**elasticsearch学习文档**

# 网址

官网：<https://www.elastic.co/cn/>

<https://www.elastic.co/cn/products/elasticsearch>

<https://www.elastic.co/cn/products/kibana>

中文社区：<https://elasticsearch.cn/>

Es教程汇总：<https://blog.csdn.net/gwd1154978352/article/details/82781731>

慕课网：<https://www.imooc.com/learn/889>

# 常用

## es：9200

### 启动

su elsearch

cd /usr/elasticsearch/elasticsearch-6.4.0/bin

sh elasticsearch -d

### 访问

<http://192.168.202.132:9200/>

## head 9100

### 启动

nohup npm run start &

### 访问

<http://192.168.202.132:9100/>

## Kibana

### 启动

cd /usr/elasticsearch /kibana-6.4.0-linux-x86\_64/bin

nohup ./kibana &

### 访问

[http://192.168.202.132:5601](http://192.168.202.132:5601/)

# 安装

<https://blog.csdn.net/gwd1154978352/article/details/82666891>

## 基础环境

1.系统版本：Centos 7

2.jdk版本：java version "1.8.0\_181"（该版本的elasticsearch必须使用jdk8及以上版本）

3.安装版本为ElasticSearch6.4.0

## 下载

<https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.4.0.tar.gz>

## 解压并安装

(1)将下载的文件放在服务器如下目录:/usr/elasticsearch/

(2)解压

tar -zxvf elasticsearch-6.4.0.tar.gz

(3)创建日志文件夹

# mkdir -p /data/logs/elasticsearch

# mkdir -p /data/elasticsearch/{data,work,plugins,scripts}

(4)创建一个非root用户

ElasticSearch有个比较特殊的地方就是不能用root权限来运行，所以我们这边需要新建一个用户以及赋予对应权限。

a.新建一个elsearch用户组

groupadd elsearch

b.新建用户elsearch，并让他加入elsearch组

useradd elsearch -g elsearch -p elsearch

c.赋予新用户对elasticsearch-6.4.0和/data/\*的操作权限

chown -R elsearch:elsearch elasticsearch-6.4.0

chown -R elsearch:elsearch /data/\*

d.切换用户

su elsearch

(5)修改配置文件（暂时可以不修改）

因为上面准备独立出日志等相关数据了，所以这边需要修改配置文件，添加如下配置：

network.host: 0.0.0.0 //监听访问地址为任意网段，也可以按自己的要求要设置对应的网段

path.data: /data/elasticsearch/data

path.logs: /data/logs/elasticsearch

#如果没有对应的插件，那么下面两个就不用配置，否则会报错

#path.plugins: /data/elasticsearch/plugins

#path.scripts: /data/elasticsearch/scripts

也可以通过如下配置来修改端口：

http.port: 9200

(6)启动任务

cd /usr/elasticsearch/elasticsearch-6.4.0/bin/

sh elasticsearch 或者用sh elasticsearch -d来后台启动

(7)验证是否运行

下面两者需同时满足

a. 输入如下脚本，有对应服务出现

ps -ef|grep elasticsearch

b.输入http//:ip:9200出现下图所示

{

"name" : "F6U\_sFR",

"cluster\_name" : "elasticsearch",

"cluster\_uuid" : "dTfp6ypnTQW-45WbwqSGTQ",

"version" : {

"number" : "6.4.0",

"build\_flavor" : "default",

"build\_type" : "tar",

"build\_hash" : "595516e",

"build\_date" : "2018-08-17T23:18:47.308994Z",

"build\_snapshot" : false,

"lucene\_version" : "7.4.0",

"minimum\_wire\_compatibility\_version" : "5.6.0",

"minimum\_index\_compatibility\_version" : "5.0.0"

},

"tagline" : "You Know, for Search"

}

## 关闭elasticsearch

需要对ES节点进行重新启动或正常关机的时候，有三种方法可以关闭ES：

1.在控制台中,使用CTRL+C组合键.

2.通过发送TERM信号终止服务器进程.

3.使用REST APIcurl -XPOST 'http://localhost:9200/\_shutdown'

## 可能异常

1.

org.elasticsearch.bootstrap.StartupException: java.lang.RuntimeException: can not run elasticsearch as root

    at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:140) ~[elasticsearch-6.4.0.jar:6.4.0]

    at org.elasticsearch.bootstrap.Elasticsearch.execute(Elasticsearch.java:127) ~[elasticsearch-6.4.0.jar:6.4.0]

    at org.elasticsearch.cli.EnvironmentAwareCommand.execute(EnvironmentAwareCommand.java:86) ~[elasticsearch-6.4.0.jar:6.4.0]

    at org.elasticsearch.cli.Command.mainWithoutErrorHandling(Command.java:124) ~[elasticsearch-cli-6.4.0.jar:6.4.0]

    at org.elasticsearch.cli.Command.main(Command.java:90) ~[elasticsearch-cli-6.4.0.jar:6.4.0]

    at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:93) ~[elasticsearch-6.4.0.jar:6.4.0]

    at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:86) ~[elasticsearch-6.4.0.jar:6.4.0]

Caused by: java.lang.RuntimeException: can not run elasticsearch as root

    at org.elasticsearch.bootstrap.Bootstrap.initializeNatives(Bootstrap.java:104) ~[elasticsearch-6.4.0.jar:6.4.0]

    at org.elasticsearch.bootstrap.Bootstrap.setup(Bootstrap.java:171) ~[elasticsearch-6.4.0.jar:6.4.0]

    at org.elasticsearch.bootstrap.Bootstrap.init(Bootstrap.java:326) ~[elasticsearch-6.4.0.jar:6.4.0]

    at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:136) ~[elasticsearch-6.4.0.jar:6.4.0]

    ... 6 more

解决方案：按照上面所述，新增用户，不要用root用户启动服务即可

2.

max virtual memory areas vm.max\_map\_count [65530] is too low, increase to at least [262144]

解决方案：

a.切换到root用户，修改sysctl.conf配置文件

vi /etc/sysctl.conf

b.添加如下配置文件

vm.max\_map\_count=655360

c.执行如下脚本后，重启elasticsearch即可

# 安装Head插件

<https://blog.csdn.net/gwd1154978352/article/details/82693174>

## 基础环境

1.已安装ElasticSearch

2.需要安装git,理论上也可以通过本地下载然后传输到相应服务器上

3.需要安装nodejs，运行elasticsearch-head会用到grunt，而grunt需要npm包管理器，所以nodejs是必须要安装的。

说明：elasticsearch-head是一个elasticsearch的集群管理、数据可视化、增删查改、查询语句可视化的工具，它是完全由html5编写的独立网页程序。elasticsearch5.0之后，elasticsearch-head不做为插件放在其plugins目录下了，所以可以单独部署一台服务器，没必要和elasticsearch部署在同一台机器，这就需要在下面的文章中配置对应的跨域访问。

