# ES简介

Shay Banon在2004年创造了Elasticsearch的前身，称为Compass。在考虑Compass的第三个版本时，他意识到有必要重写Compass的大部分内容，以“创建一个可扩展的搜索解决方案”。因此，他创建了“一个从头构建的分布式解决方案”，并使用了一个公共接口，即HTTP上的JSON，它也适用于Java以外的编程语言。Shay Banon在2010年2月发布了Elasticsearch的第一个版本。

ES是一个基于lucene实现的搜索服务.当客户端调用es的功能时,可以实现全文检索的各种操作,并且不需要关心lucene的底层代码.只要客户端支持http协议就可以了.

# ES结构

|  |
| --- |
| 计算机生成了可选文字: 接口层 http协议 一堆url访问地 REST风格 支持其他序列 化协议thr血 TCP/IP 功能扩展 es进程本身 具备的各种功能 分布式集群搭建 实现索引文件 管理功能 打开关闭索弓 存储层 ES结构 本地 分布式 对接hdfs等 分布式文件系 |

### 存储层

可以利用lucene创建索引时，决定将索引存储在哪里（都是分布式的）

本地:数据分片切分

远程hdfs:分布式文件系统

### 功能扩展

es做一个启动的服务,可以创建集群,实现高可用分布式

扩展了对lucene创建的索引管理功能

### 接口层

外界客户端可以通过哪种途径连接es操作es

命令操作:http协议

java代码:TCP/IP协议

# REST风格

### 12000年http创始人一篇论文

论文主旨:

大家正在使用的http协议方式,和我创建http协议的目的不一样---你们在乱用,瞎用,不会用

遵循REST风格---你要不会用,就了解什么是REST风格

### URI的使用：

表示资源ID，不同的URI地址应该指向不同的资源，单独看这个概念很多系统http请求接口设计就已经违反了URL定义。

例如：对同一商品使用了不同的URI进行操作

新增商品: product/save

修改商品: product/update

查询商品: product/query

满足URI：

product/manage/{productId},新增商品,修改商品,查询商品访问一个地址

### 什么是REST风格

representational state transfer(表象性状态转变).

http请求方式的使用

REST风格中定义

put请求表示新增

post请求表示整体修改覆盖

delete请求表示删除

get请求表示查询

配合uri表示资源的定义,可以使用不同的请求方式达到增删查改的操作目录

新增商品

put:product/manage/{productId}

修改商品

post:product/manage/{productId}

删除商品

delete:product/manage/{productId}

查询商品

get:product/manage/{productId}

REST风格

uri表示访问的资源是谁;

http请求方式表示要对资源做什么操作;

### 为什么要满足REST风格

系统的版本上下很容做成兼容

不满足REST风格,版本兼容就要花费更多的成本

例如:

版本1.0

post:product/manage/save进行新增

版本2.0

post:product/manage/insert进行新增

很多web应用框架都支持REST风格,比如springmvc

@RequestMapping(value="product/mange/save",method=RequestMethod.GET)

方法1:处理get请求

@RequestMapping(value="product/mange/save",method=RequestMethod.POST)

方法2:处理post请求

# ES安装启动配置

### 目录结构

**bin**:启动命令.

**config**:

elastcisearch.yml:es节点核心属性配置文件.

jvm.options:es启动的jvm配置 占用内存.

**logs**:日志文件.

**data**:一个es节点保存的所有索引,集群状态数据(存在的话会影响集群搭建)

**plugins**:IK分词器插件

### ES启动

不允许root用户启动软件.使用一个有权限操作es的普通用户 es:123/es:es

问题重要 启动logs无权限报错： Permission denied

root启动会导致logs文件中出现root创建的日志,es用户无法访问无法使用.删除这个log重新启动es

#su es 从root进入es用户

$exit 从es退出到root

启动命令

es根目录 bin文件夹下:

