# 概述

搜索指的是计算机根据用户输入的关键词进行匹配，从已有的数据库中摘录出相关的记录反馈给用户。

常见的全网搜索引擎如百度、谷歌，但是除此以外，搜索技术在垂直领域也有广泛的使用，比如淘宝、京东搜索商品，万芳、知网搜索期刊，csdn中搜索问题贴，也都是基于海量数据的搜索。

## 全文搜索和传统关系性数据库对比

**传统关系型数据库弊端**

1、对于传统的关系性数据库对于关键词的查询，只能逐字逐行的匹配，性能非常差。

2、匹配方式不合理，比如搜索“小密手机” ，如果用like进行匹配， 根本匹配不到。但是考虑使用者的用户体验的话，除了完全匹配的记录，还应该显示一部分近似匹配的记录，至少应该匹配到“手机”。

**全文搜索**

全文搜索引擎目前主流的索引技术就是倒排索引的方式。利用基于分词技术构建倒排索引，首先每个记录保存数据时，都不会直接存入数据库。系统先会对数据进行分词，然后以倒排索引结构保存。如下：



用户搜索时，把搜索的关键词也进行分词，“红海行动”分词分成“红海”和“行动”两个词。这样的话，先用红海进行匹配，得到id=1和id=2的记录编号，再用行动匹配可以迅速定位id为1,3的记录。

全文索引通常还会根据匹配程度进行打分，显然1号记录能匹配的次数更多。所以显示的时候以评分进行排序的话，1号记录会排到最前面。而2、3号记录也可以匹配到。

## elasticsearch

Elasticsearch是一个基于Apache Lucene(TM)的开源搜索引擎。无论在开源还是专有领域，Lucene可以被认为是迄今为止最先进、性能最好的、功能最全的搜索引擎库。

**特点**

分布式的实时文件存储，每个字段都被索引并可被搜索

分布式的实时分析搜索引擎--做不规则查询

可以扩展到上百台服务器，处理PB级结构化或非结构化数据

Elasticsearch也使用Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，但是它的目的是通过简单的RESTful API来隐藏Lucene的复杂性，从而让全文搜索变得简单。

**ES能做什么？**

全文检索（全部字段）、模糊查询（搜索）、数据分析（提供分析语法，例如聚合）

**交互方式**

基于HTTP协议，以JSON为数据交互格式的RESTful API

GET POST PUT DELETE HEAD

Elasticsearch官方提供了多种程序语言的客户端—java，Javascript，.NET，PHP，Perl，Python，以及 Ruby——还有很多由社区提供的客户端和插件

**简单实例**

public class Movie {

String id;

String name;

Double doubanScore;

List<Actor> actorList;

}

public class Actor{

String id;

String name;

}

{

“id”:”1”,

“name”:”operation red sea”,

“doubanScore”:”8.5”,

“actorList”:[

{“id”:”1”,”name”:”zhangyi”},

{“id”:”2”,”name”:”haiqing”},

{“id”:”3”,”name”:”zhanghanyu”}

]

}

## 同类产品

**Solr、ElasticSearch、Hermes（腾讯）（实时检索分析）**

**Solr、ES**

1. 源自搜索引擎，侧重搜索与全文检索。

2. 数据规模从几百万到千万不等，数据量过亿的集群特别少。

有可能存在个别系统数据量过亿，但这并不是普遍现象（就像Oracle的表里的数据规模有可能超过Hive里一样，但需要小型机）。

**Hermes**

1. 一个基于大索引技术的海量数据实时检索分析平台。侧重数据分析。

2. 数据规模从几亿到万亿不等。最小的表也是千万级别。

在 腾讯17 台TS5机器，就可以处理每天450亿的数据(每条数据1kb左右)，数据可以保存一个月之久。

**Solr、ES区别**

全文检索、搜索、分析。基于lucene

Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能;

Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式；

Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供；

Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于Elasticsearch，如“附近的人”功能。

## lucene与elasticsearch

处理分词、构建倒排索引等都是lucene处理的，但是lucene还不能称之为搜索引擎，lucene只是一个提供全文搜索功能类库的核心工具包，而真正使用它还需要一个完善的服务框架搭建起来的应用。lucene类似于jdk，而搜索引擎软件类似于tomcat。

目前市面上流行的搜索引擎软件就两款，分别是elasticsearch和solr,这两款都是基于lucene搭建的可以独立部署启动的搜索引擎服务软件。由于内核相同，所以两者除了服务器安装、部署、管理、集群以外，对于数据的操作，修改、添加、保存、查询等等都十分类似。

从实际企业使用情况来看，elasticsearch的市场份额逐步在取代solr，国内百度、京东、新浪都是基于elasticsearch实现的搜索功能。国外像维基百科、GitHub、Stack Overflow等等也都是基于ES的。

elasticsearch为用户提供按关键字查询的全文搜索功能。

著名的ELK框架(**ElasticSearch,Logstash,Kibana**)就是实现企业海量日志的处理分析解决方案，大数据领域的重要一份子。

# 基本概念

cluster

整个elasticsearch 默认就是集群状态，整个集群是一份完整、互备的数据。

node

集群中的一个节点，一般只一个进程就是一个node，主节点负责集群级别的操作，如加入新的节点、删除节点、创建和删除索引等；从节点负责具体数据的增删改查。

shard

分片，即使是一个节点中的数据也会通过hash算法，分成多个片存放，默认是5片。对于数据的增加、修改和删除操作，都是在主片上进行，然后同步到复制片（syn同步或asyn异步），对于数据的查询，主片和复制片都可以进行。

Index(库)

相当于rdbms的database, 对于用户来说是一个逻辑数据库，虽然物理上会被分多个shard存放，也可能存放在多个node中。

Type(表)

类似于rdbms的table，但是与其说像table，其实更像面向对象中的class , 同一Json的格式的数据集合。每个type中的字段是什么数据类型由mapping定义。但是如果没有设定mapping系统会自动，根据一条数据的格式来推断出应该的数据格式。

true/false → boolean

1020 → long

20.1 → double

“2018-02-01” → date

“hello world” → text +keyword

默认只有text会进行分词，keyword是不会分词的字符串。

mapping

除了自动定义，还可以手动定义，但是只能对新加的、没有数据的字段进行定义。一旦有了数据就无法再做修改了。注意虽然每个Field的数据放在不同的type下,但是同一个名字的Field在一个index下只能有一种mapping定义。

Document

(一条数据) 类似于rdbms的 row、面向对象里的object

Field

(字段) 相当于字段、属性

## 数据类型

每个type中的字段的数据类型由mapping定义，如果没有设定mapping，系统会自动根据数据的格式来推断出对应的数据格式。

true/false → boolean

1020 → long

20.1 → double

“2018-02-01” → date

“hello world” → text +keyword

默认只有text会进行分词，keyword是不会分词的字符串。

mapping除了自动定义，还可以手动定义，但是只能对新加的、没有数据的字段进行定义。一旦有了数据就无法再做修改了。

注意：虽然每个Field的数据放在不同的type下,但是同一个名字的Field在一个index下只能有一种mapping定义。

**核心数据类型**

字符串型：text、keyword

数值型：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

范围类型：integer\_range、float\_range、long\_range、double\_range、date\_range

**复杂数据类型**

数组类型：array

对象类型：object

嵌套类型：nested object

**地理位置数据类型**

geo\_point(点)、geo\_shape(形状)

**专用类型**

记录IP地址ip

实现自动补全completion

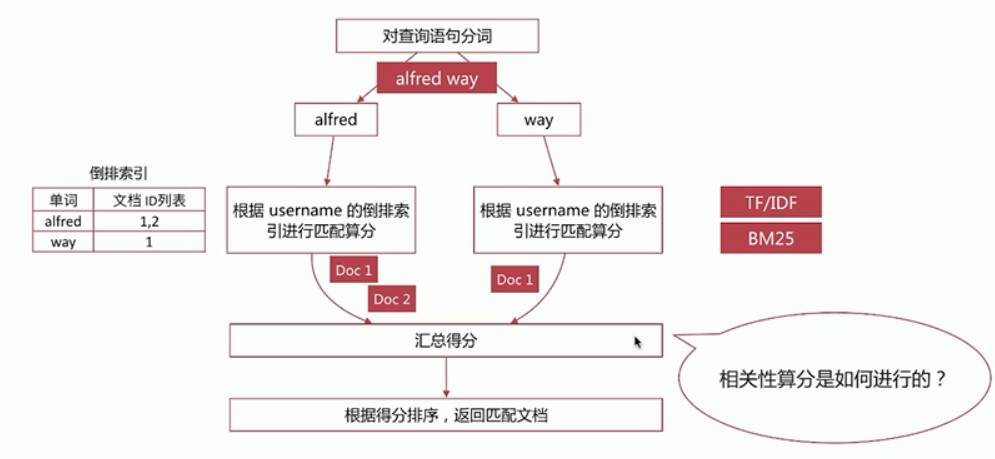
记录分词数：token\_count

记录字符串hash值母乳murmur3

**多字段特性multi-fields**

允许对同一个字段采用不同的配置，比如分词，例如对人名实现拼音搜索，只需要在人名中新增一个子字段为pinyin即可

## 相关性算分



相关性算分：指文档与查询语句间的相关度，通过倒排索引可以获取与查询语句相匹配的文档列表。将最符合用户查询需求的文档放到前列的本质问题是一个排序的问题，排序的依据是相关性算分，确定倒排索引哪个文档排在前面。

影响相关度算分的参数：

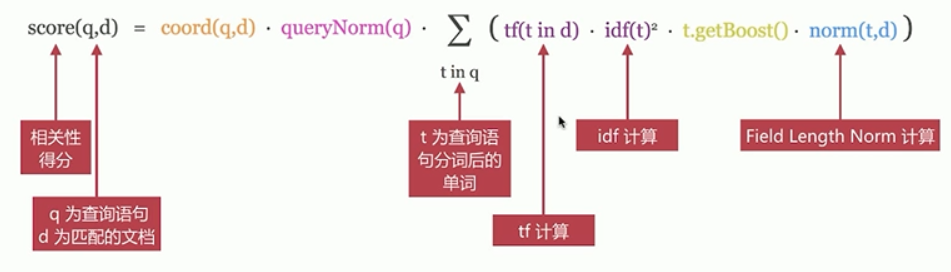
TF(**Term Frequency**)：词频，即单词在文档中出现的次数，词频越高，相关度越高

DF(**Document Frequency**)：文档词频，即单词出现的文档数

IDF(**Inverse Document Frequency**)：逆向文档词频，与文档词频相反，即1/DF。即单词出现的文档数越少，相关度越高（如果一个单词在文档集出现越少，算为越重要单词）