## 下载并安装

root权限下执行如下操作

yum install epel-release

yum install nodejs npm

cd /usr/elasticsearch/

git clone git://github.com/mobz/elasticsearch-head.git

安装依赖包

cd /usr/elasticsearch/elasticsearch-head

cnpm install //执行后会生成node\_modules文件夹

如果遇到异常cnpm不是内部或外部命令 cnpm: command not found，则运行如下脚本,使用淘宝镜像包

npm install cnpm -g --registry=https://registry.npm.taobao.org

修改配置文件

1.修改Gruntfile.js

cd /usr/elasticsearch/elasticsearch-head

vi Gruntfile.js

在该文件中添加如下，务必注意不要漏了添加“，”号，这边的hostname:’\*’，表示允许所有IP可以访问,此处也可以修改端口号

server: {

options: {

hostname: '\*',

port: 9100,

base: '.',

keepalive: true

}

}

2.修改elasticsearch-head默认连接地址

cd /usr/elasticsearch/elasticsearch-head/\_site

vi app.js

做如下修改，将ip地址修改为对应的服务器的ip地址

将localhost修改为ESdeIP地址

修改前：this.base\_uri = this.config.base\_uri;

修改后： this.base\_uri = this.config.base\_uri || this.prefs.get("app-base\_uri") || "http://you ip address:9200";

这边有个问题就是这个文件的内容很大，肉眼一时无法定位，所以要用到搜索，在vi app.js后不要进入编辑模式，使用如下脚本进行搜索“this.base\_uri”

/this.base\_uri

3.修改ElasticSearch配置文件并启动ElasticSearch

这边需要修改elasticsearch的配置文件elasticsearch.yml，以允许跨域访问，在文末追加如下代码即可

vi /usr/elasticsearch/elasticsearch-6.4.0/config/elasticsearch.yml

http.cors.enabled: true

http.cors.allow-origin: "\*"

修改完毕后重新启动ElasticSearch（注意不能使用root权限启动，具体参考第一篇博文）

su elsearch

cd /usr/elasticsearch/elasticsearch-6.4.0/bin/

./elasticsearch -d

启动elasticsearch-head

/usr/elasticsearch/elasticsearch-head/node\_modules/grunt/bin/grunt server

后台运行：

nohup npm run start &

访问elasticsearch-head

在浏览器输入网址：http://elasticsearch-head的服务器ip:9100,出现如下页面则表示配置成功

# 安装IK分词器插件

## 简介

IK Analyzer是一个开源的，基于Java语言开发的轻量级的中文分词工具包，最初的时候，它是以开源项目Lucene为应用主体的，结合词典分词和文法分析算法的中文分词组件，从3.0版本之后，IK逐渐成为面向java的公用分词组件，独立于Lucene项目，同时提供了对Lucene的默认优化实现，IK实现了简单的分词 歧义排除算法，标志着IK分词器从单纯的词典分词向模拟语义分词衍化

## 基础环境

1.基础环境建立在前两篇博客的基础之上，这边IK的版本务必要和elasticsearch一致，否则会报错

2.安装maven

## 下载

git clone https://github.com/medcl/elasticsearch-analysis-ik.git

下载方式有两种，一种是直接用git命令下载，另一种是在windows上下载好后上传到服务器上再进行解压

<https://github.com/medcl/elasticsearch-analysis-ik/releases/tag/v6.4.0>

## 打包编译

按照上述方式下载好后，将项目进行打包

执行如下脚本（需要先按照maven，此处不再赘述，网上相关博文很多）：

mvn package

编译完成之后切换路径到项目下的target/releases，找到对应zip包，我这边是elasticsearch-analysis-ik-6.4.0.zip。

将该zip文件拷贝至/usr/elasticsearch/elasticsearch-6.4.0/plugins/ik（此处ik文件夹是自己创建的）下，并进行解压

unzip elasticsearch-analysis-ik-6.4.0.zip

重启ElasticSearch

systemctl restart elasticsearch.service

## 测试IK分词器

ik 带有两个分词器

ik\_max\_word ：会将文本做最细粒度的拆分；尽可能多的拆分出词语

ik\_smart：会做最粗粒度的拆分；已被分出的词语将不会再次被其它词语占有

更多具体内容可以看官方案例：官方测试案例

注意：在新版本中需要在请求头中设置请求格式，否则会报错，错误为

"error" : "Content-Type header [application/x-www-form-urlencoded] is not supported"

另外新版本中已经不支持String了，用text代替，输入String会报下错误

org.elasticsearch.index.mapper.MapperParsingException: No handler for type [string] declared on field [content]

at org.elasticsearch.index.mapper.ObjectMapper$TypeParser.parseProperties(ObjectMapper.java:274) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.index.mapper.ObjectMapper$TypeParser.parseObjectOrDocumentTypeProperties(ObjectMapper.java:199) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.index.mapper.RootObjectMapper$TypeParser.parse(RootObjectMapper.java:131) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.index.mapper.DocumentMapperParser.parse(DocumentMapperParser.java:112) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.index.mapper.DocumentMapperParser.parse(DocumentMapperParser.java:92) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.index.mapper.MapperService.parse(MapperService.java:626) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.metadata.MetaDataMappingService$PutMappingExecutor.applyRequest(MetaDataMappingService.java:263) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.metadata.MetaDataMappingService$PutMappingExecutor.execute(MetaDataMappingService.java:229) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.service.MasterService.executeTasks(MasterService.java:639) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.service.MasterService.calculateTaskOutputs(MasterService.java:268) ~[elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.service.MasterService.runTasks(MasterService.java:198) [elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.service.MasterService$Batcher.run(MasterService.java:133) [elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.service.TaskBatcher.runIfNotProcessed(TaskBatcher.java:150) [elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.cluster.service.TaskBatcher$BatchedTask.run(TaskBatcher.java:188) [elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.common.util.concurrent.ThreadContext$ContextPreservingRunnable.run(ThreadContext.java:624) [elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.common.util.concurrent.PrioritizedEsThreadPoolExecutor$TieBreakingPrioritizedRunnable.runAndClean(PrioritizedEsThreadPoolExecutor.java:244) [elasticsearch-6.4.0.jar:6.4.0]

at org.elasticsearch.common.util.concurrent.PrioritizedEsThreadPoolExecutor$TieBreakingPrioritizedRunnable.run(PrioritizedEsThreadPoolExecutor.java:207) [elasticsearch-6.4.0.jar:6.4.0]

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) [?:1.8.0\_181]

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) [?:1.8.0\_181]

at java.lang.Thread.run(Thread.java:748) [?:1.8.0\_181]

ik\_smart正确请求方式如下（直接复制粘贴到xshell，回车即可）：

curl -H "Content-Type: application/json" 'http://localhost:9200/index/\_analyze?pretty=true' -d '

{

"analyzer": "ik\_smart",

"text": "中华人民共和国万岁万岁万万岁"

}'

返回结果：

{

"tokens" : [

{

"token" : "中华人民共和国",

"start\_offset" : 0,

"end\_offset" : 7,

"type" : "CN\_WORD",

"position" : 0

},

{

"token" : "万岁",

"start\_offset" : 7,

"end\_offset" : 9,

"type" : "CN\_WORD",

"position" : 1

},

{

"token" : "万岁",

"start\_offset" : 9,

"end\_offset" : 11,

"type" : "CN\_WORD",

"position" : 2

},

{

"token" : "万万岁",

"start\_offset" : 11,

"end\_offset" : 14,

"type" : "CN\_WORD",

"position" : 3

}

]

}

ik\_max\_word正确请求方式如下：

curl -H "Content-Type: application/json" 'http://localhost:9200/index/\_analyze?pretty=true' -d '

