# 12.ElasticSearch

## 12.1.ElasticSearch概述

### 12.1.1.介绍

lucene问题：

1.lucene是java开发的，对于不会java的人使用比较困难

2.lucene代码简化问题

3.分布式与高可用问题

ElasticSearch：

1.基于lucene的搜索服务器

2.提供一个分布式高可用多用户能力的全文检索引擎

3.基于RESTful web接口

4.java开发，但是以接口形式对外暴露

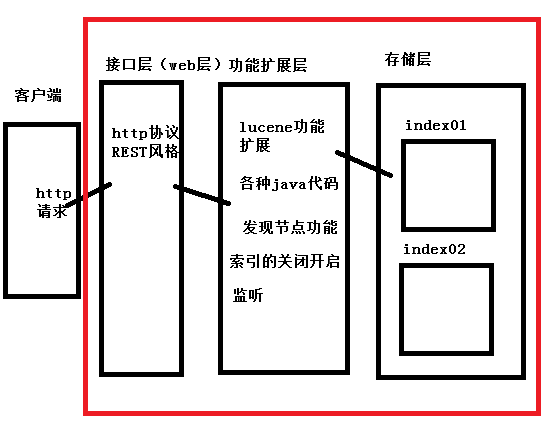
### 12.1.2.ElasticSearch结构

包括三大部分：

底层数据存储：维护索引文件，利用分布式数据切分存储，可以存在本地，也可以存在hdfs

封装lucene扩展功能：发现集群节点功能，支持各种对资源的管理

接口层：给用户提供访问接口（http协议，REST风格），用户可以通过url访问地址，通过请求方式，对索引进行CRUD操作



## 12.2.REST风格

### 12.2.1.意义

2001年，蒂姆·贝纳斯·李（http协议的创始人）发表了一篇对http使用乱象的论文，认为业界对http理解不到位，给了8种请求方式，绝大部分只使用get和post两种，在论文指出，url只表示资源，请求方式表示对资源的操作。

### 12.2.2.请求方式

put：新增

get：查询

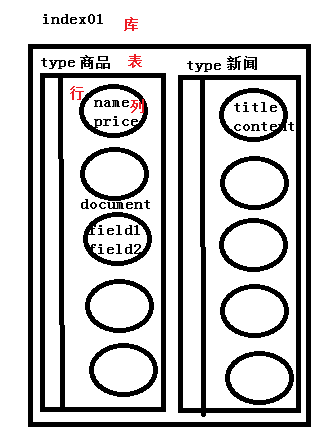
post：修改

delete：删除

Springmvc也支持REST风格

### 12.2.3.elasticsearch概念与数据库类比

|  |  |
| --- | --- |
| 数据库 | elasticsearch |
| database库 | index索引 |
| table表 | type类型 |
| rows行数据 | document文档 |
| column列数据 | field域 |
| sql语法 | DSL（REST风格的url） |
| insert | put请求方式 |
| delete | delete请求方式 |
| update | post请求方式 |
| select | get请求方式 |



## 12.3.elasticsearch安装配置

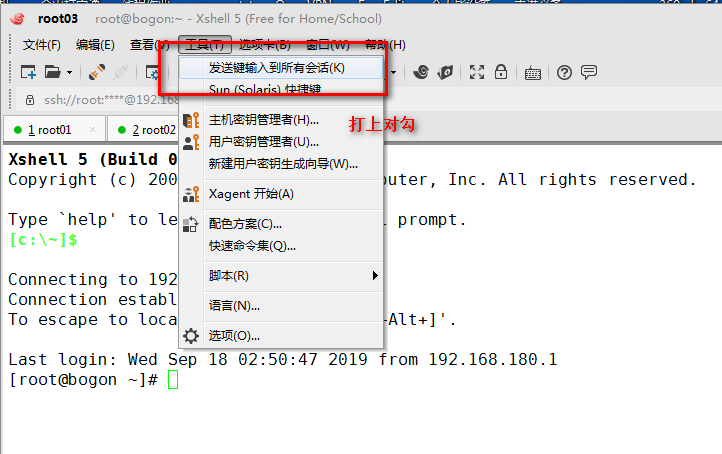
最终使用的是3台服务器

**linux的环境配置（配置3台）**

先关闭3台防火墙

[root@bogon /]# service iptables stop

[root@bogon /]# chkconfig iptables off



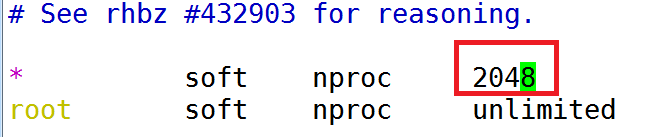
这样命令可以同步到所有窗口

注意：如果是复制，则不会同步

1.1修改root以外的用户可使用的最大线程数（3台）

elasticsearch出于安全考虑,不允许root用户来操作启动,所以需要添加一个具有操作权限的es用户,centOS 6.5对于除了root以外的线程上限默认1024,es要求至少2048;

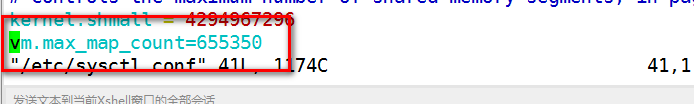
[root@10-9-100-26 ~]# vim /etc/security/limits.d/90-nproc.conf



1.2修改root以外最大虚拟内存占用量

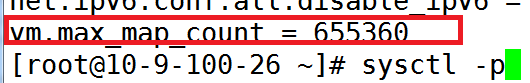
es要求vm.max\_map\_count的key对应的内存占用值是200M,默认65M

[root@10-9-100-26 ~]# vim /etc/sysctl.conf



重启linux系统之后

[root@10-9-100-26 ~]# sysctl -p



2.安装配置es

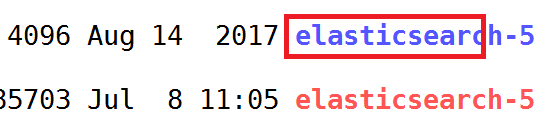
2.1安装包解压(3台)

