

# ElasticSearch精讲

日期	修订版本	修改章节	修改描述	作者
2019-04-10	V1.0			Janson



## 第一章：ES简介

### 1.1、ES简介

#### 1.1.1 ES定义

ElasticSearch是一款基于Apache Lucene构建的开源搜索引擎，它采用Java编写并使用Lucene构建索引、提供搜索功能，ElasticSearch的目标是让全文搜索变得简单，开发者可以通过它简单明了的RestFul API轻松地实现搜索功能，而不必去面对Lucene的复杂性。ES能够轻松的进行大规模的横向扩展，以支撑PB级的结构化和非结构化海量数据的处理。

一言以蔽之：ElasticSearch是一款基于Lucene的实时分布式搜索和分析引擎。

ElasticSearch设计主要用于云计算中，能够达到实时搜索、稳定、可靠、快速，安装使用也非常方便。

官网：[www.elastic.co](http://www.elastic.co)

#### 1.1.2 ES工作原理



### 1.1.3 ES轶事

Shay Banon认为自己参与Lucene完全是一种偶然，当年他还是一个待业工程师，跟随自己的新婚妻子来到伦敦，妻子想在伦敦学习做一名厨师，而自己则想为妻子开发一个方便搜索菜谱的应用，所以才接触到Lucene。直接使用Lucene构建搜索有很多问题，包含大量重复性的工作，所以Shay便在Lucene的基础上不断地进行抽象，让Java程序嵌入搜索变得更容易，经过一段时间的打磨便诞生了他的第一个开源作品“Compass”，中文即“指南针”的意思。之后，Shay找到了一份面对高性能分布式开发环境的新工作，在工作中他渐渐发现越来越需要一个易用的、高性能、实时、分布式搜索服务，于是他决定重写Compass，将它从一个库打造成了一个独立的server，并将其改名为Elasticsearch。

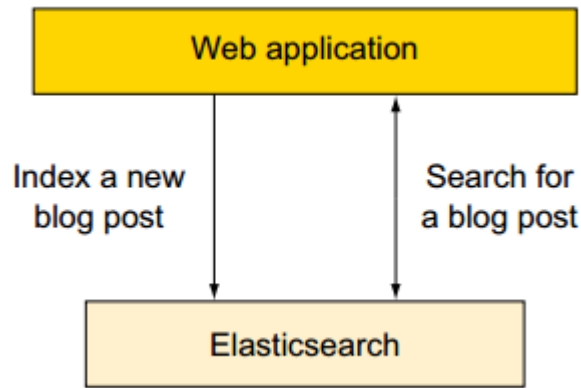
### 1.1.4 ES的适用场景概述

1. 站内搜索：主要和 Solr 竞争，属于后起之秀。
2. NoSQL Json文档数据库：主要抢占 Mongo 的市场，它在读写性能上优于 Mongo，同时也支持地理位置查询，还方便地理位置和文本混合查询。
3. 监控：统计、日志类时间序的数据存储和分析、可视化，这方面是引领者。
4. 国外：Wikipedia（维基百科）使用ES提供全文搜索并高亮关键字、StackOverflow（IT问答网站）结合全文搜索与地理位置查询、Github使用Elasticsearch检索1300亿行的代码。
5. 国内：百度（在云分析、网盟、预测、文库、钱包、风控等业务上都应用了ES，单集群每天导入30TB+数据，总共每天60TB+）、新浪、阿里巴巴、腾讯等公司均有对ES的使用。
6. 使用比较广泛的平台ELK(ElasticSearch, Logstash, Kibana)。

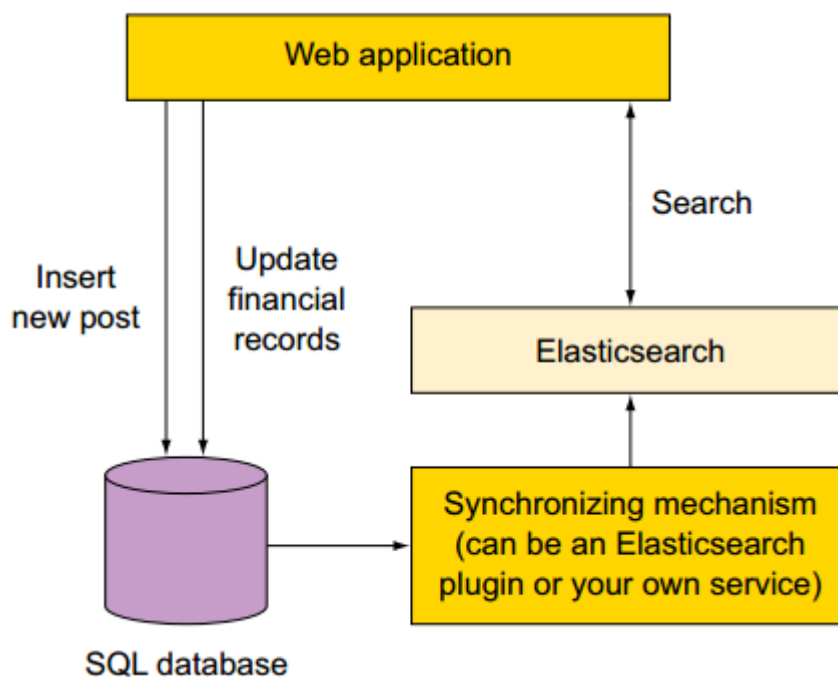
### 1.1.5 ES适用场景详解

了解了ES的使用场景，ES的研究、使用、推广才更有价值和意义。

**场景一：**使用Elasticsearch作为主要的后端 传统项目中，搜索引擎是部署在成熟的数据存储的顶部，以提供快速且相关的搜索能力。这是因为早期的搜索引擎不能提供耐用的存储或其他经常需要的功能，如统计。



**场景二：**在现有系统中增加elasticsearch 由于ES不能提供存储的所有功能，一些场景下需要在现有系统数据存储的基础上新增ES支持。



举例1：ES不支持事务、复杂的关系（至少1.X版本不支持，2.X有改善，但支持的仍然不好），如果你的系统中需要上述特征的支持，需要考虑在原有架构、原有存储的基础上的新增ES的支持。

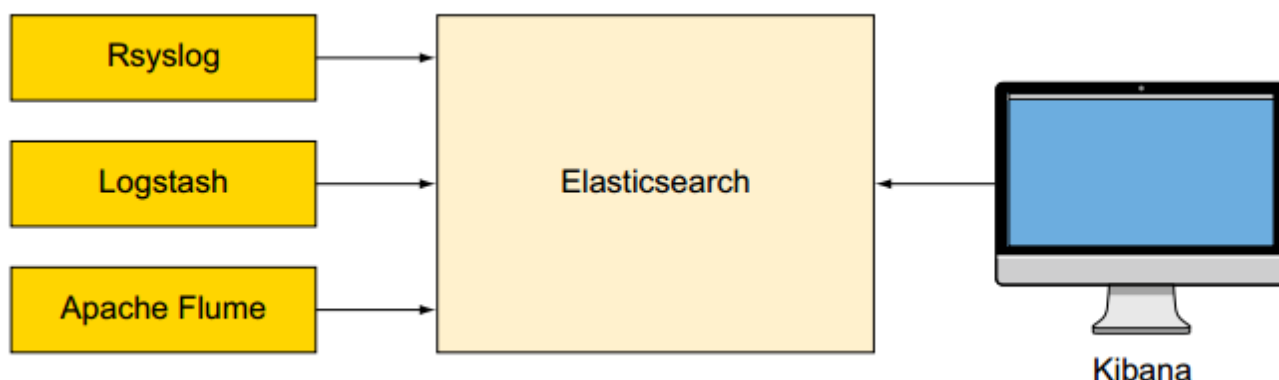
举例2：如果你已经有一个在运行的复杂的系统，你的需求之一是在现有系统中添加检索服务。一种非常冒险的方式是重构系统以支持ES。而相对安全的方式是：将ES作为新的组件添加到现有系统中。如果你使用了如下图所示的SQL数据库和ES存储，你需要找到一种方式使得两种存储之间实时同步。需要根据数据的组成、数据库选择对应的同步插件。可供选择的插件包括：1 ) mysql、oracle选择 logstash-input-jdbc 插件。2 ) mongo选择 mongo-connector工具。

假设你的在线零售商店的产品信息存储在SQL数据库中。为了快速且相关的搜索，你安装Elasticsearch。为了索引数据，您需要部署一个同步机制，该同步机制可以是Elasticsearch插件或你建立一个自定义的服务。此同步机制可以将对应于每个产品的所有数据和索引都存储在Elasticsearch，每个产品作为一个document存储（这里的document相当于关系型数据库中的一行/row数据）。

当在该网页上的搜索条件中输入“用户的类型”，店面网络应用程序通过Elasticsearch查询该信息。Elasticsearch返回符合标准的产品documents，并根据你喜欢的方式来分类文档。排序可以根据每个产品的被搜索次数所得到的相关分数，或任何存储在产品document信息，例如：最新最近加入的产品、平均得分，或者是那些插入或更新信息。所以你可以只使用Elasticsearch处理搜索。这取决于同步机制来保持Elasticsearch获取最新变化。

### 场景三：使用elasticsearch和现有的工具

在一些使用情况下，您不必写一行代码就能通过elasticsearch完成一项工作。很多工具都可以与Elasticsearch一起工作，所以你不必到你从头开始编写。例如，假设要部署一个大规模的日志框架存储，搜索，并分析了大量的事件。如图下图，处理日志和输出到Elasticsearch，您可以使用日志记录工具，如rsyslog ([www.rsyslog.com](http://www.rsyslog.com))，Logstash ([www.elastic.co/products/logstash](http://www.elastic.co/products/logstash))，或Apache Flume (<http://flume.apache.org>)。搜索和可视化界面分析这些日志，你可以使用Kibana ([www.elastic.co/产品/kibana](http://www.elastic.co/产品/kibana))。



#### 1.1.6 众多工具适配ES

为什么那么多工具适配Elasticsearch？主要原因如下：1) Elasticsearch是开源的。2) Elasticsearch提供了JAVA API接口。3) Elasticsearch提供了RESTful API接口（不管程序用什么语言开发，任何程序都可以访问）4) 更重要的是，REST请求和应答是典型的JSON（JavaScript对象符号）格式。通常情况下，一个REST请求包含一个JSON文件，其回复也都是一个JSON文件。

## 1.2、ES和Solr对比

### 1.2.1 概述

关于ES：Elasticsearch是一个实时分布式搜索和分析引擎。使用其可以以前所未有的速度处理大数据。它用于全文搜索、结构化搜索、分析以及将这三者混合使用。维基百科使用Elasticsearch提供全文搜索并高亮关键字，以及输入实时搜索(search-as-you-type)和搜索纠错(did-you-mean)等搜索建议功能。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。

关于Solr：Solr 是Apache下的一个顶级开源项目，采用Java开发，它是基于Lucene的全文搜索服务器。Solr提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展，并对索引、搜索性能进行了优化。Solr可以独立运行，运行在Jetty、Tomcat等这些Servlet容器中，Solr 索引的实现方法很简单，用 POST 方法向 Solr 服务器发送一个描述 Field 及其内容的 XML 文档，Solr根据xml文档添加、删除、更新索引。

### 1.2.2 对比说明

使用

Solr安装略微复杂一些；es基本是开箱即用，非常简单

接口 Solr类似webservice的接口；es REST风格的访问接口

分布式存储 solrCloud solr4.x才支持 es是为分布式而生的

支持的格式 Solr 支持更多格式的数据，比如JSON、XML、CSV；es仅支持json文件格式

近实时搜索

solr 查询快，但更新索引时慢（即插入删除慢），用于电商等查询多的应用；ES建立索引快（即查询慢），即实时性查询快，用于facebook新浪等搜索。

solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用。



### 1.3、ES和MySQL对比

MySQL	ElasticSearch
database(数据库)	index(索引库)
table(表)	type(类型)
row(行)	document(文档)
column(列)	field(字段)

## 第二章：REST

### 2.1、REST简介

REST全称Representational State Transfer。是一种软件的架构风格，而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务端交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

其实说白了就是类似HTTP的访问，和HTTP非常的相似。 REST操作： GET：获取对象的当前状态； PUT：改变对象的状态； POST：创建对象； DELETE：删除对象； HEAD：获取头信息。

### 2.2、Rest具体操作说明

资源	一组资源的URI，比如： <a href="http://example.com/res/">http://example.com/res/</a>	单个资源的URI，比如： <a href="http://example.com/res/123">http://example.com/res/123</a>
GET	列出URI，以及该资源组中每个资源的详细信息(后者可选)	获取指定的资源的详细信息，格式可以自选一个合适的网络媒体类型(比如：XML、JSON等)
PUT	使用给定的一组资源替换当前整组资源	替换/创建指定的资源。并将其追加到相应的资源组中。
POST	在本组资源中创建/追加一个新的资源。该操作往往返回新的URL	把指定的资源当做一个资源组，并在其下创建/追加一个新的元素，使其隶属于当前资源。
DELETE	删除整组资源	删除指定的元素

