分配用户给elasticsearch

bin/elasticsearch开动

设置conf 对外接口0.0.0.0

问题出现环境，OS版本：CentOS-7-x86\_64-Minimal-1708；ES版本：elasticsearch-6.2.2。

1、max file descriptors [4096] for elasticsearch process is too low, increase to at least [65536]

　　每个进程最大同时打开文件数太小，可通过下面2个命令查看当前数量

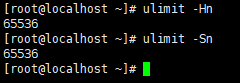
ulimit -Hn

ulimit -Sn

　　修改/etc/security/limits.conf文件，增加配置，用户退出后重新登录生效

\* soft nofile 65536

\* hard nofile 65536



2、max number of threads [3818] for user [es] is too low, increase to at least [4096]

　　问题同上，最大线程个数太低。修改配置文件/etc/security/limits.conf（和问题1是一个文件），增加配置

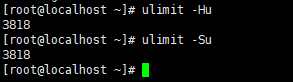
\* soft nproc 4096

\* hard nproc 4096

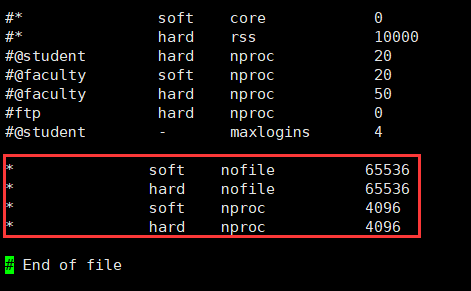
　　可通过命令查看

ulimit -Hu

ulimit -Su



修改后的文件：

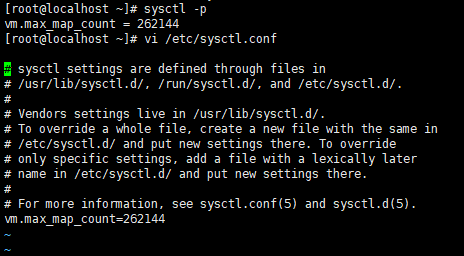


3、max virtual memory areas vm.max\_map\_count [65530] is too low, increase to at least [262144]

　　修改/etc/sysctl.conf文件，增加配置vm.max\_map\_count=262144

vi /etc/sysctl.conf  
sysctl -p

　　执行命令sysctl -p生效



 4、Exception in thread "main" java.nio.file.AccessDeniedException: /usr/local/elasticsearch/elasticsearch-6.2.2-1/config/jvm.options

　　elasticsearch用户没有该文件夹的权限，执行命令

chown -R elasticsearch:elasticsearch/usr/local/elasticsearch/

<http://10.73.11.107:9200/>

检查监控情况

10.73.11.107:9200/\_cat/health?v&pretty

现在让我们来大概看一看我们的索引：

Http请求内容

10.73.11.107:9200/\_cat/indices?v&pretty

Http请求内容：

PUT /customer?Pretty

GET /\_cat/indices?v

Curl命令

Kibana Console

第一个命令使用PUT方法创建了一个名为“customer”的索引。我们简单的在请求后面追加pretty参数来使返回值以

格式化过美观的JSON输出(如果返回值是JSON格式的话)。

然后它的返回结果为：

PUT /megacorp/employee/1

{

"first\_name" : "John",

"last\_name" : "Smith",

"age" : 25,

"about" : "I love to go rock climbing",

"interests": [ "sports", "music" ]

}

在URL中指定特殊的索引和类型进行多索引，多type搜索

1. /\_search：在所有的索引中搜索所有的类型
2. /school/\_search：在 school 索引中搜索所有的类型
3. /school,ad/\_search：在 school 和ad索引中搜索所有的类型
4. /s\*,a\*/\_search：在所有以g和a开头的索引中所有所有的类型
5. /school/student/\_search：在school索引中搜索student类型
6. /school,ad/student,phone/\_search：在school和ad索引上搜索student和phone类型
7. /\_all/student,phone/\_search：在所有的索引中搜索student和phone类型

[1 标识元字段](https://www.cnblogs.com/shoufeng/p/10679725.html" \l "1--%E6%A0%87%E8%AF%86%E5%85%83%E5%AD%97%E6%AE%B5)

* [1.1 \_index - 文档所属的索引](https://www.cnblogs.com/shoufeng/p/10679725.html" \l "11--_index---%E6%96%87%E6%A1%A3%E6%89%80%E5%B1%9E%E7%9A%84%E7%B4%A2%E5%BC%95)
* [1.2 \_uid - 包含\_type和\_id的复合字段](https://www.cnblogs.com/shoufeng/p/10679725.html" \l "12--_uid---%E5%8C%85%E5%90%AB_type%E5%92%8C_id%E7%9A%84%E5%A4%8D%E5%90%88%E5%AD%97%E6%AE%B5)
* [1.3 \_type - 文档的类型](https://www.cnblogs.com/shoufeng/p/10679725.html" \l "13--_type---%E6%96%87%E6%A1%A3%E7%9A%84%E7%B1%BB%E5%9E%8B)
* [1.4 \_id - 文档的id](https://www.cnblogs.com/shoufeng/p/10679725.html" \l "14--_id---%E6%96%87%E6%A1%A3%E7%9A%84id)

{

"\_index": "megacorp",

"\_type": "employee",

"\_id": "1",

"\_version": 3,

"\_seq\_no": 2,

"\_primary\_term": 1,

"found": true,

"\_source": {

"first\_name": "John",

"last\_name": "Smith",

"age": 25,

"about": "I love to go rock climbing",

"interests": [

"sports",

"music"

]