{

 "analyzer": "ik\_max\_word",

 "text": "中华人民共和国万岁万岁万万岁"

}'

返回结果：

{

"tokens" : [

{

"token" : "中华人民共和国",

"start\_offset" : 0,

"end\_offset" : 7,

"type" : "CN\_WORD",

"position" : 0

},

{

"token" : "中华人民",

"start\_offset" : 0,

"end\_offset" : 4,

"type" : "CN\_WORD",

"position" : 1

},

{

"token" : "中华",

"start\_offset" : 0,

"end\_offset" : 2,

"type" : "CN\_WORD",

"position" : 2

},

{

"token" : "华人",

"start\_offset" : 1,

"end\_offset" : 3,

"type" : "CN\_WORD",

"position" : 3

},

{

"token" : "人民共和国",

"start\_offset" : 2,

"end\_offset" : 7,

"type" : "CN\_WORD",

"position" : 4

},

{

"token" : "人民",

"start\_offset" : 2,

"end\_offset" : 4,

"type" : "CN\_WORD",

"position" : 5

},

{

"token" : "共和国",

"start\_offset" : 4,

"end\_offset" : 7,

"type" : "CN\_WORD",

"position" : 6

},

{

"token" : "共和",

"start\_offset" : 4,

"end\_offset" : 6,

"type" : "CN\_WORD",

"position" : 7

},

{

"token" : "国",

"start\_offset" : 6,

"end\_offset" : 7,

"type" : "CN\_CHAR",

"position" : 8

},

{

"token" : "万岁",

"start\_offset" : 7,

"end\_offset" : 9,

"type" : "CN\_WORD",

"position" : 9

},

{

"token" : "万",

"start\_offset" : 7,

"end\_offset" : 8,

"type" : "TYPE\_CNUM",

"position" : 10

},

{

"token" : "岁",

"start\_offset" : 8,

"end\_offset" : 9,

"type" : "COUNT",

"position" : 11

},

{

"token" : "万岁",

"start\_offset" : 9,

"end\_offset" : 11,

"type" : "CN\_WORD",

"position" : 12

},

{

"token" : "万",

"start\_offset" : 9,

"end\_offset" : 10,

"type" : "TYPE\_CNUM",

"position" : 13

},

{

"token" : "岁",

"start\_offset" : 10,

"end\_offset" : 11,

"type" : "COUNT",

"position" : 14

},

{

"token" : "万万岁",

"start\_offset" : 11,

"end\_offset" : 14,

"type" : "CN\_WORD",

"position" : 15

},

{

"token" : "万万",

"start\_offset" : 11,

"end\_offset" : 13,

"type" : "TYPE\_CNUM",

"position" : 16

},

{

"token" : "万岁",

"start\_offset" : 12,

"end\_offset" : 14,

"type" : "CN\_WORD",

"position" : 17

},

{

"token" : "岁",

"start\_offset" : 13,

"end\_offset" : 14,

"type" : "COUNT",

"position" : 18

}

]

}

## 结果高亮

官方git上也有案例，具体看上面链接

curl -XPOST http://localhost:9200/index/fulltext/4 -H 'Content-Type:application/json' -d'

{"content":"中国驻洛杉矶领事馆遭亚裔男子枪击 嫌犯已自首"}'

curl -XPOST http://localhost:9200/index/fulltext/\_search -H 'Content-Type:application/json' -d'

{

"query" : { "match" : { "content" : "中国" }},

"highlight" : {

"pre\_tags" : ["<tag1>", "<tag2>"],

"post\_tags" : ["</tag1>", "</tag2>"],

"fields" : {

"content" : {}

}

}

}'

返回结果

{

"took": 8,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"skipped": 0,

"failed": 0

},

"hits": {

"total": 1,

"max\_score": 0.5753642,

"hits": [{

"\_index": "index",

"\_type": "fulltext",

"\_id": "1",

"\_score": 0.5753642,

"\_source": {

"content": "中国驻洛杉矶领事馆遭亚裔男子枪击 嫌犯已自首"

},

"highlight": {

"content": ["<tag1>中</tag1><tag1>国</tag1>驻洛杉矶领事馆遭亚裔男子枪击 嫌犯已自首"]

}

}]

}

}

## 扩展配置文件

IKAnalyzer.cfg.xml can be located at {conf}/analysis-ik/config/IKAnalyzer.cfg.xml or {plugins}/elasticsearch-analysis-ik-\*/config/IKAnalyzer.cfg.xml，意思就是说IKAnalyzer.cfg.xml可以放在上述位置（不过我看了下，该文件在我的/usr/elasticsearch/elasticsearch-6.4.0/plugins/ik/config目录下自带的，相关的扩展字典也在该目录下），文件内容如下

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">

<properties>

<comment>IK Analyzer 扩展配置</comment>

<!--用户可以在这里配置自己的扩展字典 -->

<entry key="ext\_dict">custom/mydict.dic;custom/single\_word\_low\_freq.dic</entry>

<!--用户可以在这里配置自己的扩展停止词字典-->

<entry key="ext\_stopwords">custom/ext\_stopword.dic</entry>

<!--用户可以在这里配置远程扩展字典 -->

<entry key="remote\_ext\_dict">location</entry>

<!--用户可以在这里配置远程扩展停止词字典-->

<entry key="remote\_ext\_stopwords">http://xxx.com/xxx.dic</entry>

</properties>

## 热更新IK分词

目前该插件支持热更新 IK 分词，通过上面在 IK 配置文件中添加如下配置

<!--用户可以在这里配置远程扩展字典 -->

<entry key="remote\_ext\_dict">location</entry>

<!--用户可以在这里配置远程扩展停止词字典-->

<entry key="remote\_ext\_stopwords">location</entry>

其中 location 是指一个 url，比如 http://yoursite.com/getCustomDict，该请求只需满足以下两点即可完成分词热更新。

该 http 请求需要返回两个头部(header)，一个是 Last-Modified，一个是 ETag，这两者都是字符串类型，只要有一个发生变化，该插件就会去抓取新的分词进而更新词库。

该 http 请求返回的内容格式是一行一个分词，换行符用 \n 即可。

满足上面两点要求就可以实现热更新分词了，不需要重启 ES 实例。

可以将需自动更新的热词放在一个 UTF-8 编码的 .txt 文件里，放在 nginx 或其他简易 http server 下，当 .txt 文件修改时，http server 会在客户端请求该文件时自动返回相应的 Last-Modified 和 ETag。可以另外做一个工具来从业务系统提取相关词汇，并更新这个 .txt 文件。

# 安装Kibana

<https://blog.csdn.net/gwd1154978352/article/details/82755511>

## 简介

Kibana是一个针对Elasticsearch的开源分析及可视化平台，用来搜索、查看交互存储在Elasticsearch索引中的数据。使用Kibana，可以通过各种图表进行高级数据分析及展示。

Kibana让海量数据更容易理解。它操作简单，基于浏览器的用户界面可以快速创建仪表板（dashboard）实时显示Elasticsearch查询动态。

设置Kibana非常简单。无需编码或者额外的基础架构，几分钟内就可以完成Kibana安装并启动Elasticsearch索引监测。

Kibana架构为Elasticsearch定制，可以将任何结构化和非结构化数据加入Elasticsearch索引。Kibana还充分利用了Elasticsearch强大的搜索和分析功能。

