如若需要可以查看最新在线文档:

<http://www.zhenchao.cf/bigdata_elk_atguigu>

<http://www.zhenchao.tk/bigdata_elk_atguigu>

<https://zhenchao125.github.io/bigdata_elk_atguigu>

# 第 1 部分 ElasticSearch

## 第 1 章 ElasticSearch 简介

### 1.1 Elasticsearch 是什么

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。

主要特点:

1. 分布式的实时文件存储，每个字段都被索引并可被搜索
2. 分布式的实时分析搜索引擎–做不规则查询
3. 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 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）

关于什么是 REST API, 参考: https://www.zhihu.com/question/28557115

### 1.2 ElasticSearch 使用案例

1. 2013年初，GitHub抛弃了Solr，采取 ElasticSearch 来做 PB 级的搜索。 “GitHub使用ElasticSearch搜索20TB的数据，包括13亿文件和1300亿行代码”
2. 维基百科：启动以 elasticsearch 为基础的核心搜索架构SoundCloud：“SoundCloud使用ElasticSearch为1.8亿用户提供即时而精准的音乐搜索服务”
3. 百度：百度目前广泛使用 ElasticSearch 作为文本数据分析，采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示，辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线（包括casio、云分析、网盟、预测、文库、直达号、钱包、风控等），单集群最大100台机器，200个ES节点，每天导入30TB+数据
4. 新浪使用 ES 分析处理 32 亿条实时日志
5. 阿里使用 ES 构建挖财自己的日志采集和分析体系

### 1.3 与其他数据存储比较



## 第 2 章 ElasticSearch 基础知识

### 2.1 ElasticSearch 数据存储方式

#### 2.1.1 几个重要的概念

|  |  |
| --- | --- |
| 概念 | 解释 |
| 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 |
| 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。(7.0 默认是 1 片) |
| index | 相当于EDBMS 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 | 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。(（6.x只允许建一个，7.0被废弃，造成index实际相当于table级）) |
| document | 类似于 rdbms 的 row、面向对象里的object |
| field | 相当于字段、属性 |

### 2.3 倒排索引

要学习 ElasticSearch 必须学习倒排索引

什么是倒排索引呢?

请背出带“前”字的诗句, 你会发现自己几乎想不出来有哪些诗句.

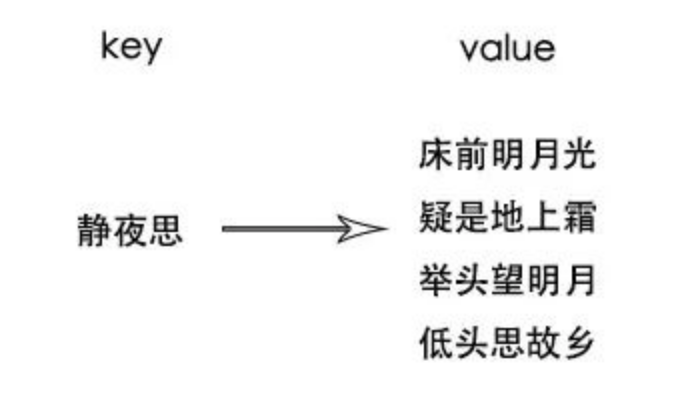
请背下李白的的“静夜思”: “窗前明月光, 疑是地上霜, …”. 再背下李白的“望庐山瀑布”: “日照香炉生紫烟, 遥看瀑布挂前川,…” 很容易就背出来了. 为什么呢?

是因为你脑袋里面使用“题目”为诗句创建了索引, 通过这些索引你在脑海里面很检索出来具体的诗句.

如果我们也使用“前”为这些诗句建立索引, 是不是也会容易的检索出来这些诗句呢, 答案是肯定的! 这就是“倒排索引”

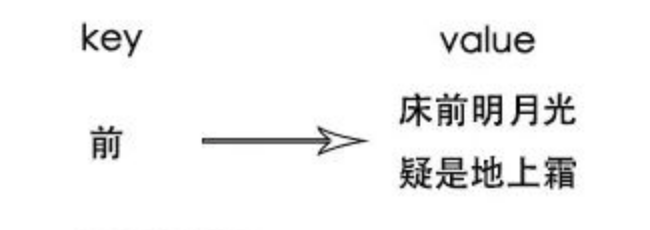
#### 普通索引(正排索引)

普通的索引是以诗名作为 key, 诗的内容作为 value 来建立的索引的.



#### 倒排索引

如果以“前”作为 key 和以诗的内容作为 value 来建立索引, 则就是倒排索引



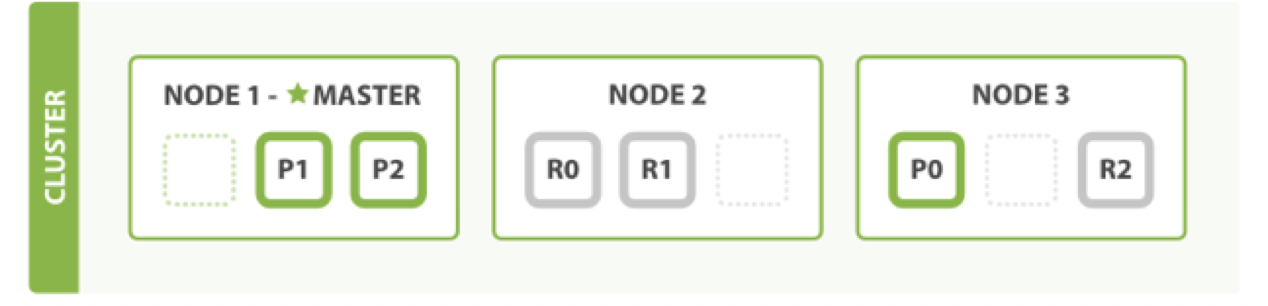
也可为其他的字词建立倒排索引.