### 2.3、ES内置REST接口

URL	描述
<code>/index/_search</code>	搜索指定索引下的数据
<code>/_aliases</code>	获取或操作索引的别名
<code>/index/</code>	查看指定索引的详细信息
<code>/index/type/</code>	创建或操作类型
<code>/index/_mapping</code>	创建或操作mapping
<code>/index/_setting</code>	创建或操作设置(number_of_shards是不可更改的)
<code>/index/_open</code>	打开指定被关闭的索引
<code>/index/_close</code>	关闭指定索引
<code>/index/_refresh</code>	刷新索引(使新加内容对搜索可见，不保证数据被写入磁盘)
<code>/index/flush</code>	刷新索引(会触发Lucene提交)

## 第三章：ES安装

### 3.1、ES的安装配置介绍

实际上ES的安装配置是非常简单的，没有繁琐的安装配置，可以称之为零配置，开箱即用。说明一点：ES新版本的操作必须要在普通用户下面进行操作 ES安装配置

下载地址 <https://www.elastic.co/downloads/past-releases/elasticsearch-6-5-3> 或者再github官网elastic项目下载都可以下载到各个版本的es <https://github.com/elastic/elasticsearch>

安装要求 JDK版本最低1.7

安装 同一个安装包既可以在windows下使用，也可以在linux下使用，我们这里就在linux下来操作。

方式1：ES的安装之默认配置单机版

步骤：

将安装包上传到Linux下，解压，在普通用户下运行elasticSearch/bin/elasticsearch 文件

注意点：

①必须是普通用户，不能是root用户（否则，报错：java.lang.RuntimeException: can not run elasticsearch as root）

②elasticSearch/bin/elasticsearch -d ~>以后台进程的方式启动es，通过jps命令，可以察觉到进程名为：Elasticsearch

③Linux命令：

useradd 新用户名 ~>新建用户

passwd 用户名 ~>设置密码

su -l 用户名 ~>用户切换

④curl：linux命令，可以模拟browser向远程的服务器发送请求，并获得反馈。

（curl:linux os中的一个命令，可以使用命令行的方式模拟browser向远程的server发送请求，并获得远程server的反馈

ip： 联网的终端设别在网络上的唯一标识

端口号：联网的终端设备上安装的具有访问网络功能的应用程序的唯一标识。 )

语法：curl -XGET 'http://127.0.0.1:9200'

## 方式2：ES的安装之手动定制单机版

配置config/elasticsearch.yml cluster.name: bigdata ~>集群名 node.name: hadoop ~>集群中当前es服务器节点名  
path.data: /home/tom/data/elastic ~> es索引库中的数据最终存储到哪个目录下，目录会自动创建 path.logs:  
/home/tom/logs/elastic ~> es进程启动后，对应的日志信息存放的目录，目录会自动创建 network.host:  
JANSON01 ~> 当前虚拟机的ip地址的别名 http.cors.enabled: true ~> 下面两个配置参数指的是es服务器允许别的插件服务访问 (插件: 对现有软件功能的一个扩展的软件) http.cors.allow-origin: "\*"

启动：（daemon: 精灵进程，后台进程的方式启动；索引库启动需要花费几秒钟的时间，等待!）

\$ELASTICSEARCH\_HOME/bin/elasticsearch -d

注意：1，若是进程启动不了，查看日志文件/home/tom/logs/elastic/bigdata.log，报错：max file descriptors [4096] for elasticsearch process is too low, increase to at least [65536]，解决方案见：Elasticsearch\1\_资料\⑥异常\I -es安装异常.txt

## 2, yml, properties:

同：都是用来操作资源文件的。

不同点：

①properties资源文件中，键与值之间使用=进行分隔（等于号）

yml资源文件中，键与值之间使用:进行分隔（冒号后面必须得添加一个半角空格）

②较之于properties资源文件中，yml资源文件书写起来更加简洁一些，通过缩进来标识层次关系。

~>properties:

db.mysql.url=txxxx

db.mysql.pwd=txxxx

~>yml:（更加精简，相同的目录只需要书写一次即可）

db.mysql.url: txxxx ~> 键和值之间使用:隔开，且至少必须有一个半角空格

pwd: txxxx ~> 通过缩进来标识层次关系

也就是：

\*.yml，与\*.properties 都属于资源文件，较之于properties文件，yml文件书写得更为简洁，文件内容中：键：值，多个键前缀相同，通过缩进来标识层次关系

验证 访问es的安装服务器，http://<es\_ip>:9200





← → ↻ ⓘ 不安全 | janson01:9200

应用 百度一下

```
{
  "name" : "hadoop",
  "cluster_name" : "bigdata",
  "cluster_uuid" : "Pm19-oHoRHqZcreNRCQ8Cw",
  "version" : {
    "number" : "6.5.3",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "159a78a",
    "build_date" : "2018-12-06T20:11:28.826501Z",
    "build_snapshot" : false,
    "lucene_version" : "7.5.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

### 3.2、ES配置文件说明

logging.yml 日志配置文件，es也是使用log4j来记录日志的，所以logging.yml里的设置按普通log4j配置来设置就行了。elasticsearch.yml es的基本配置文件,需要注意的是key和value的格式":"之后需要一个空格。修改如下配置之后，就可以从别的机器上进行访问了

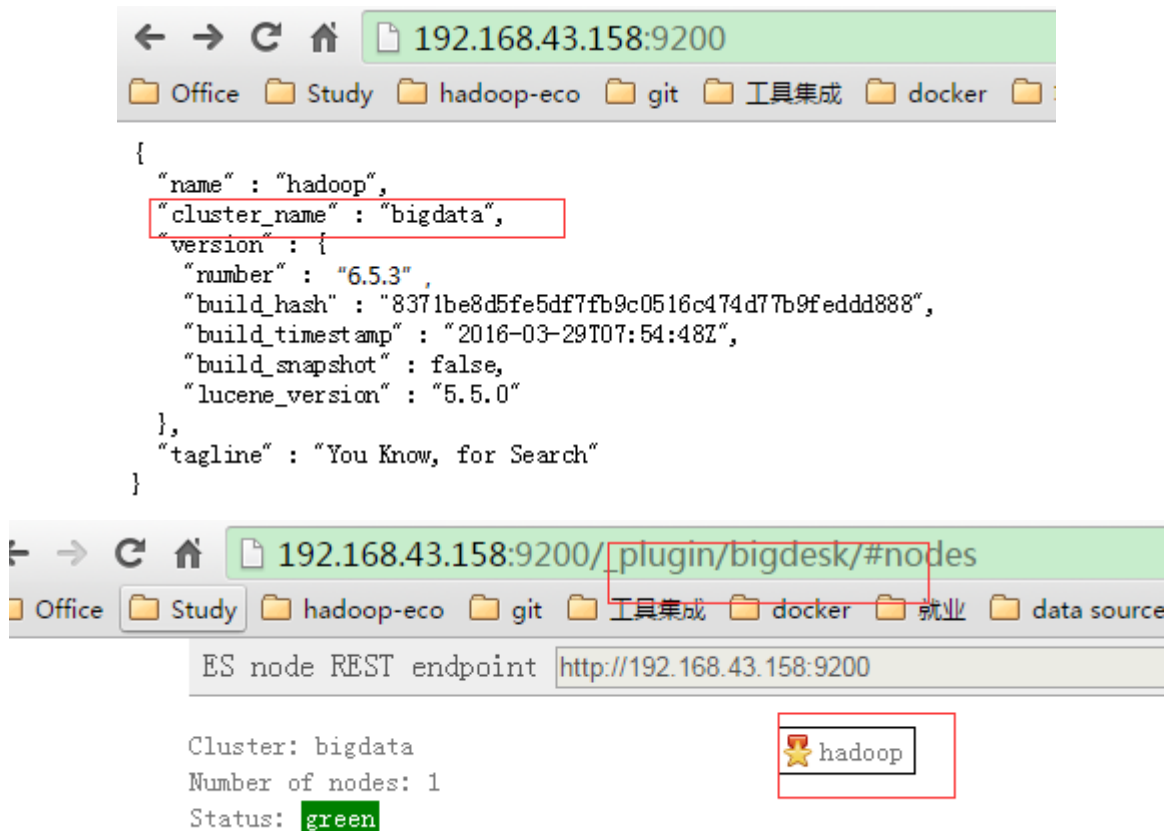
修改node.name

Transport.tcp.port: 9300 设置节点间交互的tcp端口，默认为9300

在Elasticsearch集群中可以监控统计很多信息，但是只有一个是重要的集群健康(cluster health)。Es中用三种颜色状态表示:green, yellow, red. Green: 所有主分片和副本分片都可用 Yellow: 所有主分片可用，但不是所有副本分片都可用 Red: 不是所有的主分片都可用

```
# network.host: 192.168.0.1
network.host: 0.0.0.0
#
```





## 第四章：CURL

### 4.1、CURL简介

curl是利用URL语法在命令行方式下工作的开源文件传输工具，使用curl可以简单实现常见的get/post请求。简单的认为是在命令行下面访问url的一个工具。在centos的默认库里面是有curl工具的，如果没有请yum安装即可。

curl -X 指定http的请求方法 有HEAD GET POST PUT DELETE -d 指定要传输的数据 -H 指定http请求头信息

### 4.2、使用CURL创建索引库、创建索引

curl创建索引库 curl -XPUT http://:9200/index\_name/ PUT或POST都可以创建 举例： curl -XPUT 'http://localhost:9200/bigdata'

创建索引 curl -H 'Content-Type:application/json' -XPOST 'http://localhost:9200/bigdata/product/1' -d '{"name": "hadoop", "author": "Doug Cutting", "core": ["hdfs", "mr", "yarn"], "latest\_version": 3.0}'

```
[bigdata@service ~]$ curl -XPOST 'http://localhost:9200/bigdata/product/1' -d '{"name": "hadoop", "author": "Doug Cutting", "core": ["hdfs", "mr", "yarn"], "latest_version": 3.0}'
{"_index": "bigdata", "_type": "product", "_id": "1", "_version": 1, "_shards": {"total": 2, "successful": 1, "failed": 0}, "created": true} [bigdata@service ~]$
```

### 4.3、使用CURL进行PUT和POST操作

PUT是幂等方法，POST不是。所以PUT用于更新，POST用于新增比较合适。PUT和DELETE操作是幂等的。所谓幂等是指不管进行多少次操作，结果都一样。比如用PUT修改一篇文章，然后在做同样的操作，每次操作后的结果并没有什么不同，DELETE也是一样。POST操作不是幂等的，比如常见的POST重复加载问题：当我们多次发出同样的POST请求后，其结果是创建了若干的资源。还有一点需要注意的就是，创建操作可以使用POST，也可以使用PUT，

区别就在于POST是作用在一个集合资源(/articles)之上的，而PUT操作是作用在一个具体资源之上的(/articles/123)，比如说很多资源使用数据库自增主键作为标识信息，这个时候就需要使用PUT了。而创建的资源的标识信息到底是什么，只能由服务端提供时，这个时候就必须使用POST。ES创建索引库和索引时的注意点 1)索引库名称必须要全部小写，不能以下划线开头，也不能包含逗号 2)如果没有明确指定索引数据的ID，那么es会自动生成一个随机的ID，需要使用POST参数 curl -XPOST <http://localhost:9200/bigdata/product/> -d '{"author": "Doug Cutting"}