## 下载

下载地址（版本必须和elasticsearch版本一致）

## 解压

将下载的文件放到如下目录后进行解压

tar -zxvf kibana-6.4.0-linux-x86\_64.tar.gz

## 修改配置文件

修改/config/kibaba.yml文件（不修改的话外网无法通过ip访问）

将默认的server.host: "localhost" 改成server.host: "0.0.0.0"，以供外网访问。

如果elasticsearch不在该服务器或者说端口不是默认的9200端口的话还需要修改elasticsearch.url这个配置，将其设置为对应的elasticsearch的真实地址和端口。

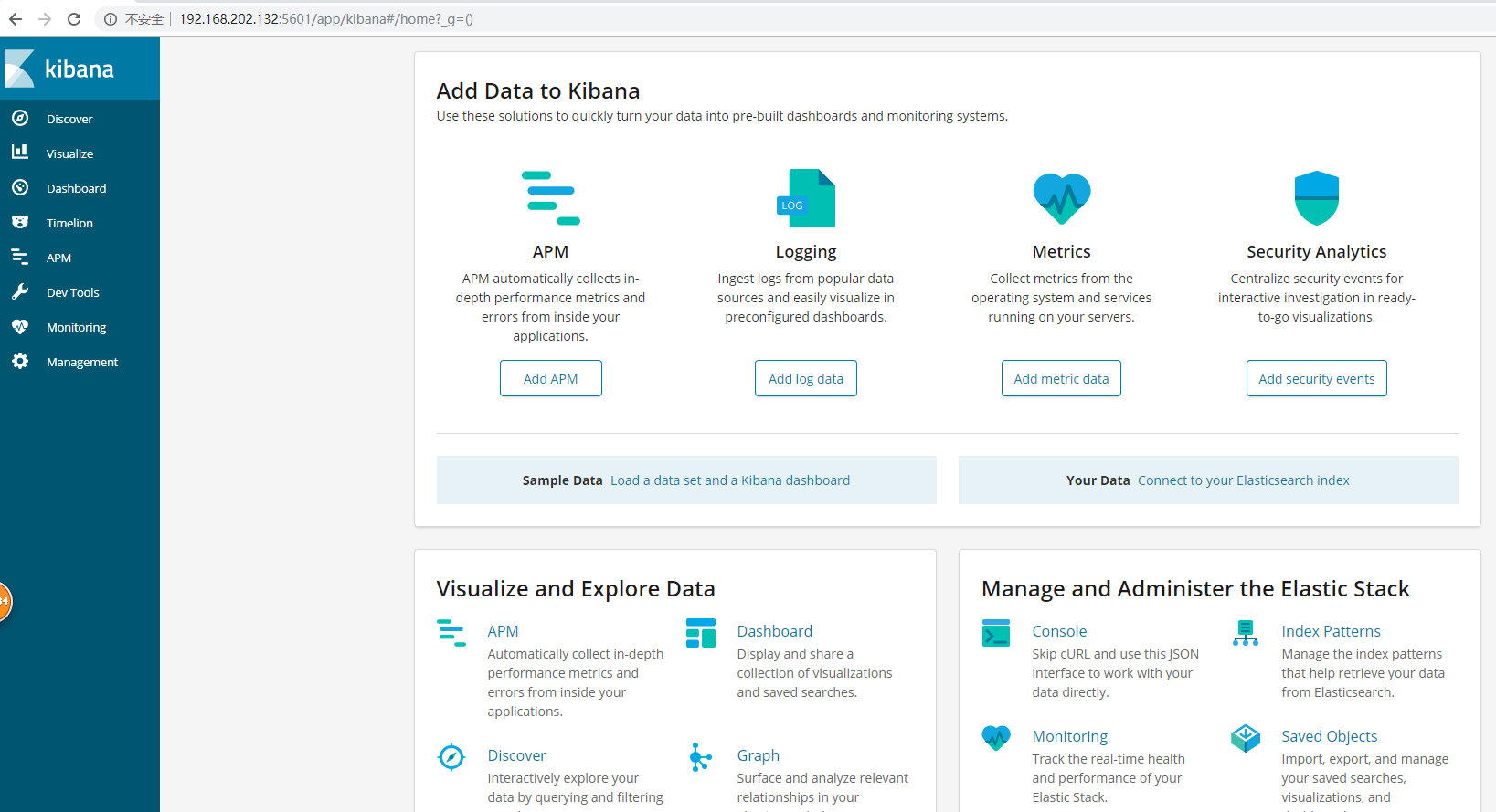
## 启动

cd /usr/elasticsearch /kibana-6.4.0-linux-x86\_64/bin

sh kibana （或者使用nohup ./kibana & 进行后台启动）

## 访问测试

在浏览器中输入http://IP:5601会有对应的kibana界面显示



# Kibana简单操作ES

<https://blog.csdn.net/gwd1154978352/article/details/82804942>

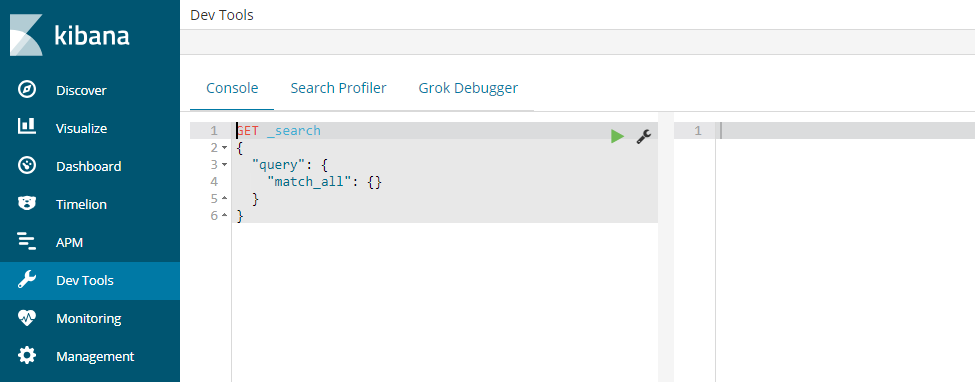
运行、打开kibana相关工具

要先运行ElasticSearch

/usr/elasticsearch/kibana-6.4.0-linux-x86\_64/bin

sh kibana

打开对应的dev Tools



## 获取所有数据

GET /\_search

返回数据含义

took：耗费了几毫秒

timed\_out：是否超时，false是没有，默认无timeout

\_shards（shard：碎片）：shards fail的条件（primary和replica全部挂掉），不影响其他shard。默认情况下来说，一个搜索请求，会打到一个index的所有primary shard上去，当然了，每个primary shard都可能会有一个或多个replic shard，所以请求也可以到primary shard的其中一个replica shard上去。

hits.total：本次搜索，返回了几条结果

hits.max\_score：score的含义，就是document对于一个search的相关度的匹配分数，越相关，就越匹配，分数也高

hits.hits：包含了匹配搜索的document的详细数据，默认查询前10条数据，按\_score降序排序

timeout这边默认是没有的，也就意味着当你搜索的时候他会直到所有搜索结束才会返回结果，但是当我们做一些时间比较敏感的搜索的时候，等待时间很久，对用户来说是非常不友好的。那我们可以通过设置timeout这个值，来定时返回已经搜索到的数据。timeout机制，指定每个shard，就只能在timeout时间范围内，将搜索到的部分数据（也可能是搜索到的全部数据），直接返回给client，而不是等到所有数据全部搜索出来后再返回。