/home/resources/elasticsearch-5.5.2.tar.gz

[root@10-9-100-26 software]# cp /home/resources/elasticsearch-5.5.2.tar.gz ./

解压安装包elasticsearch

[root@10-9-100-26 software]# tar -xf elasticsearch-5.5.2.tar.gz

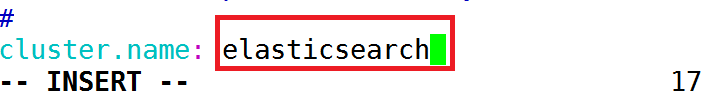


2.2单节点es配置(3台)

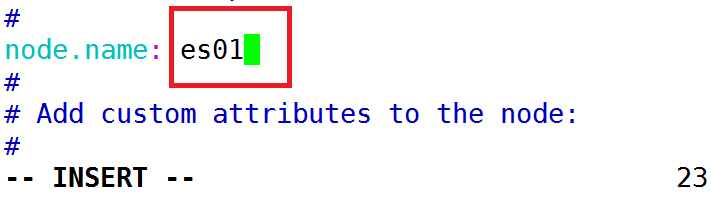
elasticsearch.yml,在es根目录config文件夹下;

[root@10-9-100-26 config]# vim elasticsearch.yml

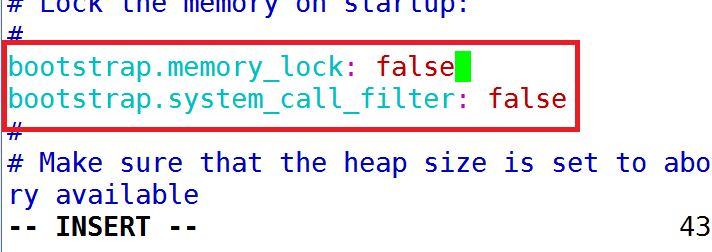
* 节点所在的集群名称默认elasticsearch



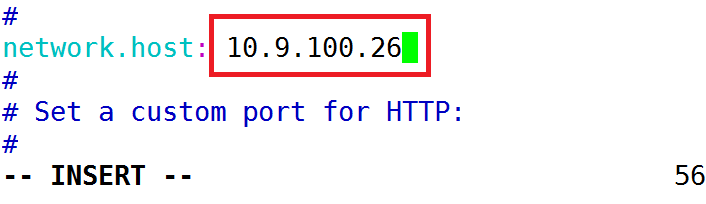
* 修改节点名称（第二、三台分别是es02和es03）



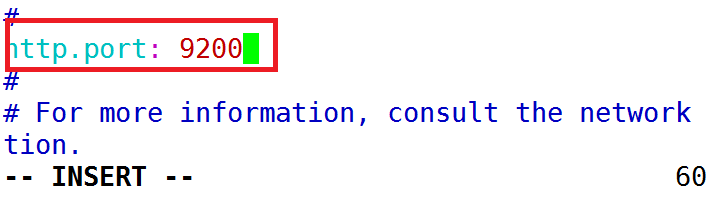
* 关闭bootstrap的插件加载(不关闭不影响,只是启动时会抛异常)



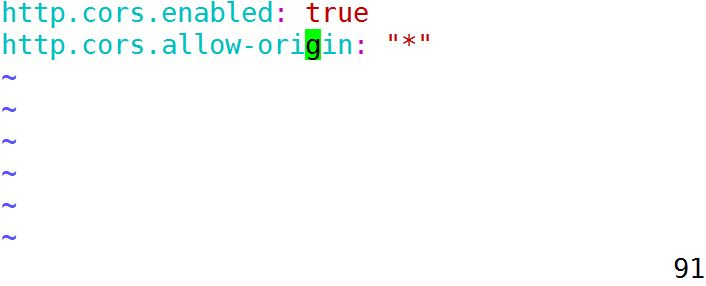
* 配置对外开启的ip访问地址,不配置只能本地访问（三台分别写自己ip）



* http协议请求的访问端口(javaclient代码默认访问端口9300)



* 配置http插件访问es的功能开启,末尾添加



3.启动es

es禁止使用root用户,需要另行添加一个用户单独操作elasticsearch.

3.1添加一个具有操作es权限的用户es

[root@10-9-100-26 config]# groupadd es

[root@10-9-100-26 config]# useradd es -g es -p es

3.2添加es用户的权限可以操作/home/software/elasticsearch文件夹

[root@localhost config]#chown -R es:es /home/software/elasticsearch-5.5.2

3.3使用es启动elasticsearch

使用root用户su到es用户无需密码;

[root@10-9-48-69 config]# su es

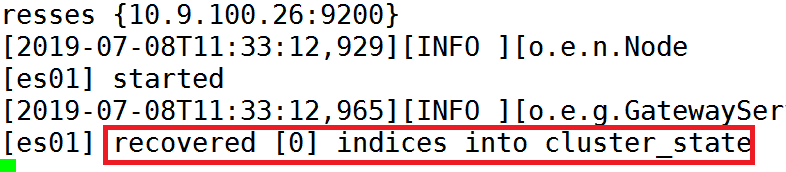
bin文件夹下的脚本文件elasticsearch;

* 后台启动(若要关闭：ps -ef|grep elasticsearch然后kill掉)

bin/elasticsearch -d

* 占用控制台看日志(我们以控制台启动)