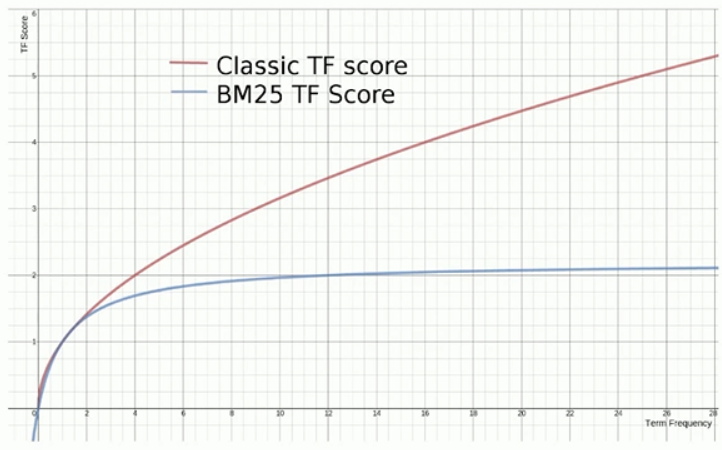
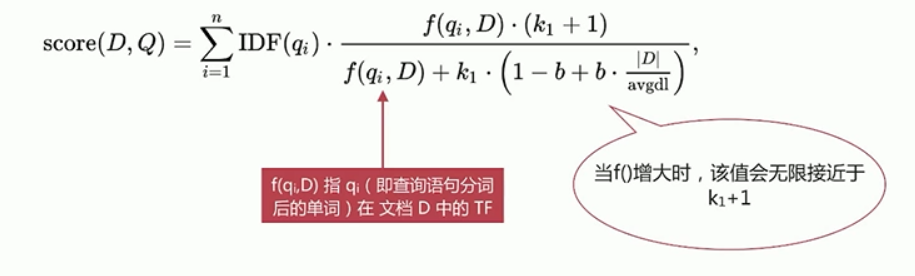
Field-length Norm：文档越短，相关度越高

**TF/IDE模型**



**BM25模型（5.X之后的默认模型）**

BM25相比TF/IDF的一大优化是降低了tf在过大时的权重，避免词频对查询影响过大。



**查看算分**

|  |
| --- |
| GET test\_search\_index/\_search  {  "explain": true,  "query":{  "match": {  "username": {  "query":"alfred way",  "operator":"and"  }  }  }  } |

# 数据存储方式

## 存储方式

**面向文档**

Elasticsearch是面向文档(**document oriented**)的，这意味着它可以存储整个对象或文档(document)。然而它不仅仅是存储，还会索引(index)每个文档的内容使之可以被搜索。在Elasticsearch中，你可以对文档（而非成行成列的数据）进行索引、搜索、排序、过滤。这种理解数据的方式与以往完全不同，这也是Elasticsearch能够执行复杂的全文搜索的原因之一。

**JSON**

ELasticsearch使用Javascript对象符号(**JavaScript Object Notation**)，也就是JSON，作为文档序列化格式。JSON现在已经被大多语言所支持，而且已经成为NoSQL领域的标准格式。它简洁、简单且容易阅读。

以下使用JSON文档来表示一个用户对象：

{

"email": "john@smith.com",

"first\_name": "John",

"last\_name": "Smith",

"info": {

"bio": "Eco-warrior and defender of the weak",

"age": 25,

"interests": [ "dolphins", "whales" ]

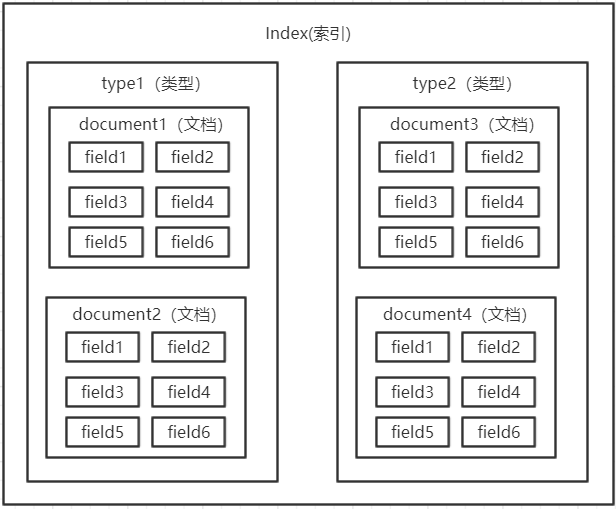
},

"join\_date": "2014/05/01"

}

尽管原始的user对象很复杂，但它的结构和对象的含义已经被完整的体现在JSON中了，在Elasticsearch中将对象转化为JSON并做索引要比在表结构中做相同的事情简单的多。

## 存储结构



### Mysql数据与ES数据转化

#### 元数据

创建文档语句

PUT atguigu/doc

{

“name”:”zhangsan”,

“age”:10

}

\_index:文档所在索引名称

\_type:文档所在类型名称

\_id:文档唯一id

\_uid:组合id，由\_type和\_id组成（6.x后，\_type不再起作用，同\_id）

\_source:文档的原始Json数据，包括每个字段的内容

\_all:将所有字段内容整合起来，默认禁用（用于对所有字段内容检索）

#### 名词解释

**索引 index**

一个索引就是一个拥有几分相似特征的文档的集合。比如说，你可以有一个客户数据的索引，另一个产品目录的索引，还有一个订单数据的索引。一个索引由一个名字来标识（必须全部是小写字母的），并且当我们要对对应于这个索引中的文档进行索引、搜索、更新和删除的时候，都要使用到这个名字。在一个集群中，可以定义任意多的索引。

**类型 type**

Es6之后，一个index中只能有一个type。

在一个索引中，你可以定义一种或多种类型。一个类型是你的索引的一个逻辑上的分类/分区，其语义完全由你来定。通常，会为具有一组共同字段的文档定义一个类型。比如说，我们假设你运营一个博客平台并且将你所有的数据存储到一个索引中。在这个索引中，你可以为用户数据定义一个类型，为博客数据定义另一个类型，当然，也可以为评论数据定义另一个类型。

**字段Field**

相当于是数据表的字段，对文档数据根据不同属性进行的分类标识

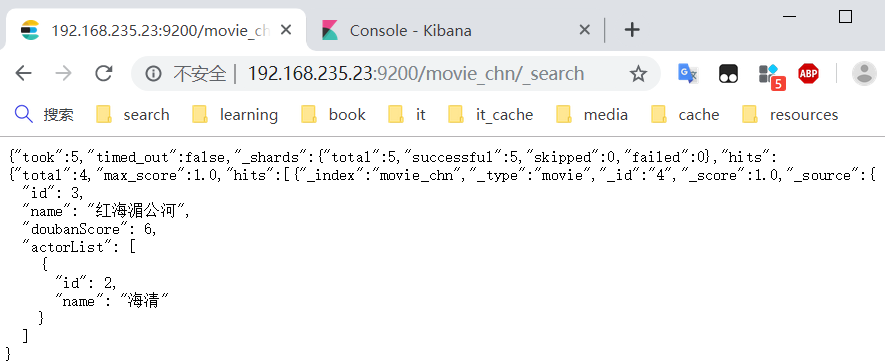
**document**

一个文档是一个可被索引的基础信息单元。比如，你可以拥有某一个客户的文档，某一个产品的一个文档，当然，也可以拥有某个订单的一个文档。文档以JSON（Javascript Object Notation）格式来表示，而JSON是一个到处存在的互联网数据交互格式。在一个index/type里面，你可以存储任意多的文档。注意，尽管一个文档，物理上存在于一个索引之中，文档必须被索引/赋予一个索引的type。

# 操作工具

## Rest风格HTTP请求

在浏览器中输入相关命令即可访问es进行搜索。



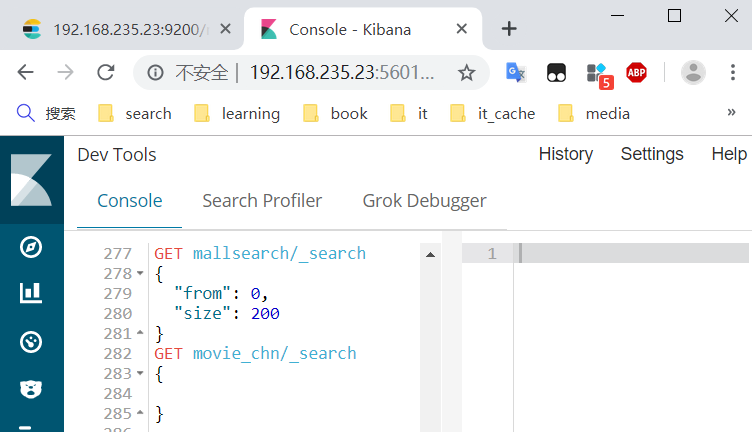
## Linux命令行

## Kibana

Kibana 是一个开源分析和可视化平台，旨在可视化操作 Elasticsearch 。Kibana可以用来搜索，查看和与存储在 Elasticsearch 索引中的数据进行交互。可以轻松地进行高级数据分析，并可在各种图表，表格和地图中显示数据。

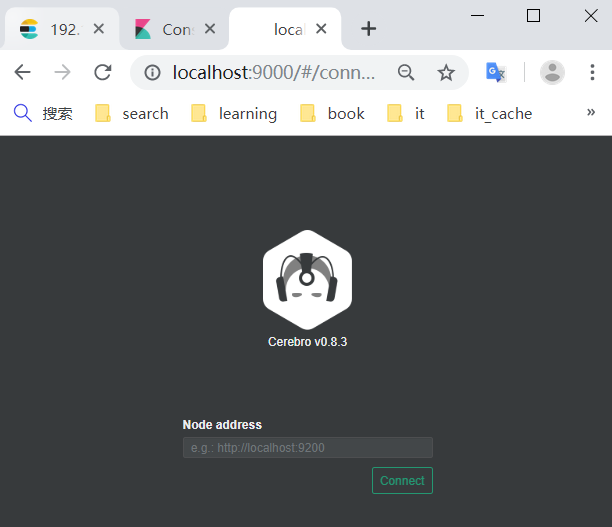
Kibana 可以轻松理解海量数据。其简单的基于浏览器的界面使您能够快速创建和共享动态仪表板，实时显示 Elasticsearch 查询的更改。

通过在Kibana图形界面中输入DSL命令来访问es进行各种操作。Kibana访问地址为http://IP:5601



## Cerebro插件

Cerebro网址的访问地址为localhost:9000



# 搜索原理

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html该网站可以查询相关算法和内存结构。

## 索引方式

### 正排索引

正排索引指的是记录文档id到文档内容、单词的关联关系

|  |  |
| --- | --- |
| **docid** | **content** |
| 1 | Java从入门到放弃 |
| 2 | PHP是世界上最好的编程语言 |
| 3 | 计算机编程语言是如何诞生的 |

### 倒排索引

记录单词到文档id的关联关系，包含：

单词词典（Term DicTionary）：记录所有文档的单词，一般比较大

倒排索引（Posting List）：记录单词倒排列表的关联信息

例如：编程语言

Term Dictionary 编程语言

Posting List

|  |  |  |  |
| --- | --- | --- | --- |
| DocId | TF | Position | Offset |
| 1 | 1 | 0 | <0,2> |
| 3 | 1 | 0 | <0,2> |

