ElasticSearch分布式搜索原理解析

# 数据检索问题

## 大规模数据如何检索？

当系统数据量上了10亿、100亿条的时候，我们在做系统架构的时候通常会从以下角度去考虑：

1）用什么数据库好？(MySQL、sybase、Oracle、达梦、神通、MongoDB、Hbase…)

2）如何解决单点故障；(lvs、F5、A10、Zookeep、MQ)

3）如何保证数据安全性；(热备、冷备、异地多活)

4）如何解决检索难题；(数据库代理中间件：mysql-proxy、Cobar、MaxScale等;)

5）如何解决统计分析问题；(离线、近实时)

## 传统数据库的应对解决方案？

对于关系型数据，我们通常采用以下或类似架构去解决查询瓶颈和写入瓶颈：

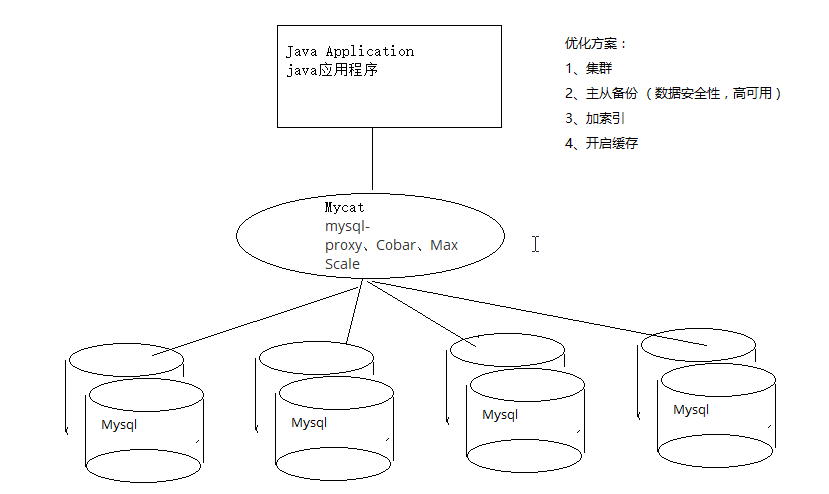
解决要点：

1）通过主从备份解决数据安全性问题；

2）通过数据库代理中间件心跳监测，解决单点故障问题；

3）通过代理中间件将查询语句分发到各个slave节点进行查询，并汇总结果

4）通过分表分库解决读写效率问题



## 非关系型数据库的解决方案？

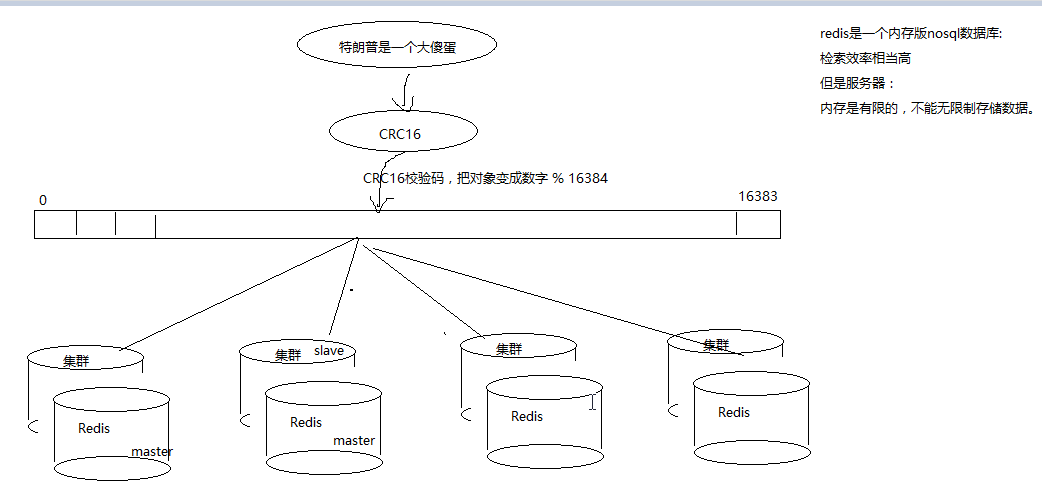
对于Nosql数据库，以redis为例，其它原理类似：

解决要点：

1）通过副本备份保证数据安全性；

2）通过节点竞选机制解决单点问题；

3）先从配置库检索分片信息，然后将请求分发到各个节点，最后由路由节点合并汇总结果



## 完全把数据放入内存怎么样？

完全把数据放在内存中是不可靠的，实际上也不太现实，当我们的数据达到PB级别时，按照每个节点96G内存计算，

在内存完全装满的数据情况下，我们需要的机器是：1PB=1024T=1048576G

节点数=1048576/96=10922个

实际上，考虑到数据备份，节点数往往在2.5万台左右。成本巨大决定了其不现实！

从前面我们了解到，把数据放在内存也好，不放在内存也好，都不能完完全全解决问题。

全部放在内存速度问题是解决了，但成本问题上来了。 为解决以上问题，从源头着手分析，通常会从以下方式来寻找方法：

1、存储数据时按有序存储；

2、将数据和索引分离；

3、压缩数据；

全文检索技术： --- 检索的数据是一些菲关系型数据，word,pdf,html

# 全文检索技术

## 什么是全文检索？

什么叫做全文检索呢？这要从我们生活中的数据说起。

我们生活中的数据总体分为两种：**结构化数据和非结构化数据**。

* **结构化数据**：指具有固定格式或有限长度的数据，如数据库，元数据等。