}

}

GET /megacorp/employee/1

全部查询查找

10.73.11.107:9200/megacorp/employee/\_search

192.168.26.135:9200/megacorp/employee/\_search?q=age:26 结果查询

ElasticSearch是基于lucene的开源搜索引擎，它的查询语法关键字跟lucene一样，如下:

* 分页:from/size
* 字段:fields
* 排序:sort
* 查询:query
* 过滤:filter
* 高亮:highlight
* 统计:facet



## **match**

最简单的一个match例子：

查询和"我的宝马多少马力"这个查询语句匹配的文档。

{

"query": {

"match": {

"content" : {

"query" : "我的宝马多少马力"

}

}

}

}

## **match\_phrase**

比如上面一个例子，一个文档"我的保时捷马力不错"也会被搜索出来，那么想要精确匹配所有同时包含"宝马 多少 马力"的文档怎么做？就要使用 match\_phrase 了

    match\_phase：会对输入做分词，但是需要结果中也包含所有的分词，而且顺序要求一样。以"hello world"为例，要求结果中必须包含hello和world，而且还要求他们是连着的，顺序也是固定的，hello that word不满足，world hello也不满足条件。

{

"query": {

"match\_phrase": {

"content" : {

"query" : "我的宝马多少马力"

}

}

}

}

完全匹配可能比较严，我们会希望有个可调节因子，少匹配一个也满足，那就需要使用到slop。

{

"query": {

"match\_phrase": {

"content" : {

"query" : "我的宝马多少马力",

"slop" : 1

}

}

}

}

如果我们希望两个字段进行匹配，其中一个字段有这个文档就满足的话，使用multi\_match

{

"query": {

"multi\_match": {

"query" : "我的宝马多少马力",

"fields" : ["title", "content"]

}

}

}

但是multi\_match就涉及到匹配评分的问题了。

## **我们希望完全匹配的文档占的评分比较高，则需要使用best\_fields**

{

"query": {

"multi\_match": {

"query": "我的宝马发动机多少",

"type": "best\_fields",

"fields": [

"tag",

"content"

],

"tie\_breaker": 0.3

}

}

}

意思就是完全匹配"宝马 发动机"的文档评分会比较靠前，如果只匹配宝马的文档评分乘以0.3的系数

## **我们希望越多字段匹配的文档评分越高，就要使用most\_fields**

{

"query": {

"multi\_match": {

"query": "我的宝马发动机多少",

"type": "most\_fields",

"fields": [

"tag",

"content"

]

}

}

}

## **我们会希望这个词条的分词词汇是分配到不同字段中的，那么就使用cross\_fields**

{

"query": {

"multi\_match": {

"query": "我的宝马发动机多少",

"type": "cross\_fields",

"fields": [

"tag",

"content"

]

}

}

}

# **term**

term是代表完全匹配，即不进行分词器分析，文档中必须包含整个搜索的词汇

{

"query": {

"term": {

"content": "汽车保养"

}

}

}

查出的所有文档都包含"汽车保养"这个词组的词汇。

使用term要确定的是这个字段是否“被分析”(analyzed)，默认的字符串是被分析的。

拿官网上的例子举例：

mapping是这样的：

PUT my\_index

{

"mappings": {

"my\_type": {

"properties": {

"full\_text": {

"type": "string"

},

"exact\_value": {

"type": "string",

"index": "not\_analyzed"

}

}

}

}

}

PUT my\_index/my\_type/1

{

"full\_text": "Quick Foxes!",

"exact\_value": "Quick Foxes!"

}

其中的full\_text是被分析过的，所以full\_text的索引中存的就是[quick, foxes]，而extra\_value中存的是[Quick Foxes!]。

那下面的几个请求：

GET my\_index/my\_type/\_search

{

"query": {

"term": {

"exact\_value": "Quick Foxes!"

}

}

}

请求的出数据，因为完全匹配

GET my\_index/my\_type/\_search

{

"query": {

"term": {

"full\_text": "Quick Foxes!"

}

}

}

## **bool联合查询: must,should,must\_not**

如果我们想要请求"content中带宝马，但是tag中不带宝马"这样类似的需求，就需要用到bool联合查询。  
联合查询就会使用到must,should,must\_not三种关键词。

这三个可以这么理解

* must: 文档必须完全匹配条件
* should: should下面会带一个以上的条件，至少满足一个条件，这个文档就符合should
* must\_not: 文档必须不匹配条件

比如上面那个需求：

{

"query": {

"bool": {

"must": {

"term": {

"content": "宝马"

}

},

"must\_not": {

"term": {

"tags": "宝马"

}

}

}

}

}



GET /website/blog/123?\_source=title,text

{"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 1,

"exists" : true,

"\_source" : {

"title": "My first blog entry" ,

"text": "Just trying this out..."

}

更新时候区别

version\_type=external，唯一的区别在于，\_version，只有当你提供的version与es中的\_version一模一样的时候，才可以进行修改，只要不一样，就报错；当version\_type=external的时候，只有当你提供的version比es中的\_version大的时候，才能完成修改

es，\_version=1，?version=1**，才能更新成功**

**es，\_version**=1，?version>1&version\_type=external，才能成功，比如说?version=2&version\_type=external

局部更新

POST /website/blog/1/\_update

{

"doc" : {

"tags" : [ "testing" ],

"views": 0

}

结果

}

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 3,

"found": true,

"\_source": {

"title": "My first blog entry",

"text": "Starting to get the hang of this...",

"tags": [ "testing" ], <1>

"views": 0 <1>

}

}

GET /website/blog/\_mget

{

"docs" : [

{ "\_id" : 2 },

{ "\_type" : "pageviews", "\_id" : 1 }

]

}GET /website/blog/\_mget

{

"ids" : [ "2", "1" ]

}