DocId：文档id，文档的原始信息

TF：单词频率，记录该词再文档中出现的次数，用于后续相关性算分

Position：位置，记录Field分词后，单词所在的位置，从0开始

Offset：偏移量，记录单词在文档中开始和结束位置，用于高亮显示等

**内存结构**

ElasticSearch的内存结构是B+Tree形式的，每个文档字段都有自己的倒排索引。

## 分词

分词指的是将文本转换成一系列单词（term or token）的过程，也可以叫做文本分析，在es里面称为Analysis。

### 分词概念

#### 分词机制

|  |  |  |
| --- | --- | --- |
| Character Filter | 对原始文本进行处理 | 例：去除html标签、特殊字符等 |
| Tokenizer | 将原始文本进行分词 | 例：培训机构-->培训，机构 |
| Token Filters | 分词后的关键字进行加工 | 例：转小写、删除语气词、近义词和同义词等 |

#### 分词API

**直接指定测试（指定分词器）**

**Request:**

POST \_analyze

{

"analyzer": "standard",

"text":"hello world"

}

**Response:**

{

"tokens": [

{

"token": "hello", #分词

"start\_offset": 0, #开始偏移

"end\_offset": 5, #结束偏移

"type": "<ALPHANUM>", #单词类型

"position": 0 #位置

},

{

"token": "world",

"start\_offset": 6,

"end\_offset": 11,

"type": "<NUM>",

"position": 1

}

]

}

**针对索引的字段进行分词测试（利用该字段的分词器）**

**Request：**

POST test/\_analyze

{

"field": "name",

"text":"hello world"

}

**Response:**

{

"tokens": [

{

"token": "hello",

"start\_offset": 0,

"end\_offset": 5,

"type": "<ALPHANUM>",

"position": 0

},

{

"token": "world",

"start\_offset": 6,

"end\_offset": 11,

"type": "<ALPHANUM>",

"position": 1

}

]

}

**自定义分词器**

**Request:**

POST \_analyze

{

"tokenizer": "standard",

"filter": ["lowercase"],

"text":"Hello WORLD"

}

**Response:**

{

"tokens": [

{

"token": "hello",

"start\_offset": 0,

"end\_offset": 5,

"type": "<ALPHANUM>",

"position": 0

},

{

"token": "world",

"start\_offset": 6,

"end\_offset": 11,

"type": "<ALPHANUM>",

"position": 1

}

]

}

##### ElasticSearch自带的分词器

|  |  |
| --- | --- |
| **分词器（Analyzer）** | **特点** |
| Standard（es默认） | 支持多语言，按词切分并做小写处理 |
| Simple | 按照非字母切分，小写处理 |
| Whitespace | 按照空格来切分 |
| Stop | 去除语气助词，如the、an、的、这等 |
| Keyword | 不分词 |
| Pattern | 正则分词，默认\w+,即非字词符号做分割符 |
| Language | 常见语言的分词器（30+） |

##### Character Filters

在进行Tokenizer之前对原始文本进行处理，如增加、删除或替换字符等，注意：进行处理后，会影响后续tokenizer解析的position和offset。

|  |  |
| --- | --- |
| HTML Strip | 去除html标签和转换html实体 |
| Mapping | 字符串替换操作 |
| Pattern Replace | 正则匹配替换 |

**Request：**

POST \_analyze

{

"tokenizer": "keyword",

"char\_filter": ["html\_strip"],

"text":"<div><h1>B<sup>+</sup>Trees</h1></div>"

}

**Response:**

{

"tokens": [

{

"token": """

B+Trees

""",

"start\_offset": 0,

"end\_offset": 38,

"type": "word",

"position": 0

}

]

}

##### Token Filter

对输出的单词（term）进行增加、删除、修改等操作

|  |  |
| --- | --- |
| Lowercase | 将所有term转换为小写 |
| stop | 删除stop words |
| NGram | 和Edge NGram连词分割 |
| Synonym | 添加近义词的term |

**Request:**

POST \_analyze

{

"tokenizer": "standard",

"text":"a Hello World",

"filter": [

"stop",

"lowercase",

{

"type":"ngram",

"min\_gram":3,

"max\_gram":4

}

]

}

**Response:**

{

"tokens": [

{

"token": "hel",

"start\_offset": 2,

"end\_offset": 7,

"type": "<ALPHANUM>",

"position": 1

},

{

"token": "hell",

"start\_offset": 2,

"end\_offset": 7,

"type": "<ALPHANUM>",

"position": 1

},

{

"token": "ell",

"start\_offset": 2,

"end\_offset": 7,

"type": "<ALPHANUM>",

"position": 1

},

{

"token": "ello",

"start\_offset": 2,

"end\_offset": 7,

"type": "<ALPHANUM>",

"position": 1

},

{

"token": "llo",

"start\_offset": 2,

"end\_offset": 7,

"type": "<ALPHANUM>",

"position": 1

},

{

"token": "wor",

"start\_offset": 8,

"end\_offset": 13,

"type": "<ALPHANUM>",

"position": 2

},

{

"token": "worl",

"start\_offset": 8,

"end\_offset": 13,

"type": "<ALPHANUM>",

"position": 2

},

{

"token": "orl",

"start\_offset": 8,

"end\_offset": 13,

"type": "<ALPHANUM>",

"position": 2

},

{

"token": "orld",

"start\_offset": 8,

"end\_offset": 13,

"type": "<ALPHANUM>",

"position": 2

},

{

"token": "rld",

"start\_offset": 8,

"end\_offset": 13,

"type": "<ALPHANUM>",

"position": 2

}

]

}

##### 自定义分词API

PUT my\_analyzer

{

"settings": {

"analysis": {

"analyzer": {

"my":{

"tokenizer":"punctuation",

"type":"custom",

"char\_filter":["emoticons"],

"filter":["lowercase","english\_stop"]

}

},

"tokenizer": {

"punctuation":{

"type":"pattern",

"pattern":"[.,!?]"

}

},

"char\_filter": {

"emoticons":{

"type":"mapping",

"mappings":[

":)=>\_happy\_",

":(=>\_sad\_"

]

}

},

"filter": {

"english\_stop":{

"type":"stop",

"stopwords":"\_english\_"

}

}

}

}

}

**测试**

POST my\_analyzer/\_analyze

{

"analyzer": "my",

"text":"l'm a :) person,and you?"

}

**结果**

{

"tokens": [

{

"token": "l'm a \_happy\_ person",

"start\_offset": 0,

"end\_offset": 15,

"type": "word",

"position": 0

},

{

"token": "and you",

"start\_offset": 16,

"end\_offset": 23,

"type": "word",

"position": 1

}

]

}

### 分词使用场景

1、索引时分词：创建或更新文档时，会对相应得文档进行分词(指定字段分词)

PUT my\_test

{

“mappings”:{

“doc”:{

“properties”:{

“title”:{

“type”:”text”,

“analyzer”:”ik\_smart”

}

}

}

}

}

2、查询时分词：查询时会对查询语句进行分词

POST my\_test/\_search

{

“query”:{

“match”:{

“message”:{

“query”:”hello”,

“analyzer”:”standard”

}

}

}

}

PUT my\_test

{

“mappings”:{

“doc”:{

“properties”:{

“title”:{

“type”:”text”,

“analyzer”:”whitespace”,

“search\_analyzer”:”standard” #查询指定分词器

}

}

}

}

}

一般不需要特别指定查询时分词器，直接使用索引时分词器即可，否则会出现无法匹配得情况，如果不需要分词将字段type设置成keyword，可以节省空间

### 中文分词

elasticsearch本身自带的中文分词就是单纯把中文一个字一个字的分开，根本没有词汇的概念。但是实际应用中，用户都是以词汇为条件，进行查询匹配的，如果能够把文章以词汇为单位切分开，那么与用户的查询条件能够更贴切的匹配上，查询速度也更加快速。

|  |  |  |  |
| --- | --- | --- | --- |
| **分词器名称** | **介绍** | **特点** | **地址** |
| IK | 实现中英文单词切分 | 自定义词库 | https://github.com/medcl/elasticsearch-analysis-ik |
| Jieba | python流行分词系统，支持分词和词性标注 | 支持繁体、自定义、并行分词 | http://github.com/sing1ee/elasticsearch-jieba-plugin |
| Hanlp | 由一系列模型于算法组成的java工具包 | 普及自然语言处理在生产环境中的应用 | https://github.com/hankcs/HanLP |
| THULAC | 清华大学中文词法分析工具包 | 具有中文分词和词性标注功能 | https://github.com/microbun/elasticsearch-thulac-plugin |

因此需要安装es分词的插件ik，其GitHub下载地址为：

https://github.com/medcl/elasticsearch-analysis-ik/releases

ik分词器分为ik\_smart（最少切分）和ik\_max\_word（最细粒度划分）两种，这两种分词器效果不同。

#### IK分词器

##### 默认情况

|  |
| --- |
| GET \_analyze{"text": "我是中国人"} |
| {  "tokens": [  {"token": "我","start\_offset": 0,"end\_offset": 1,"type": "<IDEOGRAPHIC>","position": 0},  {"token": "是","start\_offset": 1,"end\_offset": 2,"type": "<IDEOGRAPHIC>","position": 1},  {"token": "中","start\_offset": 2,"end\_offset": 3,"type": "<IDEOGRAPHIC>","position": 2},  {"token": "国","start\_offset": 3,"end\_offset": 4,"type": "<IDEOGRAPHIC>","position": 3},  {"token": "人","start\_offset": 4,"end\_offset": 5,"type": "<IDEOGRAPHIC>","position": 4}  ]  } |

##### 使用ik\_smart分词器

|  |
| --- |
| GET \_analyze{"text": "我是中国人","analyzer": "ik\_smart"} |
| {  "tokens": [  {"token": "我","start\_offset": 0,"end\_offset": 1,"type": "CN\_CHAR","position": 0},  {"token": "是","start\_offset": 1,"end\_offset": 2,"type": "CN\_CHAR","position": 1},  {"token": "中国人","start\_offset": 2,"end\_offset": 5,"type": "CN\_WORD","position": 2}  ]  } |

##### 使用ik\_max\_word分词器

|  |
| --- |
| GET \_analyze{"text": "我是中国人","analyzer": "ik\_max\_word"} |
| {  "tokens": [  {"token": "我","start\_offset": 0,"end\_offset": 1,"type": "CN\_CHAR","position": 0},  {"token": "是","start\_offset": 1,"end\_offset": 2,"type": "CN\_CHAR","position": 1},  {"token": "中国人","start\_offset": 2,"end\_offset": 5,"type": "CN\_WORD","position": 2},  {"token": "中国","start\_offset": 2,"end\_offset": 4,"type": "CN\_WORD","position": 3},  {"token": "国人","start\_offset": 3,"end\_offset": 5,"type": "CN\_WORD","position": 4}  ]  } |