**> 对结构化数据的搜索：** 如对数据库的搜索，用SQL语句。再如对元数据的搜索，如利用windows搜索对文件名，类型，修改时间进行搜索等。

* **非结构化数据**：指不定长或无固定格式的数据，如 互联网数据、邮件，word文档等。

**> 对非结构化数据的搜索：** 如用Google和百度可以搜索大量内容数据。

对非结构化数据顺序扫描很慢，对结构化数据的搜索却相对较快，那么把我们的非结构化数据想办法弄得有一定结构不就行了吗？这就是全文检索的基本思路，也即将非结构化数据中的一部分信息提取出来，重新组织，使其变得有一定结构，然后对此有一定结构的数据进行搜索，从而达到搜索相对较快的目的。这部分从非结构化数据中提取出的然后重新组织的信息，我们称之**索引** 。

**非结构化数据又一种叫法叫全文数据。**

按照数据的分类，搜索也分为两种：

对非结构化数据也即全文数据的搜索主要有两种方法：**顺序扫描法和反向索引法**。

* **顺序扫描法**：所谓顺序扫描法，就是顺序扫描每个文档内容，看看是否有要搜索的关键字，实现查找文档的功能，也就是根据文档找词。
* **反向索引法**：所谓反向索引，就是提前将搜索的关键字建成索引，然后再根据索引查找文档，也就是根据词找文档。

**这种先建立索引，再对索引进行搜索文档的过程就叫全文检索(Full-text Search)**。

## 全文检索场景

* 搜索引擎
* 站内搜索
* 系统文件搜索

## 实时搜索与传统搜索

通常来说，**传统搜索**都是一些“静态”的搜索，即用户搜索的只是从信息库里边筛选出来的信息。而百度推出的**实时搜索功能**，改变了传统意义上的静态搜索模式，用户对于搜索的结果是实时变化的。

举个例子，用户在搜索“华山”、“峨眉山”等景点时，实时观看各地景区画面。以华山景区为例，当用户在搜索框中输入“华山”时，点击右侧“实时直播——华山”，即可实时观看华山靓丽风景，并能在华山长空栈道、北峰顶、观日台三个视角之间切换。同时，该直播引入广受年轻人欢迎的“弹幕”模式，用户在观看风景时可以同时发表评论，甚至进行聊天互动。

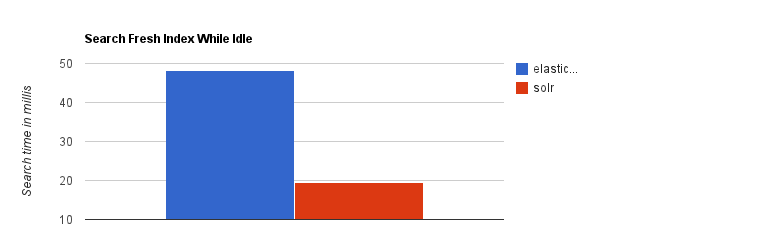
## 全文检索相关技术

1. Lucene：如果使用该技术实现，需要对Lucene的API和底层原理非常了解，而且需要编写大量的Java代码。
2. Solr：使用java实现的一个web应用，可以使用rest方式的http请求，进行远程API的调用。
3. ElasticSearch(ES)：可以使用rest方式的http请求，进行远程API的调用。