```
[bigdata@service ~]$ curl -XPOST http://localhost:9200/bigdata/product/ -d '{"author": "Doug Cutting"}'
{"_index": "bigdata", "_type": "product", "_id": "AVenCVBca5ANYnG-AvJ6", "_version": 1, "_shards": {"total": 2, "successful": 1, "failed": 0}, "created": true} [bigdata@service ~]$
```

如果想要确定创建的都是全新的数据 1：使用随机ID(POST方式) 2：在url后面添加参数 curl -XPOST [http://localhost:9200/bigdata/product/2?op\\_type=create](http://localhost:9200/bigdata/product/2?op_type=create) -d '{"name": "hbase"}' curl -XPOST <http://localhost:9200/bigdata/product/3/ create> -d '{"name": "hive"}' 如果成功创建了新的文档，ES将会返回常见的元数据以及created为true的反馈。如果存在同名文件，ES将会返回AlreadyExistsException。之前的版本如果成功创建了新的文档，Elasticsearch将会返回常见的元数据以及201 Created的HTTP反馈码。而如果存在同名文件，Elasticsearch将会返回一个409 Conflict的HTTP反馈码

#### 4.4、CURL使用之查询所有

查询所有 -GET 根据产品ID查询 curl -XGET <http://localhost:9200/bigdata/product/1?pretty> 在任意的查询url中添加pretty参数，es可以获取更易识别的json结果。检索文档中的一部分，显示特定的字段内容 curl -XGET <http://localhost:9200/bigdata/product/1?source=name,author&pretty> 获取source的数据 curl -XGET 'http://localhost:9200/bigdata/product/1/source?pretty' 查询所有 curl -XGET 'http://localhost:9200/bigdata/product/ search?pretty' 根据条件进行查询 curl -XGET 'http://localhost:9200/bigdata/product/ search?q=name:hbase&pretty'

#### 4.5、CURL使用之ES更新&删除

ES更新 ES可以使用PUT或者POST对文档进行更新，如果指定ID的文档已经存在，则执行更新操作 注意：执行更新操作的时候，ES首先将旧的文档标记为删除状态，然后添加新的文档，旧的文档不会立即消失，但是你也无法访问，ES会继续添加更多数据的时候在后台清理已经标记为删除状态的文档。局部更新 可以添加新字段或者更新已经存在字段(必须使用POST) curl -XPOST <http://localhost:9200/bigdata/product/1/ update> -d '{"doc":{"name": "apache-hadoop"}}'

普通删除，根据主键删除 curl -XDELETE <http://localhost:9200/bigdata/product/3/> 说明：如果文档存在，es属性 found : true, successful:1, \_version属性的值+1。如果文档不存在，es属性found为false，但是版本号version依然会+1，这个就是内部 管理的一部分，有点像svn版本号，它保证了我们在多个节点间的不同操作的顺序被正确标记了。注意：一个文档被删除之后，不会立即生效，他只是被标记为已删除。ES将会在你之后添加 更多索引的时候才会在后台进行删除。

#### 4.6、CURL使用之ES批量操作-bulk

Bulk api可以帮助我们同时执行多个请求 格式： action:[index|create|update|delete] metadata:index,type,id request body:source(删除操作不需要) {action:{metadata}}\n {request body}\n {action:{metadata}}\n {request body}\n create和index的区别 如果数据存在，使用create操作失败，会提示文档已经存在，使用index则可以成功执行。使用文件的方式 vi requests curl -XPOST/PUT <http://localhost:9200/index/type/ bulk> --data-binary @path 比如 curl -XPOST 'http://localhost:9200/bank/accout/ bulk?pretty' --data-binary "@data/accounts.json"

accounts数据说明.txt

银行客户账号信息文档，文档schema如下 { "account\_number": 0, "balance": 16623, "firstname": "Bradshaw", "lastname": "Mckenzie", "age": 29, "gender": "F", "address": "244 Columbus Place", "employer": "Euron", "email": "[bradshawmckenzie@euron.com](mailto:bradshawmckenzie@euron.com)", "city": "Hobucken", "state": "CO" }

可以查看一下各个索引库信息 `curl 'http://localhost:9200/_cat/indices?v'`

Bulk请求可以在URL中声明 `/index或者/index/_type` Bulk一次最大处理多少数据量 Bulk会把将要处理的数据载入内存中，所以数据量是有限制的 最佳的数据量不是一个确定的数值，它取决于你的硬件，你的文档大小以及复杂性，你的索引以及搜索的负载 一般建议是1000~5000个文档，如果你的文档很大，可以适当减少队列，大小建议是5~15MB，默认不能超过100M，可以在es的配置文件中修改这个值 `http.max_content_length:100mb`

#### 4.7、CURL使用之ES版本控制

ES版本控制 普通关系型数据库使用的是（悲观并发控制（PCC））当我们在读取一个数据前先锁定这一行，然后确保只有读取到数据的这个线程可以修改 这一行数据 ES使用的是（乐观并发控制（OCC））ES不会阻止某一数据的访问，然而，如果基础数据在我们读取和写入的间隔中发生了变化，更新就会失败，这时候就由程序来决定如何处理这个冲突。它可以重新读取新数据来进行更新，又或者将这一情况直接反馈给用户。ES如何实现版本控制(使用es内部版本号) 1：首先得到需要修改的文档，获取版本(version)号 `curl -XGET http://localhost:9200/bigdata/product/1` 2：再执行更新操作的时候把版本号传过去 `curl -XPUT http://localhost:9200/bigdata/product/1?version=1 -d '{"name":"hadoop","version":3}'` (覆盖) `curl -XPOST http://localhost:9200/bigdata/product/1/update?version=3 -d '{"doc":{"name":"apache hadoop","latest_version": 2.6}}'` (部分更新) 3：如果传递的版本号和待更新的文档的版本号不一致，则会更新失败 ES如何实现版本控制(使用外部版本号) 如果你的数据库已经存在了版本号，或者是可以代表版本的时间戳。这时就可以在es的查询url后面添加 `version_type=external` 来使用这些号码。注意：版本号码必须要是大于0小于9223372036854775807（Java中long的最大正值）的整数。

es在处理外部版本号的时候，它不再检查version是否与请求中指定的数值是否相等，而是检查当前的version是否比指定的数值小，如果小，则请求成功。 example： `curl -XPUT 'http://localhost:9200/bigdata/product/20?version=10&version_type=external' -d '{"name": "flink"}'` 注意：此处url前后的引号不能省略，否则执行的时候会报错

### 第五章：ES插件

#### 5.1、ES插件概述

ES本身服务相对比较少，其功能的强大之处就体现在插件的丰富性上。有非常多的ES插件用于ES的管理，性能的完善，下面就给大家介绍几款常用的插件。

#### 5.2、BigDesk Plugin介绍

BigDesk主要提供的是节点的实时状态监控，包括jvm的情况，linux的情况，elasticsearch的情况，推荐大家使用。

#### 5.3、BigDesk Plugin安装

下载地址：<https://github.com/hlstudio/bigdesk> 在线安装：`bin/plugin install hlstudio/bigdesk` 离线安装：`bin/plugin install file:/tmp/bigdesk-master.zip`（插件比如说在/tmp目录下面） 移除：`bin/plugin remove bigdesk`

-bigdesk : 该工具的Git地址是 : <https://github.com/lukas-vlcek/bigdesk> ~> 适用对象 : es集群的运维人员。

①BigDesk主要提供的是节点的实时状态监控, 包括jvm的情况, linux的情况, elasticsearch的情况, 推荐大家使用。

②里面可以看到集群名称, 节点列表。内存消耗情况, GC回收情况。可以自由的在各个节点之间进行切换, 自动的添加或是移除一些旧的节点。同样可以更改refresh interval刷新间隔, 图标能够显示的数据量。

安装 :

①解压下载的bigdesk插件, 注意一定不要下载到elasticsearch的plugins目录下, 可以与elasticsearch的安装目录一致。(旧版本的es插件的安装目录必须是plugins)

②进入到bigdesk的\_site目录, 在Linux命令行启动 : `python -m SimpleHTTPServer`

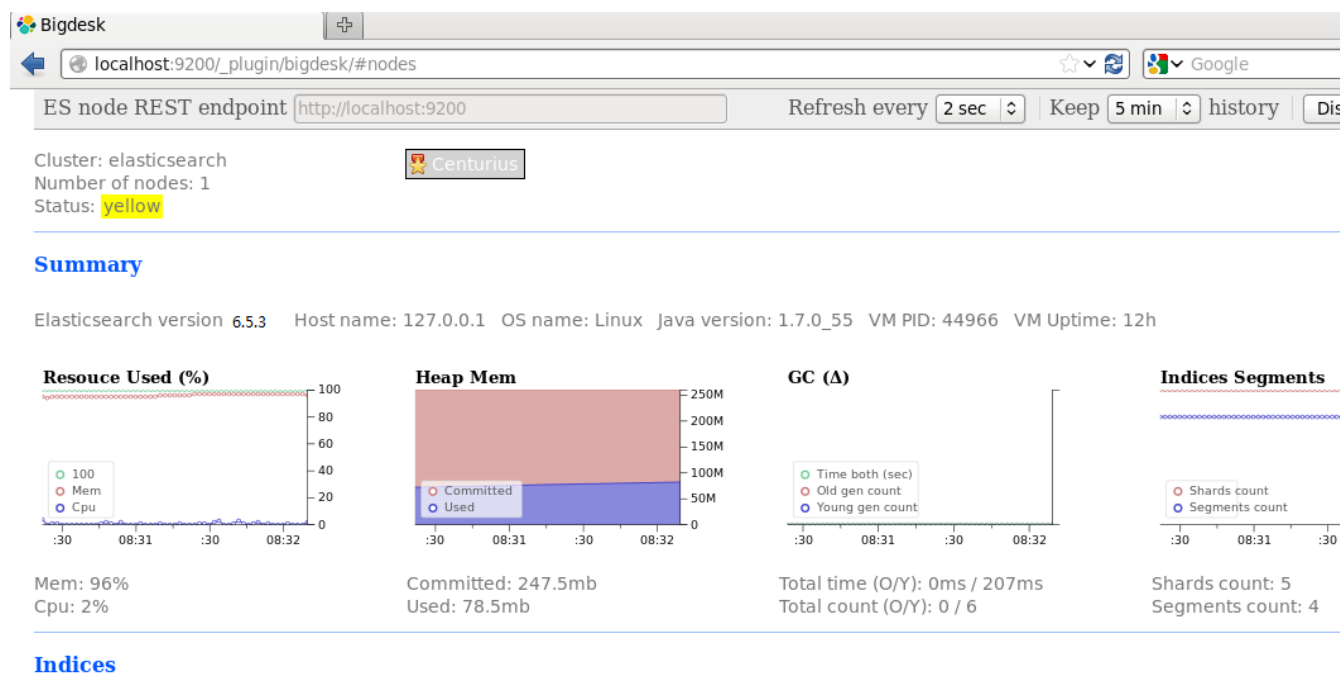
或者以后台进程的方式启动 : `nohup python -m SimpleHTTPServer > /dev/null 2>&1 &` ~>以后台进程的方式启动bigdesk监控服务 (监控es集群的)

③启动, 访问(web) : `http://janson01:8000/#nodes` ~>访问bigdesk插件, bigdesk插件再去访问es服务器 (用于监控es服务器的状况)

注意 : 浏览器的选择, 可以使用 : 火狐, 360, chrome, ...

## 5.4、BigDesk Plugin使用

访问, 在浏览器输入[http://master:9200/\\_plugin/bigdesk/](http://master:9200/_plugin/bigdesk/) 里面可以看到集群名称, 节点列表。内存消耗情况, GC回收情况。可以自由的在各个节点之间进行切换, 自动的添加或是移除一些旧的节点。同样可以更改refresh interval刷新间隔, 图标能够显示的数据量。



## 5.5、ES-Head Plugin介绍

方便对ES进行各种操作的可视化的客户端工具, 推荐大家使用

## 5.6、ES-Head Plugin安装

安装 : `bin/plugin install mobz/elasticsearch-head`

→es head:elasticsearch-head是一个elasticsearch的集群管理工具，它是完全由HTML5编写的独立网页程序，你可以通过插件把它集成到es。 ~> 使用对象：程序员使用

官方的资料：

<https://github.com/mobz/elasticsearch-head#running-with-built-in-server>

安装步骤：（注意：在root用户下安装）

①nodejs npm grunt安装（在安装html5运行的环境）

```
yum install nodejs
```

```
yum install npm
```

```
npm install -g grunt
```

```
npm install -g grunt-cli
```

# 下述的命令需要将os的当前目录设定为head插件的根目录（正式开始实施）

```
cnpm install（或是：npm install -g cnpm --
```

```
registry=https://registry.npm.taobao.org；或者是：npm install //执行后会生成node_modules文件夹）
```

注意：

★若是在线安装失败的话，需要手动下载安装包手动安装。

②修改Gruntfile.js

在该文件中添加如下，务必注意不要漏了添加“，”号，这边的hostname: '\*'，表示允许所有IP可以访问，此处也可以修改端口号

```
server: {  
    options: {  
        hostname: '*',  
        port: 9100,  
        base: '.',  
        keepalive: true  
    }  
}
```

③启动grunt server

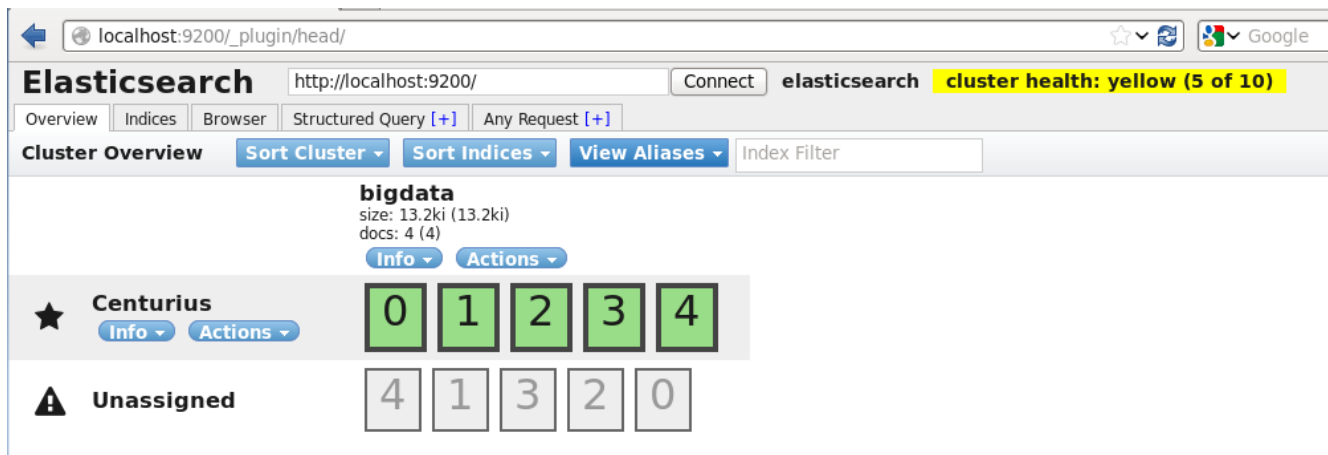
或者是以后台进程的方式启动：