#### 基于中文分词搭建索引（not now）

#### 自定义词库（not now）

修改/usr/share/elasticsearch/plugins/ik/config/中的IKAnalyzer.cfg.xml

|  |
| --- |
| <?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"></entry>  <!--用户可以在这里配置自己的扩展停止词字典-->  <entry key="ext\_stopwords"></entry>  <!--用户可以在这里配置远程扩展字典 -->  <entry key="remote\_ext\_dict">http://192.168.67.163/fenci/myword.txt</entry>  <!--用户可以在这里配置远程扩展停止词字典-->  <!-- <entry key="remote\_ext\_stopwords">words\_location</entry> -->  </properties> |

按照标红的路径利用nginx发布静态资源

在nginx.conf中配置

|  |
| --- |
| server {  listen 80;  server\_name 192.168.67.163;  location /fenci/ {  root es;  }  } |

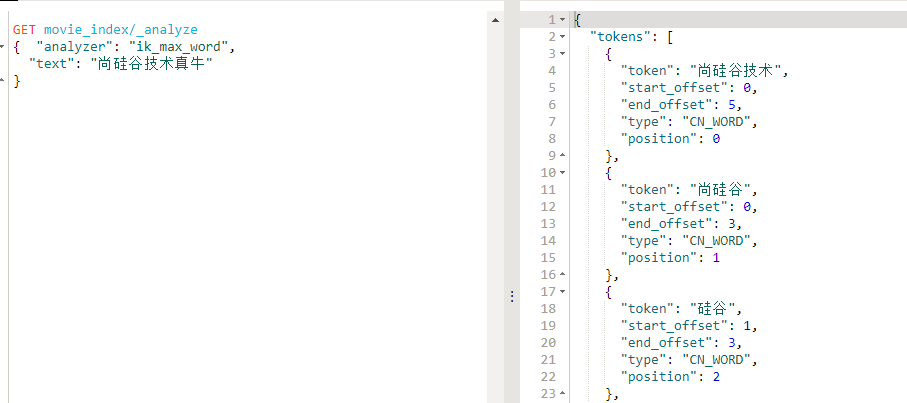
并且在/usr/local/nginx/下建/es/fenci/目录，目录下加myword.txt

myword.txt中编写关键词，每一行代表一个词。



然后重启es服务器，重启nginx。

在kibana中测试分词效果



更新完成后，es只会对新增的数据用新词分词。历史数据是不会重新分词的。如果想要历史数据重新分词。需要执行：

|  |
| --- |
| POST movies\_index\_chn/\_update\_by\_query?conflicts=proceed |

# 操作命令

## Mapping

Mapping用于定义数据库中表的结构，通过mapping来控制索引存储数据的设置，可以定义如下配置：

定义Index下的字段名（Field Name）

定义字段的类型，比如数值型、字符串型、布尔型等

定义倒排索引相关的配置，比如documentId、记录position、打分等

1、创建mapping

PUT my\_index1

{

"mappings": {

"doc":{

"properties":{

"username":{

"type": "text",

"fields": {

"pinyin":{

"type": "text"

}

}

}

}

}

}

}

2、创建文档

PUT my\_index1/doc/1

{

"username":"haha heihei"

}

3、查询

GET my\_index1/\_search

{

"query": {

"match": {

"username.pinyin": "haha"

}

}

}

Dynamic Mapping

es可以自动识别文档字段类型，从而降低用户使用成本

PUT /test\_index/doc/1

{

"username":"alfred",

"age":1

}

{

"test\_index": {

"mappings": {

"doc": {

"properties": {

"age": {

"type": "long"

},

"username": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

}

}

}

}

}

}

age自动识别为long类型，username识别为text类型

PUT test\_index/doc/1

{

"username":"samualz",

"age":14,

"birth":"1991-12-15",

"year":18,

"tags":["boy","fashion"],

"money":"100.1"

}

{

"test\_index": {

"mappings": {

"doc": {

"properties": {

"age": {

"type": "long"

},

"birth": {

"type": "date"

},

"money": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

},

"tags": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

},

"username": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

},

"year": {

"type": "long"

}

}

}

}

}

}

日期的自动识别可以自行配置日期格式，以满足各种需求

1. 自定义日期识别格式

PUT my\_index

{

"mappings":{

"doc":{

"dynamic\_date\_formats": ["yyyy-MM-dd","yyyy/MM/dd"]

}

}

}

2、关闭日期自动识别

PUT my\_index

{

"mappings": {

"doc": {

"date\_detection": false

}

}

}

字符串是数字时，默认不会自动识别为整形，因为字符串中出现数字时完全合理的，Numeric\_datection可以开启字符串中数字的自动识别。

PUT my\_index

{

"mappings":{

"doc":{

"numeric\_datection": true

}

}

}

### 默认配置

不进行配置时，自动创建的mapping

请求：

GET /atguigu/\_mapping

响应：

{

"movie": { #索引名称

"mappings": { #mapping设置

"student": { #type名称

"properties": { #字段属性

"clazz": {

"type": "text", #字段类型，字符串默认类型

"fields": { #子字段属性设置

"keyword": { #分词类型（不分词）

"type": "keyword",

"ignore\_above": 256

}

}

},

"description": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

},

"name": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

}

}

}

}

}

}

### 自定义

请求：

PUT my\_index #索引名称

{

"mappings":{

"doc":{ #类型名称

"dynamic":false,

"properties":{

"title":{

"type":"text" #字段类型

},

"name":{

"type":"keyword"

},

"age":{

"type":"integer"

}

}

}

}

}

响应：

{

"acknowledged": true,

"shards\_acknowledged": true,

"index": "my\_index"

}

### Dynamic Mapping

es依靠json文档字段类型来实现自动识别字段类型，支持的类型

|  |  |
| --- | --- |
| **JSON类型** | **es类型** |
| null | 忽略 |
| Boolean | boolean |
| 浮点类型 | float |
| 整数 | long |
| Object{} | object |
| Array[] | 由第一个非null值的类型决定 |
| String”” | 匹配为日期则设为data类型（默认开启），匹配为数字的话设为float或long类型（默认关闭），设为text类型，并附带keyword的子字段 |

**注意：**

mapping中的字段类型一旦设定后，禁止修改，因为Lucene实现的倒排索引生成后不允许修改(提高效率)。如果要修改字段的类型，需要从新建立索引，然后做reindex操作。

#### dynamic

可以设置在type下，也可以设置在字段中（object类型的字段中）

true：允许自动新增字段（默认的配置）

False：不允许自动新增字段，但是文档可以正常写入，无法对字段进行查询操作

strict：文档不能写入（如果写入会报错）

例如：

put my\_index

{

“mappings”:{

“doc”:{

“dynamic”:false,

“properties”:{

“user”:{

“properties”:{

“name”:{

“type”:”text”

},”social\_networks”:{

“dynamic”:true,

“properties”:{}

}

}

}

}

}

}

}

#### copy\_to

将该字段的值复制到目标字段，实现\_all的作用

不会出现在\_source中，只用来搜索

put my\_index

{

“mappings”:{

“doc”:{

“properties”:{

“frist\_name”:{

“type”:”text”,

“cope\_to”:”full\_name”

},”last\_name”:{

“type”:”text”,

“cope\_to”:“full\_name”

},”full\_name”:{

“type”:”text”

}

}

}

}

}

put my\_index/doc

{

“frist\_name”:”John”,

“last\_name”:”Smith”

}

GET my\_index/doc

{

“query”:{

“match”:{

“full\_name”:”John Smith”,

“operator”:”and”

}

}

}

#### Index

Index属性控制当前字段是否是索引，默认为true，即记录索引，false不记录，即不可以搜索，比如：手机号、身份证号等敏感信息，不希望被检索

例如：

创建mapping

PUT my\_index

{

"mappings": {

"doc":{

"properties": {

"cookie":{

"type":"text",

"index": false

}

}

}

}

}

创建文档

PUT my\_index/doc/1

{

"cookie":"123",

"name":"home"

}

查询

GET my\_index/\_search #报错

{

"query": {

"match": {

"cookie":"123"

}

}

}

GET my\_index/\_search #有结果

{

"query": {

"match": {

"name":"home"

}

}

}

#### Index\_options

用于控制倒排索引记录的内容，有如下4中配置

docs：只记录docid

freqs：记录docid和term frequencies（词频）

position：记录docid、term frequencies、term position

Offsets：记录docid、term frequencies、term position、character offsets

text类型默认配置为position，其默认认为docs，记录的内容越多，占用的空间越大。

## 查询命令

**搜索es中所有索引**

GET /\_cat/indices?v



health green表示集群完整，yellow表示单点正常，集群不完整，red表示单点不正常

status 表示索引状态，是否能够使用

index 索引名

uuid 索引统一编号

pri 主节点个数

rep 从节点个数

docs.count 文档个数

docs.deleted 文档被删个数

store.size 整体占空间大小

pri.store.size 主节点占空间大小

**多索引查询**

GET index1,index2/\_search

**使用type的id查找**

GET index/type/id

通过HTTP方法GET来检索文档，同样的，可以使用DELETE方法删除文档，使用HEAD方法检查某文档是否存在。如果想更新已存在的文档，只需再PUT一次。

**搜索type全部数据**

GET movie\_chn/movie/\_search

响应内容不仅会显示哪些文档被匹配到，而且这些文档内容完整的被包含在其中，给用户展示搜索结果时需要用到的所有信息都有了。

|  |
| --- |
| {  "took": 1,//耗费时间，单位为毫秒  "timed\_out": false,//是否超时  "\_shards": {  "total": 5,//发送给全部5个分片  "successful": 5,  "skipped": 0,  "failed": 0  },  "hits": {  "total": 4,//搜索出4个结果  "max\_score": 1,//最大评分  "hits": [//结果  {  "\_index": "movie\_chn",  "\_type": "movie",  "\_id": "2",  "\_score": 1,  "\_source": {  "id": 2,  "name": "湄公河行动",  "doubanScore": 8,  "actorList": [  {  "id": 3,  "name": "张涵予"  }  ]  }  }，  ...  ]  }  } |

**按条件搜索全部**

|  |
| --- |
| GET /movie\_chn/movie/\_search  {  "query": {  "match\_all": {}  }  } |

**按分词查询**

|  |
| --- |
| GET /movie\_chn/movie/\_search  {  "query": {  "match": {  "name": "红海湄公"  }  }  } |

**按分词子属性查询**

|  |
| --- |
| GET /movie\_chn/movie/\_search  {  "query": {  "match": {  "actorList.name": "海清"  }  }  } |

**按短语查询**

不再使用分词技术，直接用短语在原始数据中匹配

|  |
| --- |
| GET /movie\_chn/movie/\_search  {  "query": {  "match\_phrase": {  "name": "红海湄公"  }  }  } |