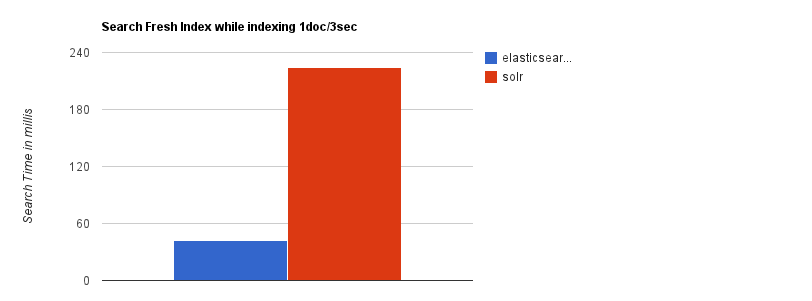
## Solr和ES的比较

### ElasticSearch vs Solr 检索速度

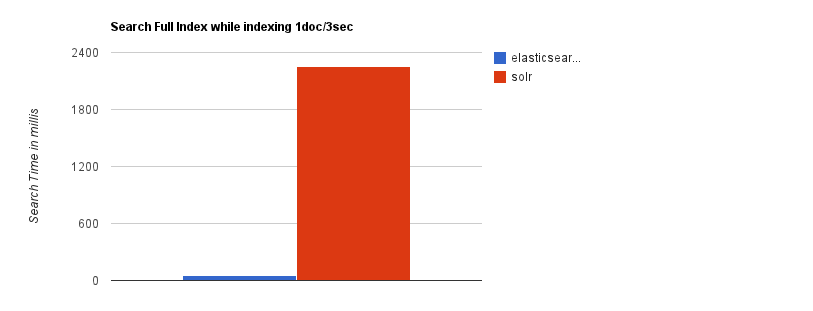
* 当单纯的对**已有数据**进行搜索时，Solr更快。



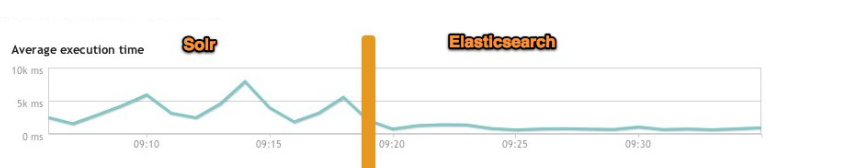
* **当实时建立索引时, Solr会产生io阻塞，查询性能较差, Elasticsearch具有明显的优势。**



* **随着数据量的增加，Solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化**。



* **大型互联网公司，实际生产环境测试，将搜索引擎从Solr转到Elasticsearch以后的平均查询速度有了50倍的提升。**



### Elasticsearch 与 Solr 的比较总结

* 二者安装都很简单；
* Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能;
* Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式；
* Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供；
* Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch。

**最终的结论：**

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。

# 全文检索的流程分析

## 什么是索引

有人可能会说，对非结构化数据顺序扫描很慢，对结构化数据的搜索却相对较快（由于结构化数据有一定的结构可以采取一定的搜索算法加快速度），那么把我们的非结构化数据想办法弄得有一定结构不就行了吗？

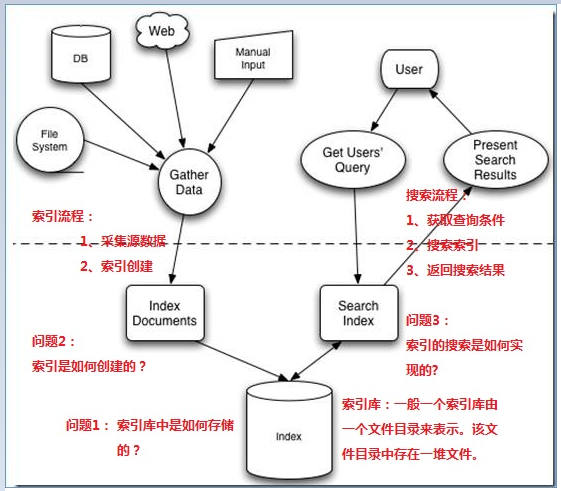
这种想法很天然，却构成了全文检索的基本思路，也即将非结构化数据中的一部分信息提取出来，重新组织，使其变得有一定结构，然后对此有一定结构的数据进行搜索，从而达到搜索相对较快的目的。