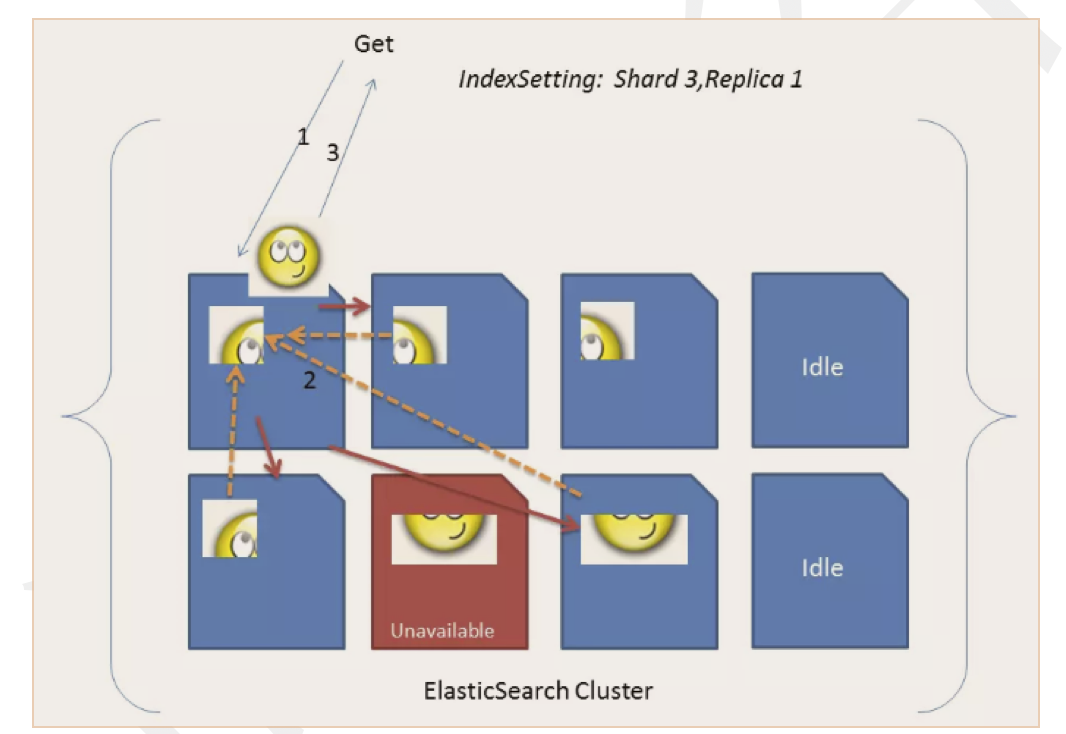
### 2.2 ElasticSearch 特点

#### 2.2.1 天然分片, 天然集群

es 把数据分成多个 shard，下图中的 P0-P2，多个shard可以组成一份完整的数据，这些shard可以分布在集群中的各个机器节点中。随着数据的不断增加，集群可以增加多个分片，把多个分片放到多个机子上，已达到负载均衡，横向扩展。



在实际运算过程中，每个查询任务提交到某一个节点，该节点必须负责将数据进行整理汇聚，再返回给客户端，也就是一个简单的节点上进行 Map 计算，在一个固定的节点上进行 Reduces 得到最终结果向客户端返回。



这种集群分片的机制造就 elasticsearch 强大的数据容量及运算扩展性。

#### 2.2.2 天然索引

1. ES 所有数据都是默认进行索引的，这点和 mysql 正好相反. (mysql 是默认不加索引，要加索引必须特别说明，ES 只有不加索引才需要说明。)
2. 而 ES 使用的是倒排索引和 Mysql 的 B+Tree 索引不同。

## 第 3 章 ElasticSearch 安装

### 步骤 1: 下载安装包

我们使用 6.3.1 版本

https://www.elastic.co/cn/downloads/past-releases/elasticsearch-6-3-1

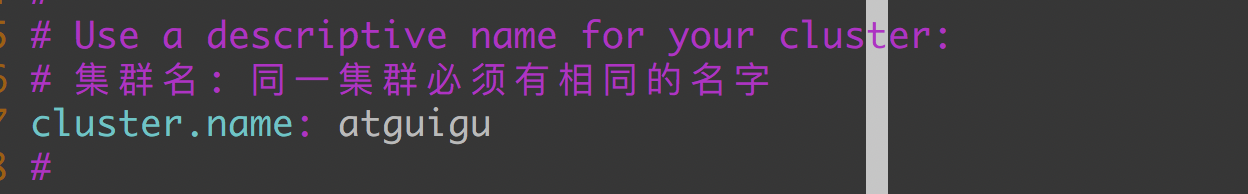
### 步骤 2: 解压

tar -zxvf elasticsearch-6.3.1.tar.gz -C /opt/module

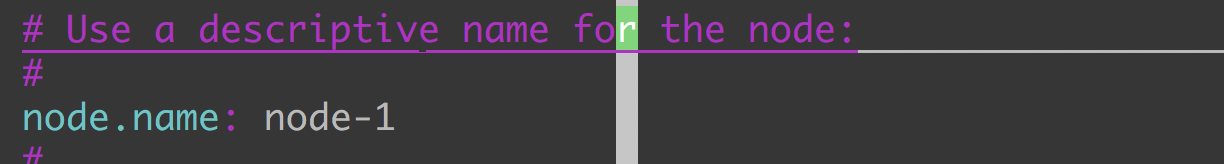
### 步骤 3: 修改配置文件config/elasticsearch.yml

配置的时候需要注意: 每行必须顶格, 不能有前置空格. “:”后面必须有一个空格.

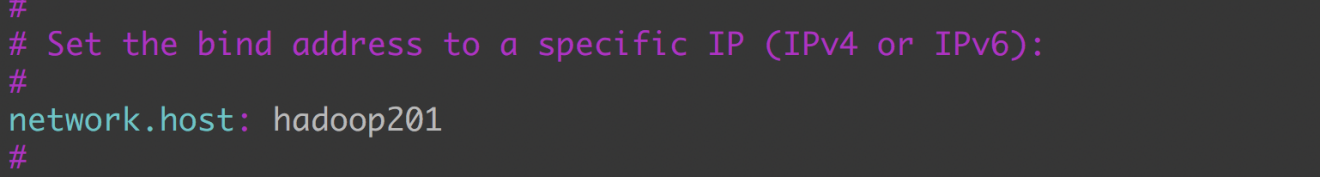
#### 1. 配置集群名



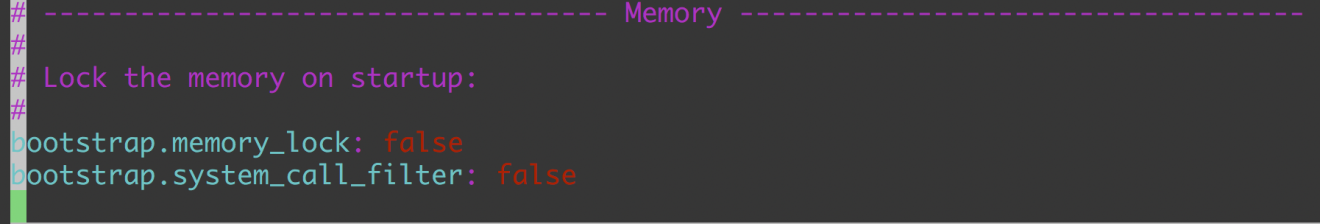
#### 2. 当前节点名: 每个节点的名字不能相同, 当分发到其他节点的时候, 需要改成不同的名字



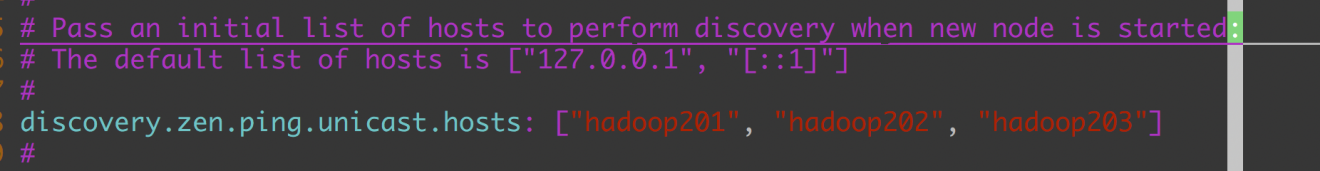
#### 3. 给当前节点绑定 ip 地址, 端口号保持默认 9200 就行



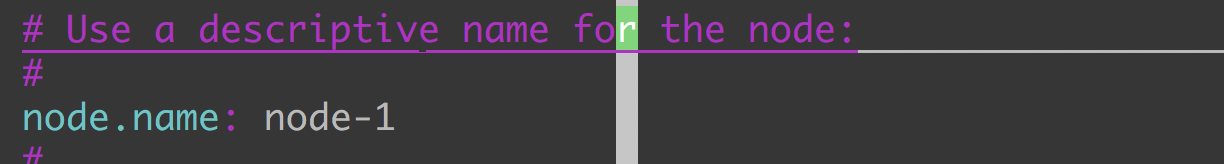
#### 4. 关掉 bootstrap 自检程序



#### 5. 集群个节点IP地址，也可以使用域名，需要各节点能够解析



### 步骤 4: 分发 ElasticSearch

注意修改每个节点的名 

### 步骤 5: 修改 linux 配置

默认 elasticsearch 是单机访问模式，就是只能自己访问自己。 但是我们之后一定会设置成允许应用服务器通过网络方式访问。这时，elasticsearch 就会因为嫌弃单机版的低端默认配置而报错，甚至无法启动。 所以我们在这里就要把服务器的一些限制打开，能支持更多并发

注意: 修改以下配置的时候需要切换到 root 用户, 每个节点都要修改.