```
nohup grunt server > /dev/null 2>&1 &
```

④访问 <http://JANSON01:9100>（建议：使用google浏览器访问）

## 5.7、ES-Head Plugin使用

访问：[http://master:9200/\\_plugin/head/](http://master:9200/_plugin/head/)



## 第六章：ES集群

### 6.1、ES集群安装介绍

集群安装非常简单，只要节点同属于一个局域网同一网段，而且集群名称相同，ES就会自动发现其他节点。主要配置项  
elasticsearch-6.5.3节点一 --> master cluster.name: bigdata http.port: 9200 network.host: 0.0.0.0  
elasticsearch-6.5.3节点二 --> slave01 cluster.name: bigdata http.port: 19200 network.host: 0.0.0.0  
transport.tcp.port: 19300 elasticsearch-6.5.3节点三 --> slave02 cluster.name: bigdata http.port: 29200  
network.host: 0.0.0.0 transport.tcp.port: 29300 配置完成之后启动三个ES节点 通过ES插件elasticsearch-head查看集群信息

### 6.2、ES集群安装演示（一）

前提：（在另外两台节点上名为tom的用户）

①基于单机版，别的节点上安装的前提事先准备好：

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

②配置tom用户到另外两台节点的免密码登录

ssh-keygen -t rsa

ssh-copy-id -i tom@janson02 ~> 将当前节点上的公钥拷贝到别的节点上

③es集群安装注意点：（集群中所有节点都需要配置）

discovery.zen.minimum\_master\_nodes: 2 <~ 防止“脑裂”（brain split），集群中至少有两台节点可用，否则，若只有一台，集群就瘫痪，计算公式：数 = 节点数/2 + 1

discovery.zen.ping.unicast.hosts: ["JANSON01", "JANSON02", "JANSON03"] <~ es集群中有哪些节点，官方文档上显示：只要集群中的所有节点在同一个网段内，所有索引服务器彼此感知到，自动组织成一个集群

### 6.3、ES集群安装演示（二）

步骤：

①将JANSON01节点上es目录拷贝到JANSON02, JANSON03上

scp -r ~/es tom@janson03:~/

mkdir -p /home/tom/data/elastic <~ 新建目录，不能直接从janson01节点拷贝，若是手动拷贝，集群会失效！（会自动将节点JANSON01上的数据自动同步到别的节点上）

mkdir -p /home/tom/logs/elastic

②修改es核心配置文件elasticsearch.yml

node.name: 集群中当前节点的名字

network.host: ip地址的别名



③验证：（不需要在别的节点上安装插件，因为插件是独立于es服务器单独存在）  
通过插件进行验证

④集群的健康状况：

Green： 所有的主分片和副分片都可用

Yellow：所有的主分片都可以不是所有的副分片都可用

Red： 不是所有的主分片和副分片都可用

## 6.4、ES集群演示

Elasticsearch <http://service.bigdata.jack.cn:9200/> 连接 bigdata 集群健康值: green (0)

概览 索引 数据浏览 基本查询 [+] 复合查询 [+]

集群概览 集群排序 Sort Indices View Aliases Index Filter

★ Psiklop ☆代表master节点，○代表slave节点  
信息 动作

● Fabian Cortez  
信息 动作

● Green Goblin 都可查阅集群节点的具体信息  
信息 动作

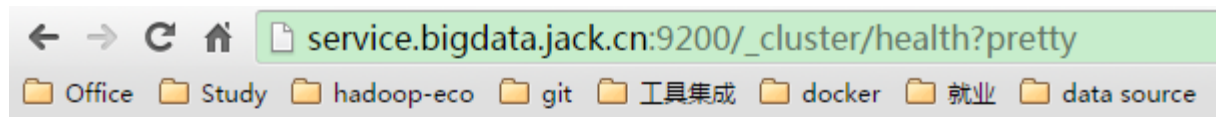
## 第七章：ES核心概念

### 7.1、ES核心概念概述

### 7.2、ES核心概念之Cluster

Cluster 代表一个集群，集群中有多个节点，其中有一个为主节点，这个主节点是可以通过选举产生的，主从节点是对于集群内部来说的。ES的一个概念就是去中心化，字面上理解就是无中心节点，这是对于集群外部来说的，因为从外部来看ES集群，在逻辑上是个整体，你与任何一个节点的通信和与整个ES集群通信是等价的。主节点的职责是负责管理集群状态，包括管理分片的状态和副本的状态，以及节点的发现和删除。只需要在同一个网段之内启动多个ES节点，就可以自动组成一个集群。默认情况下ES会自动发现同一网段内的节点，自动组成集群。集群的查看状态 `http://<ip|host>:9200/_cluster/health?pretty`





```
{
  "cluster_name" : "bigdata",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 3,
  "number_of_data_nodes" : 3,
  "active_primary_shards" : 0,
  "active_shards" : 0,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

### 7.3、ES核心概念之shards

代表索引分片，ES可以把一个完整的索引分成多个分片，这样的好处是可以把一个大的索引拆分成多个，分布到不同的节点上，构成分布式搜索。分片的数量只能在索引创建前指定，并且索引创建后不能更改。可以在创建索引库的时候指定 `curl -XPUT 'localhost:9200/test1/' -d '{"settings":{"number_of_shards":3}}'` 默认是一个索引库有5个分片 `index.number_of_shards:5`

### 7.4、ES核心概念之replicas

代表索引副本，ES可以给索引设置副本，副本的作用一是提高系统的容错性，当某个节点某个分片损坏或丢失时可以从副本中恢复。二是提高ES的查询效率，ES会自动对搜索请求进行负载均衡。可以在创建索引库的时候指定 `curl -XPUT 'localhost:9200/test2/' -d '{"settings":{"number_of_replicas":2}}'` 默认是一个分片有1个副本 `index.number_of_replicas:1`

### 7.5、ES核心概念之recovery & gateway

代表数据恢复或者叫数据重新分布，ES在有节点加入或退出时会根据机器的负载对索引分片进行重新分配，挂掉的节点重新启动时也会进行数据恢复。

代表ES索引的持久化存储方式，ES默认是先把索引存放到内存中，当内存满了时再持久化到硬盘。当这个ES集群关闭在重新启动是就会从gateway中读取索引数据。Es支持多种类型的gateway，有本地文件系统(默认)，分布式文件系统，Hadoop的HDFS和amazon的s3云存储服务。

### 7.6、ES核心概念之discovery.zen

代表ES的自动发现节点机制，ES是一个基于p2p的系统，它先通过广播寻找存在的节点，再通过多播协议来进行节点之间的通信，同时也支持点对点的交互。 \*\*如果是不同网段的节点如果组成ES集群 禁用自动发现机制 `discovery.zen.ping.multicast.enabled: false` 设置新节点被启动时能够发现的注解列表 `discovery.zen.ping.unicast.hosts: ["master:9200", "slave01:9200"]`

### 7.7、ES核心概念之Transport

代表ES内部节点或集群与客户端的交互方式，默认内部是使用tcp协议进行交互，同时它支持http协议(json格式)、thrift、servlet、memcached、zeroMQ等传输协议(通过插件方式集成)。

## 第八章：ES JavaAPI

### 8.1、ES JavaAPI概述

前提：涉及到哪些核心的api: TransportClient <~ 封装了对es服务器的各种操作

简单的crud:(C: 新增, create; R: 查询, retrieve; U:更新, update; D: 删除, delete) TransportClient实例.prepareIndex(索引库,类型,标识) <~ 新增索引 TransportClient实例.prepareDelete(索引库,类型,标识) <~ 删除索引 TransportClient实例.prepareUpdate(索引库,类型,标识) <~ 更新索引 TransportClient实例.prepareGet(索引库,类型,标识) <~ 查询单条索引信息

查询多条索引信息：TransportClient实例.prepareSearch

### 8.2、ES JavaAPI之连接ES

通过TransportClient接口，我们可以不启动节点就可以和ES集群进行通信，它需要指定 ES集群中其中一台或者多台机器IP地址和端口(默认9300) public class ElasticSearchTest { private static final int PORT = 9300; private TransportClient client; @Before public void setUp() { client = TransportClient.builder().build(); InetSocketAddress ista = new InetSocketAddress(new InetSocketAddress("master", PORT)); client.addTransportAddresses(ista); System.out.println("cluster.name = " + client.settings().get("cluster.name")); } @After public void cleanUp() {client.close();} }

前提：Maven pom依赖： org.elasticsearch elasticsearch 6.5.3

com.fasterxml.jackson.core jackson-databind 2.7.0 org.dom4j dom4j 2.0.0

- 1) 如果需要使用其他名称的集群 (默认是 elasticsearch)，需要如下设置

```
Settings settings = Settings.builder()
    .put("cluster.name", "nCName").build();
TransportClient client = TransportClient.builder()
    .settings(settings).build()
    .addTransportAddress(new
        InetSocketAddress("host", 9300));
```

```
16- @Before
17- public void setUp() {
18-     Settings settings = Settings.builder().put("cluster.name", "bigdata").build();
19-     client = TransportClient.builder().settings(settings).build();
```

Markers Properties Servers Data Source Explorer Snippets Problems Console JUnit Package Explorer  
<terminated> ElasticSearchTest [JUnit] D:\Program Files\Java\64bit\jdk1.7.0\_75\bin\javaw.exe (2016年10月10日 上午10:30:27)  
十月10, 2016 10:30:28 上午 org.elasticsearch.plugins.PluginsService <init>  
信息: [Julie Power] modules [], plugins [], sites []  
十月10, 2016 10:30:30 上午 org.elasticsearch.client.transport.TransportClientNodesService\$Si  
警告: [Julie Power] node {#transport#-1}{192.168.43.158}{service.bigdata.jack.cn/192.168.43  
cluster.name = bigdata

- 2) 通过 TransportClient 这个接口，自动嗅探整个集群的状态，ES 会自动把集群中其它机器的 IP 地址添加到客户端中。

```
Settings settings = Settings.builder()
    .put("client.transport.sniff", true).build();
TransportClient client = TransportClient.builder()
    .settings(settings).build()
    .addTransportAddress(new
        InetSocketAddress("host", 9300));
```

### 8.3、ES JavaAPI之增加索引之json方式

### 8.4、ES JavaAPI之增加索引之map方式

增加索引的数据格式有 4 种, json、map、bean、es helper

- JSON

@Test

```
public void testAddIndexJSON() {  
    String source = "{\"name\":\"hadoop\", \"author\" : \"Doug  
Couting\"}";  
    IndexResponse response = client  
        .prepareIndex(index, type, "1").setSource(source).get();  
    System.out.println("version: " + response.getVersion());  
}
```

- Map

@Test

```
public void testAddIndexMap() {  
    Map<String, Object> map = new HashMap<String, Object>();  
    map.put("name", "HBase");  
    map.put("version", 1.1);  
    IndexResponse response = client  
        .prepareIndex(index, type, "2").setSource(map).get();  
    System.out.println("version: " + response.getVersion());  
}
```

### 8.5、ES JavaAPI之增加索引之bean方式

### 8.6、ES JavaAPI之增加索引之es helper介绍

### 8.7、ES JavaAPI之增加索引之es helper方式

- Bean

```
@Test
public void testAddIndexBean() throws Exception {
    BigdataProduct bp = new BigdataProduct("spark", "apache", "1.6");
    ObjectMapper oMapper = new ObjectMapper();
    byte[] bytes = oMapper.writeValueAsBytes(bp);
    IndexResponse response = client.prepareIndex(index, type,
"3").setSource(bytes).get();
    System.out.println("version: " + response.getVersion());
}
```

- ES Helper

```
@Test
public void testAddIndexHelper() throws Exception {
    XContentBuilder xBuilder = XContentFactory.jsonBuilder()
        .startObject()
        .field("name", "flume")
        .field("version", "1.6")
        .field("author", "apache")
        .endObject();
    IndexResponse response = client.prepareIndex(index, type,
"4").setSource(xBuilder).get();
    System.out.println("version: " + response.getVersion());
}
```