#elasticsearch 在控制台输出日志

#elasticsearch -d在后台运行

|  |
| --- |
|  |

### es的配置文件

在ES根目录config文件夹下，编辑配置文件elasticsearch.yml

* + 17行:

cluster.name

自定义的集群名称,默认elasticsearch. 同一个集群所有节点,名字一致

* + 23行:

node.name

节点名称,同一个集群节点名字相互不一样

* + 56行/60行

network.host

当前服务器被外部访问的ip地址可以是域名，默认127.0.0.1 只允许本机访问这个es节点

http.port

http协议访问操作es时需要通过这个端口.默认9200

* + 69行

discovery.zen.ping.unicast.hosts

集群发现节点的协调器list,保存了集群中所有协调器的ip地址--搭建集群使用

* + 73行

discovery.zen.minimum\_master\_nodes

防止集群脑裂的配置，默认已被注释的

# ES索引分片结构

### http协议创建索引观察结构

* + 调用创建索引的命令

curl 支持在linux系统发送http请求

-X 请求方式 默认GET

curl -XGET url

curl -XPOST url

 ….

-d 添加请求体数据

curl -XPOST -d '请求体数据' url

创建一个索引index01：没有数据，是空的

|  |
| --- |
| [root@10-42-175-170 ~]# curl -XPUT http://10.42.175.170:9200/index01  {"acknowledged":true,"shards\_acknowledged":true} |

### 索引分片结构

上例中创建完index01索引后会创建以下目录，建了5个索引文件

分布式中可以把5个索引文件中的某几个放到其它节点。

1DiLOGd3ReCgZi5X9jZifQ索引index01的id

|  |
| --- |
|  |

es默认对每一个索引文件都计算分片,5个分片.每一个数据分片都是一个完整的lucene创建的索引。

|  |
| --- |
|  |

### 数据的分片计算逻辑

如果我们向index01写入数据 放进去一批document。会根据document的id值进行hash取 对5取余，每一个document都会对应0-4号分片其中的一个，最后某个document放到一个分片中。

分片不会少一直5个，所以用不着一致性hash计算

|  |
| --- |
| 计算机生成了可选文字: index01  shards num—5  S:  lucene:writer.commit() 0 号 索 弓 |  lucene:writer.commit() 1 ． 号 索 弓 丨  lucene:writer.commit() 2 号 索 弓 |  lucene:writer.commit() 3 ． 号 索 弓 |  lucene:writer.commit() 4 号 索 弓 |  囗 囗 |

### 分片高可用

es除了计算默认5个分片,默认为每一个分片在其它节点创建一个副本分片(主从复制)

副本默认1个。

|  |
| --- |
| 计算机生成了可选文字: index01  shards num—5  S:  lucene:writer.commit() 0 号 索 弓 |  lucene:writer.commit() 1 ． 号 索 弓 丨  lucene:writer.commit() 2 号 索 弓 |  lucene:writer.commit() 3 ． 号 索 弓 |  lucene:writer.commit() 4 号 索 弓 | |

因为现在只有一个es节点,即使计算除了从分片,没地方放.

### es的head插件使用（查看数据）

/home/presoftware/elasticsearch-head-master文件夹.

保证当前运行这个head插件的服务器的hosts文件中有正确的myhost映射ip

在head插件的根目录运行启动

#grunt server ：myshost为系统本地域名，对应127.0.0.1，外部访问用外网ip

|  |
| --- |
| [root@10-42-175-170 elasticsearch-head-master]# grunt server  Running "connect:server" (connect) task  Waiting forever...  Started connect web server on http://myhost:9100 |

网页访问：

|  |
| --- |
|  |

# ES http协议命令

通过REST风格的http请求,实现对ES的各种操作.

### 索引管理

#### 创建索引

新增一个index02的索引

|  |
| --- |
| [root@10-42-175-170 ~]# curl -XPUT http://10.42.175.170:9200/index02  {"acknowledged":true,"shards\_acknowledged":true} |

#### 索引读限制

es在创建索引的底层使用lucene逻辑,但分配更多线程资源管理索引，例如对一个索引的读的限制.可以在访问一个索引时携带一个参数等。

{"blocks.read":true/false} ：true表示开启读限制,false不开启读限制.

|  |
| --- |
| [root@10-42-175-170 ~]# curl -XPUT -d '{"blocks.read":true}' http://10.42.175.170:9200/index01/\_settings  {"acknowledged":true} |

读取时卡主：表示限制读取了

|  |
| --- |
|  |

#### 查询索引

查询index01索引

|  |
| --- |
| [root@10-42-175-170 ~]# curl -XGET http://10.42.175.170:9200/index01 |
|  |