#### 问题1： max file descriptors [4096] for elasticsearch process likely too low, increase to at least [65536] elasticsearch

原因：系统允许 Elasticsearch 打开的最大文件数需要修改成 65536 解决：

vim /etc/security/limits.conf 添加内容：

\* soft nofile 65536

\* hard nofile 131072

\* soft nproc 2048

\* hard nproc 65536

注意：“\*” 不要省略掉

#### 问题2：max number of threads [1024] for user [judy2] likely too low, increase to at least [2048] （CentOS7.x 不用改）

原因：允许最大进程数修该成4096 解决：vim /etc/security/limits.d/90-nproc.conf  
修改如下内容： \* soft nproc 1024

修改为

* soft nproc 4096

#### 问题3：max virtual memory areas vm.max\_map\_count [65530] likely too low, increase to at least [262144] （CentOS7.x 不用改）

原因：一个进程可以拥有的虚拟内存区域的数量。 解决： 在 /etc/sysctl.conf 文件最后添加一行 vim /etc/sysctl.conf

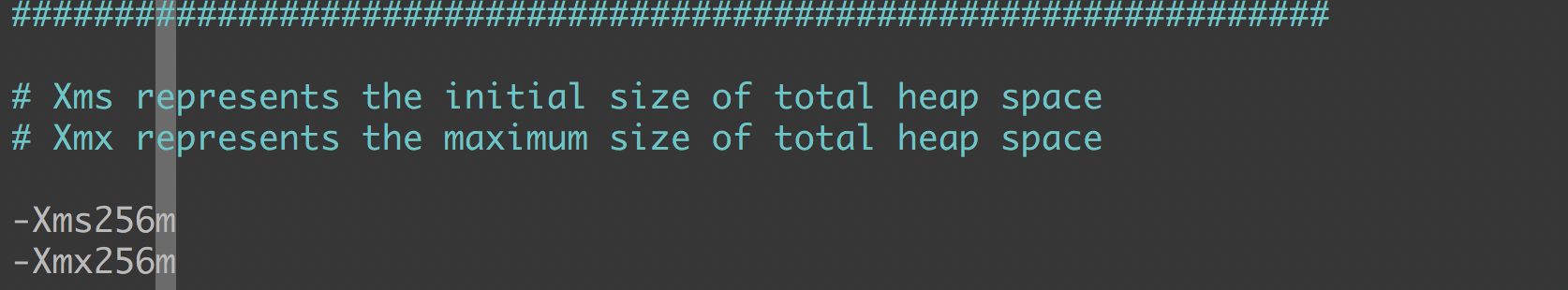
vm.max\_map\_count=262144

即可永久修改

### 步骤 6 教学环境启动优化

ES是用在Java虚拟机中运行的，虚拟机默认启动占用2G内存。但是如果是装在PC机学习用，实际用不了2个G。所以可以改小一点内存。

vim config/jvm.options



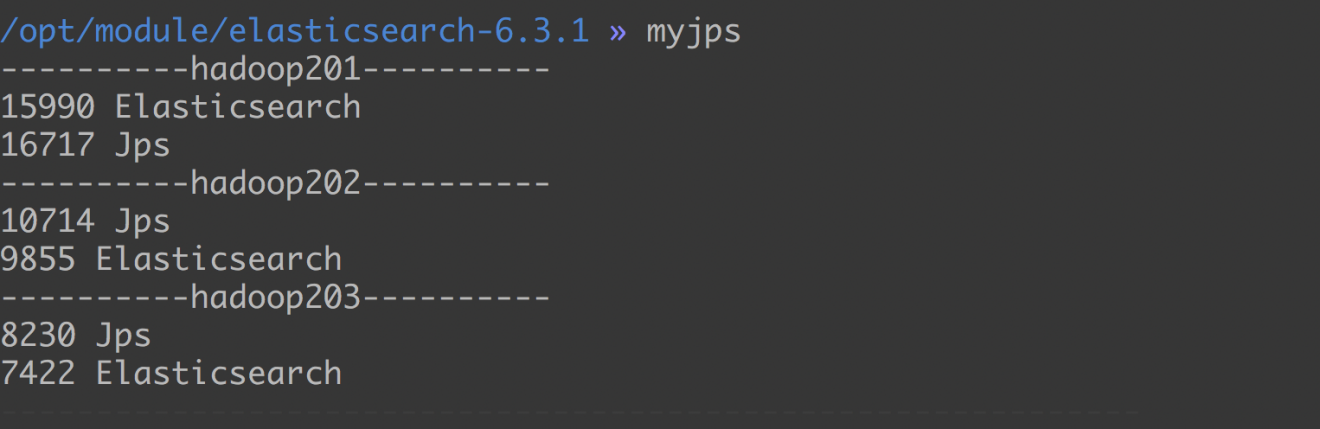
注意: 注意别忘了分发修改后的配置文件jvm.options

### 步骤 7 启动 ElasticSearch

分别在 3 台设备上启动 ElasticSearch

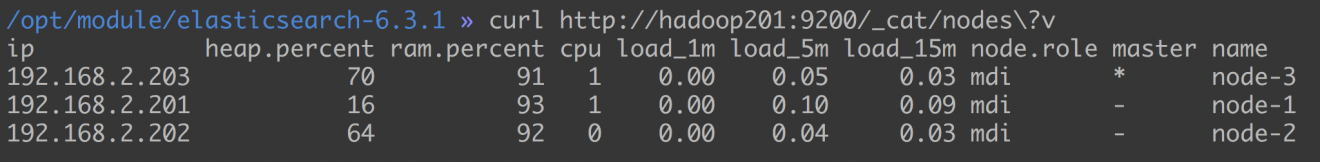
bin/elasticsearch &

### 步骤 8 查看 ElasticSearch 是否启动成功



### 步骤 9 测试是否可以连接到 ElasticSearch

curl http://hadoop104:9200/\_cat/nodes?v



或者在浏览器中输入地址: http://hadoop104:9200/\_cat/nodes?v

## 第 4 章 Kibana 安装

### 4.1 ElasticSearch 交互方式

1. 基于 HTTP 协议，以 JSON 为数据交互格式的 RESTful API

* GET POST PUT DELETE HEAD

1. 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

在学习测试阶段使用 Kibana 操作 ElasticSearch 是最方便的. 下面学习如何安装和使用 Kibana.

Kibana 可以看出是一个操作 ElasticSeach 的客户端.