可以通过如下方式进行设置

timeout=10ms，timeout=1s，timeout=1m

GET /\_search?timeout=10m

## 创建Document

PUT /ecommerce/product/1

{

"name" : "gaolujie yagao",

"desc" : "gaoxiao meibai",

"price" : 30,

"producer" : "gaolujie producer",

"tags": [ "meibai", "fangzhu" ]

}

PUT /ecommerce/product/2

{

"name" : "jiajieshi yagao",

"desc" : "youxiao fangzhu",

"price" : 25,

"producer" : "jiajieshi producer",

"tags": [ "fangzhu" ]

}

PUT /ecommerce/product/3

{

"name" : "zhonghua yagao",

"desc" : "caoben zhiwu",

"price" : 40,

"producer" : "zhonghua producer",

"tags": [ "qingxin" ]

}

## 检索文档（查询）

GET /ecommerce/product/1

## 替换文档（全量替换）

PUT /ecommerce/product/1

{

"name" : "jiaqiangban gaolujie yagao",

"desc" : "gaoxiao meibai",

"price" : 30,

"producer" : "gaolujie producer",

"tags": [ "meibai", "fangzhu" ]

}

## 注意点

document是不可变的，如果要修改document的内容，可以通过全量替换，直接对document重新建立索引，替换里面所有的内容。

es会将老的document标记为deleted（逻辑删除），然后新增我们给定的一个document，当我们创建越来越多的document的时候，es会在适当的时机在后台自动删除（物理删除）标记为deleted的document。

替换必须带上所有的field，否则其他数据会丢失。

## 更新文档（修改）

POST /ecommerce/product/1/\_update

{

"doc": {

"name": "jiaqiangban gaolujie yagao"

}

}

## 删除文档（删除）

DELETE /ecommerce/product/1

## 请求分类

1、query string search

类似这种 搜索全部商品：GET /ecommerce/product/\_search（参数直接拼接在请求url上，不带json参数的）

query string search的由来，因为search参数都是以http请求的query string来附带的

搜索商品名称中包含yagao的商品，而且按照售价降序排序：GET /ecommerce/product/\_search?q=name:yagao&sort=price:desc

适用于临时的在命令行使用一些工具，比如curl，快速的发出请求，来检索想要的信息；但是如果查询请求很复杂，是很难去构建的，所以在生产环境中，几乎很少使用query string search

2、query DSL

DSL：Domain Specified Language，特定领域的语言

http request body：请求体，可以用json的格式来构建查询语法，比较方便，可以构建各种复杂的语法，比query string search肯定强大多了。

更加适合生产环境的使用，可以构建复杂的查询

## 查询query

### 查询所有match\_all

查询所有的商品

GET /ecommerce/product/\_search

{

  "query": { "match\_all": {} }

}

### 排序 sort

查询名称包含yagao的商品，同时按照价格降序排序

GET /ecommerce/product/\_search

{

    "query" : {

        "match" : {

            "name" : "yagao"

        }

    },

    "sort": [

        { "price": "desc" }

    ]

}

### 分页查询 from size

总共3条商品，假设每页就显示1条商品，现在显示第2页，所以就查出来第2个商品

GET /ecommerce/product/\_search

{

  "query": { "match\_all": {} },

  "from": 1,

  "size": 1

}

from 从0开始

### 聚合查询

GET /book/\_search

{

"aggs":{

"group\_by\_word\_count":{

"terms":{

"field":"word\_count"

}

},

"group\_by\_publish\_date":{

"terms":{

"field":"publish\_date"

}

}

}

}



{

"aggs":{

"grades\_word\_count":{

"stats":{

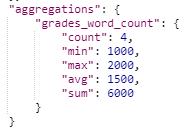
"field":"word\_count"

}

}

}

}



{

"aggs":{

"grades\_word\_count":{

"min":{

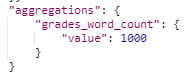
"field":"word\_count"

}

}

}

}



4.查询指定项

指定要查询出来商品的名称和价格

GET /ecommerce/product/\_search

{

  "query": { "match\_all": {} },

  "\_source": ["name", "price"]

}

### full-text search（全文检索） match

GET /ecommerce/product/\_search

{

    "query" : {

        "match" : {

            "producer" : "yagao producer"

        }

    }

}

### phrase search（短语搜索）

跟全文检索相对应，相反，全文检索会将输入的搜索串拆解开来，去倒排索引里面去一一匹配，只要能匹配上任意一个拆解后的单词，就可以作为结果返回

phrase search，要求输入的搜索串，必须在指定的字段文本中，完全包含一模一样的，才可以算匹配，才能作为结果返回

GET /ecommerce/product/\_search

{

    "query" : {

        "match\_phrase" : {

            "producer" : "yagao producer"

        }

    }

}

### 多字段匹配

GET /book/\_search

{

"query": {

"multi\_match": {

"query": "张三",

"fields": [ "author","title"]

}

}

}



### 语法查询

GET /book/\_search

{

"query": {

"query\_string": {

"query": "(入门 AND Java) OR Python OR 张三",

"fields":["author","title"]

}

}

}



### 字段级别查询\_term

GET /book/\_search

{

"query": {

"term": {

"word\_count":1000

}

}

}



GET /book/\_search

{

"query": {

"range": {

"word\_count": {

"gte": 1000,

"lte": 2000

}

}

}

}

### 过滤查询 bool\_filter

{

"query": {

"bool": {

"filter": {

"term": {

"word\_count": 1000

}

}

}

}

}

### 固定分数查询constant\_score\_boost

{

"query": {

"constant\_score": {

"filter": {

"term": {

"word\_count": 1000

}

},

"boost":2

}

}

}

### 布尔查询（多条件查询）

名字中有"yagao"，描述上可以有fangzhu也可以没有，价格不能是25元

must表示一定要满足；

should表示可以满足也可以不满足；

must\_not表示不能满足该条件；

"minimum\_should\_match": 1，表示最小匹配度，可以设置为百分百，详情看源文档Elasticsearch Reference [6.4] » Query DSL » Minimum Should Match，设置了这个值的时候就必须满足should里面的设置了，另外注意这边should里面同一字段设置的多个值（意思是当这个值等于X或者等于Y的时候都成立，务必注意格式）

GET /ecommerce/\_search

{

"query": {

"bool": {

"must": [

{

"match": {

"name": "yagao"

}

}

],

"should": [

{

"match": {

"desc": "fangzhu"

}

},

{

"match": {

"desc": "caoben"