bin/elasticsearch



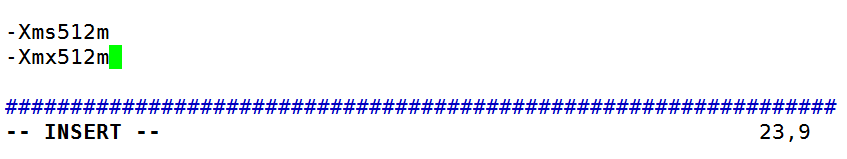
如果是自己的虚拟机，可能会jvm内存不够，切换到root用户，可以修改：

# vim [elasticsearch]/config/jvm.options

-Xms2g

-Xmx2g

修改为

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

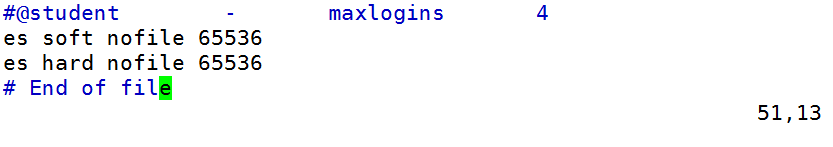
编辑该文件

/etc/security/limits.conf

后面加上：

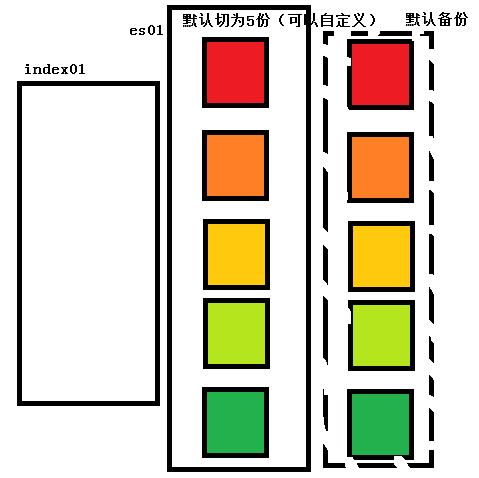
es soft nofile 65536

es hard nofile 65536



其中，es为用户名

4.索引文件的管理结构



## 12.4.es的dsl命令

### 12.4.1.linux的curl命令

linux提供的一个支持http协议请求的命令，可以发起get、post、put、delete请求

选项：

-X请求方式

-d请求体参数

例如：[root@bogon ~]# curl -XGET http://www.baidu.com

### 12.4.2.DSL命令入门

可以在课前资料中获取大部分es命令：

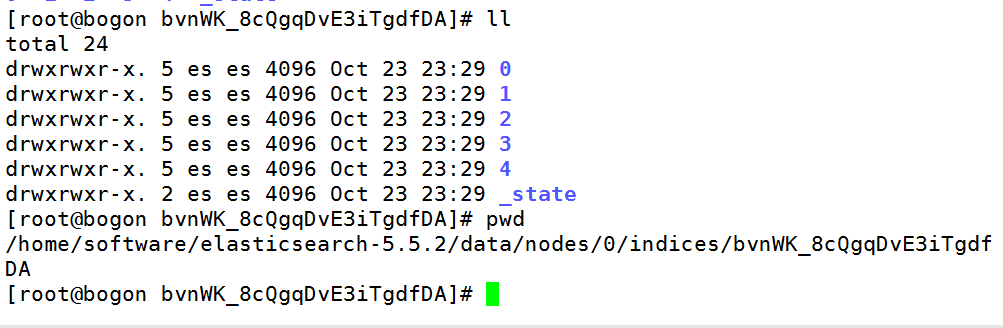
课前资料\01-课前资料\10-elasticsearch\02-其他资料\es使用命令.txt

将http://10.9.17.153全部替换为http://192.168.171.143（自己的ip）

案例一：新建索引

curl -XPUT <http://192.168.171.143:9200/index01>

es默认会将索引分布式存储为5个分片，查看是否分片：



由5个目录管理5个分片

由于所有内容观察不便，可以利用插件查看

### 12.4.3.可视化插件

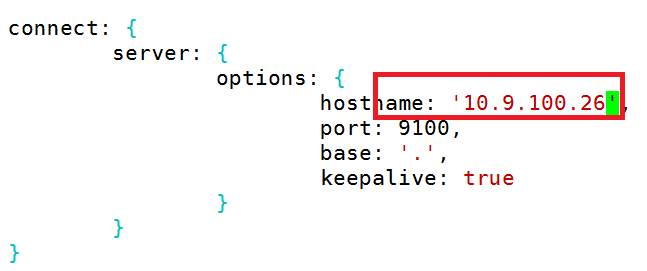
开启每个es对http插件支持(已经做了 http.cors)

配置head插件(html5 node.js)（已提前配置好）,视图结构

/home/presoftware/elasticsearch-head-master的内容

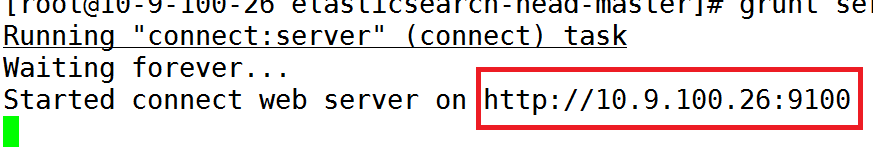
[root@10-9-100-26 elasticsearch-head-master]# vim Gruntfile.js

93行左右,修改host的ip地址为当前节点ip地址;



* 保存退出直接运行

[root@10-9-100-26 elasticsearch-head-master]# grunt server



* 在浏览器访问这个地址url
* 页面结构



### 12.4.4.DSL索引操作

**（1）创建索引**

curl -XPUT <http://192.168.171.143:9200/索引名称>

创建完成后，es会在底层切分为5个分片和5个副本

**（2）读写权限**

curl -XPUT -d '{"blocks.read":false}' http://192.168.171.143:9200/index01/\_settings

blocks.read：true表示禁止读，默认false

blocks.read\_only：true表示只读

blocks.write：true表示禁止写

**（3）查看索引**

查看单个索引：

curl -XGET <http://192.168.171.143:9200/index01/_settings>

查看多个索引：

curl -XGET [http://192.168.171.143:9200/index01,index02/\_settings](http://192.168.171.143:9200/index01,blog/_settings)

**（4）删除索引**

curl -XDELETE <http://192.168.171.143:9200/index02>

**（5）开启、关闭索引**

关闭

curl -XPOST http://192.168.171.143:9200/index01/\_close

打开

curl -XPOST http://192.168.171.143:9200/index01/\_open

es每次启动都会分配线程管理监听索引文件，开启和关闭索引，其实就是开启和关闭这些监听的线程，有时候，有些索引长期不用，就可以关闭，如果需要使用时，再开闭

### 12.4.5.文档操作命令

**（1）新建文档**

curl -XPUT -d '{"域名":域值,"title":"es简介","content":"es好用好用真好用"}' <http://192.168.171.143:9200/索引名称/type类型/文档id>