还可以查询多个索引

|  |
| --- |
| curl -XGET <http://10.9.48.69:9200/index01,index02> |

#### 删除索引

|  |
| --- |
| curl -XDELETE <http://10.9.48.69:9200/index02> |

#### 打开/关闭索引

关闭

|  |
| --- |
| #curl -XPOST <http://10.9.48.69:9200/index01/_close> |

打开

|  |
| --- |
| #curl -XPOST <http://10.9.48.69:9200/index01/_open> |

一旦打开,操作正常执行,可以访问index所有数据,状态,document搜索

一旦关闭,在当前es里就再也不能执行与这个所有有关的任何命令.

一些陈旧历史数据,不能删除数据,可以减少系统对他管理的线程资源分片,关闭

### 文档管理

操作完索引后,可以通过es的命令,和接口创建删除等管理索引文件,但是任何一个索引文件中都需要保存大量的document文档;客户端连接es的服务器,传递给es一批文档数据,es才会按照要去写入到索引中.

#### 新增文档

格式：curl –XPUT –d 文档内容json url

|  |
| --- |
| curl -XPUT -d '{"id":10,"title":"es简介","content":"es好用好用真好用"}' <http://10.9.48.69:9200/index01/article/1> |

index01：索引名

article：type

1：文档id

重要问题 type类型是什么? es中维护管理的数据整体结构,完全可以对照mysql理解

|  |  |
| --- | --- |
| mysql | ES |
| database | index |
| tables | type:同一个类型的document域属性结构一致的 |
| rows | document |
| columns | fields |

返回结果：

|  |
| --- |
| {  "\_index": "index02",  "\_type": "article",  "\_id": "1",  "\_version": 1,对同一个文档的写操作增删改都会在版本version进行自增1,目的是分布式高可用中保持数据复制的有效判断.数据一致性。  "result": "created",  "\_shards": {  "total": 2,  "successful": 1,  "failed": 0  },  "created": true  } |

#### 获取文档

明确你获取文档的3个坐标属性 index type id

##### 单个文档

|  |
| --- |
| curl -XGET <http://10.9.48.69:9200/index02/article/1> |
| {  "\_index": "index02",  "\_type": "article",  "\_id": "1",  "\_version": 1,  "found": true,  "\_source": {  "id": 10,  "title": "es简介",  "content": "es好用好用真好用"  }  } |

##### 获取多个文档

需要在请求体中,以参数形式,携带多个文档对应的index/type/id

|  |
| --- |
| 格式：Curl –XGET json数组 url |
| Curl -XGET -d '{"docs":[{"\_index":"index02","\_type":"article","\_id":"1"},{"\_index":"index02","\_type":"article","\_id":"2"}]}' <http://10.9.48.69:9200/_mget> |

#### 删除文档

为了在集群中对主从分片复制有一致性判断,version在删除时也会自增1

|  |
| --- |
| curl -XDELETE <http://10.9.48.69:9200/index01/article/1> |

### 搜索管理

#### match\_all

查询：

|  |
| --- |
| query结构  {  "query": {  "match\_all": {    }  }  } |
| curl -XGET http://10.42.175.170:9200/index01/\_search '{"query": {"match\_all": {}}}' |

返回结果：查询返回结果格式基本相同

|  |
| --- |
| {  "took": 8,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {包含的数据就是通过前查询方式计算的,读到的所有内容  "total": 1, topDocs.getTotalHits  "max\_score": 1.0,  "hits": [{数组,相当于topDocs属性scoreDocs ScoreDoc[]  "\_index": "index01",三个坐标  "\_type": "product",  "\_id": "123456",  "\_score": 1.0,匹配分数  "\_source": {source的值就是新增 -d携带的字符串  "productId": "123456",  "productName": "华为手机",  }  }]  }  } |

#### term\_query

词项查询,最基本的查询方式之一.可以使用term\_query验证一下当前数据写入索引时使用的分词器---默认Standard

|  |
| --- |
| {  "query": {  "term": {  "title": "简介"  }  }  } |
| curl -XGET http:// 10.42.175.170:9200/index01/\_search -d '{"query":{"term":{"title":"简介"}}}' |

重要问题 默认如果是standard,不满足我们定义数据分词计算需求,就要使用中文分词器,怎么指定.