}

}

],

"must\_not": [

{

"match": {

"price": 25

}

}

],

"minimum\_should\_match": 1

}

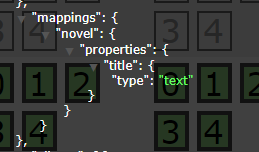
}

}

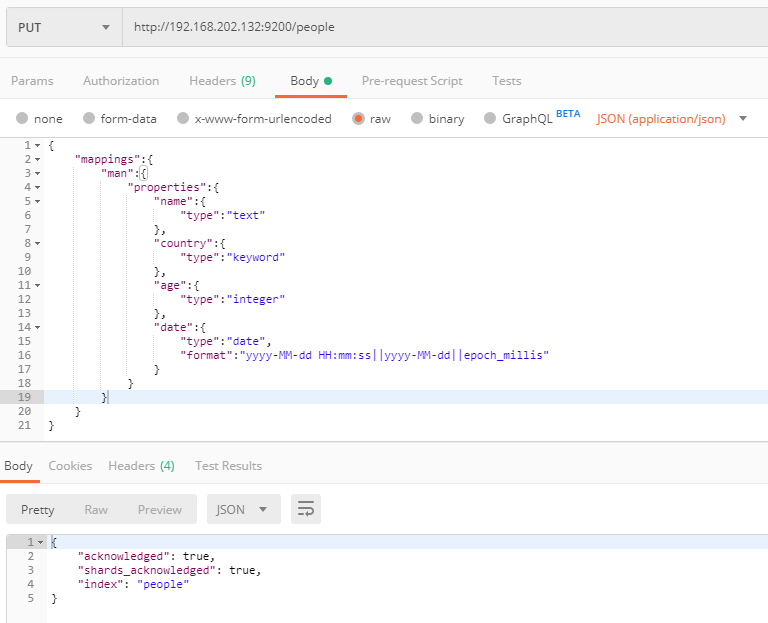
# Postman操作es

## 创建结构化索引mappings：PUT /索引名

索引信息中mapping信息不为空，则为结构化

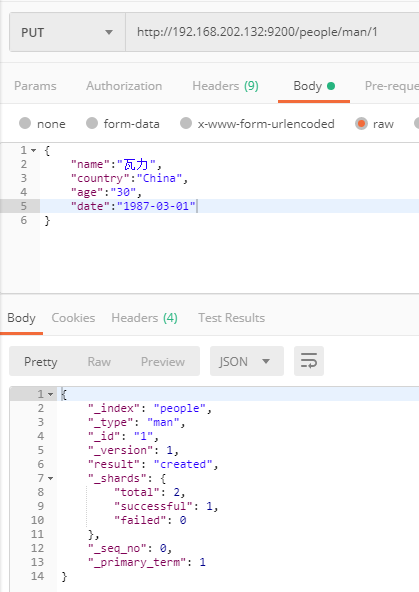
利用postman创建结构化索引

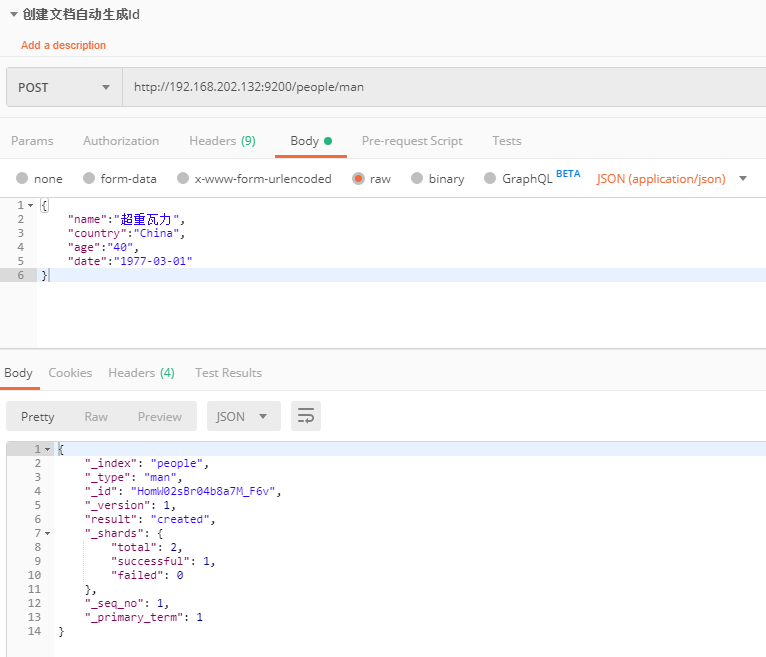


## 创建文档

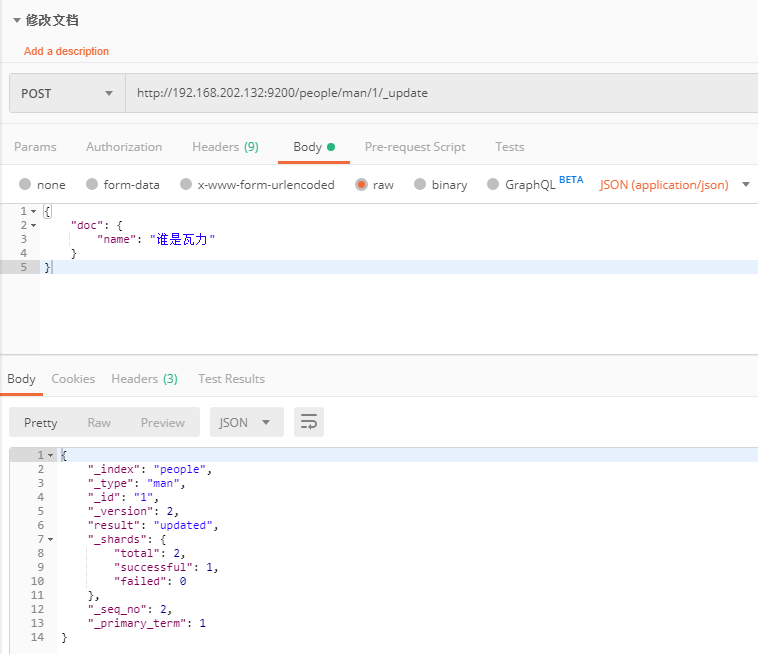
指定Id：PUT /索引名/类型名/文档Id

自动生成Id:POST /索引名/类型名

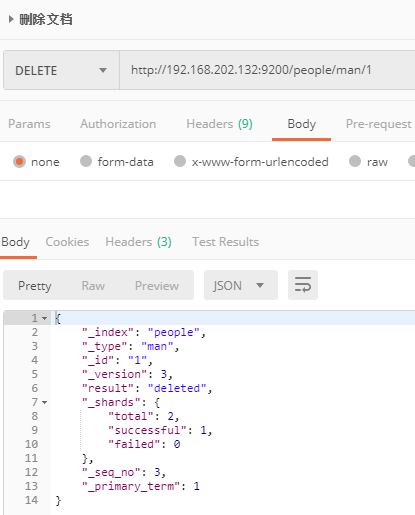




## 修改文档 doc

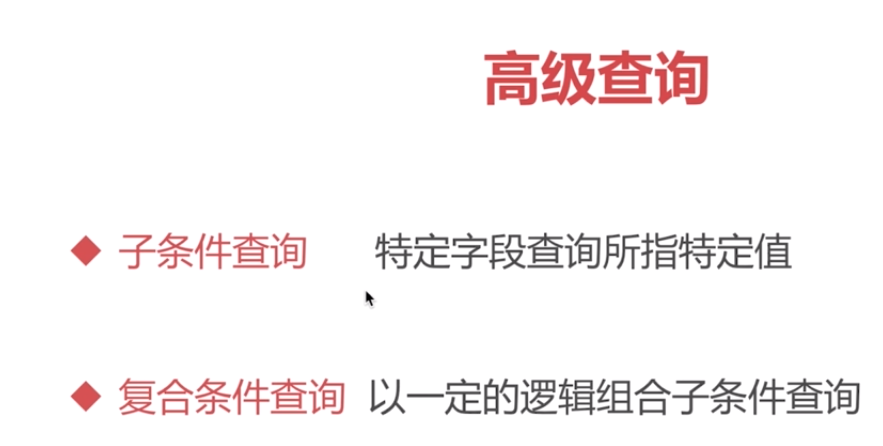


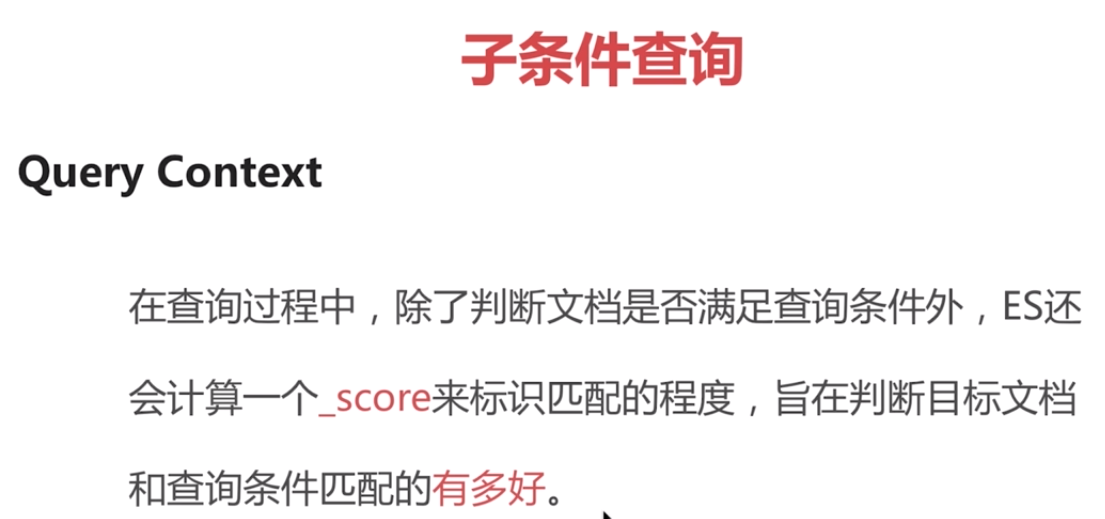
## 删除文档

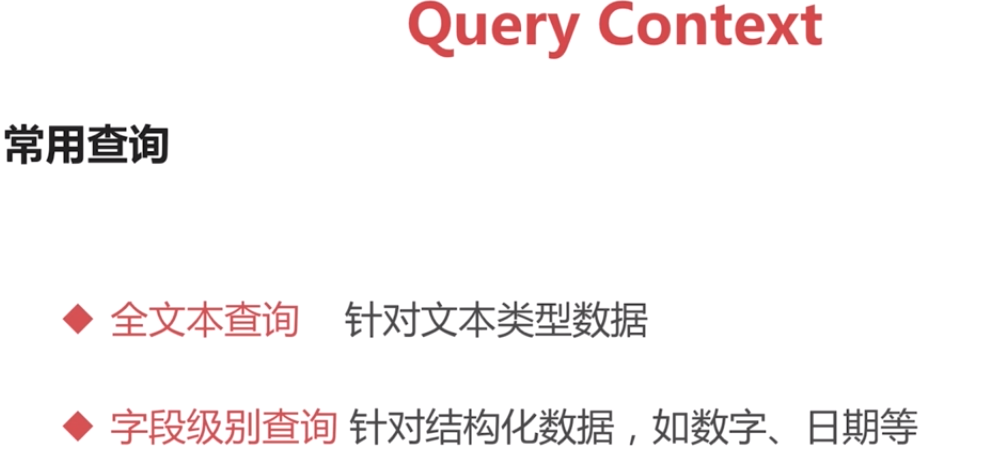


## 查询

### 简单查询







# 高亮

<https://blog.csdn.net/gwd1154978352/article/details/82761419>

# Java操作

## pom.xml

<?xml version=*"1.0"*?>

<project

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*

xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.lys.demo</groupId>

<artifactId>demo-elasticSearch-master</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>demo-elasticSearch-n0001</artifactId>

<name>demo-elasticSearch-n0001</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<elsaticsearch.version>6.4.0</elsaticsearch.version>

</properties>

<dependencies>

<dependency>

<groupId>org.elasticsearch.client</groupId>

<artifactId>elasticsearch-rest-client</artifactId>

<version>${elsaticsearch.version}</version>

</dependency>

<dependency>

<groupId>org.elasticsearch.client</groupId>

<artifactId>transport</artifactId>

<version>${elsaticsearch.version}</version>

</dependency>

<dependency>

<groupId>org.elasticsearch.plugin</groupId>

<artifactId>transport-netty4-client</artifactId>

<version>${elsaticsearch.version}</version>

</dependency>

<dependency>

<groupId>org.elasticsearch</groupId>

<artifactId>elasticsearch</artifactId>

<version>${elsaticsearch.version}</version>

</dependency>

<dependency>

<groupId>org.elasticsearch.client</groupId>

<artifactId>elasticsearch-rest-high-level-client</artifactId>

<version>${elsaticsearch.version}</version>

<type>pom</type>

</dependency>

<dependency>

<groupId>commons-lang</groupId>