**fuzzy模糊查询**

校正匹配分词，当一个单词都无法准确匹配，es通过一种算法对非常接近的单词也给与一定的评分，能够查询出来，但是消耗更多的性能。

|  |
| --- |
| GET /movie\_chn/movie/\_search  {  "query": {  "fuzzy": {  "name": "公"  }  }  } |

**过滤-查询后过滤**

|  |
| --- |
| GET /movie\_chn/movie/\_search  {  "query": {  "match": {  "name": "红海"  }  },  "post\_filter": {  "term": {  "actorList.name": "译"  }  }  } |

**过滤-查询前过滤（推荐使用）**

|  |
| --- |
| GET /movie\_chn/movie/\_search  {  "query": {  "bool": {  "filter": {  "term":{  "actorList.id":3  }  },  "must": [  {  "match": {  "name": "湄公河"  }  }  ]  }  }  } |

**过滤-按范围过滤**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| gt | 大于 | lt | 小于 | gte | 大于等于 | lte | 小于等于 |

|  |
| --- |
| GET movie\_chn/movie/\_search  {  "query": {  "bool": {  "filter": {  "range": {  "doubanScore": {  "gte": 6,  "lte": 8  }  }  }  }  }  } |

**排序**

asc表示升序。desc表示降序

|  |
| --- |
| GET movie\_chn/movie/\_search  {  "query": {  "match": {  "name": "行动"  }  },  "sort": [  {  "doubanScore": {  "order": "asc"  }  }  ]  } |

**分页查询**

|  |
| --- |
| GET movie\_chn/movie/\_search  {  "query": {  "match\_all": {}  },  "from": 0,  "size": 2  } |

**指定字段查询**

|  |
| --- |
| GET movie\_chn/movie/\_search  {  "query": {"match\_all": {}},  "\_source": ["name","id"]  } |

**将查询结果高亮**

|  |
| --- |
| GET movie\_chn/movie/\_search  {  "query": {  "match": {  "name": "行动"  }  },  "highlight": {  "fields": {  "name": {}  }  }  } |

## 增删改命令

**增加一个索引（库）**

PUT /movie\_index

**删除一个索引（库）**

DELETE /movie\_index

**新增文档**

PUT /index/type/id

新增文档时如果之前没有建过index或type，es会自动创建。

|  |
| --- |
| PUT /movie\_chn/movie/1  {  "id": 1,  "name": "红海行动",  "doubanScore": 8.5,  "actorList": [  {  "id": 1,  "name": "张译"  },  {  "id": 2,  "name": "海清"  },  {  "id": 3,  "name": "张涵予"  }  ]  } |

**修改（整体）替换**

此时和新增一样

|  |
| --- |
| PUT /movie\_chn/movie/1  {  "id": 1,  "name": "湄公河行动",  "doubanScore": 9,  "actorList": [  {  "id": 1,  "name": "张译"  }  ]  } |

**修改某个字段**

|  |
| --- |
| POST /movie\_chn/movie/4/\_update  {  "doc": {  "id": 4  }  } |

**删除一个document**

DELETE movie\_index/movie/3

## 文档操作

### 创建文档

文档通过index API被索引，使数据可以被存储和搜索。文档通过其\_index、\_type、\_id唯一确定，\_id可以自定义，也可以使用index API 生成。

PUT {index}/{type}/{id}

{

“”:””

}

**自定义ID**

如果文档有自然的标识符（例如user\_account字段或者其他值表示文档），就可以自定义\_id，使用这种形式的index API：

PUT /{index}/{type}/{id}

{

"field": "value",

...

}

例如索引为“website”，类型叫做“blog”，自定义ID是“123”，那么这个索引如下所示：

PUT /website/blog/123

{

"title": "My first blog entry",

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

"date": "2014/01/01"

}

Elasticsearch的响应：

{

"\_index": "website",

"\_type": "blog",

"\_id": "123",

"\_version": 1,

"created": true

}

Elasticsearch中每个文档都有版本号，每当文档变化（包括删除）都会使\_version增加，\_version确保程序的一部分不会覆盖掉另一部分所做的更改。

**自增ID**

如果没有自定义ID，可以让Elasticsearch自动生成。PUT方法（在这个URL中存储文档）变成了POST方法（在这个类型下存储文档）。原来是把文档存储到某个ID对应的空间，现在是把这个文档添加到某个\_type下）。

URL现在只包含\_index和\_type两个字段：

POST /website/blog/

{

"title": "My second blog entry",

"text": "Still trying this out...",

"date": "2014/01/01"

}

响应内容与刚才类似，只有\_id字段变成了自动生成的值：

{

"\_index": "website",

"\_type": "blog",

"\_id": "wM0OSFhDQXGZAWDf0-drSA",

"\_version": 1,

"created": true

}

自动生成的ID有22个字符长，URL-safe, Base64-encoded string universally unique identifiers, 或者叫 [UUIDs](http://en.wikipedia.org/wiki/Uuid)。

### 获取文档

使用同样的\_index、\_type、\_id从Elasticsearch中获取文档，但是HTTP方法改为GET

GET /website/blog/123?pretty

响应包含元数据节点，增加了\_source字段，\_source字段包含在创建索引时发送给Elasticsearch的原始文档。

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 1,

"found" : true,

"\_source" : {

"title": "My first blog entry",

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

"date": "2014/01/01"

}

}

#### pretty

在查询字符串中增加pretty参数，会让Elasticsearch美化输出(pretty-print)JSON响应以便更加容易阅读。\_source字段不会被美化，与输入的一致。

GET请求返回的响应内容包括{"found": true}。这意味着文档已经找到。如果我们请求一个不存在的文档，依旧会得到一个JSON，不过found值变成了false。此外，HTTP响应状态码也会变成'404 Not Found'代替'200 OK'。我们可以在curl后加-i参数得到响应头：

curl -i -XGET http://localhost:9200/website/blog/124?pretty

现在响应类似于这样：

HTTP/1.1 404 Not Found

Content-Type: application/json; charset=UTF-8

Content-Length: 83

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "124",

"found" : false

}

#### 检索部分文档

请求个别字段可以使用\_source参数，多个字段可以使用逗号分隔：

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

\_source字段现在只包含我们请求的字段，而且过滤了date字段：

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 1,

"exists" : true,

"\_source" : {

"title": "My first blog entry" ,

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

}

}

或者只想得到\_source字段而不要其他的元数据，可以请求：

GET /website/blog/123/\_source

它仅仅返回:

{

"title": "My first blog entry",

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

"date": "2014/01/01"

}

### 更新文档

POST /website/blog/123

{

"title": "My first blog entry",

"text": "I am starting to get the hang of this...",

"date": "2014/01/02"

}

在响应中，可以看到\_version增加了。

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 2,

"created": false <1>

}

### 删除文档

删除文档的语法模式与之前基本一致，只不过要使用DELETE方法：

DELETE /website/blog/123

### 局部更新

POST /website/blog/1/\_update

{

"doc" : {

"tags" : [ "testing" ],

"views": 0

}

}

如果请求成功，将看到类似index请求的响应结果：

{

"\_index" : "website",

"\_id" : "1",

"\_type" : "blog",

"\_version" : 3

}

检索文档文档显示被更新的\_source字段：

{

"\_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>

}

}

### 批量插入

每个json之间不能有换行\n

POST test\_search\_index/doc/\_bulk

{

"index":{

"\_id":1

}

}

{

"username":"alfred way",

"job":"java engineer",

"age":18,

"birth":"1991-12-15",

"isMarried":false

}

{

"index":{

"\_id":2

}

}

{

"username":"alfred",

"job":"java senior engineer and java specialist",

"age":28,

"birth":"1980-05-07",

"isMarried":true

}

{

"index":{

"\_id":3

}

}

{

"username":"lee",

"job":"java and ruby engineer",

"age":22,

"birth":"1985-08-07",

"isMarried":false

}

### 检索多个文档

Elasticsearch检索多个文档非常快，合并多个请求可以避免每个请求单独的网络开销。如果需要从Elasticsearch中检索多个文档，相对于一个一个的检索，更快的方式是在一个请求中使用multi-get或者mget API。

mget API参数是一个docs数组，数组的每个节点定义一个文档的\_index、\_type、\_id元数据。如果你只想检索一个或几个确定的字段，也可以定义一个\_source参数：

POST /\_mget

{

"docs" : [

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : 2

},

{

"\_index" : "website",

"\_type" : "pageviews",

"\_id" : 1,

"\_source": "views"

}

]

}

响应也包含一个docs数组，每个文档还包含一个响应，并按照请求定义的顺序排列。每个这样的响应与单独使用get request响应体相同：

{

"docs" : [

{

"\_index" : "website",

"\_id" : "2",

"\_type" : "blog",

"found" : true,

"\_source" : {

"text" : "This is a piece of cake...",

"title" : "My first external blog entry"

},

"\_version" : 10

},

{

"\_index" : "website",

"\_id" : "1",

"\_type" : "pageviews",

"found" : true,

"\_version" : 2,

"\_source" : {

"views" : 2

}

}

]

}

如果检索的文档在同一个\_index中（甚至在同一个\_type中），你就可以在URL中定义一个默认的/\_index或者/\_index/\_type。

可以通过简单的ids数组来代替完整的docs数组：

POST /website/blog/\_mget

{

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

}

请求的第二个文档并不存在，定义了类型为blog，但是ID为1的文档类型为pageviews。这个不存在的文档会在响应体中被告知。

{

"docs" : [

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "2",

"\_version" : 10,

"found" : true,

"\_source" : {

"title": "My first external blog entry",

"text": "This is a piece of cake..."