### 4.2 安装 Kibana

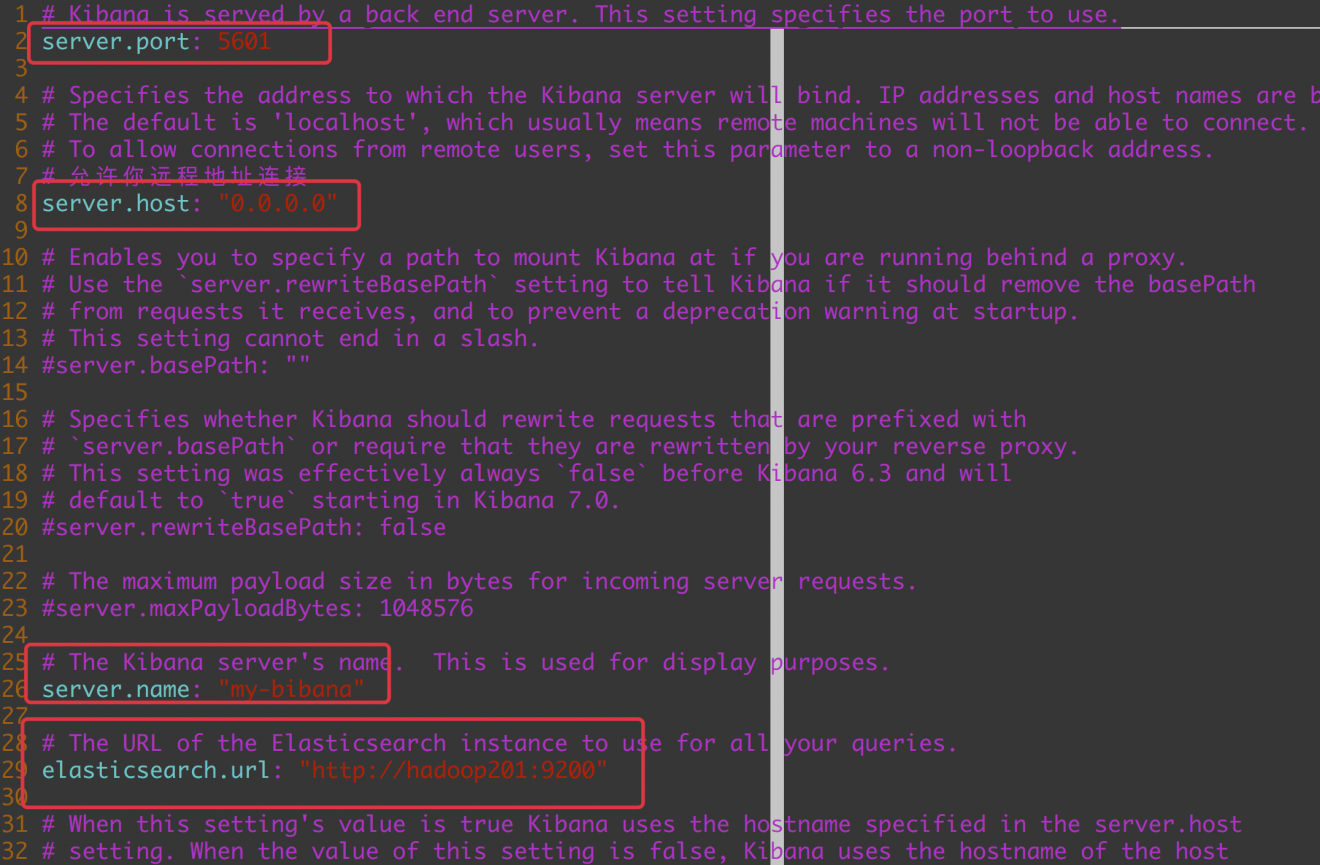
#### 步骤 1: 下载 Kibana

https://www.elastic.co/cn/downloads/past-releases/kibana-6-3-1

#### 步骤 2: 解压 Kibana

tar -zxvf kibana-6.3.1-linux-x86\_64.tar.gz -C /opt/module

#### 步骤 3: 配置config/kibana.yml

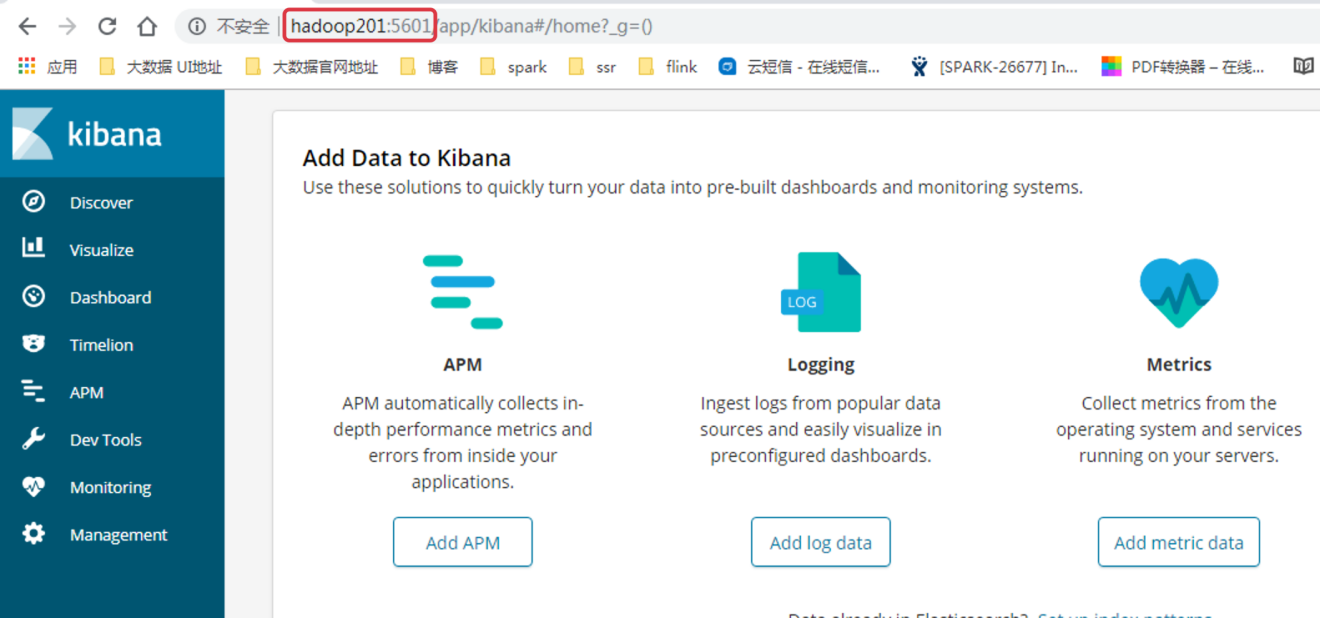


#### 步骤 4: 启动 Kibana

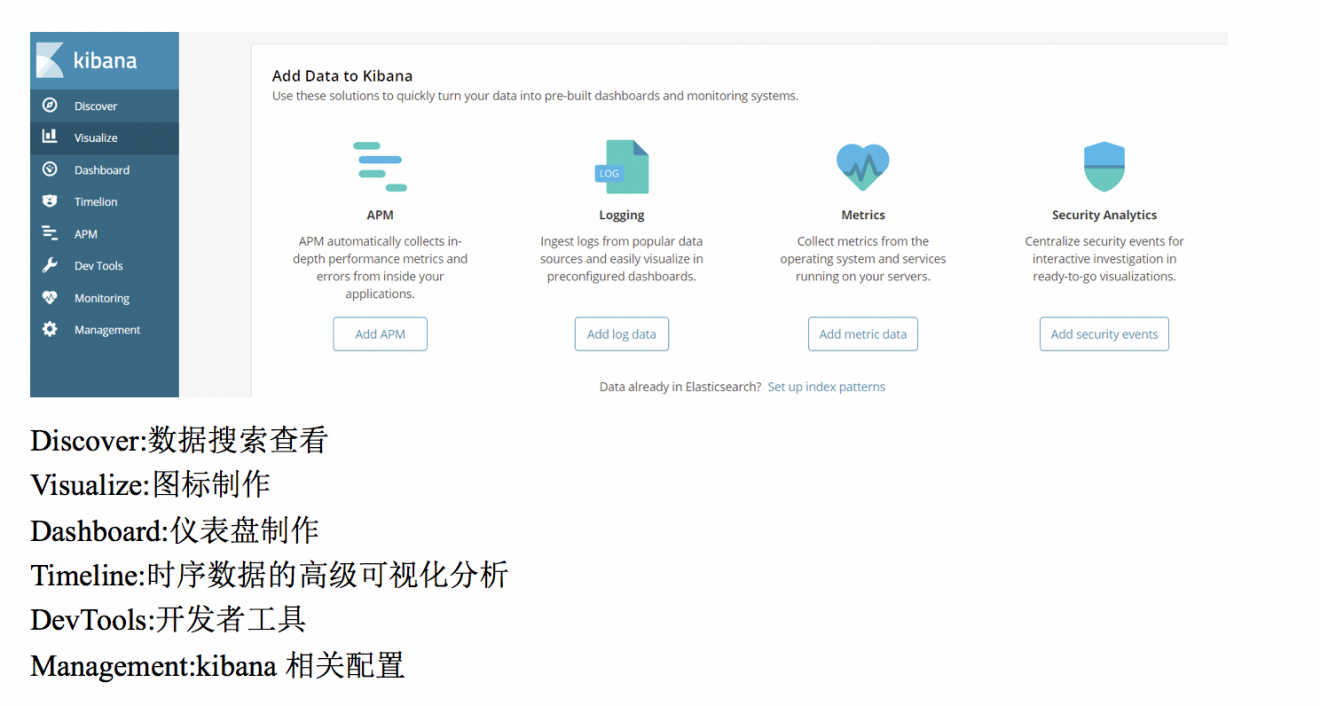
bin/kibana

#### 步骤 5: 在浏览器中打开 Kibana

http://hadoop104:5601



### 4.3 Kibana 功能简介



## 第 5 章 制作 Elastic 和 Kibana 的统一启停脚本

为了方便启动 ElasticSearch 和 Kibana, 制作统一启动脚本

#!/bin/bash  
es\_home=/opt/module/elasticsearch-6.3.1  
kibana\_home=/opt/module/kibana-6.3.1  
case $1 in  
 "start") {  
 for i in hadoop201 hadoop202 hadoop203  
 do  
 ssh $i "source /etc/profile;${es\_home}/bin/elasticsearch >$es\_home/logs/es.log 2>&1 &"  
  
 done  
  
 nohup ${kibana\_home}/bin/kibana >$kibana\_home/kibana.log 2>&1 &  
  
 };;  
 "stop") {  
 ps -ef|grep ${kibana\_home} |grep -v grep|awk '{print $2}'|xargs kill  
  
 for i in hadoop201 hadoop202 hadoop203  
 do  
 ssh $i "ps -ef|grep $es\_home |grep -v grep|awk '{print \$2}'|xargs kill" >/dev/null 2>&1  
 done  
  
  
 };;  
 \*){  
 echo "你启动的姿势不正确, 请使用参数 start 来启动es集群, 使用参数 stop 来关闭es集群"  
 };;  
esac



## 第 6 章 Elastic Restfull Api

### 6.1 ElasticSearch 中保存的数据结构

假设有两个对象:

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

这两个对象如果放在关系型数据库保存，会被拆成 2 张表，但是 elasticsearch 是用一个 json 来表示一个 document。

所以在 ES 中是这样保存的:

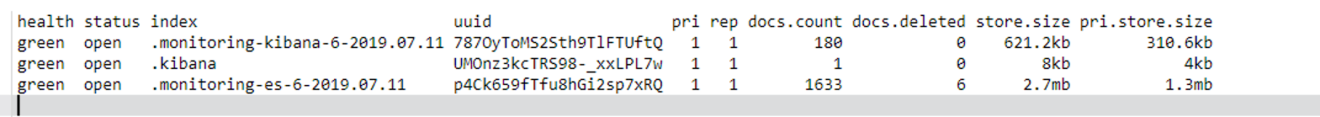
{  
 “id”:”1”,  
 “name”:”operation red sea”,  
 “doubanScore”:”8.5”,  
 “actorList”:[   
 {“id”:”1”,”name”:”zhangyi”},  
 {“id”:”2”,”name”:”haiqing”},  
 {“id”:”3”,”name”:”zhanghanyu”}  
 ]  
}

### 6.2 操作 ElasticSearch 中的数据

#### 6.2.1 查看 ES 中有哪些索引

GET /\_cat/indices?v

结果:

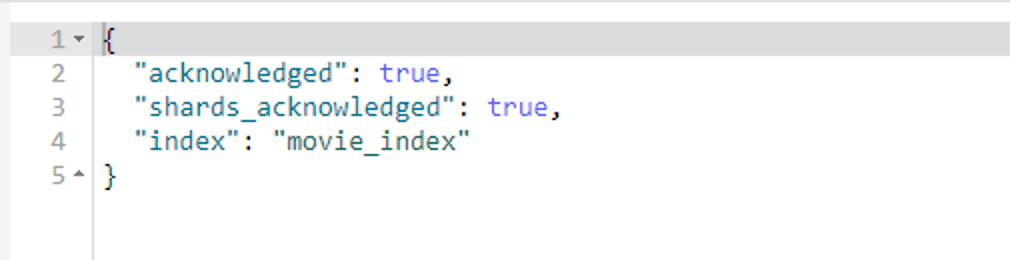


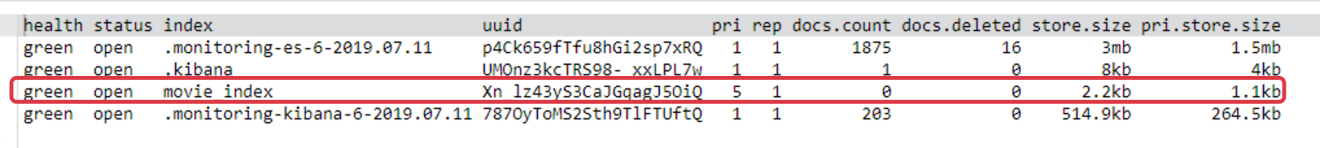
表头含义:

|  |  |
| --- | --- |
| health | green(集群完整) yellow(单点正常、集群不完整) red(单点不正常) |
| status | 是否能使用 |
| index | 索引名 |
| uuid | 索引统一编号 |
| pri | 主节点几个 |
| rep | 从节点几个 |
| docs.count | 文档数 |
| docs.deleted | 文档被删了多少 |
| store.size | 整体占空间大小 |
| pri.store.size | 主节点占 |

#### 6.2.2 增加索引

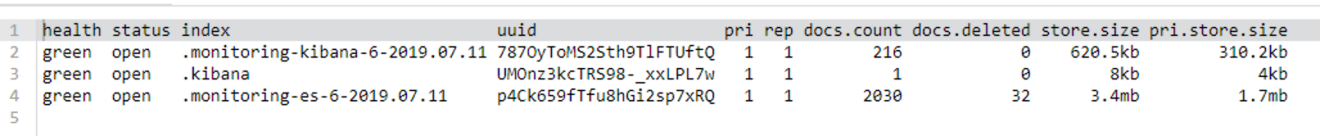
PUT /movie\_index





#### 6.2.3 删除索引

DELETE /movie\_index



#### 6.2.4 新增文档

PUT /movie\_index/movie/1  
{ "id":1,  
 "name":"operation red sea",  
 "doubanScore":8.5,  
 "actorList":[   
 {"id":1,"name":"zhang yi"},  
 {"id":2,"name":"hai qing"},  
 {"id":3,"name":"zhang han yu"}  
 ]  
}  
PUT /movie\_index/movie/2  
{  
 "id":2,  
 "name":"operation meigong river",  
 "doubanScore":8.0,  
 "actorList":[   
 {"id":3,"name":"zhang han yu"}  
 ]  
}  
  
