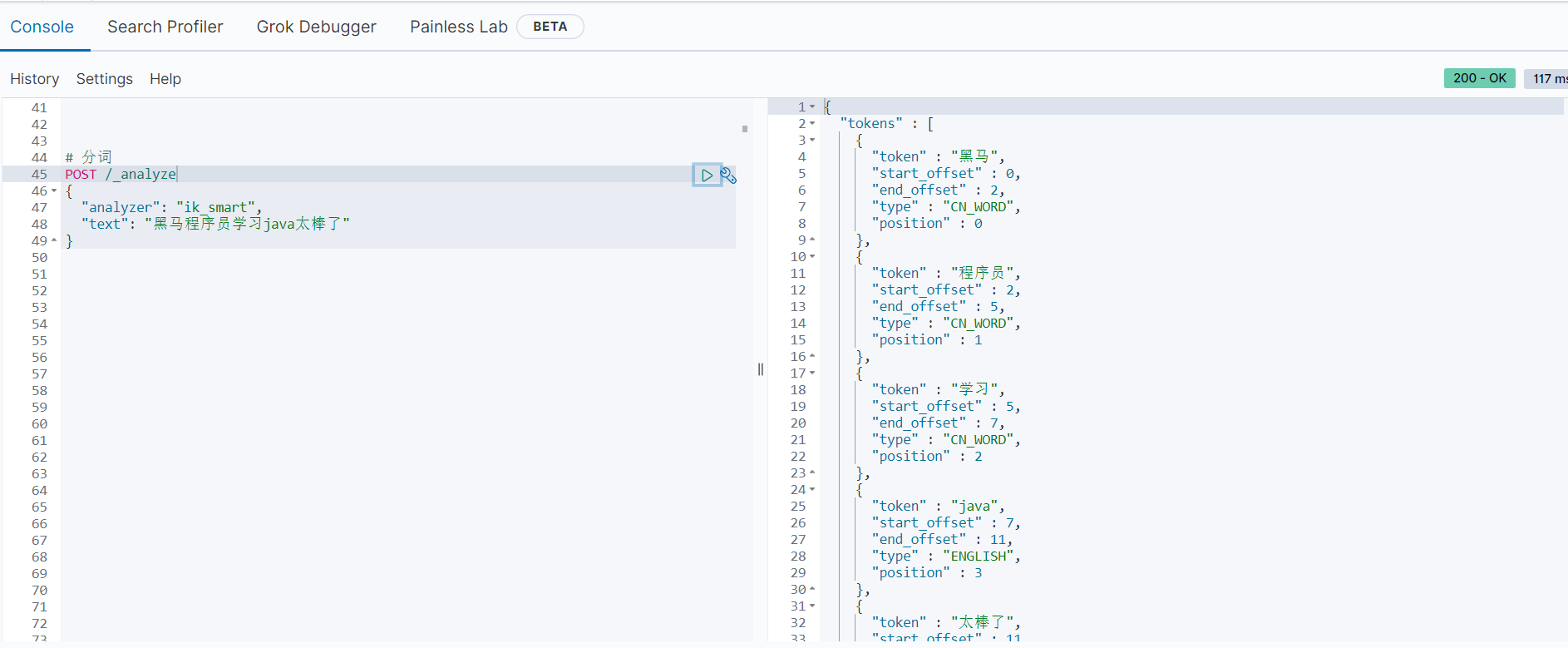
##### 5：查看是否安装IK分词器成功：查看es日志，docker logs -f es

##### 6：测试IK分词器

IK分词器包含两种模式：

ik\_smart：最少切分，粗粒度，占用内存少，搜索命中概率较低

ik\_max\_word：最细切分，细粒度，占用内存多，搜索命中概率高



# GET表示请求方式；\_search表示请求资源 下面的json表示请求数据

GET /\_search

{

"query": {

"match\_all": {}

}

}

GET \_search

{

"query": {

"match\_all": {}

}

}

#模拟

GET /

GET /

# 分词

# ik\_smert分词器 粗粒度

POST /\_analyze

{

"analyzer": "ik\_smart",

"text": "黑马程序员学习java太棒了"

}

# ik\_max\_word 细粒度

POST \_analyze

{

"analyzer": "ik\_max\_word",

"text": "黑马程序员学习java太棒了"