这部分从非结构化数据中提取出的然后重新组织的信息，我们称之**索引**

这种说法比较抽象，举几个例子就很容易明白，比如字典，字典的拼音表和部首检字表就相当于字典的索引，对每一个字的解释是非结构化的，如果字典没有音节表和 部首检字表，在茫茫辞海中找一个字只能顺序扫描。然而字的某些信息可以提取出来进行结构化处理，比如读音，就比较结构化，分声母和韵母，分别只有几种可以 一一列举，于是将读音拿出来按一定的顺序排列，每一项读音都指向此字的详细解释的页数。我们搜索时按结构化的拼音搜到读音，然后按其指向的页数，便可找到 我们的非结构化数据——也即对字的解释

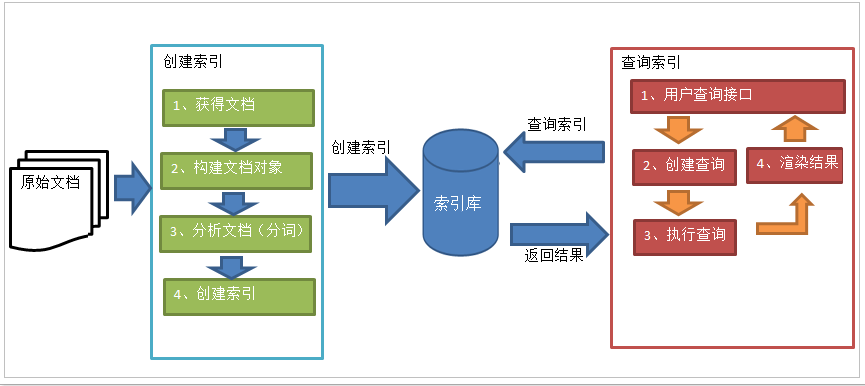
## 流程总览

搜索基本的流程实现：



**全文检索的流程分为两大流程：索引创建、搜索索引**

* 索引创建：将现实世界中所有的结构化和非结构化数据提取信息，创建索引的过程。
* 搜索索引：就是得到用户的查询请求，搜索创建的索引，然后返回结果的过程。



**想搞清楚全文检索，必须要搞清楚下面三个问题：**

**1. 索引库里面究竟存些什么？(Index)**

**2. 如何创建索引？(Indexing)**

**3. 如何对索引进行搜索？(Search)**

## 原始内容

**原始内容是指要索引和搜索的内容**。

原始内容包括互联网上的网页、数据库中的数据、磁盘上的文件等。

## 获得文档

也就是采集数据，从互联网上、数据库、文件系统中等获取需要搜索的原始信息，这个过程就是**信息采集。**

**采集数据的目的是为了将原始内容存储到Document对象中**。

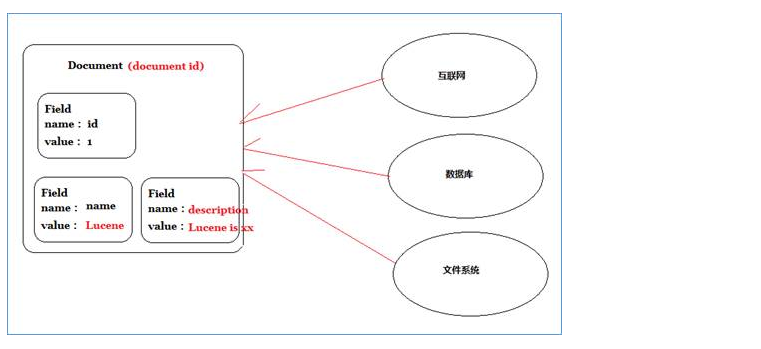
**如何采集数据？**

1. 对于互联网上网页，可以使用工具将网页抓取到本地生成html文件。
2. 数据库中的数据，可以直接连接数据库读取表中的数据。
3. 文件系统中的某个文件，可以通过I/O操作读取文件的内容。