#### match\_query

可以不关心分词器，只提供查询的文本，es会使用分词器对文本先解析，然后对每个词项用termquery挨个查询。

|  |
| --- |
| {  "query": {  "match": {  "title": "大数据简介"  }  }  } |
| curl -XGET http:// 10.42.175.170:9200/index01/\_search -d '{"query":{"match":{"title":"大数据简介"}}}' |

title创建多个TermQuery

首先对title进行分词，默认Standard。大数据简介-->分词器分词-->大 数 据 简 介

5个termQuery 挨个查,所有结果返回

# IK分词器和Mapping设置

### IK分词器插件

#### 插件安装

es中没有ik分词器支持,无法指定这个分词器的使用.所以先来安装插件.

所有分词器代码都会按照es能够加载的插件封装.将插件解压到对应es根目录plugins中.

ik插件包 /home/resources/elasticsearch-analysis-ik-5.5.2.zip

|  |
| --- |
| [root@10-9-48-69 resources]# unzip /home/resources/elasticsearch-analysis-ik-5.5.2.zip -d /home/software/elasticsearch-5.5.2/plugins/ |

#### 插件测试

重启es ：让es重新加载plugins下的内容

发送一个请求 使用一个已有的索引,计算分词器的效果

|  |
| --- |
| http://10.42.175.170:9200/index02/\_analyze?analyzer=ik\_max\_word&text=中华人民共和国 |
|  |
|  |

参数

analyzer：分词器名字 ik\_max\_word

text：计算测试的文本

### Mapping映射

当es安装完了ik分词器之后,并不会在创建索引document数据时自动使用ik分词器。

创建数据之前配置一个叫做mapping--映射，决定数据document结构（可以理解位数据库的表结构）。

类似我们之前用lucene创建时,指定域属性。

#### Mapping是什么

是每个索引中的一个属性值(索引管理中查询索引时可以看到)，定义了当前索引中某个类型的具体结构。

联想数据库的表格结构(schema),不同的是：table的结构是在创建表格之前就定义好的。mapping有2种类型.

动态mapping

静态mapping

#### 动态Mapping

对于一个索引中的某个类型中的某一批document,可以通过在添加数据时自动生成mapping的映射关系（域属性类型，计算分词分词器等）

新增索引时不会创建mapping，第一次添加document时会生成

比如:

新增数据时,{"name":"wanglaoshi"},对应动态mapping把数据按照字符串自动生成

下面是查询索引返回的一部分数据：

|  |
| --- |
| curl -XGET http://10.42.175.170:9200/index01 |
| "mappings": {  "product": {  "properties": {  "name": {  "type": "text",  "fields": {  "keyword": {  "type": "keyword",  "ignore\_above": 256  }  }  }  }  }  } |

json是字符串时,默认type:text底层就是lucene的TextField

json是个数字数据,默认type:long 底层lucene创建document使用 LongPoint+StringField

json是个时间字符串, 默认type:date 底层都是lucene的StringField

**type:** 生成type类型值 "text" 表示这个name域属性是个字符串

**fields**:是额外的指定

type:"keyword" 当前有fields属性的name除了text特性

还有keyword 底层就是StringField以完整的内容保存name字符串

ignore\_above:256 但是当name字符串值长度超过256字节时,不会在保存完整字符串.

**总结动态mapping**:

mapping索引中定义类型里document结构的一个属性

动态mapping:什么时候写数据,就会在什么时候自动生成对应的动态结构

字符串: type:text 表示底层会按照分词计算,还有附加属性type:keyword 并且在不大于256字节长度时保存完整字段值

整数数字类型 type:long

浮点数:type:float

日期类型:type:date

地理位置类型….

动态mapping中默认指定了一个属性，标准分词器分词

analyzer:standard

如果动态mapping满足所有数据的创建需求,则可以继续使用,不满足可以换成静态mapping(自定义数据类型结构)

#### 静态Mapping

手动定义mapping：创建索引时,人为定义mapping结构。这样就不会再生成动态的mapping。.