}

# 默认分词器

POST /\_analyze

{

"analyzer": "standard",

"text": "黑马程序员学习java太棒了！"

}

POST /\_analyze

{

"analyzer": "ik\_max\_word",

"text": "今天的天气好晴朗啊，处处good风光！广东粤安的员工黄远辉说的哦，好想白嫖啊"

}

POST /\_analyze

{

"analyzer": "ik\_smart",

"text": "哈登，我想对你说：你是我的神！"

}

# 创建索引库、映射mapping

PUT /heima

{

"mappings": {

"properties": {

"info": {

"type": "text",

"analyzer": "ik\_smart"

},

"email": {

"type": "keyword",

"index": false

},

"name": {

"type": "object",

"properties": {

"firstName": {

"type": "keyword"

},

"lastName": {

"type": "keyword"

}

}

}

}

}

}

PUT /student

{

"mappings": {

"properties": {

"info": {

"type": "text",

"analyzer": "ik\_smart"

},

"email": {

"type": "keyword",

"index": false

},

"grade": {

"type": "integer"

},

"age": {

"type": "integer"

},

"gender": {

"type": "keyword"

},

"address": {

"type": "text",

"analyzer": "ik\_smart"

},

"name": {

"type": "keyword"

}

}

}

}

GET /student

# 查询索引库

GET /heima

# 修改索引库（只支持增加属性）

PUT /heima/\_mapping

{

"properties": {

"age": {

"type": "integer",

"index": false

}

}

}

DELETE /student

# 删除索引库

DELETE /heima

# 新增文档 (注意加id，否则es会默认生成id)

POST /heima/\_doc/1

{

"info": "黑马程序员Java学员信息",

"email": "2278181701@qq.com",

"name": {

"firstName": "远辉",

"lastName": "黄"

},

"age": 28

}

# 查询文档

GET /heima/\_doc/1

# 修改文档：全量修改，会删除旧文档，添加新文档

PUT /heima/\_doc/1

{

"info": "黑马程序员的Java学员信息",

"email": "2278181701@qq.com",

"name": {

"firstName": "远辉",

"lastName": "黄"

},

"age": 28

}

# 修改文档：局部修改（增量修改，修改指定字段值）

POST /heima/\_update/1

{

"doc": {

"age": 29,

"name": {

"firstName": "YuanHui",

"lastName": "Huang"

}

}

}

# 删除文档

DELETE /heima/\_doc/1

# 创建案例项目映射

PUT /hotel

{

"mappings": {

"properties": {

"id": {

"type": "keyword"

},

"name": {

"type": "text",

"analyzer": "ik\_max\_word",

"copy\_to": "all"

},

"address": {

"type": "keyword",

"index": false

},

"price": {

"type": "integer"

},

"sorce": {

"type": "integer"

},

"brand": {

"type": "keyword",

"copy\_to": "all"

},

"city": {

"type": "keyword"

},

"starName": {

"type": "keyword"

},

"business": {

"type": "keyword",

"copy\_to": "all"

},

"location": {

"type": "geo\_point"

},

"pic": {

"type": "keyword",

"index": false

},

"all": {

"type": "text",

"analyzer": "ik\_max\_word"

}

}

}

}

DELETE /hotel

GET /hotel

GET /hotel/\_doc/61083

# 查询所有，默认一次只会查20条

GET /hotel/\_search

# 1：查询所有，匹配所有

GET /hotel/\_search

{

"query": {

"match\_all": {

}

}

}

GET /hotel/\_search

{

"query": {

"match\_all": {}

}

}

# 2：全文检索查询：利用分词器对用户输入内容分词，然后去倒排索引库中匹配，常用于搜索框搜索

# 2.1全文检索查询 match\_query 单字段查询

GET /hotel/\_search

{

"query": {

"match": {

"all": "如家外滩"

}

}

}

GET /hotel/\_search

{

"query": {

"match": {

"all": "如家外滩"

}

}

}

#2.2 multi\_match\_query 全文检索查询：多匹配查询 多字段查询 此时和上面结果一致，推荐用上面的，因为参与搜索的字段越多，效率越低

GET /hotel/\_search

{

"query": {

"multi\_match": {

"query": "如家外滩",

"fields": ["brand", "name", "business"]