## 8.8、ES JavaAPI之查询

```
@Test public void testGet() { GetResponse response = client.prepareGet(index, type, "1").get(); Map<String,
Object> map = response.getSource(); System.out.println("version: " + response.getVersion());
for(Map.Entry<String, Object> me : map.entrySet()) { System.out.println(me.getKey() + "=" + me.getValue()); } }
```

## 8.9、ES JavaAPI之更新

```
@Test public void testUpdate() throws Exception { XContentBuilder source= XContentFactory.jsonBuilder()
.startObject() .field("name", "hadoop") .field("author", "CDH") .field("version", 2.7) .endObject();
client.prepareUpdate(index, type, "1").setDoc(source).get(); testGet(); }
```

## 8.10、ES JavaAPI之删除

```
@Test public void testDelete() { DeleteResponse response = client.prepareDelete(index, type, "4").get();
System.out.println("version: " + response.getVersion()); }
```

## 8.11、ES JavaAPI之索引文档条数

```
@Test public void testCount() { //该方法过时，请使用prepareSearch long count =
client.prepareCount("bigdata").get().getCount(); System.out.println(count); }
```

## 8.12、ES JavaAPI之批量操作介绍

## 8.13、ES JavaAPI之批量操作bulk

```
@Test public void testBulkInsert() { String deptDev = "{\"name\":\"研发部\", \"deptNo\" : 20}"; String deptMarket = "  
{\"name\":\"市场部\", \"deptNo\" : 30}"; String deptOffice = "{\"name\":\"行政部\", \"deptNo\" : 40}"; client.prepareBulk()  
.add(new IndexRequest(index, "dep", "1").source(deptDev)).add(new IndexRequest(index, "dep",  
"2").source(deptMarket)).add(new IndexRequest(index, "dep", "3").source(deptOffice)).add(new  
DeleteRequest(index, type, "3")).get(); }
```

### 8.14、ES JavaAPI之全文索引概述

ES是基于Lucene的开源搜索引擎，其查询语法关键字部分和lucene大致一样：分页:from/size、字段:fields、排序:sort、查询:query 过滤:filter、高亮:highlight、统计:facet Search Type 查询query 中文分词

### 8.15、SearchType详解

es的搜索类型有4种 query and fetch(速度最快)(返回N倍数据量) query then fetch (默认搜索方式) DFS query and fetch DFS query then fetch(可以更精确控制搜索打分和排名。)DFS解释：见备注 总结一下，从性能考虑 QUERY\_AND\_FETCH是最快的，DFS\_QUERY\_THEN\_FETCH是最慢的。从搜索的准确度来说，DFS要比非DFS的准确度更高。

DFS是什么缩写？这个D可能是Distributed，F可能是frequency的缩写，至于S可能是Scatter的缩写，整个单词可能是分布式词频率和文档频率散发的缩写。

初始化散发是一个什么样的过程？从es的官方网站我们可以发现，初始化散发其实就是在进行真正的查询之前，先把各个分片的词频率和文档频率收集一下，然后进行词搜索的时候，各分片依据全局的词频率和文档频率进行搜索和排名。显然如果使用DFS\_QUERY\_THEN\_FETCH这种查询方式，效率是最低的，因为一个搜索，可能要请求3次分片。但，使用DFS方法，搜索精度应该是最高的。

### 8.16、SearchType案例演示

```
/**  
 * 测试：检索类型，以及分页检索  
 */  
@Test  
public void testSearchTypeAndSplitPage() {  
    //案例1：检索bigdata索引库中，product type中的字段name为hive的索引信息。学习知识点：检索类  
    型，分页检索  
  
    SearchResponse response = client.prepareSearch(indices)  
        //指定所关注的type  
        .setTypes(TYPE_PRODUCT)  
        //设定searchType  
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)  
        //设置分页（查询第一页，每页显示2条记录），公式：开始索引 = （页码-1）*pageSize  
        .setFrom((1 - 1) * 2)  
        .setSize(2)  
        //设置查询的条件  
        .setQuery(QueryBuilders.termQuery("name", "hive"))  
        .get();  
  
    //从结果中显示所有满足条件的记录  
    SearchHits hits = response.getHits();  
    for (SearchHit hit : hits) {
```

```

        logger.info("检索到的document信息是：" + hit.getSourceAsString());
    }
}

```

## 8.17、ES查询之Query概述

### 1, 查询所有

matchAllQuery() 匹配所有文件

match\_all 查询是Elasticsearch中最简单的查询之一。它使我们能够匹配索引中的所有文件。

```

SearchResponse searchResponse = client.prepareSearch("blog2")
    .setTypes("article").setQuery(QueryBuilders.matchAllQuery())
    .get();

```

SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象

### 2, 解析查询字符串

相比其他可用的查询, query\_string 查询支持全部的Apache Lucene查询语法

针对多字段的query\_string 查询

```

SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.queryStringQuery("全面")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象

```

### 3, 通配符查询 (wildcardQuery)

\* 匹配多个字符, ? 匹配1个字符

注意: 避免\* 开始, 会检索大量内容造成效率缓慢

```

SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.wildcardQuery("content", "elasc*?")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象

```

### 4, 词条查询 (termQuery)

词条查询是Elasticsearch中的一个简单查询。它仅匹配在给定字段中含有该词条的文档, 而

且是确切的、未经分析的词条

termQuery("key", obj) 完全匹配

termsQuery("key", obj1, obj2..) 一次匹配多个值, 只要有一个值是正确的, 就可以查询出数据

```

// SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
//     .setQuery(QueryBuilders.termQuery("content", "搜索")).get();
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.termsQuery("content", "搜索", "全文")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象

```

### 5, 字段匹配查询

matchQuery("key", obj) 单个匹配, field 不支持通配符, 前缀具高级特性

match 查询把query参数中的值拿出来, 加以分析, 然后构建相应的查询。使用match查询时, Elasticsearch将对一个字段选择合适的分析器, 所以可以确定, 传给match查询的词条将被建立索引时相同的分析器处理。

multiMatchQuery("text", "field1", "field2"..); 匹配多个字段, field 有通配符查询功能

```

// SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
//     .setQuery(QueryBuilders.matchQuery("content", "搜索")).get();
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.multiMatchQuery("搜索", "title", "content")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象

```



#### 6, 只查询ID (标识符查询)

标识符查询是一个简单的查询, 仅用提供的标识符来过滤返回的文档。此查询针对内部的`_uid`字段运行, 所以它不需要启用`_id`字段

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.idsQuery().ids("1")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象
```

#### 7, 相似度查询

fuzzy查询是模糊查询中的第三种类型, 它基于编辑距离算法来匹配文档

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.fuzzyQuery("content", "elasticsearxx")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象
```

#### 8, 范围查询

范围查询使我们能够找到在某一字段值在某个范围里的文档, 字段可以是数值型, 也可以是基于字符串的

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.rangeQuery("content").from("我们").to("解决方案")
    .includeLower(true).includeUpper(true)).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象
```

`includeLower(true)`: 包含上界

`includeUpper(true)`: 包含下界

#### 9, 跨度查询

下面代码表示, 从首字母开始, 查询content字段=问题的数据, 问题前面的词为300个, 可以测试30看是否能查询出数据。

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")

.setQuery(QueryBuilders.spanFirstQuery(QueryBuilders.spanTermQuery("content", "问题"),
300)).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象
```

#### 10, 组合查询 (复杂查询)

```
must(QueryBuilders) : AND
mustNot(QueryBuilders): NOT
should(QueryBuilders):OR
```

在定义json: 放置到Elasticsearch的插件中

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.boolQuery().must(QueryBuilders.termQuery("title", "搜索"))
    .must(QueryBuilders.wildcardQuery("content", "elastic*ch"))).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象
```

#### 10, 排序查询

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.matchAllQuery())
    .addSort("id", SortOrder.DESC).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数, 查询结果有多少对象
```



## 8.18、ES查询之boolQuery() 布尔查询，可以用来组合多个查询条件

组合查询（复杂查询）

```
must(QueryBuilders) : AND
mustNot(QueryBuilders): NOT
should(QueryBuilders):OR
```

在定义json：放置到Elasticsearch的插件中

```
{
  "query":{
    "bool":{
      "must":{
        "term":{
          "title":"elasticsearch"
        }
      },
      "should":{
        "range":{
          "id":{
            "from":1,
            "to":2
          }
        }
      }
    }
  }
}
```

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.boolQuery().must(QueryBuilders.termQuery("title", "搜索"))
    .must(QueryBuilders.wildcardQuery("content", "elastic*ch"))).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
```

## 8.19、ES查询之fuzzyQuery() 模糊查询

相似度查询

fuzzy查询是模糊查询中的第三种类型，它基于编辑距离算法来匹配文档

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.fuzzyQuery("content", "elasticsearchx")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
```

## 8.20、ES查询之matchAllQuery() 查询所有数据

matchAllQuery()匹配所有文件

match\_all查询是Elasticsearch中最简单的查询之一。它使我们能够匹配索引中的所有文件。

```
SearchResponse searchResponse = client.prepareSearch("blog2")
    .setTypes("article").setQuery(QueryBuilders.matchAllQuery())
    .get();
```

```
SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
```

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.matchAllQuery())
    .addSort("id", SortOrder.DESC).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
```

## 8.21、ES查询之regexpQuery() 正则表达式查询

假设现在你想匹配在w地域(Area)的所有邮政编码，那我们怎样来匹配呢？我们这一节就来介绍一下regexp匹配。

```
curl -XGET 'http://localhost:9200/my_index/address/_search -d '
{
  "query": {
    "regexp": {
      "postcode": "w[0-9].+"
    }
  }
}
```

这个正则表达式的规定了词条需要以w开头，紧跟着一个0到9的数字，然后是一个或者多个其它字符。

regexp所要匹配的字段要以正则式的形式出现。如以上代码中"w[0-9].+"。

## 8.22、ES查询之termQuery() 词条查询

词条查询是Elasticsearch中的一个简单查询。它仅匹配在给定字段中含有该词条的文档，而且是确切的、未经分析的词条

termQuery("key", obj) 完全匹配

termsQuery("key", obj1, obj2..) 一次匹配多个值，只要有一个值是正确的，就可以查询出数据

```
// SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
// .setQuery(QueryBuilders.termQuery("content", "搜索")).get();
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.termsQuery("content", "搜索", "全文")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
```

## 8.23、ES查询之wildcardQuery() 通配符查询

\*匹配多个字符，?匹配1个字符

注意：避免\* 开始，会检索大量内容造成效率缓慢

```
SearchResponse searchResponse = client.prepareSearch("blog2").setTypes("article")
    .setQuery(QueryBuilders.wildcardQuery("content", "elas*c?")).get();
SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
```

## 8.24、ES查询详解之分页from/size

核心代码：

setFrom(0).setSize(1)

```
/**
 * 测试：检索类型，以及分页检索
 */
@Test
public void testSearchTypeAndSplitPage() {
    //案例1：检索bigdata索引库中，product type中的字段name为hive的索引信息。学习知识点： 检索类型，
    分页检索

    SearchResponse response = client.prepareSearch(indices)
        //指定所关注的type
        .setTypes(TYPE_PRODUCT)
        //设定searchType
        .setSearchType(SearchType.DFS_QUERY_THEN_FETCH)
        //设置分页（查询第一页，每页显示2条记录），公式：开始索引 = （页码-1）*pageSize
        .setFrom((1 - 1) * 2)
        .setSize(2)
        //设置查询的条件
        .setQuery(QueryBuilders.termQuery("name", "hive"))
        .get();

    //从结果中显示所有满足条件的记录
    SearchHits hits = response.getHits();
    for (SearchHit hit : hits) {
        logger.info("检索到的document信息是：" + hit.getSourceAsString());
    }
}
```

## 8.25、ES查询详解之排序sort

.addSort("age", SortOrder.DESC)

```
/**
 * 排序查询演示
 */
@Test
```

```

public void sortSearch() {
    //需求：检索索引库bank中的type为account中所有男性客户，根据得分降序排列，若得分相同，然后根据每个账户的余额升序排列。

    //步骤：
    //1) 检索
    SearchResponse response = client.prepareSearch(indices)
        .setTypes("account")
        //设置检索的条件
        .setQuery(QueryBuilders.termQuery("gender.keyword", "M"))
        //定制排序规则
        //根据得分降序排列
        .addSort(SortBuilders.scoreSort().order(SortOrder.DESC))
        //若得分相同，根据银行存款余额的升序排列
        .addSort(SortBuilders.fieldSort("balance").order(SortOrder.ASC))
        //设置分页
        .setFrom(0)
        .setSize(1000)
        //触发检索
        .get();

    //2) 分析并显示检索后的结果
    SearchHits hits = response.getHits();
    for (SearchHit hit : hits) {
        float score = hit.getScore();
        String source = hit.getSourceAsString();
        System.out.printf("当前索引的得分是：%.8f，内容是：%s\n", score, source);
    }
}

```

## 8.26、ES查询详解之过滤filter