新建一个index06 将已经编辑好的mapping 放进去作为请求体数据.

analyzer：指定分词器

|  |
| --- |
| {  "mappings": {  "book": {  "properties": {  "content": {  "type": "text",  "analyzer": "ik\_max\_word"  },  "id": {  "type": "integer"  },  "title": {  "type": "text",  "analyzer": "ik\_max\_word"  }  }  }  }  } |
| curl -XPUT -d '{"mappings":{"book":{"properties":{"content":{"type":"text","analyzer":"ik\_max\_word"},"id":{"type":"integer"},"title":{"type":"text","analyzer":"ik\_max\_word"}}}}}' [http:// 10.42.175.170:9200/index06](http://10.9.48.69:9200/index06) |

验证mapping有没有设置成功

* 写入一个document 携带content id title
* termQuery查询 title:简介 查询有结果说明ik分词器生效 mapping生效的.

|  |
| --- |
| curl -XGET <http://10.9.48.69:9200/index06/_search> -d '{"query":{"term":{"title":"简介"}}}' |

# ES集群

### es集群特点

es天生支持集群的配置,即使我们只启动一个es节点,也是以集群的逻辑运行的.相对于之前的redis mycat集群搭建简单多了。

### es集群结构和角色

#### 主节点（master）

功能:是集群的一个大脑。主要功能是计算管理一批集群元数据 。

**元数据：**

meta-data,描述数据的数据

master中对es集群管理的元数据包括：分片的个数，分片的名字，分片的存储位置,从分片个数，从分片的存储位，,document文档元数据(index,type,id,version) 包括集群状态等。

|  |
| --- |
| 计算机生成了可选文字: Master  Master  Master |

Master节点需要在集群中指定多个,同一时间只能选举其中一个作为leader，其他master节点同步元数据,一旦leader宕机完蛋,剩余master选举顶替—保证元数据高可用

元数据生成:

集群中的基础数据（状态,节点信息 ）在创建集群时,元数据文件就生成了

写入索引:master节点leader开始计算(分片个数,从分片,分布式计算等等)

#### 数据节点（data）

* 功能:读写索引文件

当master计算完元数据库,更新完元数据,开始对数据节点进行数据读写功能;

|  |
| --- |
| 计算机生成了可选文字: index01  Master  dataNode  Master  Master  乙 曰 囗  旧 囗 囗 |

#### 负载均衡节点

功能:同步元数据,给外界链接使用的节点

|  |
| --- |
| 计算机生成了可选文字: index01  Master  dataNode  Master  Master  乙 曰 囗 |

#### 协调器

* 功能:链接集群的枢纽

协调器可以通过链接一个集群的所有节点,将集群链接成一整个服务.

进群工作中,所有角色节点都要链接协调器才能完成集群的后续逻辑

|  |
| --- |
| 计算机生成了可选文字: index01  Master  dataNode  Master  Master  乙 曰 曰  曰 曰 |

### 启动创建集群

在单个节点配置基础之上完善集群的创建

* 准备3个节点每个主机运行一个es进程
  + 删除es根目录的data文件夹.不删除可能会有数据冲突
  + 保证3个节点的plugins环境是一致（例如IK分词器插件）
* 配置文件修改 elasticsearch.yml
  + 确保同一个集群加到一起 即每个cluster.name值一致
  + network.host（es被外部访问的地址）根据主机定义
  + 协调器的list:在3个es中选择至少1个作为协调器运行,保证协调器高可用则至少2个

discovery.zen.ping.unicast.hosts: ["es1的地址","es2的地址"]

* 防止脑裂发生的配置（73行） 最小有效master个数 2(3个master过半)

discovery.zen.minimum\_master\_nodes: 2

* 挨个启动每一个es节点
* 效果查看：三个主机，新增三个索引index01/2/3，都进行了数据分片
  + - 五角星的位leader

|  |
| --- |
|  |

# ES集群原理—脑裂

在es集群中,由leader管理元数据,并且其他master同步备份的元数据，这种结构容易出现脑裂

### 什么是脑裂

es脑裂是集群执行选举时发生的，即一个集群出现多个leader。问题严重在于多个leader保存多个元数据且不一致。其它master只会从其中一个leader同步元数据。.