}

}

}

GET /hotel/\_search

{

"query": {

"multi\_match": {

"query": "如家外滩",

"fields": ["brand", "name", "business"]

}

}

}

# 3：精确查询 查不分词的字段 如：keyword bool 数值 日期等

# 3.1：精确匹配 词条查询

GET /hotel/\_search

{

"query": {

"term": {

"city": {

"value": "上海"

}

}

}

}

GET /hotel/\_search

{

"query": {

"term": {

"brand": {

"value": "如家"

}

}

}

}

# 3.2：范围查询

GET /hotel/\_search

{

"query": {

"range": {

"price": {

"gte": 1000,

"lte": 3000

}

}

}

}

GET /hotel/\_search

{

"query": {

"range": {

"score": {

"gte": 40,

"lte": 45

}

}

}

}

# 4：地理查询

# 4.1: 中心点附近查询 distance查询

GET /hotel/\_search

{

"query": {

"geo\_distance": {

"distance": "10km",

"location": "31.21,121.5"

}

}

}

# 4.2：区域范围查询

GET /hotel/\_search

{

"query": {

"geo\_bounding\_box": {

"location": {

"top\_left": {

"lat": 31.1,

"lon": 121.5

},

"bottom\_right": "30.9,121.7"

}

}

}

}

# 5：复合查询：上面的都是简单查询，复合查询可以将其他简单查询组合起来，实现更复杂的搜索逻辑

# 5.1:function\_socre 算分排序 控制排序 百度竞价

GET /hotel/\_search

{

"query": {

"function\_score": {

"query": {

"match": {

"all": "外滩"

}

},

"functions": [

{

"filter": {

"term": {

"brand": "如家"

}

},

"weight": 2

}

],

"boost\_mode": "sum"

}

}

}

# 5.2：复合查询（布尔查询），一个或者多个子查询的组合 参与算分：must should 不参与算分：moust\_not filter。参与算分条件越多，性能越差

GET /hotel/\_search

{

"query": {

"bool": {

"must": [

{

"term": {

"city": {

"value": "上海"

}

}

}

],

"should": [

{"term": {

"brand": {

"value": "皇冠假日"

}

}},

{"term": {

"brand": {

"value": "华美达"

}

}}

],

"must\_not": [

{

"range": {

"price": {

"lte": 500

}

}

}

],

"filter": [

{

"range": {

"score": {

"gte": 45

}

}

}

]

}

}

}

GET /hotel/\_search

{

"query": {

"bool": {

"must": [

{

"match": {

"name": "如家"

}

}

],

"must\_not": [

{

"range": {

"price": {

"gt": 400

}

}

}

],

"filter": [

{

"geo\_distance": {

"distance": "10km",

"location": {

"lat": 31.21,

"lon": 121.5

}

}

}

]

}

}

}

# 6：搜索结果出了：排序 分页 高亮

# 6.1.1：排序 制定了排序，es会放弃默认算分

GET /hotel/\_search

{

"query": {

"match\_all": {}

},

"sort": [

{

"score": {

"order": "desc"

},

"price": "desc"

}

]

}

GET /hotel/\_search

{

"query": {

"match\_all": {}

},

"sort": [

{

"score": {

"order": "desc"

}

},

{

"price": {

"order": "desc"

}

}

]

}

# 6.1.2 地理排序

GET /hotel/\_search

{

"query": {

"match\_all": {}

},

"sort": [

{

"price": "asc"

},

{

"\_geo\_distance": {

"location": {

"lat": 31.034661,

"lon": 121.612282

},

"order": "asc",

"unit": "km"

}

}

]

}

GET /hotel/\_search

{

"query": {

"match\_all": {}

},

"sort": [

{

"\_geo\_distance": {

"location": "31.034661,121.612282",

"order": "asc",

"unit": "km"

}

}

]

}

# 6.3：分页查询

GET /hotel/\_search

{

"query": {"match\_all": {}},

"sort": [

{

"price": {

"order": "asc"

}

}

],

"from": 3,

"size": 3

}

GET /hotel/\_search

{

"query": {

"match": {

"brand": "如家"

}

},

"sort": [

{

"score": "desc"