PUT /movie\_index/movie/3  
{  
 "id":3,  
 "name":"incident red sea",  
 "doubanScore":5.0,  
 "actorList":[   
 {"id":4,"name":"zhang chen"}  
 ]  
}

注意: 如果之前没建过 index 或者 type，es 会自动创建。

#### 6.2.5 搜索 type 全部数据

GET /movie\_index/movie/\_search

{  
 "took": 129,  
 "timed\_out": false,  
 "\_shards": {  
 "total": 5,  
 "successful": 5,  
 "skipped": 0,  
 "failed": 0  
 },  
 "hits": {  
 "total": 3,  
 "max\_score": 1,  
 "hits": [  
 {  
 "\_index": "movie\_index",  
 "\_type": "movie",  
 "\_id": "2",  
 "\_score": 1,  
 "\_source": {  
 "id": 2,  
 "name": "operation meigong river",  
 "doubanScore": 8,  
 "actorList": [  
 {  
 "id": 3,  
 "name": "zhang han yu"  
 }  
 ]  
 }  
 },  
 {  
 "\_index": "movie\_index",  
 "\_type": "movie",  
 "\_id": "1",  
 "\_score": 1,  
 "\_source": {  
 "id": 1,  
 "name": "operation red sea",  
 "doubanScore": 8.5,  
 "actorList": [  
 {  
 "id": 1,  
 "name": "zhang yi"  
 },  
 {  
 "id": 2,  
 "name": "hai qing"  
 },  
 {  
 "id": 3,  
 "name": "zhang han yu"  
 }  
 ]  
 }  
 },  
 {  
 "\_index": "movie\_index",  
 "\_type": "movie",  
 "\_id": "3",  
 "\_score": 1,  
 "\_source": {  
 "id": 3,  
 "name": "incident red sea",  
 "doubanScore": 5,  
 "actorList": [  
 {  
 "id": 4,  
 "name": "zhang chen"  
 }  
 ]  
 }  
 }  
 ]  
 }  
}

#### 6.2.2 查找指定 id 的 document 数据

GET /movie\_index/movie/1

{  
 "\_index": "movie\_index",  
 "\_type": "movie",  
 "\_id": "1",  
 "\_version": 1,  
 "found": true,  
 "\_source": {  
 "id": 1,  
 "name": "operation red sea",  
 "doubanScore": 8.5,  
 "actorList": [  
 {  
 "id": 1,  
 "name": "zhang yi"  
 },  
 {  
 "id": 2,  
 "name": "hai qing"  
 },  
 {  
 "id": 3,  
 "name": "zhang han yu"  
 }  
 ]  
 }  
}

#### 6.2.7 修改 document

修改分两种: 整体替换和只修改某个字段

##### 整体替换

和新增文档没有区别

PUT /movie\_index/movie/3  
{  
 "id":"3",  
 "name":"incident red sea",  
 "doubanScore":"8.0",  
 "actorList":[   
 {"id":"1","name":"zhang chen"}  
 ]  
}

##### 只修改某个字段

使用post方法

POST /movie\_index/movie/3/\_update  
{  
 "doc": {  
 "doubanScore":"8.1"  
 }  
}

#### 6.2.8 删除一个 document

DELETE /movie\_index/movie/3

#### 6.2.9 按条件查询(全部)

GET /movie\_index/movie/\_search  
{  
 "query": {  
 "match\_all": {}  
 }  
}

#### 6.2.10 按照字段的分词查询

GET /movie\_index/movie/\_search  
{  
 "query": {  
 "match": {  
 "name": "sea"  
 }  
 }  
}

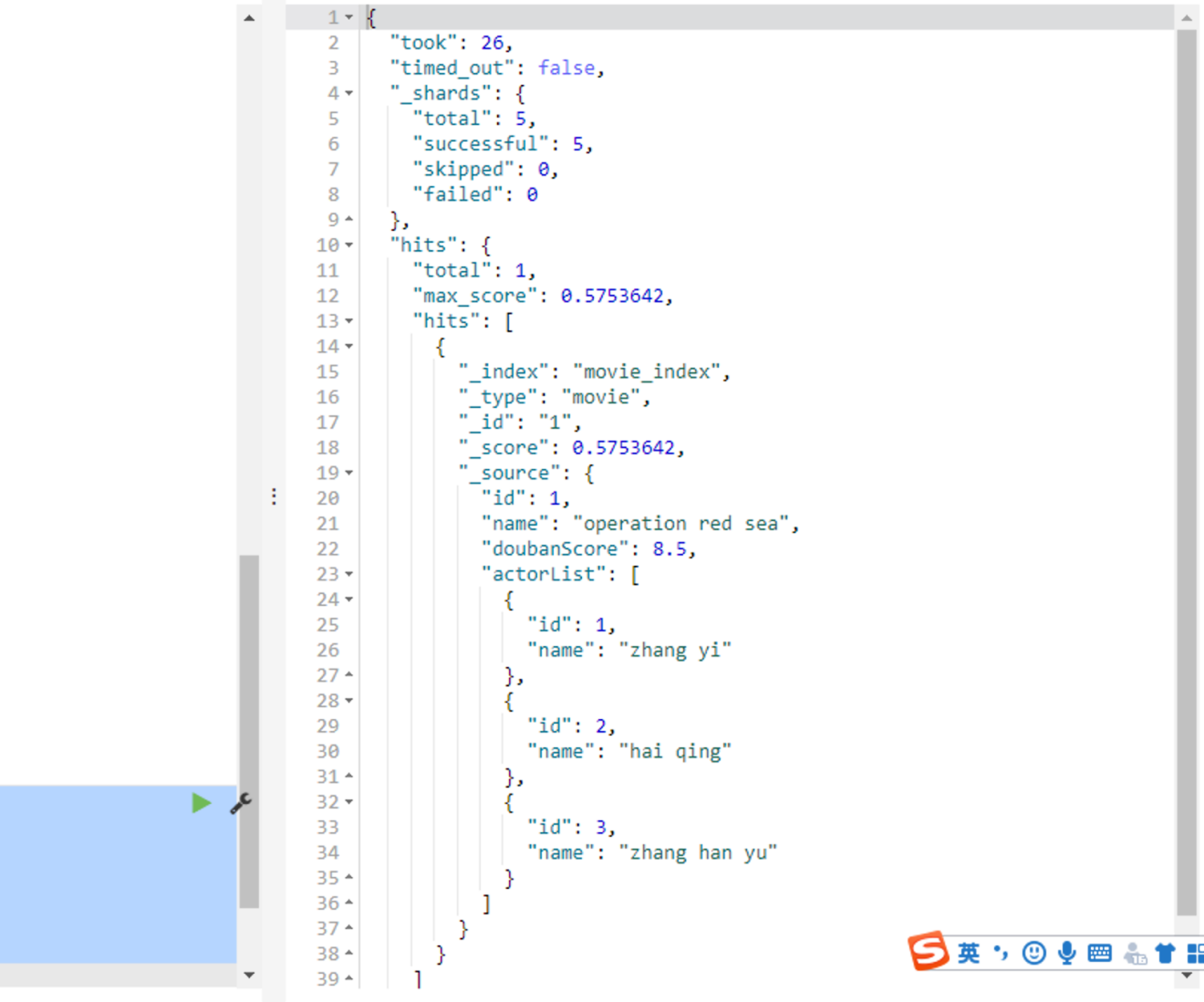
#### 6.2.11 按照分词子属性查询

GET /movie\_index/movie/\_search  
{  
 "query": {  
 "match": {  
 "actorList.name": "zhang"  
 }  
 }  
}