例如：

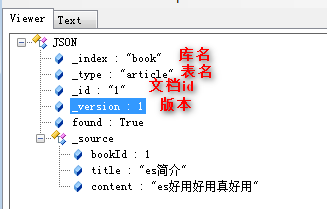
curl -XPUT -d '{"bookId":1,"title":"es简介","content":"es好用好用真好用"}' <http://192.168.171.143:9200/book/article/1>

**（2）获取文档**

获取一个文档

curl -XGET <http://192.168.171.143:9200/索引名称/type类型/文档id>

curl -XGET <http://192.168.171.143:9200/book/article/1>



\_version：对此文档的每次操作，version都会增加1

作用：es集群中通过version可以避免读取错误数据

获取多个文档

curl -XGET -d '{"docs":[{"\_index":"book","\_type":"article","\_id":"1"},{"\_index":"book","\_type":"article","\_id":"2"}]}' <http://192.168.171.143:9200/_mget>

**（3）删除文档**

curl -XDELETE <http://192.168.171.143:9200/book/article/1>

### 12.4.6.搜索索引

准备一些文档数据

curl -XPUT -d '{"id":1,"title":"es简介","content":"es好用好用真好用"}' http://192.168.171.143:9200/index01/article/1

curl -XPUT -d '{"id":1,"title":"java编程思想","content":"这就是个工具书"}' http://192.168.171.143:9200/index01/article/2

curl -XPUT -d '{"id":1,"title":"大数据简介","content":"你知道什么是大数据吗，就是大数据"}' <http://192.168.171.143:9200/index01/article/3>

**（1）查询所有match\_all**

查询某个索引的全部文档

curl -XGET http://192.168.171.143:9200/index01/\_search -d '{"query": {"match\_all": {}}}'

**（2）词项查询term query**

1.curl -XGET http://192.168.171.143:9200/index01/\_search -d '{"query":{"term":{"title":"java"}}}'

2.curl -XGET http://192.168.171.143:9200/index01/\_search -d '{"query":{"term":{"title":"java编程思想"}}}'

3.curl -XGET http://192.168.171.143:9200/jtdb\_item/\_search -d '{"query":{"term":{"title":"双卡双"}}}'

第一条java可以查到数据

第二条java编程思想查不到数据

第三条报错找不到索引

从上可以看出，es默认的对document保存时使用的分词器是标准分词器（standarAnalyzer）

如何将分词器设置为IK分词器？

**（3）match query**

curl -XGET http://192.168.171.143:9200/index01/\_search -d '{"query":{"match":{"title":"java编程思想"}}}'

可以搜到结构，matchQuery会对搜索关键字进行分词处理，只要搜索的内容中包含这些分词结果的部分，都可以匹配

### 12.4.7.配置IK分词器

**（1）IK分词器插件安装**

1.将插件资源解压到es根目录的plugins中,修改文件夹analysis-ik

/home/software/elasticsearch-5.5.2/plugins

[root@10-9-100-26 plugins]# cp /home/resources/elasticsearch-analysis-ik-5.5.2.zip ./

2.解压默认文件夹名称elasticsearch，需修改文件名称

这里使用unzip命令,需要unzip/zip插件支持,如果空镜像,可以通过yum获取插件 yum -y install unzip zip

[root@10-9-100-26 plugins]# unzip elasticsearch-analysis-ik-5.5.2.zip

[root@10-9-100-26 plugins]# rm -f elasticsearch-analysis-ik-5.5.2.zip

[root@10-9-100-26 plugins]# mv elasticsearch/ analysis-ik

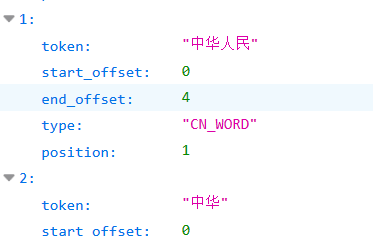
重启elasticsearch

**（2）ik分词器的测试使用**

访问es提供的接口地址,传递参数使用测试iK分词器;

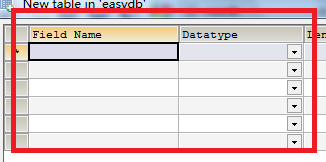
浏览器http://192.168.171.143:9200/index01/\_analyze?analyzer=ik\_max\_word&text=中华人民共和国

返回结果可以看到ik的词项数据



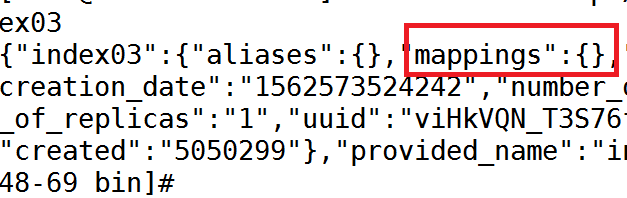
### 12.4.8.mapping设置

es中除了索引,类型,文档,域,词项等概念以外,mapping表示对一个索引的结构做指定设置的概念,例如,存储的document每个域属性,到低是int 还是long ,是string还是text,需要分词,分词器指定某一个设置.---相当于数据库中表格的schema



默认的mapping查询

在刚刚创建的索引中,mapping是空



当使用默认mapping(动态mapping)存储数据,会根据数据的值的形态确定类型,数字--long,字符串--使用standard分词器,keyword(String类型)保存一个域数据的整体字符串



* 人为指定静态mapping

设置数字是int,设置字符串不要keyword就要text,分词器IK\_MAX\_WORD

* 新建一个索引index04

[root@192.168.171.143 bin]# curl -XPUT [http://192.168.171.143:9200/index04](http://10.9.100.26:9200/index04)

* 对index04的article的类型的mapping定义properties

{"properties":