}

},

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "1",

"found" : false <1>

}

]

}

## 其他

Liunx执行：curl 'http://localhost:9200/?pretty'

浏览器访问：http://localhost:9200/?pretty

# 状态查看命令

语法：ip:post/\_cat/[args](?v|?format=json&pretty)

（?v表示显示字段说明,?format=json&pretty表示显示成json格式）

查看所有索引

GET \_cat/indices?v

查看es集群状态

GET \_cat/health?v

GET /\_search #查询所有索引文档

GET /my\_index/\_search #查询指定索引文档

GET /my\_index1,my\_index2/\_search #多索引查询

GET /my\_\*/\_search

2019-03-xxx

2019-04-vvv

2019-05-xxx

### URI查询方式（查询有限制，很多配置不能实现）

GET /my\_index/\_search?q=user:alfred #指定字段查询

GET /my\_index/\_search?q=keyword&df=user&sort=age:asc&from=4&size=10&timeout=1s

q : 指定查询的语句，例如q=aa或q=user:aa

df:q中不指定字段默认查询的字段，如果不指定，es会查询所有字段

Sort：排序，asc升序，desc降序

timeout：指定超时时间，默认不超时

from，size：用于分页

### term与phrase

term相当于单词查询，phrase相当于词语查询

term：Alfred way等效于alfred or way

phrase：”Alfred way” 词语查询，要求先后顺序

### 泛查询

Alfred等效于在所有字段去匹配该term(不指定字段查询)

指定字段

name:alfred

Group分组设定（），使用括号指定匹配的规则

（quick OR brown）AND fox：通过括号指定匹配的优先级

status:(active OR pending) title:(full text search)：把关键词当成一个整体

GET test\_search\_index/\_search?q=alfred

查询语句执行计划查看

GET test\_search\_index/\_search?q=alfred

{

"profile":true

}

term查询

GET test\_search\_index/\_search?q=username:alfred way #alfred OR way

phrase查询

GET test\_search\_index/\_search?q=username:"alfred way"

group查询

GET test\_search\_index/\_search?q=username:(alfred OR way)

### 布尔操作符

AND(&&),OR(||),NOT(!)

name:(tom NOT lee) #表示name字段中可以包含tom但一定不包含lee

+、-分别对应must和must\_not

name:(tom +lee -alfred) #表示name字段中，一定包含lee，一定不包含alfred，可以包含tom

注意：+在url中会被解析成空格，要使用encode后的结果才可以，为%2B

GET test\_search\_index/\_search?q=username:(alfred %2Bway)

### 范围查询，支持数值和日期

区间：闭区间：[]，开区间:{}

age:[1 TO 10] #1<=age<=10

age:[1 TO 10} #1<=age<10

age:[1 TO ] #1<=age

age:[\* TO 10] #age<=10

算术符号写法

age:>=1

age:(>=1&&<=10)或者age:(+>=1 +<=10)

### 通配符查询

?:1个字符

\*:0或多个字符

例如：name:t?m

name:tom\*

name:t\*m

注意：通配符匹配执行效率低，且占用较多内存，不建议使用，如无特殊要求，不要讲?/\*放在最前面

### 正则表达式

name:/[mb]oat/

模糊匹配fuzzy query

name:roam~1 [0,1,2]

匹配与roam差1个character的词，比如foam、roams等

### 近似度查询

proximity search

“fox quick”~5

以term为单位进行差异比较，比如”quick fox” “quick brown fox”

### 聚合

**开启聚合**

PUT gmall/\_mapping/SkuInfo/

{

"properties": {

"skuAttrValueList.valueId": {

"type": "text",

"fielddata": true

}

}

}

**聚合**

"aggregations" : {

"groupby\_attr" : {

"terms" : {

"field" : "skuAttrValueList.valueId"

}

}

}

**聚合效果**

"aggregations": {

"groupby\_attr": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 31,

"buckets": [

{

"key": "51",

"doc\_count": 63

},

{

"key": "43",

"doc\_count": 60

},

{

"key": "48",

"doc\_count": 60

},

{

"key": "39",

"doc\_count": 59

},

{

"key": "41",

"doc\_count": 13

},

{

"key": "54",

"doc\_count": 13

},

{

"key": "59",

"doc\_count": 12

},

{

"key": "45",

"doc\_count": 9

},

{

"key": "50",

"doc\_count": 8

},

{

"key": "52",

"doc\_count": 7

}

]

}

}

# ElasticSearch集群

一个节点(node)就是一个Elasticsearch实例，而一个集群(cluster)由一个或多个节点组成，它们具有相同的cluster.name，它们协同工作，分享数据和负载。当加入新的节点或者删除一个节点时，集群就会感知到并平衡数据（同步）。

## 集群概念

### 集群节点

1、集群中一个节点会被选举为主节点(master)

2、主节点临时管理集群级别的一些变更，例如新建或删除索引、增加或移除节点等

3、主节点不参与文档级别的变更或搜索，这意味着在流量增长的时候，该主节点不会成为集群的瓶颈

4、任何节点都可以成为主节点。

5、用户能够与集群中的任何节点通信，包括主节点

6、每一个节点都知道文档存在于哪个节点上，它们可以转发请求到相应的节点上。

7、访问的节点负责收集各节点返回的数据，最后一起返回给客户端。这一切都由Elasticsearch处理

### 集群健康

在Elasticsearch集群中可以监控统计很多信息，只有集群健康(cluster health)是最重要的，集群健康有三种状态：green、yellow或red。status字段提供一个综合的指标来表示集群的的服务状况，三种颜色各自的含义：

| **颜色** | **意义** |
| --- | --- |
| green | 所有主要分片和复制分片都可用 |
| yellow | 所有主要分片可用，但不是所有复制分片都可用 |
| red | 不是所有的主要分片都可用 |

### 集群分片

索引只是一个用来指向一个或多个分片(**shards**)的“逻辑命名空间(**logical namespace**)”。

分片(**shard**)是一个最小级别“工作单元(**worker unit**)”,它只是保存了索引中所有数据的一部分，是一个Lucene实例，并且它本身就是一个完整的搜索引擎。文档存储在分片中，并且在分片中被索引，但是应用程序不会直接与它们通信，而是直接与索引通信。

分片是Elasticsearch在集群中分发数据的关键。把分片想象成数据的容器。文档存储在分片中，然后分片分配到你集群中的节点上。当你的集群扩容或缩小，Elasticsearch将会自动在你的节点间迁移分片，以使集群保持平衡。

**主分片**

索引中的每个文档属于一个单独的主分片，所以主分片的数量决定了索引最多能存储多少数据。

理论上主分片能存储的数据大小是没有限制的，限制取决于你实际的使用情况。分片的最大容量完全取决于你的使用状况：硬件存储的大小、文档的大小和复杂度、如何索引和查询你的文档，以及你期望的响应时间。

**副分片**

复制分片只是主分片的一个副本，它可以防止硬件故障导致的数据丢失，同时可以提供读请求，比如搜索或者从别的shard取回文档。

当索引创建完成的时候，主分片的数量就固定了，但是复制分片的数量可以随时调整。

创建分片：

PUT /blogs

{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

}

}

增加副分片：

PUT /blogs/\_settings

{

"number\_of\_replicas" : 2

}

主分片设置后不能进行修改，只能修改副本分片

集群的健康状态yellow表示所有的主分片(**primary shards**)启动并且正常运行了——集群已经可以正常处理任何请求——但是复制分片(**replica shards**)还没有全部可用。事实上所有的三个复制分片现在都是unassigned状态——它们还未被分配给节点。在同一个节点上保存相同的数据副本是没有必要的，如果这个节点故障了，那所有的数据副本也会丢失。

### 故障转移

在单一节点上运行意味着有单点故障的风险——没有数据备份。要防止单点故障，唯一需要做的就是启动另一个节点。

第二个节点已经加入集群，三个复制分片(replica shards)也已经被分配了——分别对应三个主分片，这意味着在丢失任意一个节点的情况下依旧可以保证数据的完整性。

文档的索引将首先被存储在主分片中，然后并发复制到对应的复制节点上。这可以确保我们的数据在主节点和复制节点上都可以被检索。

## 集群操作原理

### 路由

ElasticSearch通过算法决定文档属于哪个分片以及创建新文档时要存储在哪个分片上：

shard = hash(routing) % number\_of\_primary\_shards

routing值是一个任意字符串，它默认是\_id，但也可以自定义。所有的文档API（get、index、delete、bulk、update、mget）都接收一个routing参数，它用来自定义文档到分片的映射。自定义路由值可以确保所有相关文档——例如属于同一个人的文档——被保存在同一分片上。

主分片的数量只能在创建索引时定义且不能修改，是因为如果主分片的数量在未来改变了，所有先前的路由值就失效了，文档也就永远找不到了。

### 操作数据节点工作流程

每个节点都有能力处理任意请求。每个节点都知道任意文档所在的节点，所以也可以将请求转发到需要的节点。

新建、索引和删除请求都是写(write)操作，它们必须在主分片上成功完成才能复制到相关的复制分片上。



1. 客户端给Node 1发送新建、索引或删除请求。
2. 节点使用文档的\_id确定文档属于分片0。它转发请求到Node 3，分片0位于这个节点上。
3. Node 3在主分片上执行请求，如果成功，它转发请求到相应的位于Node 1和Node 2的复制节点上。当所有的复制节点报告成功，Node 3报告成功到请求的节点，请求的节点再报告给客户端。

**replication**

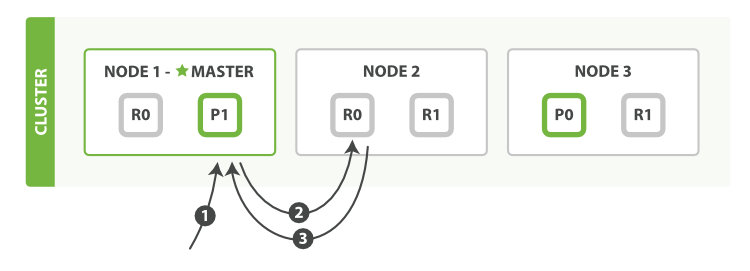
复制默认的值是sync。这将导致主分片得到复制分片的成功响应后才返回。

如果你设置replication为async，请求在主分片上被执行后就会返回给客户端。它依旧会转发请求给复制节点，但你将不知道复制节点成功与否。

上面的这个选项不建议使用。默认的sync复制允许Elasticsearch强制反馈传输。async复制可能会因为在不等待其它分片就绪的情况下发送过多的请求而使Elasticsearch过载。

### 检索流程

文档能够从主分片或任意一个复制分片被检索。



1. 客户端给Node 1发送get请求。
2. 节点使用文档的\_id确定文档属于分片0。分片0对应的复制分片在三个节点上都有。此时，它转发请求到Node 2。
3. Node 2返回文档(document)给Node 1然后返回给客户端。

对于读请求，为了平衡负载，请求节点会为每个请求选择不同的分片——它会循环所有分片副本。

可能的情况是，一个被索引的文档已经存在于主分片上却还没来得及同步到复制分片上。这时复制分片会报告文档未找到，主分片会成功返回文档。一旦索引请求成功返回给用户，文档则在主分片和复制分片都是可用的。

# 安装和配置

## 安装ES单机版

### 安装ES

1、将相关压缩包上传到Linux虚拟机中

cerebro.zip集群管理工具

elasticsearch.tar.gz主程序

elasticsearch-analysis-ik.rar中文分词器

kibana.tar.gz可视化第三方工具

2、解压主程序压缩包并修改解压后的目录中所有文件的权限

|  |
| --- |
| [root@xmm opt]# tar -zxvf elasticsearch-6.3.1.tar.gz  [root@xmm opt]# chmod 777 -R elasticsearch-6.3.1 |

3、配置解压目录下的config目录下的jvm.options文件，在测试环境中可以将-Xms1g修改的小一些，但是如果其值过小会导致es功能无法完全开启。（此步可选）

|  |
| --- |
| [root@xmm opt]# cd elasticsearch-6.3.1/config/  [root@xmm config]# vim jvm.options  ...  # Xms represents the initial size of total heap space  # Xmx represents the maximum size of total heap space  -Xms1g  -Xmx1g  ... |

4、因为es6无法使用root用户来启动，因此需要切换用户，如果没有其他用户则可以使用adduser来添加用户，然后使用su来切换用户。如果想要切换回root用户也是使用su命令并需要输入root用户的密码。

|  |
| --- |
| [root@xmm bin]# adduser es  [root@xmm bin]# su es  [es@xmm bin]$ su root  密码：  [root@xmm bin]# |