}

],

"from": 0,

"size": 5

}

#3：高亮，默认情况下，es搜索字段必须和高亮字段一致，否则不会高亮

GET /hotel/\_search

{

"query": {

"match": {

"all": "如家"

}

},

"highlight": {

"fields": {

"name": {

"require\_field\_match": "false"

}

}

}

}

GET /hotel/\_search

{

"query": {

"match": {

"all": "外滩如家"

}

},

"highlight": {

"fields": {

"brand": {

"require\_field\_match": "false"

},

"name": {

"require\_field\_match": "false"

}

}

}

}

# 增加isAD字段 此方法会增加字段isAD

POST /hotel/\_update/2056126831

{

"doc": {

"isAD": true

}

}

POST /hotel/\_update/1989806195

{

"doc": {

"isAD": true

}

}

POST /hotel/\_update/2056105938

{

"doc": {

"isAD": true

}

}

# 桶聚合

GET /hotel/\_search

{

"size": 0,

"aggs": {

"brandAgg": {

"terms": {

"field": "brand",

"size": 20

}

},

"cityAgg": {

"terms": {

"field": "city",

"size": 20

}

},

"starAgg": {

"terms": {

"field": "starName",

"size": 20

}

}

}

}

# 桶聚合 自定义排序规则

GET /hotel/\_search

{

"size": 0,

"aggs": {

"brandAgg": {

"terms": {

"field": "brand",

"size": 20,

"order": {

"\_count": "asc"

}

}

}

}

}

# 桶聚合 限定聚合范围

GET /hotel/\_search

{

"query": {

"range": {

"price": {

"lte": 200

}

}

},

"size": 0,

"aggs": {

"brandAgg": {

"terms": {

"field": "brand",

"size": 20,

"order": {

"\_count": "asc"

}

}

}

}

}

# 度量聚合 (此处在brands聚合内部使用子聚合，就是分组后对每组分别计算)

GET /hotel/\_search

{

"size": 0,

"aggs": {

"brandAgg": {

"terms": {

"field": "brand",

"size": 20

},

"aggs": {

"score\_stats": {

"stats": {

"field": "score"

}

}

}

}

}

}

# 度量聚合 对度量聚合结果排序

GET /hotel/\_search

{

"size": 0,

"aggs": {

"brandAgg": {

"terms": {

"field": "brand",

"size": 20,

"order": {

"scoreAgg.avg": "desc"

}

},

"aggs": {

"scoreAgg": {

"stats": {

"field": "score"

}

}

}

}

}

}

# 测试py分词器

POST /test/\_analyze

{

"text": ["如家酒店还不错！"],

"analyzer": "my\_analyzer"

}

# 自定义拼音分词器

PUT /test

{

"settings": {

"analysis": {

"analyzer": {

"my\_analyzer": {

"tokenizer": "ik\_max\_word",

"filter": "py"

}

},

"filter": {

"py": {

"type": "pinyin",

"keep\_full\_pinyin": false,

"keep\_joined\_full\_pinyin": true,

"keep\_original": true,

"limit\_first\_letter\_length": 16,

"remove\_duplicated\_term": true,

"none\_chinese\_pinyin\_tokenize": false

}

}

}

},

"mappings": {

"properties": {

"name": {

"type": "text",

"analyzer": "my\_analyzer",

"search\_analyzer": "ik\_smart"

}

}

}

}

DELETE /test

POST /test/\_doc/1

{

"id": 1,

"name": "狮子"

}

POST /test/\_doc/2

{

"id": 2,

"name": "虱子"

}

GET /test/\_search

{

"query": {

"match": {

"name": "shizi"

}

}

}

# 自动补全的索引库

PUT /test2

{

"mappings": {

"properties": {

"title":{

"type": "completion"

}

}

}

}

GET /test2

# 示例数据

POST test2/\_doc

{

"title": ["Sony", "WH-1000XM3"]

}

POST test2/\_doc

{

"title": ["SK-II", "PITERA"]

}

POST test2/\_doc

{

"title": ["Nintendo", "switch"]

}

# 自动补全查询

GET /test2/\_search

{

"suggest": {

"titleSuggest": {

"text": "sw",

"completion": {

"field": "title",

"skip\_duplicates": true,

"size": 10

}

}

}

}