{

"id":{"type":"integer"},

"content":{"type":"text","analyzer":"ik\_max\_word"},

"title":{"type":"text","analyzer":"ik\_max\_word"}

}

}

curl -XPUT [http://192.168.171.143:9200/index04/\_mapping/article](http://10.9.100.26:9200/index04/_mapping/article) -d '{"properties":{"id":{"type":"integer"},"content":{"type":"text","analyzer":"ik\_max\_word"},"title":{"type":"text","analyzer":"ik\_max\_word"}}}'

返回true表示mapping设置成功;验证ik分词器指定是否正确

可以在静态mapping对应的索引添加一些数据,通过词项查询,查看是否有对应的词项数据,如果有,ik写数据时一定进行了分词计算;

## 12.5.elasticsearch集群配置

单节点es并不支持高并发,高可用,需要es的集群实现分布式高可用的结构,由于es存在发现节点的功能模块,只要配置单节点时按照集群设置的规则配置elasticsearch.yml,直接启动多个节点自动加入一个集群进行功能的提供;

1.配置多个节点(3)集群

问题:第一个节点已经配置了ik分词器,但是第二第三节点如果不配置ik加入集群后会出现问题;做到ik分词器配置统一,数据计算使用的分词器也要统一(之前ik分词器使用的所有索引数据清空)

1.1各自配置3个单独es节点（已经做过）

按照elasticseasrch的环境搭建页准备3个单独节点;

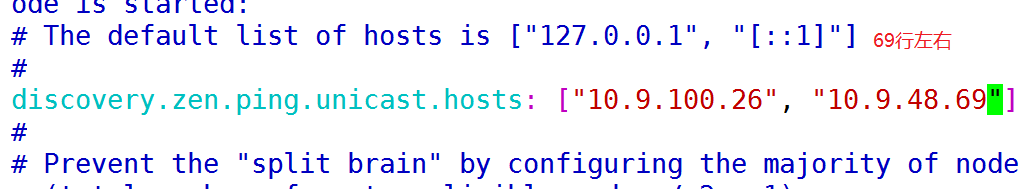
1.2集群的配置内容

* 三个节点的集群名称要相同

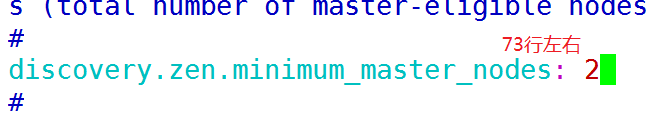
elasticsearch

* 节点名称在同一个集群不同es01 es02 es03
* 对集群节点添加一些配置内容
  + 协调器list(3台机器都配置，内容相同，ip从三台机器随便选取两个)

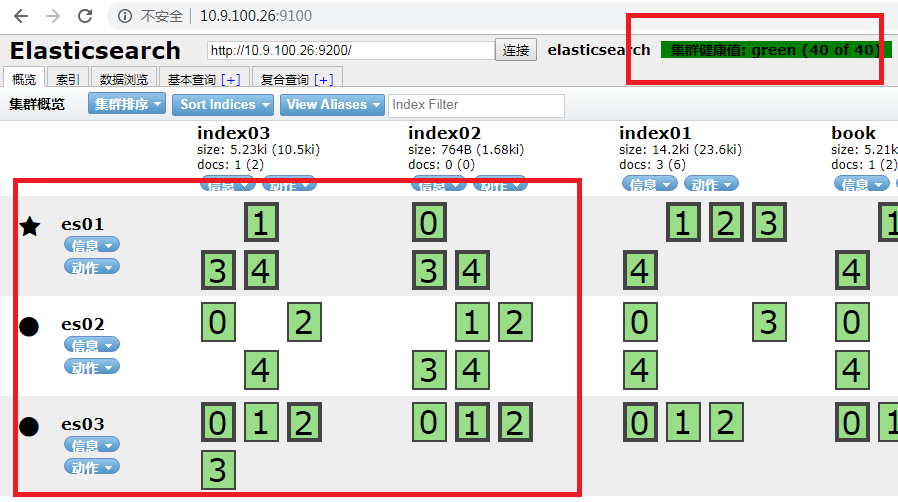
整个集群加入发起人;从3台节点中选取2个节点作为配置协调器,三个节点的协调器配置保持一致

[root@bogon config]# vim elasticsearch.yml 

* 最小master的数量:2



* 启动三个节点观察集群效果

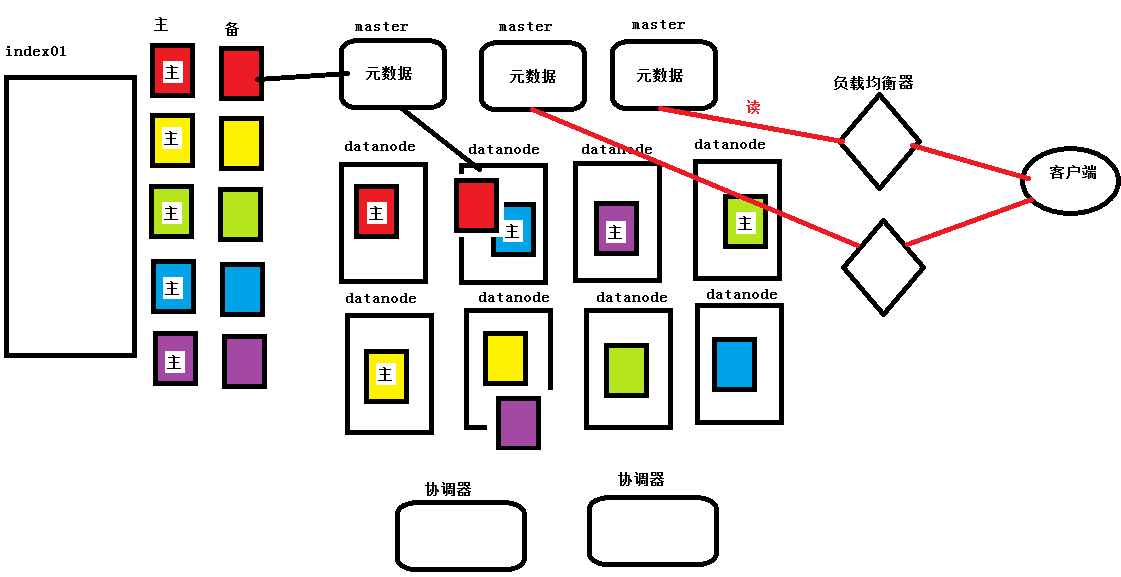


* 分片: 默认5个主分片,做了切分处理,由不同节点维护,从分片做了切分处理,并且同一个号码的主从分片不在同一个节点

节点越多的时候,提升分布式并发能力,分片数量适当提高,副本越多,高可用能力越高.