<artifactId>commons-lang</artifactId>

<version>2.6</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.9.1</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

<version>2.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.47</version>

</dependency>

</dependencies>

</project>

## 测试类

package org.demo.elasticSearch.n0001;

import java.io.IOException;

import java.net.InetAddress;

import java.util.concurrent.ExecutionException;

import org.elasticsearch.action.delete.DeleteResponse;

import org.elasticsearch.action.get.GetResponse;

import org.elasticsearch.action.index.IndexResponse;

import org.elasticsearch.action.search.SearchRequestBuilder;

import org.elasticsearch.action.search.SearchResponse;

import org.elasticsearch.action.search.SearchType;

import org.elasticsearch.action.update.UpdateRequest;

import org.elasticsearch.client.transport.TransportClient;

import org.elasticsearch.common.settings.Settings;

import org.elasticsearch.common.text.Text;

import org.elasticsearch.common.transport.TransportAddress;

import org.elasticsearch.common.xcontent.XContentBuilder;

import org.elasticsearch.common.xcontent.XContentFactory;

import org.elasticsearch.common.xcontent.XContentType;

import org.elasticsearch.index.query.BoolQueryBuilder;

import org.elasticsearch.index.query.QueryBuilder;

import org.elasticsearch.index.query.QueryBuilders;

import org.elasticsearch.index.query.RangeQueryBuilder;

import org.elasticsearch.search.SearchHit;

import org.elasticsearch.search.SearchHits;

import org.elasticsearch.search.fetch.subphase.highlight.HighlightBuilder;

import org.elasticsearch.transport.client.PreBuiltTransportClient;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import com.alibaba.fastjson.JSONObject;

public class ESTest {

private TransportClient client = null;

// 获取客户端

@SuppressWarnings("resource")

@Before

public void getClient() throws Exception {

Settings settings = Settings.builder().put("cluster.name", "elasticsearch").build();

client = new PreBuiltTransportClient(settings)

.addTransportAddress(new TransportAddress(InetAddress.getByName("192.168.202.132"), 9300));