#### 6.2.12 按照短语查询

安装短语查询的意思是指, 匹配某个 field 的整个内容, 不再利用分词技术

GET /movie\_index/movie/\_search  
{  
 "query": {  
 "match\_phrase": {  
 "name": "operation red"  
 }  
 }  
}

说明: 把operation red作为一个整体来看待 

对比:

下面的表示包含 operation 或者 red 的都会被查找出来

GET /movie\_index/movie/\_search  
{  
 "query": {  
 "match": {  
 "name": "operation red"  
 }  
 }  
}

#### 6.2.13 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\_index/movie/\_search  
{  
 "query": {  
 "fuzzy": {  
 "name": "red"  
 }  
 }  
}

#### 6.2.14 过滤(查询后过滤)

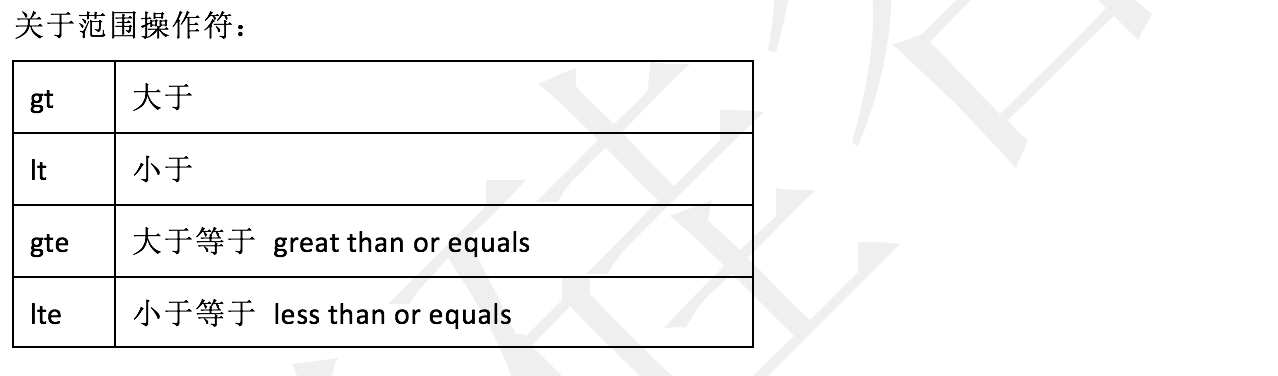
GET /movie\_index/movie/\_search  
{  
 "query": {  
 "match": {  
 "name": "red"  
 }  
 },  
 "post\_filter": {  
 "term": {  
 "actorList.id": "3"  
 }  
 }  
}

#### 6.1.15 查询前过滤(推荐使用)

GET movie\_index/movie/\_search  
{  
 "query": {  
 "bool": {  
 "filter": [  
 {"term":   
 {"actorList.id": 3}  
 },  
 {  
 "term":  
 {"actorList.id": 1}  
 }  
 ],  
 "must":   
 {"match": {  
 "name": "zhang"  
 }}  
   
 }  
 }  
}

#### 6.1.16 按范围过滤

GET movie\_index/movie/\_search  
{  
 "query": {  
 "bool": {  
 "filter": {  
 "range": {  
 "doubanScore": {  
 "gt": 5,  
 "lt": 9  
 }  
 }  
 }  
 }  
 }  
}



#### 6.1.17 排序

GET movie\_index/movie/\_search  
{  
 "query":{  
 "match": {"name":"red operation"}  
 }  
 , "sort": [  
 {  
 "doubanScore": {  
 "order": "desc"  
 }  
 }  
 ]  
}

#### 6.1.18 分页查询

GET movie\_index/movie/\_search  
{  
 "query": { "match\_all": {} },  
 "from": 1,  
 "size": 1  
}

#### 6.2.19 指定查询的字段

GET movie\_index/movie/\_search  
{  
 "query": { "match\_all": {} },  
 "\_source": ["name", "doubanScore"]  
}

#### 6.2.20 聚合

##### 每个演员参演了多少部电影

GET movie\_index/movie/\_search  
{  
 "aggs": {  
 "groupby\_actor": {  
 "terms": {  
 "field": "actorList.name.keyword"  
 }  
 }  
 }  
}

##### 每个演员参演电影的平均分是多少，并按评分排序

GET movie\_index/movie/\_search  
{   
 "aggs": {  
 "groupby\_actor\_id": {  
 "terms": {  
 "field": "actorList.name.keyword" ,  
 "order": {  
 "avg\_score": "desc"  
 }  
 },  
 "aggs": {  
 "avg\_score":{  
 "avg": {  
 "field": "doubanScore"   
 }  
 }  
 }  
 }   
 }  
}

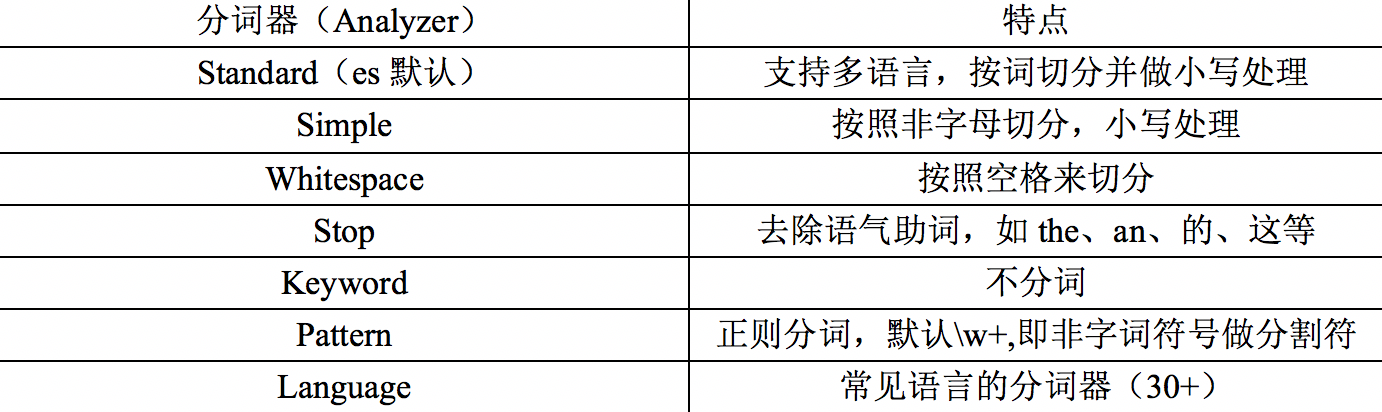
## 第 7 章 中文分词

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。

### 7.1 分词器

#### Elasticsearch自带的分词器



#### 中文分词器



### 7.1 安装 IK 分词器

#### 步骤 1: 下载中文分词器

使用 IK Analysis for Elasticsearch. 下载与 ElasticSearch 的版本匹配的分词器版本.