## 12.6.elasticsearch集群结构

### 12.6.1.结构图



### 12.6.2.角色介绍

**master：**

会在启动时进行选举，最终只有一个现役master，其他master同步现役master的数据

元数据：存储在master中，用来记录文件块（索引分片）的大小、位置、分片数、备份数等信息

**datanode：**

存数据

**负载均衡器：**

客户端连接负载均衡器，通过负载均衡器实现对集群master的负载均衡的读

**协调器：**

任意一个节点加入时，需要协调器协调，还有例如集群通信

**集群的每台服务器都可以具备哪些角色？**

可以在配置文件中指定每个节点的角色

如果不指定，每台服务器默认会具备以下角色

master：默认是true

datanode：默认是true

负载均衡器：默认是true

协调器：需要在配置文件中指定

**分析当前三台服务器具备的角色：**

es01：具备master、datanode、负载均衡器

es02：具备master、datanode、负载均衡器、协调器

es03：具备master、datanode、负载均衡器、协调器

### 12.6.3.选举逻辑（RBULLY算法）

1.连接协调器，获取当前集群的所有状态，包括master选举相关状态

2.判断当前集群的activeMaster是否为空

如果不为空，选举结束，跳到第五步（老大已经选完）

如果为空，进入第三步

3.将所有master作为候选人添加到cadidateMaster中

4.判断cadidate是否满足最小master数量

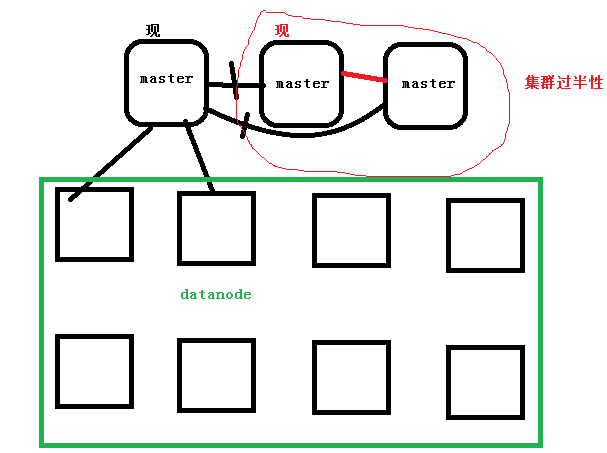
如果不满足，跳到第一步重新选举

如果满足，直接从候选人中找id最小的放入activeMaster中

5.选举完成

### 12.6.4.脑裂

split brain，高可用分布式技术常见的一种管理机制错误



**es主节点机制**

主节点管理整个集群，现役master计算和生成metadata（元数据），其他master同步数据，如果主节点宕机，其他master需要选举出新的主节点，经过RBULLY算法

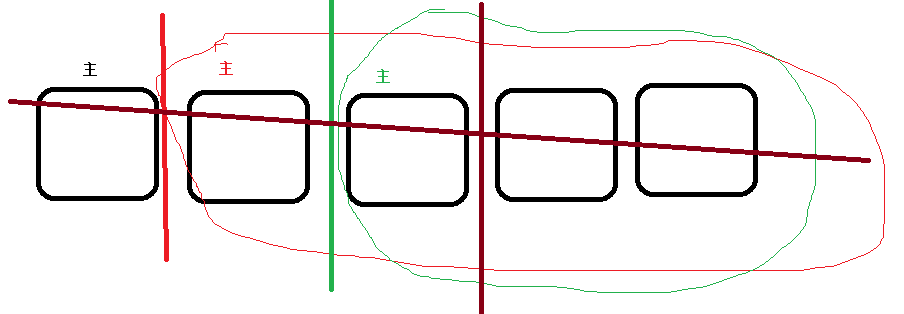
**脑裂**

由于网络波动，导致其他节点错误的认为主节点宕机，从而经过选举逻辑，选出第二个主节点，此时集群中就会存在多个主节点（现役master），该听谁的--脑裂

**es解决脑裂的办法**

主节点所在master集群必须满足过半性，需要配置最小master数量，过半数量：master可用数量/2+1

例如：



当集群的某个主节点发生脑裂，其他节点会选举出第二个主节点，由于第二个主节点所在集群满足过半，所以由第二个主节点作为集群主节点，如果主节点集群都不能满足过半性，则集群失效

## 12.7.java客户端transportclient

### 12.7.1.操作索引

**（1）创建maven工程**

注意：不要和lucene的测试工程在同一个工程，底层可能会有版本冲突

直接将工程整合到easymall中

1.quickstart（easymall-common-elasticsearch）

2.pom继承easymall-parent，依赖transportclient

|  |
| --- |
| <parent>  <groupId>cn.tedu</groupId>  <artifactId>easymall-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <!-- es核心依赖 -->  <dependency>  <groupId>org.elasticsearch</groupId>  <artifactId>elasticsearch</artifactId>  <version>5.5.2</version>  </dependency>  <!-- es的java客户端 -->  <dependency>  <groupId>org.elasticsearch.client</groupId>  <artifactId>transport</artifactId>  <version>5.5.2</version>  </dependency> |