### 脑裂的原因

es的master的leader是通过选举产生的，一旦发现现役leader宕机,则从其他master里选出新的leader.有一种情况会在现役master没有宕机时,就从其他master选择新的leader,例如：网络波动（误以为宕机）

三台master设备ABC在互相通信的时候可能会出现网络波动导致分区为A|BC，即会被误判断为A down机，A也认为BC连不上了。这个时候A会自己对自己进行选举，BC也会进行选举，如果没有设置脑裂值为大于等于集群节点的半数，可能会出现一个集群有两个master作为leader出现，又由于master之间只会从一个leader的元数据进行数据同步保证数据的一致性，现在有两个leader会出现两个leader各自保存的数据不一致最后整个集群的数据将会混乱

### 脑裂的防止

配置文件中 配置了一个最小有效master个数:集群中所有master节点个数过半数.在不满足最小master个数的选举中,leader是无效的.当这个值生效时.

集群无论如何选举,总会至多一个有效master存在

例如：在脑裂的参数原因的例子中，如果设置了有效master为过半数2，当A被误以为BC宕机，此时A不会再自我选举，因为只有1个节点不满足过半数2。

# ES集群原理—选举机制

链接协调器执行选举逻辑,分为4步.

|  |  |
| --- | --- |
| 第一步: | 启动一个master之后立刻链接协调器，获取集群的所有master节点信息，信息中activeMaster值保存着当前集群的leader节点，进入第二步 |
| 第二步: | 判断activeMaster是否为空。若集群没有选举leader则activeMaster为空。若集群已经选举则activeMaster不为空。为空进入第三步，不为空当前节点选举结束(所有节点都会结束在这一步) |
| 第三步: | 将当前集群中所有master节点信息,放到一个后备master集合中（candidateList），进入第四步 |
| 第四步: | 判断candidateList中的size是否大于等于最小master数量（防止脑裂的哪个设置？）.如果满足将会把candidateList中的节点进行对比，id值大的会暂定为activeMaster 重新回到第一步.  如果不满足，直接重新回到第一步 |

如果集群master节点宕机

master会不断链接协调器.

leader宕机:

协调器发现,立刻将activeMaster清空,剩余2个节点发现activeMaster为空,重新执行选举逻辑,直到从剩余两个中选择一个

备用master宕机:

协调器发现,其他master就发现了.

宕机一个,candidateList少了一个,剩余2个,满足最小master数量

宕机2个,candidateList少了2个,剩余1个,不满足最小master.

# JAVA客户端TransportClient

Transportclient对象,底层需要java的Socket支持,创建连接时需要通过Socket代码获取ip地址和端口.

|  |
| --- |
| **public void** init***()* throws** UnknownHostException ***{*** *//给client赋值  //传递一个setting值，定义连接的集群名称elasticsearch  //Settings settings = Settings.builder().put("cluster.name", "elasticsearch").build();  //默认集群名称就是elasticsearch，锁不用手动设置了* **transportClient**=**new** PreBuiltTransportClient***(***Settings.***EMPTY)***;  *//创建一个节点的ip 端口*   InetSocketTransportAddress address=  **new** InetSocketTransportAddress***(***InetAddress.*getByName****(*"10.42.175.170"*)***,9300***)***;  **transportClient**.addTransportAddress***(***address***)***; ***}*** |

### 引入依赖

|  |
| --- |
| *<!--#####搜索 elasticsearch#############-->  <!--elasticsearch-->* ***<*dependency*>  <*groupId*>***org.elasticsearch.client***</*groupId*>  <*artifactId*>***transport***</*artifactId*>  <*version*>***5.5.2***</*version*>  <*exclusions*>  <*exclusion*>*** *<!--不使用默认的2.4.6版本-->* ***<*groupId*>***org.elasticsearch***</*groupId*>  <*artifactId*>***elasticsearch***</*artifactId*>  </*exclusion*>  </*exclusions*>  </*dependency*>*** *<!--使用5.5.2版本-->* ***<*dependency*>  <*groupId*>***org.elasticsearch***</*groupId*>  <*artifactId*>***elasticsearch***</*artifactId*>  <*version*>***5.5.2***</*version*>  </*dependency*>*** |