```

.setPostFilter(FilterBuilders.rangeFilter("age").from(1).to(19))

```

## 8.27、ES查询详解之高亮highlight

```

public void testSearch() {
    String indices = "bigdata";//指的是要搜索的哪一个索引库
    SearchRequestBuilder builder = client.prepareSearch(indices)
        .setSearchType(SearchType.DEFAULT)
        .setFrom(0)
        .setSize(5)//设置分页
        .addHighlightedField("name");//设置高亮字段
        .setHighlighterPreTags("<font style='color:red;size=35'>")
        .setHighlighterPostTags("</font>");//高亮风格
    builder.setQuery(QueryBuilders.fuzzyQuery("name", "hadoop"));
    SearchResponse searchResponse = builder.get();
    SearchHits searchHits = searchResponse.getHits();
    SearchHit[] hits = searchHits.getHits();
    long total = searchHits.getTotalHits();
    System.out.println("总共条数： " + total);//总共查询到多少条数据
    for (SearchHit searchHit : hits) {

```

```

        Map<String, Object> source = searchHit.getSource();
        Map<String, HighlightField> highlightFields =
            searchHit.getHighlightFields();
        System.out.println("-----");
        String name = source.get("name").toString();
        String author = source.get("author").toString();
        System.out.println("name=" + name);
        System.out.println("author=" + author);
        HighlightField highlightField =
            highlightFields.get("name");
        if(highlightField != null) {
            Text[] fragments = highlightField.fragments();
            name = "";
            for (Text text : fragments) {
                name += text.toString();
            }
        }
        System.out.println("name: " + name);
        System.out.println("author: " + author);
    }
}

```

## 8.28、ES查询详解之aggregations

根据字段进行分组统计 根据字段分组，统计其他字段的值 size设置为0，会获取所有数据，否则，只会返回10条。

```

/**
 * 聚合查询演示
 */
@Test
public void aggregationSearch() {
    //需求：查询bank索引库中，所有女性员工数，最年轻的员工的年龄，最低薪水(银行存款余额)，以及平均薪水。

    //步骤：
    //1) 检索
    SearchResponse response = client.prepareSearch(indices)
        .setTypes("account")
        //设置检索的条件
        .setQuery(QueryBuilders.termQuery("gender.keyword", "F"))
        //设置聚合操作
        //①员工数
        .addAggregation(new ValueCountAggregationBuilder("cntEmp",
ValueTypes.LONG).field("account_number"))
        //②最年轻的（女性）员工的年龄
        .addAggregation(new MinAggregationBuilder("minAge").field("age"))
        //③（女性员工）最少银行存款余额
        .addAggregation(new MinAggregationBuilder("minBalance").field("balance"))
        //④（女性员工）平均银行存款余额
        .addAggregation(new AvgAggregationBuilder("avgBalance").field("balance"))
        //触发检索
        .get();
}

```

```
//2) 分析检索后的结果
Aggregations aggregations = response.getAggregations();

ValueCount cntEmp = aggregations.get("cntEmp");
Min minAge = aggregations.get("minAge");
Min minBalance = aggregations.get("minBalance");
Avg avgBalance = aggregations.get("avgBalance");

//3) 显示结果
//%d, 占位符, 同时也是格式符, 用来格式化整数的; %f: 格式化小数; %s: 格式化字符串; %n: 用来换行
System.out.printf("bank索引库中, 所有女性员工数是: %d, 最年轻的员工的年龄: %.0f, 最低银行存款余额: %.2f, 以及平均银行存款余额: %.2f",
    cntEmp.getValue(), minAge.getValue(), minBalance.getValue(),
    avgBalance.getValue());
}
```

## 第九章：中文分词

### 9.1、中文分词概述

我们在上面的执行过程中看到了，查询中文基本查询不出数据，那是因为ES都是需要对每一句话进行分词，拆分后才能够进行查询解析。因为底层依赖lucene，所以中文分词效果不佳，但是有比较好的分词插件，比较好的中文分词有IK，庖丁解牛中文分词等等。

### 9.2、中文分词必要性详解

先不安装任何中文分词的插件，使用termQuery查询，看能否查询到需要的结果 新建索引库ok

```
##添加数据
curl -H 'Content-Type:application/json' -XPOST http://JANSON01:9200/ok/news/1 -d '{"content": "美国留给伊拉克的是个烂摊子吗"}'
curl -H 'Content-Type:application/json' -XPOST http://JANSON01:9200/ok/news/2 -d '{"content": "公安部：各地校车将享最高路权"}'
curl -H 'Content-Type:application/json' -XPOST http://JANSON01:9200/ok/news/3 -d '{"content": "中韩渔警冲突调查：韩警平均每天扣1艘中国渔船"}'
curl -H 'Content-Type:application/json' -XPOST http://JANSON01:9200/ok/news/4 -d '{"content": "中国驻洛杉矶领事馆遭亚裔男子枪击 嫌犯已自首"}'
```

### 9.3、中文分词插件安装演示

安装了中文分词的插件（如：ik），使用termQuery查询，看能否查询到需要的结果 具体步骤：①Elasticsearch-6.5.3安装配置完毕。②下载IK-6.5.3 源码下载：<https://github.com/medcl/elasticsearch-analysis-ik/releases> 编译版本：elasticsearch-analysis-ik-6.5.3.zip

在线安装：./bin/elasticsearch-plugin install <https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v6.5.3/elasticsearch-analysis-ik-6.5.3.zip>

③在线安装后，在相应的目录下会出现安装后的结果：a)/home/tom/es/config/analysis-ik ~> 包含常用的中文词组的字典文件，如：main.dic b)/home/tom/es/plugins/analysis-ik ~> 包含一些jar包以及资源文件

若是离线安装时，需要解压到ES\_HOME/plugins/ik目录下面(直接包含一个conf文件夹和一堆.jar包)

注意：安装完毕后，需要将安装好的目录跨节点拷贝到另外两台节点上

④重新启动ES ⑤启动 看到try load config .....IK相关信息，说明启动完成和安装IK插件完成。

修改ES\_HOME/config/elasticsearch.yml文件，添加index.analysis.analyzer.default.type: ik(把IK设置为默认分词器,这一步是可选的)

```
95
96 index.analysis.analyzer.default.type: ik
```

```
@Before
public void setUp() {
    Settings settings = Settings.settingsBuilder()
        .put("analyzer", "ik").build();
    client = TransportClient.builder().settings(settings).build();
    InetSocketAddress ista = new
        InetSocketAddress(new InetSocketAddress(HOST, PORT));
    client.addTransportAddress(ista);
}
```

#### 9.4、IK分词器，自定义词库

测试例子：~>在Kibana的Dev Tools中，运行 POST \_analyze { "analyzer":"ik\_smart", "text":"中国人民警察的服务宗旨" }

~>使用head插件来测试

```
http://JANSON01:9200/_analyze
{
  "analyzer":"ik_smart",
  "text":"天团S.H.E昨在两厅院艺文广场举办17万人露天音乐会，3人献唱多首经典好
歌，让现场粉丝听得如痴如醉"
}
```

注意点：

①针对于es集群中已经存在的历史索引库，不会进行重新分词，分词插件不起作用。

②新建索引库，以及索引库下的type时，要指定相应的中文分词插件，才会起作用。

会根据分词插件，对新增的索引信息进行分词，存储到es集群中。

③需要将安装好的ik中文分词插件拷贝到集群中别的节点上。

④给es集群安装插件时，优先安装中文分词插件（建议排在第一位！！）。

⑤windows下的换行符是\r\n，Linux os下的换行符是\n；将windows下的指令拷贝到linux命令行下执行，往往会报错，不能正常执行，

应对方案是：a)先将内容粘贴到Linux下的临时文件中；b)然后从linux临时文件中拷贝

~~>

##创建索引库，并指定相应分词法

1.create a index，新建索引库

```
curl -XPUT http://janson01:9200/chinese
```

2.create a mapping，要指定索引库下的type对应的元数据信息（指定相应的中文分词插件，此处为ik）

```
curl -XPOST http://JANSON01:9200/chinese/hot/_mapping -H 'Content-Type:application/json' -d'
{
```



```

    "properties": {
      "content": {
        "type": "text",
        "analyzer": "ik_max_word",
        "search_analyzer": "ik_max_word"
      }
    }
  }
}

```

}'

3, 添加数据

```

curl -H 'Content-Type:application/json' -XPOST
http://JANSON01:9200/chinese/hot/1 -d '{"content": "美国留给伊拉克的是个烂摊子吗"}'
curl -H 'Content-Type:application/json' -XPOST
http://JANSON01:9200/chinese/hot/2 -d '{"content": "公安部：各地校车将享最高路权"}'
curl -H 'Content-Type:application/json' -XPOST
http://JANSON01:9200/chinese/hot/3 -d '{"content": "中韩渔警冲突调查：韩警平均每天扣1艘中国渔船"}'
curl -H 'Content-Type:application/json' -XPOST
http://JANSON01:9200/chinese/hot/4 -d '{"content": "中国驻洛杉矶领事馆遭亚裔男子枪击 嫌犯已自首"}'

```

4, 执行查询

```

curl -H 'Content-Type:application/json' -XPOST
'http://JANSON01:9200/chinese/hot/_search?pretty' -d '{
  "query" : { "match" : { "content" : "中国" } },
  "highlight" : {
    "pre_tags" : [ "<font>", "<u>" ],
    "post_tags" : [ "</font>", "</u>" ],
    "fields" : {
      "content" : {}
    }
  }
}'

```

如何自定义词库呢？

此电脑 > 系统 (D:) > software > elasticsearch-6.5.3 > plugins > analysis-ik > config >

名称	修改日期	类型	大小
custom	2017/5/4 17:56	文件夹	
IKAnalyzer.cfg.xml	2016/9/1 19:38	XML 文档	1 KB
main.dic	2016/9/1 19:38	DIC 文件	2,987 KB
preposition.dic	2016/9/1 19:38	DIC 文件	1 KB
quantifier.dic	2016/9/1 19:38	DIC 文件	2 KB
stopword.dic	2016/9/1 19:38	DIC 文件	1 KB
suffix.dic	2016/9/1 19:38	DIC 文件	1 KB
surname.dic	2016/9/1 19:38	DIC 文件	1 KB

修改IKAnalyzer.cfg.xml

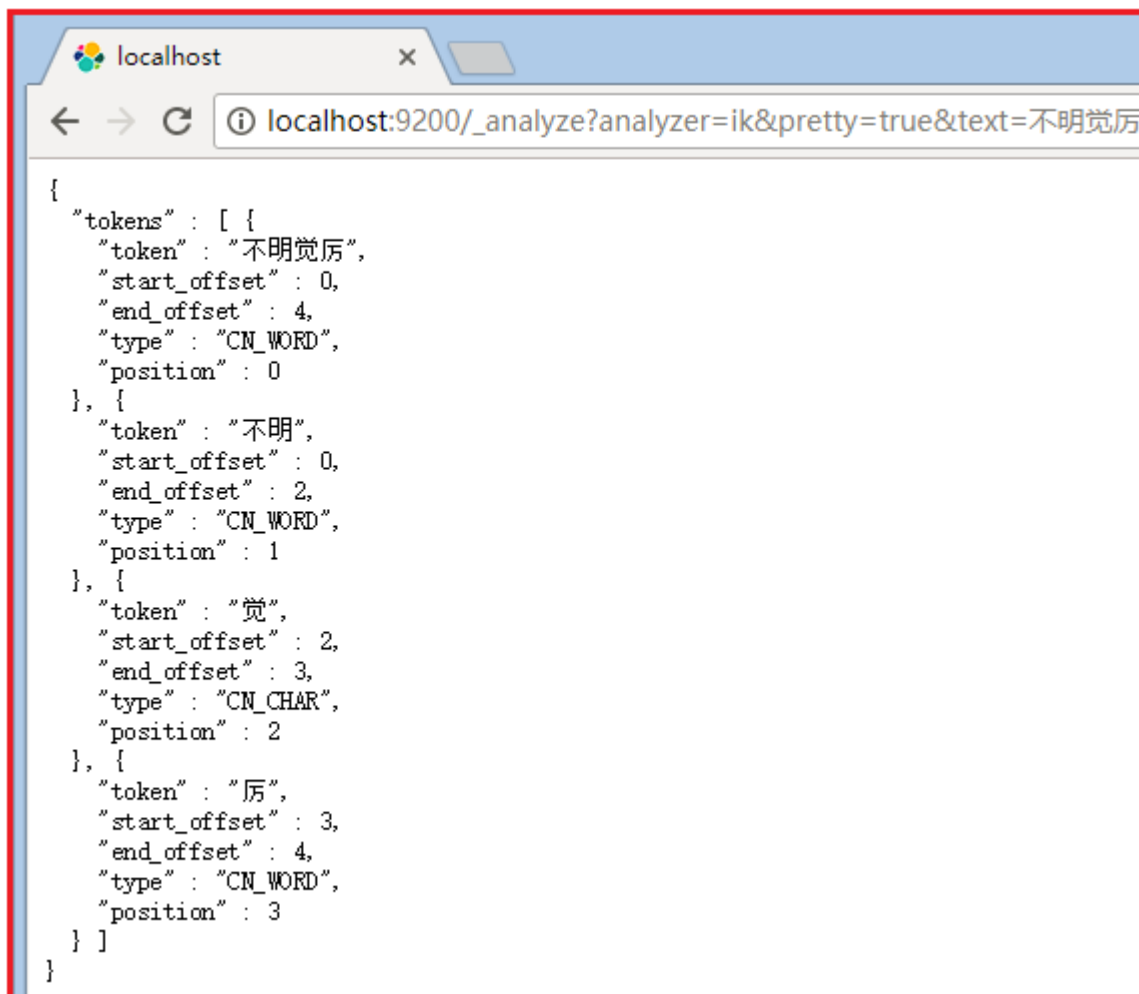
```
<properties>
  <comment>IK Analyzer 扩展配置</comment>
  <!--用户可以在这里配置自己的扩展字典 -->
  <entry key="ext_dict">custom/mydict.dic;custom/single_word_low_freq.dic</entry>
  <!--用户可以在这里配置自己的扩展停止词字典-->
  <entry key="ext_stopwords">custom/ext_stopword.dic</entry>
  <!--用户可以在这里配置远程扩展字典 -->
  <!-- <entry key="remote_ext_dict">words_location</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>
```

打开custom文件夹，mydict.dic，编辑文件



Line	Term
1	medcl
2	elastic
3	elasticsearch
4	kogstash
5	kibana
6	marvel
7	shield
8	watcher
9	beats
10	packetbeat
11	filebeat
12	topbeat
13	metrixbeat
14	kimchy
15	不明觉厉

重新启动es。