**（2）transport连接es**

测试案例：

连接es集群：

|  |
| --- |
| **package** cn.tedu.elasticsearch.test;  **import** java.net.InetSocketAddress;  **import** org.elasticsearch.client.transport.TransportClient;  **import** org.elasticsearch.common.settings.Settings;  **import** org.elasticsearch.common.transport.InetSocketTransportAddress;  **import** org.elasticsearch.transport.client.PreBuiltTransportClient;  **import** org.junit.Before;  **import** org.junit.Test;  **public** **class** TransportTest {  /\*  \* 连接es集群  \*/  **private** TransportClient client;  @Before  **public** **void** connection(){  //准备客户端对象  //settings是配置对象，empty是默认配置（默认连接名称为elasticsearch的集群）  //Settings.builder().put("cluster.name", "es").build();  client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  //添加3个负载均衡节点  client.addTransportAddress(  **new** InetSocketTransportAddress(  **new** InetSocketAddress("192.168.171.138",9300)));  client.addTransportAddress(  **new** InetSocketTransportAddress(  **new** InetSocketAddress("192.168.171.142",9300)));  client.addTransportAddress(  **new** InetSocketTransportAddress(  **new** InetSocketAddress("192.168.171.143",9300)));  }  } |

利用client操作索引

|  |
| --- |
| /\*  \* 索引操作  \*/  @Test  **public** **void** indexManage(){  //获取一个管理对象  IndicesAdminClient admin = client.admin().indices();  //创建索引  CreateIndexRequestBuilder request =  admin.prepareCreate("emindex");  //admin.prepareDelete("index02").get();  //admin.prepareExists("index01");  //发送请求  CreateIndexResponse response = request.get();  System.***out***.println(response.isShardsAcked());  } |

### 12.7.2.操作文档

easymall中索引都是商品，可以将商品对象（源数据）放入索引中

准备一个对象article

|  |
| --- |
| **package** cn.tedu.elasticsearch.test;  **public** **class** Article {  **private** Integer id;  **private** String title;  **private** String content;  get&set  } |

新增文档

|  |
| --- |
| /\*  \* 新增文档  \*/  @Test  **public** **void** documentManage() **throws** Exception{  //假设数据是源数据(从数据库获取的)  Article a1 = **new** Article();  a1.setId(2);  a1.setTitle("美国发起贸易战");  a1.setContent("导致全球变暖");  //curl -XPUT -d '{"bookId":1,"title":"es简介","content":"es好用好用真好用"}'  //http://192.168.171.143:9200/book/article/1  //将对象转换为json  ObjectMapper mapper = **new** ObjectMapper();  String jsonString = mapper.writeValueAsString(a1);  IndexRequestBuilder request =  client.prepareIndex("index01","article",a1.getId()+"");  IndexResponse response = request.~~setSource~~(jsonString).get();  System.***out***.println(response.getVersion());  System.***out***.println(response.getId());  System.***out***.println(response.getType());  System.***out***.println(response.getShardInfo());  } |

获取文档

|  |
| --- |
| @Test  **public** **void** getDocument(){  //curl -XGET http://192.168.171.143:9200/book/article/1  GetRequestBuilder request =  client.prepareGet("index01", "article","1");  GetResponse response = request.get();  System.***out***.println(response.getSourceAsString());  Map<String, Object> map = response.getSource();  System.***out***.println(map.get("id"));  System.***out***.println(map.get("title"));  System.***out***.println(map.get("content"));  } |

### 12.7.3.搜索索引

|  |
| --- |
| /\*  \* 通过不同的query进行搜索  \*/  @Test  **public** **void** searchTest(){  //查询对象,可以构造不同的查询对象  MatchQueryBuilder query =  QueryBuilders.*matchQuery*("title", "美国死没死");  //指定从哪个索引（库）中搜，指定查询对象，指定从哪里开始，搜多少条  SearchRequestBuilder request = client.prepareSearch("index01").setQuery(query)  .setFrom(0).setSize(5);  SearchResponse response = request.get();  SearchHits hits = response.getHits();  //总共搜到多少条  System.***out***.println(hits.totalHits);  //最大评分  System.***out***.println("最大"+hits.getMaxScore());  SearchHit[] hit = hits.getHits();  **for** (SearchHit searchHit : hit) {  System.***out***.println(searchHit.getScore());  System.***out***.println(searchHit.getSourceAsString());  }  } |

## 12.8.easymall搜索功能

### 12.8.1.elasticsearch依赖工程

延续easymall-common-elasticsearch工程

1.quickstart（完成）

2.pom（完成）

3.配置文件（这是依赖工程，由子工程配置，这里不配）

4.读取配置文件的配置类，将配置内容设置到java客户端，并交给spring维护@Bean

|  |
| --- |
| **package** cn.tedu.elasticsearch.config;  **import** java.net.InetSocketAddress;  **import** java.util.List;  **import** org.elasticsearch.client.transport.TransportClient;  **import** org.elasticsearch.common.settings.Settings;  **import** org.elasticsearch.common.transport.InetSocketTransportAddress;  **import** org.elasticsearch.transport.client.PreBuiltTransportClient;  **import** org.springframework.boot.context.properties.ConfigurationProperties;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  @Configuration  @ConfigurationProperties(prefix="elasticsearch")  **public** **class** ESConfig {  **private** List<String> nodes;  @Bean  **public** TransportClient initESTransportClient(){  TransportClient client = **new**  PreBuiltTransportClient(Settings.***EMPTY***);  **for** (String node : nodes) {  String ipAddr = node.split(":")[0];  Integer port = Integer.*parseInt*(node.split(":")[1]);  client.addTransportAddress(  **new** InetSocketTransportAddress(  **new** InetSocketAddress(ipAddr,port)));  }  **return** client;  }  **public** List<String> getNodes() {  **return** nodes;  }  **public** **void** setNodes(List<String> nodes) {  **this**.nodes = nodes;  }  } |

### 12.8.2.搜索功能搭建

1.quickstart（easymall-microservice-search）

2.pom文件继承easymall-parent，依赖elasticsearch工程、resources、repository（本可以不用，因为搜索时只展示索引的数据，我们这里需要将数据导入到索引，直接去数据库了）