在Internet上采集信息的软件通常称为爬虫或蜘蛛，也称为网络机器人，爬虫访问互联网上的每一个网页，将获取到的网页内容存储起来。

## 创建文档对象

**创建文档的目的是统一数据格式（Document），方便文档分析。**

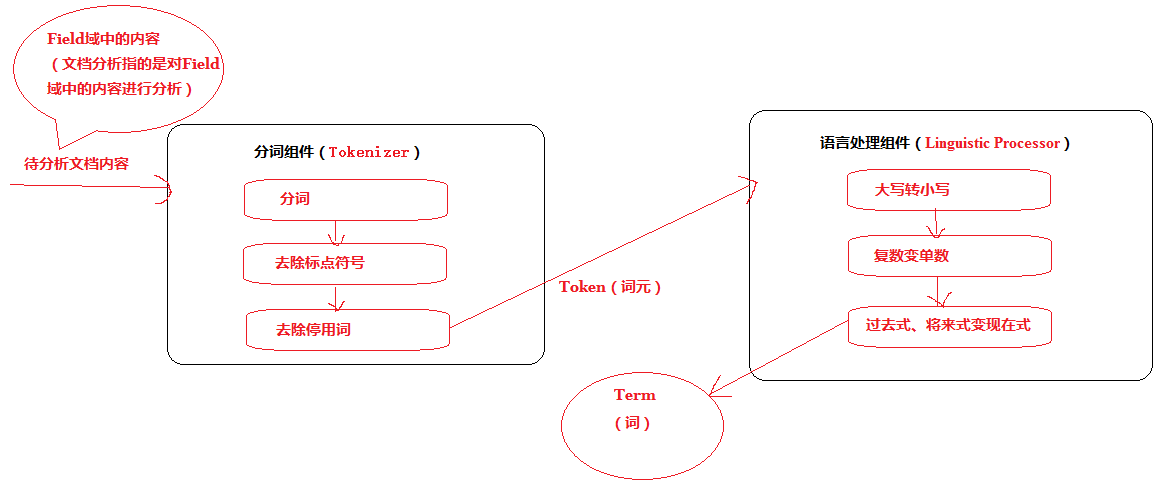


**说明：**

1. 一个Document文档中包括多个域（Field），域（Field）中存储内容。
2. 这里我们可以将数据库中一条记录当成一个Document，一列当成一个Field

## 分析文档（重点）

**分析文档主要是对Field域进行分析，分析文档的目的是为了索引。**



**说明：**分析文档主要通过**分词组件（Tokenizer）**和**语言处理组件（Linguistic Processor）**完成

### 分词组件

**分词组件工作流程（此过程称之为Tokenize）**

1. 将Field域中的内容进行分词（不同语言有不同的分词规则）
2. 去除标点符号。
3. 去除停用词（stop word）。

**经过分词（Tokenize）之后得到的结果成为词元（Token）。**

**所谓停词(Stop word)**就是一种语言中最普通的一些单词，由于没有特别的意义，因而大多数情况下不能成为搜索的关键词，因而创建索引时，这种词会被去掉而减少索引的大小。

英语中停词(Stop word)如：“the”,“a”，“this”等。

**对于每一种语言的分词组件(Tokenizer)，都有一个停词(stop word)集合**。

**示例(Document1的Field域和Document2的Field域是同名的)：**

* Document1的Field域：

 Students should be allowed to go out with their   friends, but not allowed to drink beer.

* Document2的Field域：

 My friend Jerry went to school to see his students but   found them drunk which is not allowed.