### 索引管理

|  |
| --- |
| public void indexManage() throws IOException {   //创建索引 http://10.9.151.60:9200/index01 put  //获取admin 可以管理集群cluster也可以管理indices  *AdminClient* admin = transportClient.admin();  //1、获取操作index的管理对象  *IndicesAdminClient* indices = admin.indices();  *ClusterAdminClient* cluster = admin.cluster();  //########建一个索引 index07#################  //indices.create();//方法调用,命令就发送  //2、获取request对象没有发送命令   CreateIndexRequestBuilder createIndexRequest =indices.prepareCreate("index07");  //prepare\*\*方法调用没法送命令,但是获取了返回值request对象  //想把代码命令发送出去，则需要用get()  //3、发送请求  CreateIndexResponse createIndexResponse = createIndexRequest.get();  //response解析就是创建索引的返回值 {"acknowlaged","shards\_acknowlaged"}  //4、操作返回值  createIndexResponse.isShardsAcked();  createIndexResponse.isAcknowledged();   //其他的reques对象，可以做对应的相关操作  GetAliasesRequestBuilder GetAliasesRequest = indices.prepareGetAliases("index07");  GetMappingsRequestBuilder getMappingsRequest = indices.prepareGetMappings("index02");  IndicesExistsRequestBuilder indicesExistesRequest = indices.prepareExists("index07");  GetMappingsResponse getMappingsResponse = getMappingsRequest.get();  System.*out*.println("index07存在吗:"+indicesExistesRequest.get().isExists());  //常用的方法,创建索引,不存在时创建索引  //prepareCreate prepareExists prepareDelete; } |

### 文档管理

|  |
| --- |
| public void docManage() throws JsonProcessingException {  //1、随便构造一点数据存到es中  Product product=new Product();  product.setProductCategory("手机");  product.setProductId("好用");  product.setProductId("123456");  product.setProductImgurl("http://image.jt.com");  product.setProductName("华为手机");  product.setProductNum(50);  product.setProductPrice(50.0);  //http://10.9.151.60:9200/index/article/1 -d  //2、将对象转化为json  ObjectMapper mapper=new ObjectMapper();  String json = mapper.writeValueAsString(product);  //3、通过client的prepareIndex方法获取request  //传递坐标的三个参数 索引名，类型，文档id  IndexRequestBuilder request = transportClient.prepareIndex("index07", "product", product.getProductId());  //4、设置request请求数据 json字符串  request.setSource(json);  //5、发送请求  IndexResponse indexResponse = request.get();  //6、处理返回值  //新建document返回字符串json \_index \_type \_id \_version  long version = indexResponse.getVersion();  System.*out*.println("创建version:"+version);  //删除document  transportClient.prepareDelete("index07","product","123456");  //获取  transportClient.prepareGet("index07","product","123456"); } |

### 搜索功能

|  |
| --- |
| public void search1() {   //通过query对象的封装 实现不同搜索逻辑  //搜索的过程的分页时浅查询  TermQueryBuilder query = QueryBuilders.*termQuery*("title", "介");  //搜索  SearchRequestBuilder request = transportClient.prepareSearch("index02");  //搜索条件  request.setQuery(query);  //搜索分页数据,其实位置,条数  request.setFrom(0);//limit start  request.setSize(5);  SearchResponse searchResponse = request.get();  //searchResponse解析查询结果  SearchHits topDocs = searchResponse.getHits();  System.*out*.println("查询总数:" + topDocs.totalHits);  //遍历数组  SearchHit[] scoreDocs = topDocs.getHits();  for (SearchHit hit : scoreDocs) {  //获取一个doument返回结果  //{"id":"","title"}  System.*out*.println("source的json:" + hit.getSourceAsString());  } } |