问题：没有去本地库，而是去阿里库下载去了

原因：是springboot自带的版本2.4.6（本地库是5.5.2）

解决：

修改库为5.5.2

方式一：可以直接在子工程修改

方式二：在父工程的dependencyManagement标签下的dependencies中添加如下依赖，这是声明式依赖，子工程如果使用，只需定义资源，无需版本

可以将依赖工程pom中es相关依赖的版本删除

|  |
| --- |
| <!-- es核心依赖 -->  <dependency>  <groupId>org.elasticsearch</groupId>  <artifactId>elasticsearch</artifactId>  <version>5.5.2</version>  </dependency>  <!-- es的java客户端 -->  <dependency>  <groupId>org.elasticsearch.client</groupId>  <artifactId>transport</artifactId>  <version>5.5.2</version>  </dependency> |

3.配置文件

|  |
| --- |
| server.port=10006  spring.datasource.driverClassName=com.mysql.jdbc.Driver  spring.datasource.url=jdbc:mysql:///easydb  spring.datasource.username=root  spring.datasource.password=root  spring.datasource.type=com.alibaba.druid.pool.DruidDataSource  spring.datasource.initialSize=5  spring.datasource.maxActive=50  spring.datasource.maxIdle=10  spring.datasource.minIdle=5  mybatis.typeAliasesPackage=com.jt.common.pojo  mybatis.mapperLocations=classpath:mapper/\*.xml  mybatis.configuration.mapUnderscoreToCamelCase=true  mybatis.configuration.cacheEnables=false  spring.application.name=searchservice  eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka  elasticsearch.nodes=192.168.171.138:9300,192.168.171.142:9300,192.168.171.143:9300 |

4.启动类

|  |
| --- |
| **package** cn.tedu;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication  @EnableEurekaClient  @MapperScan("cn.tedu.search.mapper")  **public** **class** StarterSearchCenter {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(StarterSearchCenter.**class**, args);  }  } |

### 12.8.3.将商品数据存入索引

将商品的全部数据读取并输出到索引文件：emindex/product

自定义一个接口文件：

|  |  |
| --- | --- |
| 访问路径 | http://localhost:10006/search/manage/create |
| 后台接收路径 | /search/manage/create |
| 请求参数 | 无 |
| 返回值 | String ok\fail |

访问到此接口时，将数据库所有商品数据存入索引中（emindex/product）

SearchController

|  |
| --- |
| **package** cn.tedu.search.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/search/manage/")  **public** **class** SearchController {  @Autowired  **private** SearchService searchService;  @RequestMapping("/create")  **public** String createEmindex(){  **try** {  searchService.createEmindex();  **return** "ok";  } **catch** (Exception e) {  **return** "fail";  }  }  } |

SearchService

|  |
| --- |
| **package** cn.tedu.search.service;  **import** java.util.List;  **import** org.elasticsearch.action.index.IndexRequestBuilder;  **import** org.elasticsearch.client.transport.TransportClient;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** com.jt.common.pojo.Product;  **import** com.jt.common.utils.MapperUtil;  **import** cn.tedu.search.mapper.SearchMapper;  @Service  **public** **class** SearchService {  @Autowired  **private** SearchMapper searchMapper;  @Autowired  **private** TransportClient client;  **public** **void** createEmindex() **throws** Exception {  //读取数据库product表的所有内容  List<Product> plist = searchMapper.queryAll();  //将每个对象转换为json  **for** (Product product : plist) {  String pjson = MapperUtil.***MP***.writeValueAsString(product);  //存入索引  IndexRequestBuilder request = client.prepareIndex(  "emindex","product",product.getProductId());  request.~~setSource~~(pjson).get();  }  }  } |

SearchMapper

|  |
| --- |
| **package** cn.tedu.search.mapper;  **import** java.util.List;  **import** com.jt.common.pojo.Product;  **public** **interface** SearchMapper {  List<Product> queryAll();  } |

SearchMapper.xml

|  |
| --- |
| <select id=*"queryAll"* resultType=*"Product"*>  select \* from t\_product;  </select> |

启动测试：

启动search微服务，浏览器访问http://localhost:10006/search/manage/create

如果出现ok，说明成功

可以在es页面看到数据

### 12.8.4.搜索功能开发

|  |  |
| --- | --- |
| js请求地址 | <http://www.easymall.com/searchs/query?query=>三星&page=1&rows=5 |
| 后台接收 | /search/manage/query?query=三星&page=1&rows=5 |
| 请求方式 | get |
| 请求参数 | String query,Integer page,Integer rows |
| 返回数据 | List<Product>查询返回结果 |

SearchController

|  |
| --- |
| @RequestMapping("/query")  **public** List<Product> queryProduct(String query,Integer page,Integer rows){  **return** searchService.queryProduct(query,page,rows);  } |

SearchService

|  |
| --- |
| **public** List<Product> queryProduct(String query, Integer page, Integer rows) {  **try** {  //构造查询对象  MatchQueryBuilder matchQuery =  QueryBuilders.*matchQuery*("productName", query);  //去索引查询,指定哪个索引，从哪里开始，查多少个数据  SearchResponse response = client.prepareSearch(  "emindex").setQuery(matchQuery)  .setFrom((page-1)\*rows).setSize(rows).get();  SearchHits hits = response.getHits();  System.***out***.println("一共查到："+hits.totalHits+"条");  SearchHit[] hit = hits.getHits();  //声明返回的List  List<Product> plist = **new** ArrayList<Product>();  **for** (SearchHit searchHit : hit) {  String pjson = searchHit.getSourceAsString();  Product product = MapperUtil.***MP***.readValue(pjson, Product.**class**);  plist.add(product);  }  **return** plist;  } **catch** (Exception e) {  e.printStackTrace();  **return** **null**;  }  } |

### 12.8.5.页面整合

**（1）nginx整合**

location /searchs {

proxy\_pass http://127.0.0.1:9005/zuul-search/search/manage;

add\_header 'Access-Control-Allow-Credentials' 'true';

add\_header 'Access-Control-Allow-Origin' '\*';

}

**（2）zuul整合**

zuul.routes.zuul-search.path=/zuul-search/\*\*

zuul.routes.zuul-search.serviceId=searchservice

### 12.8.6.测试

开启eureka、nginx、zuul、search、elasticsearch