* 在我们的例子中，便得到以下**词元(Token)**：

 “Students”，“allowed”，“go”，“their”，“friends”，“allowed”，“drink”，“beer”，“My”，“friend”，“Jerry”，“went”，“school”，“see”，“his”，“students”，“found”，“them”，“drunk”，“allowed”。

**将得到的词元(Token)传给语言处理组件(Linguistic Processor)**

### 语言处理组件

*语言处理组件****(linguistic processor)****主要是对得到的词元****(Token)****做一些同语言相关的处理。*

**对于英语，语言处理组件(Linguistic Processor)一般做以下几点：**

1. 变为小写(Lowercase)。
2. 将单词缩减为词根形式，如“cars”到“car”等。这种操作称为：stemming。
3. 将单词转变为词根形式，如“drove”到“drive”等。这种操作称为：lemmatization。

**语言处理组件(linguistic processor)的结果称为词(Term)。Term是索引库的最小单位。**

* 在我们的例子中，经过语言处理，得到的词**(Term)**如下：

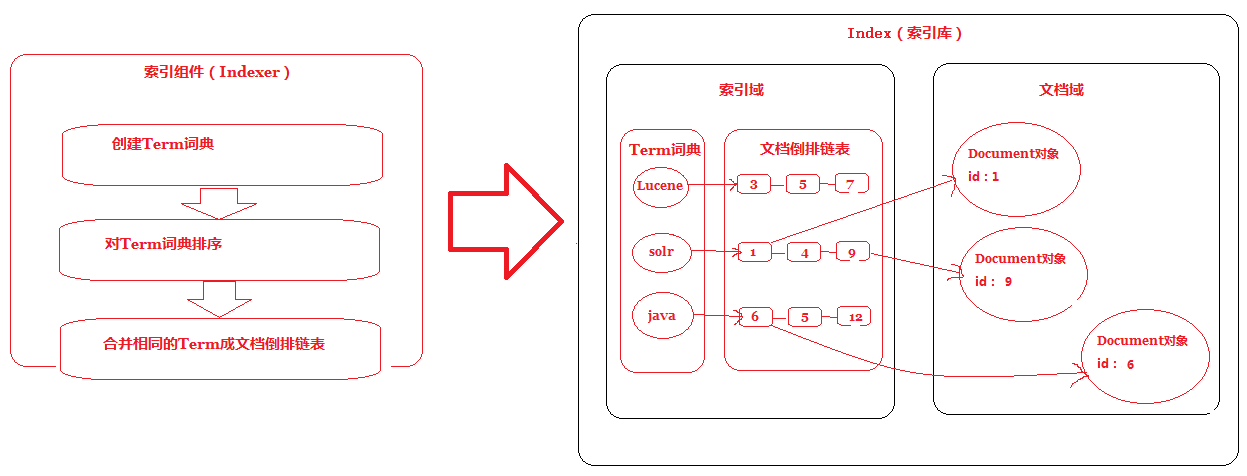
 “student”，“allow”，“go”，“their”，“friend”，“allow”，“drink”，“beer”，“my”，“friend”，“jerry”，“go”，“school”，“see”，“his”，“student”，“find”，“them”，“drink”，“allow”。

也正是因为有语言处理的步骤，才能使搜索drove，而drive也能被搜索出来。

## 索引文档

### 索引库创建

**索引的目的是为了搜索**。



**说明：将得到的词(Term)传给索引组件(Indexer)，索引组件(Indexer)主要做以下几件事情：**

### 创建Term字典

在我们的例子中字典如下：

| **Term** | **Document ID** |
| --- | --- |
| Student | 1 |
| Allow | 1 |
| Go | 1 |
| Their | 1 |
| Friend | 1 |
| Allow | 1 |
| Drink | 1 |
| Beer | 1 |
| My | 2 |
| Friend | 2 |
| Jerry | 2 |
| Go | 2 |
| School | 2 |
| See | 2 |
| His | 2 |
| Student | 2 |
| Find | 2 |
| Them | 2 |
| Drink | 2 |
| Allow | 2 |