5、进入解压目录下的bin目录启动es，启动时会提醒配置过低，导致外网访问的功能没有开启；es默认是以集群的方式启动的，启动时会将自身设置为master。

可以在启动命令后添加参数 -d设置为后台启动，即 ./elasticsearch -d

|  |
| --- |
| [es@xmm bin]$ ./elasticsearch  ...  publish\_address {127.0.0.1:9300}, bound\_addresses {[::1]:9300}, {127.0.0.1:9300}  ...  publish\_address {127.0.0.1:9200}, bound\_addresses {[::1]:9200}, {127.0.0.1:9200}  ...  max file descriptors [4096] for elasticsearch process is too low, increase to at least [65536]  max virtual memory areas vm.max\_map\_count [65530] is too low, increase to at least [262144]  ...  [o.e.c.s.ClusterApplierService] [oNJM644] new\_master |

6、在内网中输入curl命令来访问9200端口，测试是否启动成功并查看es相关信息；此时如果访问9300端口，会提醒this is not a http port。

|  |
| --- |
| [root@xmm config]# curl http://127.0.0.1:9200  {  "name" : "oNJM644",  "cluster\_name" : "elasticsearch",  "cluster\_uuid" : "WiyUDncSQfyZLrj8XY70Og",  "version" : {  "number" : "6.3.1",  "build\_flavor" : "default",  "build\_type" : "tar",  "build\_hash" : "eb782d0",  "build\_date" : "2018-06-29T21:59:26.107521Z",  "build\_snapshot" : false,  "lucene\_version" : "7.3.1",  "minimum\_wire\_compatibility\_version" : "5.6.0",  "minimum\_index\_compatibility\_version" : "5.0.0"  },  "tagline" : "You Know, for Search"  }  [root@xmm config]# curl http://127.0.0.1:9300  This is not a HTTP port[root@xmm config]# |

7、修改config目录下的elasticsearch.yml配置文件中的IP地址为虚拟机的IP地址。

|  |
| --- |
| [root@xmm config]# cp elasticsearch.yml elasticsearch.yml.bak  [root@xmm config]# vim elasticsearch.yml  # ---------------------------------- Network -----------------------------------  #  # Set the bind address to a specific IP (IPv4 or IPv6):  #  network.host: 192.168.235.23 |

8、修改完IP地址后启动elasticsearch会报错，并提醒内存不足。接下来修改Linux虚拟机的配置，该配置文件是位于/etc/security目录下的limits.conf。配置完成后使用source命令刷新配置。

nofile 打开文件的最大数目

nproc 进程的最大数目

soft 指的是当前系统生效的设置值

hard 表明系统中所能设定的最大值

|  |
| --- |
| [root@xmm security]# cp limits.conf limits.conf.bak  [root@xmm security]# vim limits.conf  ...  #@student - maxlogins 4  \* hard nofile 655360  \* soft nofile 131072  \* hard nproc 4096  \* soft nproc 2048  # End of file  [root@xmm security]# source /etc/security/limits.conf  bash: access.conf: 未找到命令...  bash: access.conf: 未找到命令...  bash: access.conf: 未找到命令...  bash: access.conf: 未找到命令... |

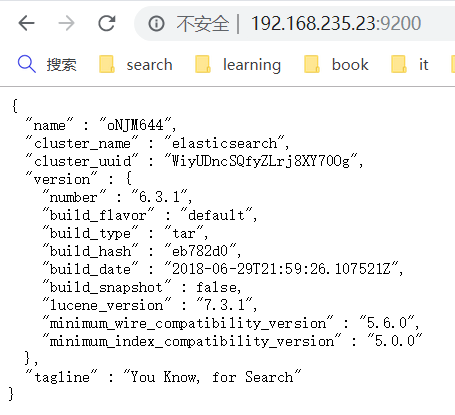
9、配置位于/etc目录下的sysctl.conf配置文件，配置完成后使用sysctl -p命令刷新配置。

vm.max\_map\_count=655360 此缺省配置下，单个jvm能开启的最大线程数为其一半

file-max 设置系统所有进程一共可以打开的文件数量

|  |
| --- |
| [root@xmm etc]# cp sysctl.conf sysctl.conf.bak  [root@xmm etc]# vim sysctl.conf  # For more information, see sysctl.conf(5) and sysctl.d(5).  vm.max\_map\_count=655360  fs.file-max=655360  [root@xmm etc]# sysctl -p  vm.max\_map\_count = 655360  fs.file-max = 655360 |

10、在外网的浏览器地址栏中输入http://192.168.235.23:9200/并访问，测试是否启动成功，浏览器中显示elasticsearch相关信息则表示启动成功。



11、配置开机自启，在/etc/init.d目录下通过vim创建elasticsearch文件，并输入如下内容

|  |
| --- |
| #!/bin/sh  #chkconfig: 2345 80 05  #description: elasticsearch  export JAVA\_HOME=/opt/jdk1.8.0\_121  export JAVA\_BIN=/opt/jdk1.8.0\_121/bin  export PATH=$PATH:$JAVA\_HOME/bin  export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  export JAVA\_HOME JAVA\_BIN PATH CLASSPATH  case "$1" in  start)  su es<<!  cd /opt/elasticsearch-6.3.1  ./bin/elasticsearch -d  !  echo "elasticsearch startup"  ;;  stop)  es\_pid=`ps aux|grep elasticsearch | grep -v 'grep elasticsearch' | awk '{print $2}'`  kill -9 $es\_pid  echo "elasticsearch stopped"  ;;  restart)  es\_pid=`ps aux|grep elasticsearch | grep -v 'grep elasticsearch' | awk '{print $2}'`  kill -9 $es\_pid  echo "elasticsearch stopped"  su es<<!  cd /opt/elasticsearch-6.3.1  ./bin/elasticsearch -d  !  echo "elasticsearch startup"  ;;  \*)  echo "start|stop|restart"  ;;  esac  exit $? |

修改elasticsearch自启动脚本权限并将该脚本注册到服务中，然后启动该服务。

|  |
| --- |
| [root@xmm init.d]# chmod 777 elasticsearch  [root@xmm init.d]# chkconfig --add elasticsearch  [root@xmm init.d]# chkconfig elasticsearch on |

### 安装Kibana

kibana对于elasticsearch就像SQLyog和Navicat for MySQL对于MySQL，是便于用户使用的图形化界面。

1、解压kibana的压缩包

|  |
| --- |
| [root@xmm opt]# tar -zxvf kibana-6.3.1-linux-x86\_64.tar.gz |

2、修改kibana的配置文件，具体修改位置如下所示

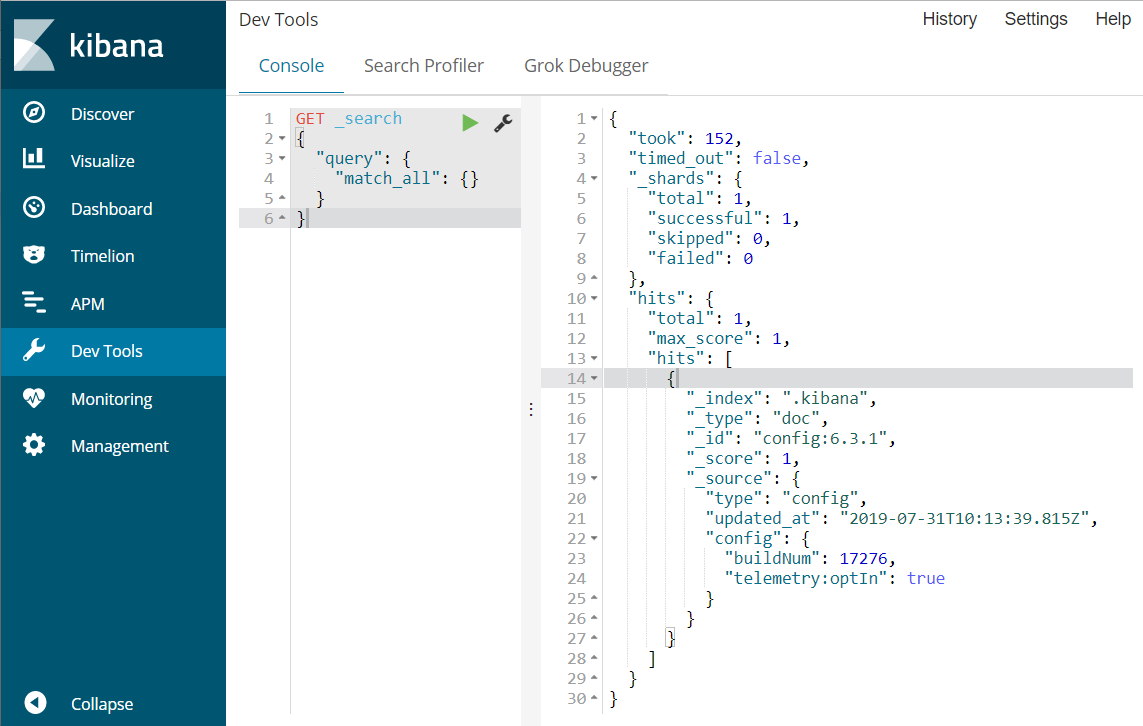
|  |
| --- |
| [root@xmm config]# cp kibana.yml kibana.yml.bak  [root@xmm config]# vim kibana.yml  ...  # To allow connections from remote users, set this parameter to a non-loopback address.  server.host: "0.0.0.0"  ...  # The URL of the Elasticsearch instance to use for all your queries.  elasticsearch.url: "http://192.168.235.23:9200" |

3、启动kibana，显示Server running at http://0.0.0.0:5601则表示启动成功

|  |
| --- |
| [root@xmm opt]# cd kibana-6.3.1-linux-x86\_64/bin/  [root@xmm bin]# ./kibana  ...  [info][listening] Server running at http://0.0.0.0:5601 |

4、在浏览器地址栏中输入http://192.168.235.23:5601访问kibana图形化界面，然后在侧边栏中选择Dev Tools，在Console中开始编写dsl语句。

执行get \_cluster/health，在右边的结果中的status为yellow或green则表示elasticsearch启动正常，并且与kibana连接正常。



### 安装ik

es会对所有存入的数据进行分词处理(分出的单词有位置、长度、出现频率等属性)。分词的目的在于将来在进行关键字匹配时，可以根据分词的结果计算与关键字的相关度的评分，评分直接影响当前数据和搜索关键字之间的匹配度高低，一般搜索引擎会将匹配度较高的结果排列在前面。

es中默认使用英文分词器，需要在es的plugins加入中文分词插件elasticsearch-analysis-ik，分词插件解压后目录放在plugins下的一个目录中。

添加完成后重启elasticsearch即可，在kibana进行测试。如果elasticsearch是后台启动方式启动的，则可以通过查看进程号并关闭该进程的方式关闭elasticsearch。