```
{
  "tokens" : [ {
    "token" : "不明觉厉",
    "start_offset" : 0,
    "end_offset" : 4,
    "type" : "CN_WORD",
    "position" : 0
  }, {
    "token" : "不明",
    "start_offset" : 0,
    "end_offset" : 2,
    "type" : "CN_WORD",
    "position" : 1
  }, {
    "token" : "觉",
    "start_offset" : 2,
    "end_offset" : 3,
    "type" : "CN_CHAR",
    "position" : 2
  }, {
    "token" : "厉",
    "start_offset" : 3,
    "end_offset" : 4,
    "type" : "CN_WORD",
    "position" : 3
  } ]
}
```

重新修改文档，再次搜索，可以查询到结果。

## 第十章：ES Rest

### 10.1、ES Rest之通过REST概述

有两种方式：一种方式是通过REST 请求URI，发送搜索参数；另外一种是通过REST请求体，发送搜索参数，而请求体允许你包含更容易表达和可阅读的JSON格式。

### 10.2、ES Rest之通过REST请求URI

curl '[http://localhost:9200/bank/\\_search?q=&pretty](http://localhost:9200/bank/_search?q=&pretty)' q=: 参数告诉elasticsearch，在bank索引中匹配所有的文档  
pretty: 参数告诉elasticsearch，返回形式打印JSON结果

```

yellow open kibana 1 1 1 0 3.1KB
[bigdata@service ~]$ curl 'http://localhost:9200/bank/_search?q=*&pretty'
{
  "took" : 9,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : 1.0,
    "hits" : [ {
      "index" : "bank",

```

### 10.3、ES Rest之通过REST请求体

上述匹配所有数据可以改成如下写法 `curl -XPOST 'localhost:9200/bank/_search?pretty' -d '{"query":{"match_all":{}}}'` 与第一种方式不同是在URI中替代传递`q=*`，使用POST方式提交，请求体包含JSON格式搜索。

```

[bigdata@service ~]$ curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
> {
> "query": {"match_all": {}}
> }'
{
  "took" : 29,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },

```

### 10.4、ES Rest之查询语言介绍

elasticsearch提供JSON格式领域特定语言执行查询。可参考Query DSL。`{"query":{"match_all":{}}}` query：告诉我们定义查询 `match_all`：运行简单类型查询指定搜索中的所有文档 除了指定查询参数，还可以指定其他参数来影响最终结果。

### 10.5、ES Rest之match\_all & 只返回第一个文档

`curl -XPOST 'localhost:9200/bank/_search?pretty' -d '{"query":{"match_all":{}},"size":1}'` 如果不指定size，默认是返回10条文档信息

```
[bigdata@service ~]$ curl -XPOST 'localhost:9200/bank/_search?pretty' -d '{"query": {"match_all": {}}, "size": 1}'
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "bank",
      "_type" : "accout",
      "_id" : "25",
      "_score" : 1.0,
      "_source" : {
        "account_number" : 25,
        "balance" : 40540,
        "firstname" : "Virginia",
        "lastname" : "Ayala",
        "age" : 39,
        "gender" : "F",
        "address" : "171 Putnam Avenue",
        "employer" : "Filodyne",
        "email" : "virginiaayala@filodyne.com",
        "city" : "Nicholson",
        "state" : "PA"
      }
    } ]
  }
}
```

match\_all & 返回11到20个文档信息（分页） curl -XPOST 'localhost:9200/bank/search?pretty' -d '{"query": {"match\_all": {}}, "from": 10 "size": 10}' from : 指定文档索引从哪里开始，默认从0开始 size : 从from开始，返回多个文档 这基本上就为分页奠定了基础。 match\_all & 根据account的balance 降序排序&返回10个文档（默认10个） curl -XPOST 'localhost:9200/bank/search?pretty' -d '{"query": {"match\_all": {}}, "sort": {"balance": {"order": "desc"}}}'

## 10.6、ES Rest之返回account\_number and balance

执行搜索 默认的，我们搜索返回完整的JSON文档。而source(\_source字段搜索点击量)。如果 我们不想返回完整的JSON文档，我们可以使用source返回指定字段。

```
curl -XPOST 'localhost:9200/bank/search?pretty' -d '{"query": {"match_all": {}}, "source": ["balance", "account_number"]}'
```

## 10.7、ES Rest之返回account\_number

match查询，可以作为基本字段搜索查询 返回account\_number=20: curl -XPOST 'localhost:9200/bank/\_search?pretty' -d '{"query": {"match": {"account\_number": 20}}}'

返回address=mill : curl -XPOST 'localhost:9200/bank/search?pretty' -d '{"query": {"match": {"address": "mill"}}}'  
 返回address=mill or address=lane curl -XPOST 'localhost:9200/bank/search?pretty' -d '{"query": {"match": {"address": "mill lane"}}}' 返回 短语匹配 address=mill lane curl -XPOST 'localhost:9200/bank/\_search?pretty' -d '{"query": {"match\_phrase": {"address": "mill lane"}}}'

## 10.8、ES Rest之布尔值(bool)查询

返回匹配address=mill&address=lane curl -XPOST 'localhost:9200/bank/\_search?pretty' -d '{"query": {"bool": {"must": [ {"match": {"address": "mill"}}, {"match": {"address": "lane"}} ] } } }' must:要求所有条件都要满足（类似于&&） should : 任何一个满足就可以（类似于||） must\_not:所有条件都不能满足（类似于!(&&)）

返回 匹配address=mill or address=lane curl -XPOST 'localhost:9200/bank/search?pretty' -d '{ "query": { "bool": { "should": [ { "match": { "address": "mill" } }, { "match": { "address": "lane" } } ] } } }' should : 任何一个满足就可以 ( 类似于|| ) 返回 不匹配address=mill & address=lane curl -XPOST 'localhost:9200/bank/search?pretty' -d '{ "query": { "bool": { "must\_not": [ { "match": { "address": "mill" } }, { "match": { "address": "lane" } } ] } } }' must\_not:所有条件都不能满足 ( 类似于! (&&) ) 返回 age=40 & state!=ID curl -XPOST 'localhost:9200/bank/\_search?pretty' -d '{ "query": { "bool": { "must": [ { "match": { "age": 40 } }, "must\_not": [ { "match": { "state": "ID" } } ] } } }' }

## 10.9 、 ES Rest之并行过滤器

文档中score(\_score字段是搜索结果)。score是一个数字型的，是一种相对方法匹配查询文档结果。分数越高，搜索关键字与该文档相关性越高；越低，搜索关键字与该文档相关性越低。在elasticsearch中所有的搜索都会触发相关性分数计算。如果我们不使用相关性分数计算，那就要使用另一种查询能力，构建过滤器。过滤器是类似于查询的概念，除了得以优化,更快的执行速度的两个主要原因：

1. 过滤器不计算得分，所以他们比执行查询的速度
2. 过滤器可缓存在内存中，允许重复搜索 为了便于理解过滤器，先介绍过滤器搜索(like match\_all, match, bool, etc.)，可以与其他普通查询搜索组合一个过滤器。 range filter,允许我们通过一个范围值来过滤文档，一般用于数字或日期过滤使用过滤器搜索返回 balances[ 20000,30000]。换句话说，balance>=20000 && balance<=30000。

curl -XPOST 'localhost:9200/bank/\_search?pretty' -d '{ "query": { "filtered": { "query": { "match\_all": {} }, "filter": { "range": { "balance": { "gte": 20000, "lte": 30000 } } } } }' 过滤查询包含match\_all查询(查询部分)和一系列过滤(过滤部分)。可以代替任何其他查询到查询部分以及其他过滤器过滤部分。在上述情况下,过滤器范围智能，因为文档落入range所有匹配“平等”，即比另一个更相关,没有文档。一般情况，最明智的方式决定是否使用filter or query，就看你是否关心相关性分数。如果相关性不重要，那就使用filter，否则就使用query。 queries and filters很类似于关系型数据库中的“SELECT WHERE clause”

## 第十一章：Settings和Mappings

### 11.1 、 Settings和Mappings概述

简单的说，就是

settings是修改分片和副本数的。

mappings是修改字段和类型的。

记住，可以用url方式来操作它们，也可以用java方式来操作它们。建议用url方式，因为简单很多。

### 11.2 、 Settings详解

维护索引库默认配置，当然经常用来修改默认配置。例如：分片数量，副本数量 查看：curl -XGET [http://localhost:9200/bigdata/\\_settings?pretty](http://localhost:9200/bigdata/_settings?pretty) 操作不存在的索引: curl -XPUT 'localhost:9200/bigdata/' -d '{"settings": {"number\_of\_shards": 3, "number\_of\_replicas": 2}}' 操作已存在的索引: curl -XPUT 'localhost:9200/bigdata/\_settings' -d '{"index": {"number\_of\_replicas": 2}}'

### 11.3 、 Mappings详解

就是对索引库中索引的字段名称及其数据类型进行定义，类似于关系数据库中表建立时要定义字段名及其数据类型那样，(和solr中的schme类似)不过es的mapping比数据库灵活很多，它可以动态添加字段。一般不需要指定mapping都可以，因为es会自动根据数据格式定义它的类型，如果你需要对某些字段添加特殊属性（如：定义使用其它分词器、是否分词、是否存储等），就必须手动添加mapping 查询索引库的mapping信息 `curl -XGET http://localhost:9200/bigdata/dep/_mapping?pretty` mappings修改字段相关属性，见备注 例如：字段类型，使用哪种分词工具 mappings 注意：下面可以使用indexAnalyzer定义分词器，也可以使用index\_analyzer定义分词器

操作不存在的索引

```
curl -XPUT 'localhost:9200/bigdata'
-d '{"mappings":{"emp":{"properties":{"name":{"type":"string","indexAnalyzer":"ik","searchAnalyzer":"ik"}}}}}'
```

操作已存在的索引