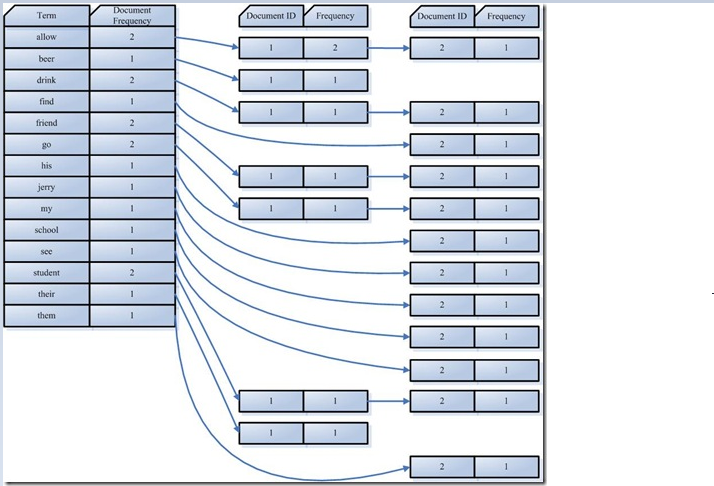
### 排序Term字典

对字典按字母顺序进行排序

| **Term** | **Document ID** |
| --- | --- |
| Allow | 1 |
| Allow | 1 |
| Allow | 2 |
| Beer | 1 |
| Drink | 1 |
| Drink | 2 |
| Find | 2 |
| Friend | 1 |
| Friend | 2 |
| Go | 1 |
| Go | 2 |
| His | 2 |
| Jerry | 2 |
| My | 2 |
| School | 2 |
| See | 2 |
| Student | 1 |
| Student | 2 |
| Their | 1 |
| Them | 2 |

### 合并Term字典

合并相同的词(Term)成为文档倒排(Posting List)链表

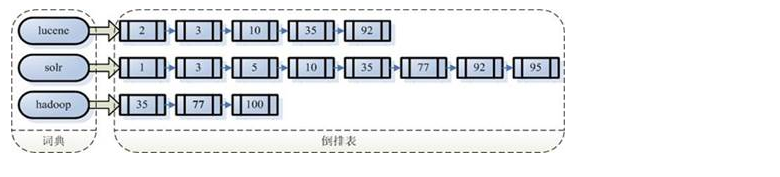


在此表中，有几个定义：

* **Document Frequency** 即文档频次，表示总共有多少文件包含此词(Term)。
* **Frequency** 即词频率，表示此文件中包含了几个此词(Term)。