### 整合Java

目前市面上有两类客户端，一类是TransportClient 为代表的ES原生客户端（ES7.0后废除），不能执行原生dsl语句，必须使用它的Java api方法。

另一种是以Rest Api为主的missing client，最典型的就是jest。这种客户端可以直接使用dsl语句拼成的字符串，直接传给服务端，然后返回json字符串再解析，该客户端也与springboot完美集成。

1、创建SpringBoot项目，然后添加相关依赖。其中jest和jna将版本号部分纳入parent模块中管理。spring-boot-starter-data-elasticsearch不用管理版本号，其版本跟随springboot的大版本号。

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-elasticsearch</artifactId> </dependency> <dependency>  <groupId>io.searchbox</groupId>  <artifactId>jest</artifactId>  <version>5.3.3</version> </dependency> <dependency>  <groupId>net.java.dev.jna</groupId>  <artifactId>jna</artifactId>  <version>4.5.1</version> </dependency> |

2、在application.properties配置文件中添加相关配置，其192.168.235.23表示es安装的Linux虚拟机的IP地址，9200表示es提供外网访问的端口号。

|  |
| --- |
| spring.elasticsearch.jest.uris=http://192.168.235.23:9200 |

3、创建查询数据

|  |  |  |
| --- | --- | --- |
| PUT /movie\_chn/movie/1  {  "id": 1,  "name": "红海行动",  "doubanScore": 8.5,  "actorList": [  {  "id": 1,  "name": "张译"  },  {  "id": 2,  "name": "海清"  },  {  "id": 3,  "name": "张涵予"  }  ]  } | PUT /movie\_chn/movie/2  {  "id": 2,  "name": "湄公河行动",  "doubanScore": 8,  "actorList": [  {  "id": 3,  "name": "张涵予"  }  ]  } | PUT /movie\_chn/movie/3  {  "id": 3,  "name": "红海事件",  "doubanScore": 5,  "actorList": [  {  "id": 4,  "name": "张晨"  }  ]  } |

4、创建测试类进行测试，在控制台上输出“红海行动”和“红海事件”则表示整合成功。

|  |
| --- |
| @RunWith(SpringRunner.class) @SpringBootTest public class MallSearchServiceApplicationTests {  @Autowired  JestClient jestClient;  @Test  public void contextLoads() throws IOException {  Search search = new Search.Builder("{\n" +  " \"query\": {\n" +  " \"bool\": {\n" +  " \"filter\": [\n" +  " {\n" +  " \"terms\": {\"actorList.id\": [\"1\",\"2\",\"3\",\"4\"]}\n" +  " }\n" +  " ],\n" +  " \"must\": [\n" +  " {\"match\": {\n" +  " \"name\": \"红海\"\n" +  " }}\n" +  " ]\n" +  " }\n" +  " }\n" +  "}").addIndex("movie\_chn").addType("movie").build();  SearchResult execute = jestClient.execute(search);  List<Movie> movies = new ArrayList<>();  List<SearchResult.Hit<Movie, Void>> hits = execute.getHits(Movie.class);  for (SearchResult.Hit<Movie, Void> hit : hits) {  Movie source = hit.source;  System.*out*.println(source.getName());  movies.add(source);  }  } } |

|  |
| --- |
| public class Movie {  private String id;  private String name;  private Double doubanScore;  private List<Actor> actorList;  }  public class Actor {  private String id;  private String name;  } |

## 安装ES集群版

### 安装es集群

1、修改集群中es的配置文件elasticsearch.yml，每个es除了node.name、network.host和discovery.zen.ping.unicast.hosts不同外，其他配置均最好保持一致。

|  |
| --- |
| cluster.name: my-application #所有集群中的es必须相同  node.name: xmm192.168.235.23 #节点名称，同一个集群中不能出现相同的名称  node.master: true #指定该节点可以是master节点  node.data: true #指定该节点可以是数据节点  path.data: /opt/data #数据的默认存放路径（自定义）  path.logs: /opt/logs #日志的默认存放路径  network.host: 192.168.235.23 #当前节点的IP地址  http.port: 9200 #对外提供服务的端口  transport.tcp.port: 9300 #为集群服务的端口  discovery.zen.ping.unicast.hosts: ["192.168.235.21", "192.168.235.22","192.168.235.24"]  # 集群个节点IP地址，也可以使用域名，需要各节点能够解析  discovery.zen.minimum\_master\_nodes: 2  # 为了避免脑裂，集群主节点数最少为半数+1 |

2、配置完成后，分别清空各自设置的data和logs目录下的数据

3、启动elasticsearch

#### 异常

1、with the same id but is a different node instance

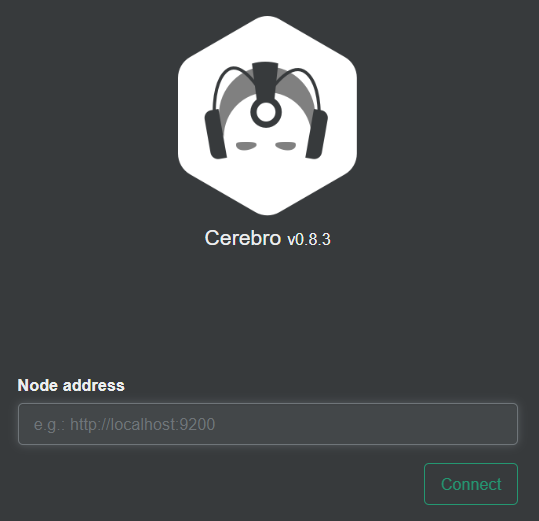
出现上述异常可以尝试将集群配置的data目录下的Nodes目录删除即可。

### cerebro管理集群

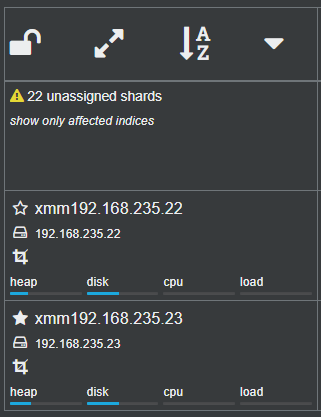
1、将cerebro压缩包解压，并进入bin目录，运行bin目录下的cerebro.bat文件

|  |
| --- |
| [info] play.api.Play - Application started (Prod) (no global state)  [info] p.c.s.AkkaHttpServer - Listening for HTTP on /0:0:0:0:0:0:0:0:9000 |

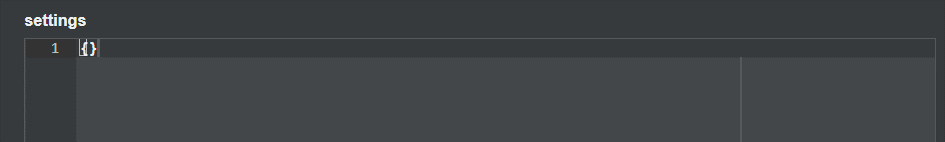
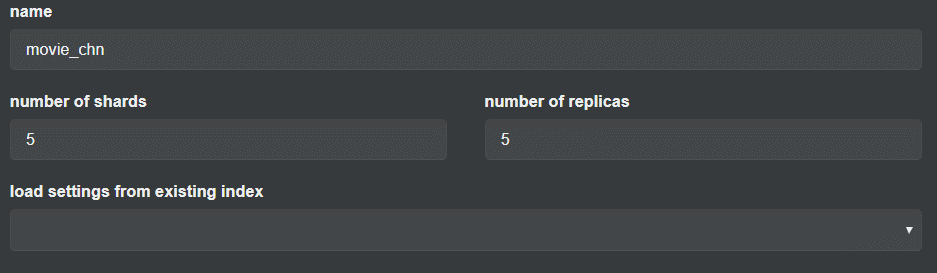
2、在浏览器地址栏中输入localhost:9000访问cerebro网址



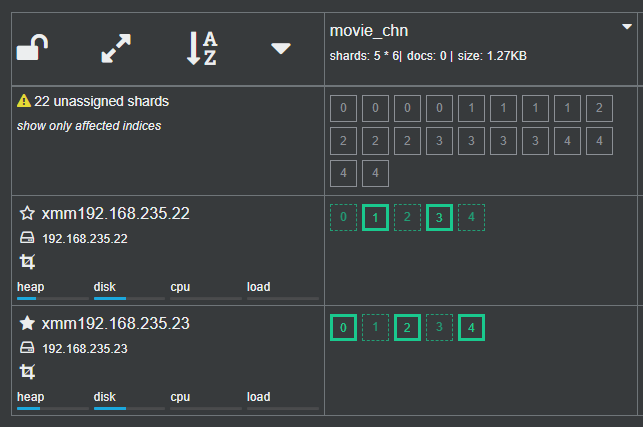
3、在Node address中输入任意集群中任意一个es的IP地址和端口号（如<http://192.168.235.23:9200/>），进行连接。成功连接后进入图形化界面，会显示出elasticsearch集群中所有的elasticsearch，如下图所示，其中镂空五角星标注的es表示非主节点，实心五角星标注的es表示主节点。



4、在上方的【more】选项中选择【create index】进入创建索引的界面，在【settings】中输入{}表示不进行特殊设置。



5、设置完成后选择【overview】返回主页面查看刚刚创建的索引



### 安装head插件（not now）

* 下载head插件

wget [https://github.com/mobz/elasticsearch-head/archive/elasticsearch-head-master.zip](https://github.com/mobz/elasticsearch-head/archive/master.zip)

也可以用git下载，前提yum install git

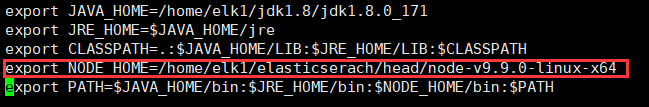
unzip elasticsearch-head-master.zip

* 安装node.js

wget <https://npm.taobao.org/mirrors/node/latest-v4.x/node-v4.4.7-linux-x64.tar.gz>

tar -zxvf node-v9.9.0-linux-x64.tar.gz

* 添加node.js到环境变量



source /etc/profile

* 测试

node -v

npm -v

* 安装grunt（grunt是一个很方便的构建工具，可以进行打包压缩、测试、执行等等的工作）

进入到elasticsearch-head-master

npm install -g grunt-cli

npm install

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

* 修改Elasticsearch配置文件

编辑elasticsearch-6.3.1/config/elasticsearch.yml,加入以下内容：

http.cors.enabled: true

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

* 修改Gruntfile.js（注意’，’）

打开elasticsearch-head-master/Gruntfile.js，找到下面connect属性，新增hostname:’\*’:

connect: {

server: {

options: {

hostname: '\*',

port: 9100,

base: '.',

keepalive: true

}

}

}

* 启动elasticsearch-head

进入elasticsearch-head目录，执行命令：grunt server

* 后台启动elasticsearch-head

nohup grunt server &exit

* 关闭head插件

ps -aux|grep head

kill 进程号