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

#### 步骤 2: 解压分词器

需要解压到 ES 的plugins目录下

unzip elasticsearch-analysis-ik-6.3.1.zip -d /opt/module/elasticsearch-6.3.1/plugins/ik

#### 步骤 3: 重启启动 ElasticSearch

### 7.2 测试使用

#### 使用默认分词器

GET movie\_index/\_analyze  
{   
 "text": "我是中国人"  
}

结果:



#### 使用ik\_smart分词器

GET movie\_index/\_analyze  
{   
 "analyzer": "ik\_smart",   
 "text": "我是中国人"  
}

结果:



#### 使用 ik\_max\_word

GET movie\_index/\_analyze  
{   
 "analyzer": "ik\_max\_word",   
 "text": "我是中国人"  
}



## 第 8 章 关于 Mapping

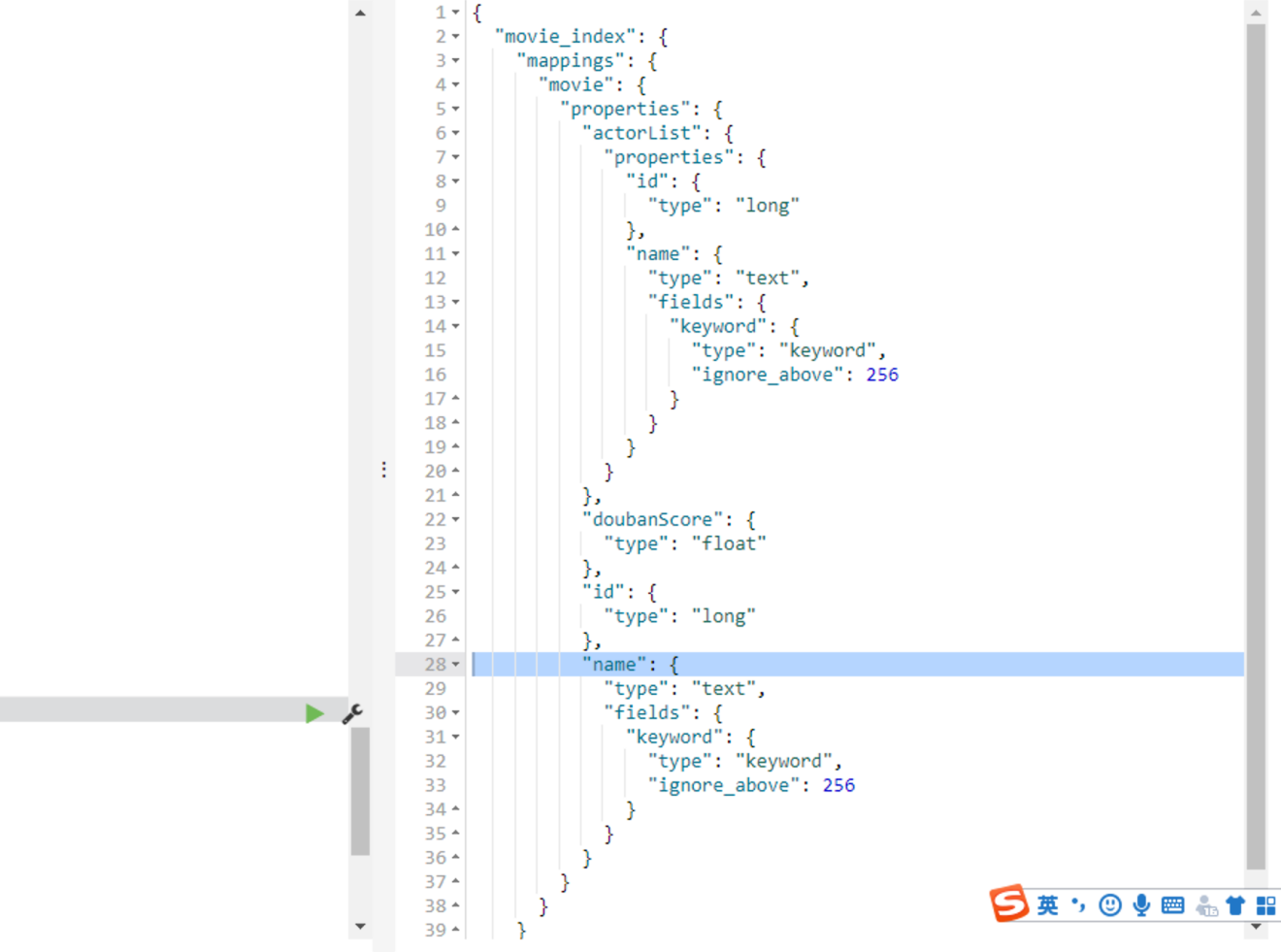
### 查看 Mapping

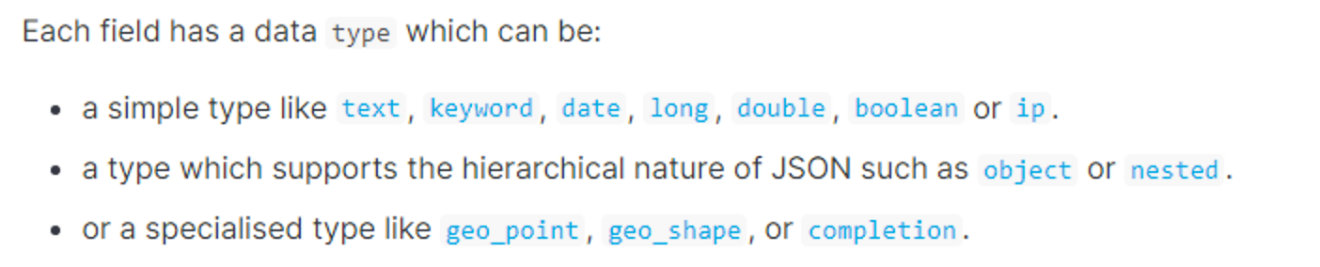
之前说 type 可以理解为table，那每个字段的数据类型是如何定义的呢

默认情况下, 是由插入的第一条数据的类型来自动推断来设定的!

可以通过 Mapping 来设置和查看每个字段的数据类型.

GET movie\_index/movie/\_mapping





true/false → boolean  
1020 → long  
20.1 → double  
“2018-02-01” → date  
“hello world” → text +keyword

### 手动指定 mapping

#### 搭建索引

PUT movie\_chn  
{  
 "mappings": {  
 "movie":{  
 "properties": {  
 "id":{  
 "type": "long"  
 },  
 "name":{  
 "type": "text"  
 , "analyzer": "ik\_smart"  
 },  
 "doubanScore":{  
 "type": "double"  
 },  
 "actorList":{  
 "properties": {  
 "id":{  
 "type":"long"  
 },  
 "name":{  
 "type":"keyword"  
 }  
 }  
 }  
 }  
 }  
 }  
}

#### 插入数据

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":8.0,  
 "actorList":[   
 {"id":3,"name":"张涵予"}  
 ]  
}  
  
PUT /movie\_chn/movie/3  
{  
 "id":3,  
 "name":"红海事件",  
 "doubanScore":5.0,  
 "actorList":[   
 {"id":4,"name":"张晨"}  
 ]  
}

#### 查询

GET /movie\_chn/movie/\_search  
{  
 "query": {  
 "match": {  
 "name": "红海"  
 }  
 }  
}  
  
GET /movie\_chn/movie/\_search  
{  
 "query": {  
 "term": {  
 "actorList.name": "张"  
 }  
 }  
}