```
curl -XPOST http://localhost:9200/bigdata/dep/_mapping
-d '{"properties":{"name":{"type":"string","indexAnalyzer":"ik","searchAnalyzer":"ik"}}}'
```

## 第十二章：ES优化

### 12.1、ES优化概述

Elasticsearch是目前大数据领域最热门的技术栈之一，经过近8年的发展，已从0.0.X版升级至6.X版本，虽然增加了很多的特性和功能，但是在主体架构上，还是没有太多的变化。

#### 12.1.1 硬件环境选择：

如果有条件，尽可能使用SSD硬盘，不错的CPU。ES的厉害之处在于ES本身的分布式架构以及lucene的特性。IO的提升，会极大改进ES的速度和性能。

#### 12.1.2 系统拓扑设计：

ES集群在架构拓扑时，一般都会采用Hot-Warm的架构模式，即设置3种不同类型的节点：Master节点、Hot节点和Warm节点。

Master节点设置：一般会设置3个专用的master节点，以提供最好的弹性扩展能力。当然，必须注意discovery.zen.minimum\_master\_nodes 属性的设置，以防split-brain问题，使用公式设置： $N/2+1$ (N为候选master节点数)。该节点保持: node.data: false；因为master节点不参与查询、索引操作，仅负责对于集群管理，所以在CPU、内存、磁盘配置上，都可以比数据节点低很多。

Hot节点设置：索引节点（写节点），同时保持近期频繁使用的索引。属于IO和CPU密集型操作，建议使用SSD的磁盘类型，保持良好的写性能；节点的数量设置一般是大于等于3个。将节点设置为hot

类型:node.attr.box\_type: hot

针对index, 通过设置index.routing.allocation.require.box\_type: hot 可以设置将索引写入hot节点。

Warm节点设置：用于不经常访问的read-only索引。由于不经常访问，一般使用普通的磁盘即可。内存、CPU的配置跟Hot节点保持一致即可；节点数量一般也是大于等于3个。



当索引不再被频繁查询时，可通过`index.routing.allocation.require.box_type: warm`，将索引标记为warm，从而保证索引不写入hot节点，以便将SSD磁盘资源用在刀刃上。一旦设置这个属性，ES会自动将索引合并到warm节点。同时，也可以在`elasticsearch.yml`中设置`index.codec: best_compression`保证warm节点的压缩配置。

Coordinating节点：协调节点用于做分布式里的协调，将各分片或节点返回的数据整合后返回。在ES集群中，所有的节点都有可能是协调节点，但是，可以通过设置`node.master`、`node.data`、`node.ingest`都为false来设置专门的协调节点。需要较好的CPU和较高的内存。

### 12.1.3 ES的内存设置：

由于ES构建基于lucene，而lucene设计强大之处在于lucene能够很好的利用操作系统内存来缓存索引数据，以提供快速的查询性能。lucene的索引文件segments是存储在单文件中的，并且不可变，对于OS来说，能够很友好地将索引文件保持在cache中，以便快速访问；因此，我们很有必要将一半的物理内存留给lucene；另一半的物理内存留给ES（JVM heap）。所以，在ES内存设置方面，可以遵循以下原则：

1. 当机器内存小于64G时，遵循通用的原则，50%给ES，50%留给lucene。
2. 当机器内存大于64G时，遵循以下原则：

- a. 如果主要的使用场景是全文检索，那么建议给ES Heap分配 4~32G的内存即可；其它内存留给操作系统，供lucene使用（segments cache），以提供更快的查询性能。
- b. 如果主要的使用场景是聚合或排序，并且大多数是numerics，dates，geo\_points 以及not\_analyzed的字符类型，建议分配给ES Heap分配 4~32G的内存即可，其它内存留给操作系统，供lucene使用(doc values cache)，提供快速的基于文档的聚类、排序性能。
- c. 如果使用场景是聚合或排序，并且都是基于analyzed 字符数据，这时需要更多的 heap size，建议机器上运行多ES实例，每个实例保持不超过50%的ES heap设置（但不超过32G，堆内存设置32G以下时，JVM使用对象指标压缩技巧节省空间），50%以上留给lucene。

3. 禁止swap，一旦允许内存与磁盘的交换，会引起致命的性能问题。通过：在`elasticsearch.yml`中`bootstrap.memory_lock: true`，以保持JVM锁定内存，保证ES的性能。

#### 4. GC设置原则：

- a. 保持GC的现有设置，默认设置为：Concurrent-Mark and Sweep (CMS)，别换成G1GC，因为目前G1还有很多BUG。
- b. 保持线程池的现有设置，目前ES的线程池较1.X有了较多优化设置，保持现状即可；默认线程池大小等于CPU核心数。如果一定要改，按公式 $((\text{CPU核心数} * 3) / 2) + 1$ 设置；不能超过CPU核心数的2倍；但是不建议修改默认配置，否则会对CPU造成硬伤。

### 12.1.4 集群分片设置：

ES一旦创建好索引后，就无法调整分片的设置，而在ES中，一个分片实际上对应一个lucene索引，而lucene索引的读写会占用很多的系统资源，因此，分片数不能设置过大；所以，在创建索引时，合理配置分片数是非常重要的。一般来说，我们遵循一些原则：

1. 控制每个分片占用的硬盘容量不超过ES的最大JVM的堆空间设置（一般设置不超过32G，参加上文的JVM设置原则），因此，如果索引的总容量在500G左右，那分片大小在16个左右即可；当然，最好同时考虑原则2。
2. 考虑一下node数量，一般一个节点有时候就是一台物理机，如果分片数过多，大大超过了节点数，很可能会导致一个节点上存在多个分片，一旦该节点故障，即使保持了1个以上的副本，同样有可能会造成数据丢失，集群无法恢复。所以，一般都设置分片数不超过节点数的3倍。

### 12.1.5 Mapping建模

1. 尽量避免使用nested或 parent/child，能不用就不用；nested query慢，parent/child query 更慢，比nested query慢上百倍；因此能在mapping设计阶段搞定的（大宽表设计或采用比较smart的数据结构），就不要用父子关系的mapping。
2. 如果一定要使用nested fields，保证nested fields字段不能过多，目前ES默认限制是50。参考：  
index.mapping.nested\_fields.limit : 50  
因为针对1个document, 每一个nested field, 都会生成一个独立的document, 这将使Doc数量剧增，影响查询效率，尤其是JOIN的效率。
3. 避免使用动态值作字段(key), 动态递增的mapping，会导致集群崩溃；同样，也需要控制字段的数量，业务中不使用的字段，就不要索引。控制索引的字段数量、mapping深度、索引字段的类型，对于ES的性能优化是重中之重。以下是ES关于字段数、mapping深度的一些默认设置：

index.mapping.nested\_objects.limit:10000

index.mapping.total\_fields.limit:1000

index.mapping.depth.limit: 20

### 12.1.6 索引优化设置：

1. 设置refresh\_interval 为-1，同时设置number\_of\_replicas 为0，通过关闭refresh间隔周期，同时不设置副本来提高写性能。
2. 修改index\_buffer\_size 的设置，可以设置成百分数，也可设置成具体的大小，大小可根据集群的规模做不同的设置测试。  
indices.memory.index\_buffer\_size : 10% (默认)  
indices.memory.min\_index\_buffer\_size : 48mb (默认)  
indices.memory.max\_index\_buffer\_size
3. 修改translog相关的设置：
  - a. 控制数据从内存到硬盘的操作频率，以减少硬盘IO。可将sync\_interval的时间设置大一些。  
index.translog.sync\_interval : 5s(默认)。
  - b. 控制tranlog数据块的大小，达到threshold大小时，才会flush到lucene索引文件。  
index.translog.flush\_threshold\_size : 512mb(默认)
4. id字段的使用，应尽可能避免自定义id，以避免针对ID的版本管理；建议使用ES的默认ID生成策略或使用数字类型ID做为主键。
5. all 字段及source字段的使用，应该注意场景和需要，all 字段包含了所有的索引字段，方便做全文检索，如果无此需求，可以禁用；source存储了原始的document内容，如果没有获取原始文档数据的需求，可通过设置includes、excludes 属性来定义放入\_source的字段。
6. 合理的配置使用index属性，analyzed 和not\_analyzed，根据业务需求来控制字段是否分词或不分词。只有groupby需求的字段，配置时就设置成not\_analyzed, 以提高查询或聚类的效率。

### 12.1.7 查询优化：

1. query\_string 或 multi\_match的查询字段越多，查询越慢。可以在mapping阶段，利用copy\_to属性将多字段的值索引到一个新字段，multi\_match时，用新的字段查询。
2. 日期字段的查询，尤其是用now 的查询实际上是不存在缓存的，因此，可以从业务的角度来考虑是否一定要用now, 毕竟利用query cache 是能够大大提高查询效率的。

3. 查询结果集的大小不能随意设置成大得离谱的值，如query.setSize不能设置成 Integer.MAX\_VALUE，因为ES内部需要建立一个数据结构来放指定大小的结果集数据。
4. 尽量避免使用script，万不得已需要使用的话，选择painless & expressions 引擎。一旦使用script查询，一定要注意控制返回，千万不要有死循环（如下错误的例子），因为ES没有脚本运行的超时控制，只要当前的脚本没执行完，该查询会一直阻塞。

如：{

```
"script_fields" : {  
  "test1" : {  
    "lang" : "groovy",  
    "script" : "while ( true ) {print 'don't use script'}"  
  }  
}
```

5. 避免层级过深的聚合查询，层级过深的group by，会导致内存、CPU消耗，建议在服务层通过程序来组装业务，也可以通过pipeline的方式来优化。
6. 复用预索引数据方式来提高AGG性能：如通过 terms aggregations 替代 range aggregations，如要根据年龄来分组，分组目标是：少年（14岁以下）青年（14-28）中年（29-50）老年（51以上），可以在索引的时候设置一个age\_group字段，预先将数据进行分类。从而不用按age来做range aggregations, 通过age\_group字段就可以了。
7. Cache的设置及使用：
  - a) QueryCache: ES查询的时候，使用filter查询会使用query cache, 如果业务场景中的过滤查询比较多，建议将querycache设置大一些，以提高查询速度。indices.queries.cache.size：10%（默认），可设置成百分比，也可设置成具体值，如256mb。当然也可以禁用查询缓存（默认是开启），通过index.queries.cache.enabled：false设置。
  - b) FieldDataCache: 在聚类或排序时，field data cache会使用频繁，因此，设置字段数据缓存的大小，在聚类或排序场景较多的情形下很有必要，可通过indices fielddata.cache.size：30% 或具体值10GB来设置。但是如果场景或数据变更比较频繁，设置cache并不是好的做法，因为缓存加载的开销也是特别大的。
  - c) ShardRequestCache: 查询请求发起后，每个分片会将结果返回给协调节点(Coordinating Node), 由协调节点将结果整合。

如果有需求，可以设置开启; 通过设置index.requests.cache.enable: true来开启。

不过，shard request cache只缓存hits.total, aggregations, suggestions类型的数据，并不会缓存hits的内容。也可以通过设置indices.requests.cache.size: 1%（默认）来控制缓存空间大小。

### 12.1.8 集群运维及恢复

## 12.2、ES优化之关于创建和删除

### ——关于创建

调大系统的"最大打开文件数",建议32K甚至是64K ulimit -a (查看) ulimit -n 32000(设置) 修改配置文件调整ES的JVM内存大小 1：修改bin/elasticsearch.in.sh中ES\_MIN\_MEM和ES\_MAX\_MEM的大小，建议设置一样大，避免频繁的分配内存，根据服务器内存大小，一般分配60%左右(默认 256M) 2：如果使用searchwrapper插件启动es的话则修改[过时，在es1.x中有用] bin/service/elasticsearch.conf(默认1024M，2.x以后不用考虑) 设置mlockall来锁定进程

的物理内存地址 避免交换 ( swapped ) 来提高性能 修改文件conf/elasticsearch.yml bootstrap.mlockall: true 分片多的话, 可以提升建立索引的能力, 5-20个比较合适。

如果分片数过少或过多, 都会导致检索比较慢。分片数过多会导致检索时打开比较多的文件, 另外也会导致多台服务器之间通讯。而分片数过少会导至单个分片索引过大, 所以检索速度慢。建议单个分片最多存储20G左右的索引数据, 所以, 分片数量=数据总量/20G 副本多的话, 可以提升搜索的能力, 但是如果设置很多副本的话也会对服务器造成额外的压力, 因为需要同步数据。所以建议设置2-3个即可。要定时对索引进行优化, 不然segment越多, 查询的性能就越差 索引量不是很大的话情况下可以将segment设置为1 curl -XPOST '[http://localhost:9200/\\_crxy/\\_optimize?max\\_num\\_segments=1](http://localhost:9200/_crxy/_optimize?max_num_segments=1)' java代码:

```
client.admin().indices().prepareOptimize("bigdata").setMaxNumSegments(1).get();
```

### ——关于删除

删除文档: 在Lucene中删除文档, 数据不会马上在硬盘上除去, 而是在lucene索引中产生一个.del的文件, 而在检索过程中这部分数据也会参与检索, lucene在检索过程会判断是否删除了, 如果删除了再过滤掉。这样也会降低检索效率。所以可以执行清除删除文档 curl -XPOST '[http://localhost:9200/bigdata/\\_optimize?only\\_expunge\\_deletes=true](http://localhost:9200/bigdata/_optimize?only_expunge_deletes=true)' client.admin().indices().prepareOptimize("bigdata").setOnlyExpungeDeletes(true).get(); 如果在项目开始的时候需要批量入库大量数据的话, 建议将副本数设置为0。因为es在索引数据的时候, 如果有副本存在, 数据也会马上同步到副本中, 这样会对es增加压力。待索引完成后将副本按需要改回来。这样可以提高索引效率。

### 12.3、ES优化之关于配置

去掉mapping中all域, Index中默认会有all的域, (相当于solr配置文件中的拷贝字段text), 这个会给查询带来方便, 但是会增加索引时间和索引尺寸 "\_all":{"enabled":"false"} log输出的水平默认为trace, 即查询超过500ms即为慢查询, 就要打印日志, 造成cpu和mem, io负载很高。把log输出水平改为info, 可以减轻服务器的压力。修改 ES\_HOME/conf/logging.yaml文件 或者修改ES\_HOME/conf/elasticsearch.yaml