# 集成SpringbOOT

### 3.1springboot自动配置

redis：RedisAutoConfiguration

Elasticsearch: ElasticsearchAutoConfiguration

1.5.9.RELEASE-->2.4.6版本的TransportClient

### 手动配置

原理:让框架来管理连接对象.spring的配置类

让框架管理一个连接es集群的Transportclient对象

声明一个配置类，spring加载读取配置文件并创建bean

|  |
| --- |
| */\*\*  \* @作者 舒新胜  \* @项目 easymall-2002-all  \* @创建时间 2020/6/9 14:09  \*/* @Configuration //配合@Configuration实现配置类属性初始化赋值 //和@Value作用一样可以读取properties //自定义前缀，多级赋值 @ConfigurationProperties("easymall.es") public class ESConfig {   public *List*<String> getNodes() {  return nodes;  }   public void setNodes(*List*<String> nodes) {  this.nodes = nodes;  }   //如果有多个参数，赋值时逗号分隔：easymall.es.nodes=SHU,XIN  //然后会把分割过后的值交给这个list  //###ESeasymall.es.nodes=10.42.175.170:9300  private *List*<String> nodes;  @Bean  public TransportClient initClient() {  System.*out*.println("开始创建ES连接客户端");  System.*out*.println(Collections.*singletonList*(nodes));  PreBuiltTransportClient transportClient = new PreBuiltTransportClient(Settings.*EMPTY*);  for (String node : nodes) {  String[] split = node.split(":");  //ip  String host=split[0];  //端口  int port=Integer.*parseInt*(split[1]);  //封装地址  InetSocketTransportAddress address = null;  try {  address = new InetSocketTransportAddress(InetAddress.*getByName*(host), port);  } catch (UnknownHostException e) {  e.printStackTrace();  continue;  }   transportClient.addTransportAddress(address);  }  return transportClient;   } } |

@Configuration：标识为配置类  
//配合@Configuration实现配置类属性初始化赋值  
//和@Value作用一样可以读取properties  
//自定义前缀，多级赋值

//加载配置类时，会读取配置文件的easymall.es.\*\*的值赋值给\*\*变量

//如果变量是另一个配置类，则可以在配置文件中通过easymall.es.另一个配置类.\*\*给这个配置类中的属性赋值。  
@ConfigurationProperties("easymall.es")

@Bean：该注解指定的方法返回值会交给spring容器

配置文件设置值，spring自动赋值

|  |
| --- |
| ###ES easymall.es.nodes=10.42.175.170:9300,10.9.151.60:9300 |

当使用配置逻辑完成对象的框架管理过程,开发者使用的代码还是最底层客户端代码.一般作为架构师都会对其做2次封装,让开发过程变得简单

|  |
| --- |
| @Component public class ESUtils {  @Autowired  private TransportClient client;  //一个叫做创建索引,  public boolean createIndexByName(String indexName){  return client.admin().indices().prepareCreate(indexName).get().isAcknowledged();  }  //一个叫做判断索引存在 } |

# ELK家族

### ELK介绍

数据采集和整理过程在上述的结构中,没有实现定时执行(手动访问的)

数据源不同,java代码就要编写很多个.

es中的数据--索引文件,价值不是体现太好.非常庞大的索引文件,数据量大到一定程度,但是只能用来搜索.

E:elasticsearch,作用就是管理一堆的索引文件

L:logstash,日志数据采集器.作用就是对接数据源(对接很多种数据源),将数据源源源不断的输出到es中

Kibana:数据分析插件,作用:链接es实现数据的分析.

### 数据采集整理一致性

如果ELK使用不用业务层面考虑一致性,logstash很容易解决,定时执行任务就行;

在数据源发生变动时,一并调用处理缓存和索引的逻辑

# CAP理论

是分布式基础理论.设计一些技术选型的逻辑

C:Consistency 一致性

A:avalibility 可用性

P:partition tolerency/partition 分区容忍度和分区

结论:分布式系统中CAP三个方面无法同时满足,只能满足最多2个 CA CP AP

P:分区,在分布式环境中,由于网络限制,导致系统之间各个部分相互不可连接,形成分区.分区是一种常态

C:对数据的一致性要求高---强一致性

A:对系统的可用性要求高.

当分区出现时P,对数据一致性要求高 ---CP理论

舍弃可用性,对系统加锁,加事务,保证数据修改是一致性

当分区出现时P,对系统可用性要求高 ---AP

数据一致性要求不高,对系统可用性没有影响

分区不存在网络连接畅通可以AC

什么时候要求可用性:

微服务调用--可用性

eureka中的注册信息是一致性体现吗?当一个服务关闭后,保持数据一致性,在eureka注册中西立刻剔除

互联网项目系统的普通功能

搜索,查询,登录

什么时候要求一致性:

支付:

客户端数据/服务器钱---第三方支付平台/银行