}

@Test

public void testCreateDoc() throws Exception {

XContentBuilder content = XContentFactory.jsonBuilder().startObject().field("author", "王五")

.field("title", "es入门").field("word\_count", 1000).field("publish\_date", "2019-08-01").endObject();

IndexResponse response = client.prepareIndex("book", "novel", "6")

.setSource(content)

.get();

System.out.println("索引名称：" + response.getIndex());

System.out.println("类型：" + response.getType());

System.out.println("文档ID：" + response.getId()); // 第一次使用是1

System.out.println("当前实例状态：" + response.status());

}

@Test

public void testCreateDocJSONObject() throws Exception {

JSONObject jsonObject = new JSONObject();

jsonObject.put("author", "赵六");

jsonObject.put("title", "activi入门");

jsonObject.put("word\_count", "2000");

jsonObject.put("publish\_date", "2018-08-01");

IndexResponse response = client.prepareIndex("book", "novel","7").setSource(jsonObject.toString(), XContentType.JSON)

.get();

System.out.println("索引名称：" + response.getIndex());//qq

System.out.println("类型：" + response.getType());//tweet

System.out.println("文档ID：" + response.getId()); // 第一次使用是1

System.out.println("当前实例状态：" + response.status());//CREATED

}

@Test

public void testGet() {

GetResponse response = client.prepareGet("book", "novel", "1").get();

System.out.println(response.getSource());

}

/\*\*

\* 删除一个文档

\* @param client

\*/

@Test

public void testDelete(){

DeleteResponse response=client.prepareDelete("book", "novel","1").get();

if(response.status().getStatus()==200){

System.out.println("删除成功");

}

}

@Test

public void testUpdate() throws IOException, InterruptedException, ExecutionException{

UpdateRequest request = new UpdateRequest("book", "novel","1");

XContentBuilder content = XContentFactory.jsonBuilder().startObject();

content.field("author", "赵六");

content.endObject();

request.doc(content);

client.update(request).get();

}

@Test

public void testSearch() throws Exception {

BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();

boolQuery.must(QueryBuilders.matchQuery("author", "赵六"));//模糊查询

boolQuery.must(QueryBuilders.matchQuery("title", "入门"));

RangeQueryBuilder rangeQuery = QueryBuilders.rangeQuery("word\_count").from(1000).to(2000);//字段级别查询

boolQuery.filter(rangeQuery);

SearchRequestBuilder searchRequestBuilder = client.prepareSearch("book")// 创建查询索引,参数productindex表示要查询的索引库为blog、index

.setTypes("novel") // 设置type

.setSearchType(SearchType.DFS\_QUERY\_THEN\_FETCH)// 设置查询类型 1.SearchType.DFS\_QUERY\_THEN\_FETCH = 精确查询

// 2.SearchType.SCAN =扫描查询,无序

.setQuery(boolQuery) // 设置查询项以及对应的值

.setFrom(0).setSize(10);// 设置分页;

System.out.println(searchRequestBuilder);

SearchResponse response = searchRequestBuilder.get();

SearchHits hits = response.getHits();

System.out.println("总数：" + hits.getTotalHits());

for (SearchHit hit : hits.getHits()) {

System.out.println(hit.getSourceAsMap());

}

System.out.println(response.toString());

closeClient();

}

@Test

public void testHighLight() throws Exception {

QueryBuilder matchQuery = QueryBuilders.matchQuery("title", "入门");

HighlightBuilder hiBuilder=new HighlightBuilder();

hiBuilder.preTags("<h2>");

hiBuilder.postTags("</h2>");

hiBuilder.field("title");

// 搜索数据

SearchRequestBuilder searchRequestBuilder = client.prepareSearch("book").setTypes("novel") // 设置type

.setQuery(matchQuery)

.highlighter(hiBuilder);

System.out.println("查询json："+searchRequestBuilder);

SearchResponse response = searchRequestBuilder.execute().actionGet();

//获取查询结果集

SearchHits searchHits = response.getHits();

System.out.println("共搜到:"+searchHits.getTotalHits()+"条结果!");

//遍历结果

for(SearchHit hit:searchHits){

System.out.println("String方式打印文档搜索内容:");

System.out.println(hit.getSourceAsString());

System.out.println("Map方式打印高亮内容");

System.out.println(hit.getHighlightFields());

System.out.println("遍历高亮集合，打印高亮片段:");

Text[] text = hit.getHighlightFields().get("title").getFragments();

for (Text str : text) {

System.out.println(str.string());

}

}

}

// 关闭客户端

@After

public void closeClient() {

if (this.client != null) {

this.client.close();

}

}

}

# 实例

文档可配置查看的角色（多个）-数组字段

## 创建结构化索引

<http://192.168.202.132:9200/book>

{

"mappings": {

"lunatic": {

"properties": {

"word\_count": {

"type": "integer"

},

"dept\_id": {

"type": "keyword"

},

"author": {

"type": "keyword"

},

"title": {

"type": "text",

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

"search\_analyzer": "ik\_smart"

},

"content": {

"type": "text",

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

"search\_analyzer": "ik\_smart"

},

"publish\_date": {

"type": "date",

"format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch\_millis"

},

"roles": {

"type": "keyword",

"fields": {

"keyword": {

"type": "keyword"

}

}

}

}

}

}

}

## 插入数据

<http://192.168.202.132:9200/book/lunatic/1>

{

"author": "林以宋",

"title": "林以宋",

"word\_count": "1000",

"publish\_date": "2000-10-01",

"dept\_id": "1",

"conent": "中国宪法",

"roles": [

"role1",

"role2"

]

}

<http://192.168.202.132:9200/book/lunatic/2>

{

"author":"中国刑法",

"title":"中国刑法",

"word\_count":"1000",

"publish\_date":"2000-10-01",

"dept\_id":"2",

"conent":"中国刑法",

"roles": [

"role3"

]

}

## 查询

<http://192.168.202.132:9200/book/_search>

{

"query": {

"bool": {

"should": [

{

"term": {

"roles": "role1"

}

},

{

"term": {

"roles": "role2"

}

}

]

}

}

}

Java

@Test

**public** **void** testSearchArray() **throws** Exception {

BoolQueryBuilder boolQuery = QueryBuilders.*boolQuery*();

boolQuery.should(QueryBuilders.*termQuery*("roles", "role1"));

boolQuery.should(QueryBuilders.*termQuery*("roles", "role2"));

SearchRequestBuilder searchRequestBuilder = client.prepareSearch("book")// 创建查询索引,参数productindex表示要查询的索引库为blog、index

.setTypes("lunatic") // 设置type

.setSearchType(SearchType.***DFS\_QUERY\_THEN\_FETCH***)// 设置查询类型 1.SearchType.DFS\_QUERY\_THEN\_FETCH = 精确查询

// 2.SearchType.SCAN =扫描查询,无序

.setQuery(boolQuery) // 设置查询项以及对应的值

.setFrom(0).setSize(10);// 设置分页;

System.***out***.println(searchRequestBuilder);

SearchResponse response = searchRequestBuilder.get();

SearchHits hits = response.getHits();

System.***out***.println("总数：" + hits.getTotalHits());

**for** (SearchHit hit : hits.getHits()) {

System.***out***.println(hit.getSourceAsMap());

}

System.***out***.println(response.toString());

closeClient();

}