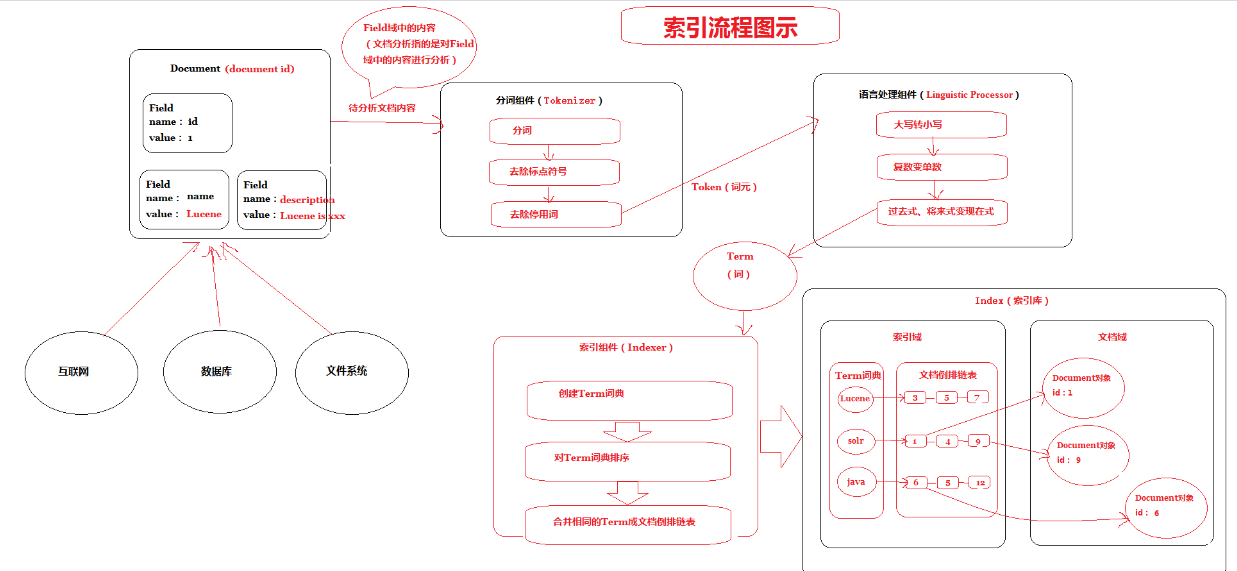
**到此为止，索引已经创建好了。**

**最终的索引结构是一种倒排索引结构也叫反向索引结构，包括索引和文档两部分，索引即词汇表，它的规模较小，而文档集合较大。**

**倒排索引结构**是根据内容（词汇）找文档，如下图： 

## 创建索引流程

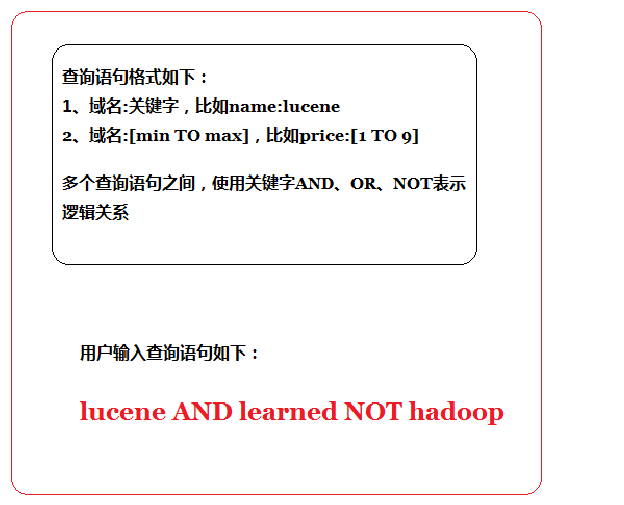
分词及检索的详细的流程：



**一次索引，多次使用。**

## 搜索索引流程

### 图1：查询语句



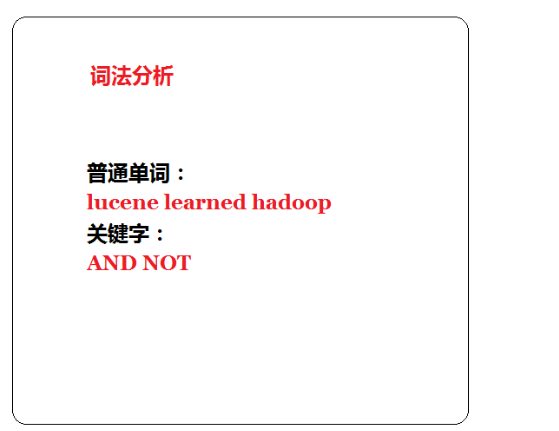
### 图2：执行搜索

**第一步：对查询语句进行词法分析、语法分析及语言处理。**

**1、词法分析**

如上述例子中，经过词法分析，得到单词有lucene，learned，hadoop, 关键字有AND, NOT。

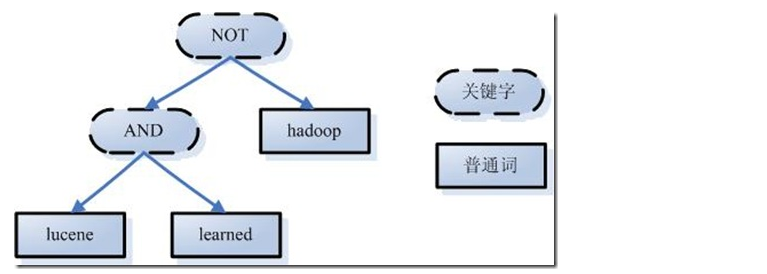
***注意：关键字必须大写，否则就作为普通单词处理。***



**2、语法分析**

如果发现查询语句不满足语法规则，则会报错。如lucene NOT AND learned，则会出错。

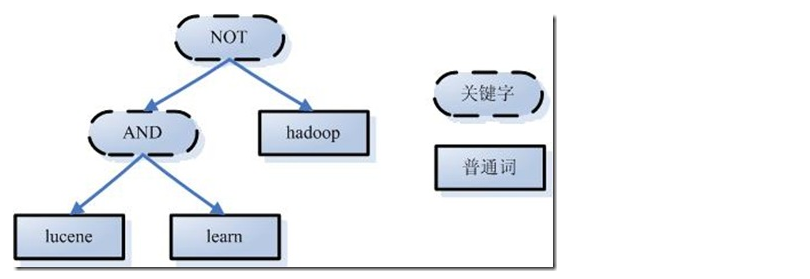
如上述例子，lucene AND learned NOT hadoop形成的语法树如下：



**3、语言处理**

如learned变成learn等。

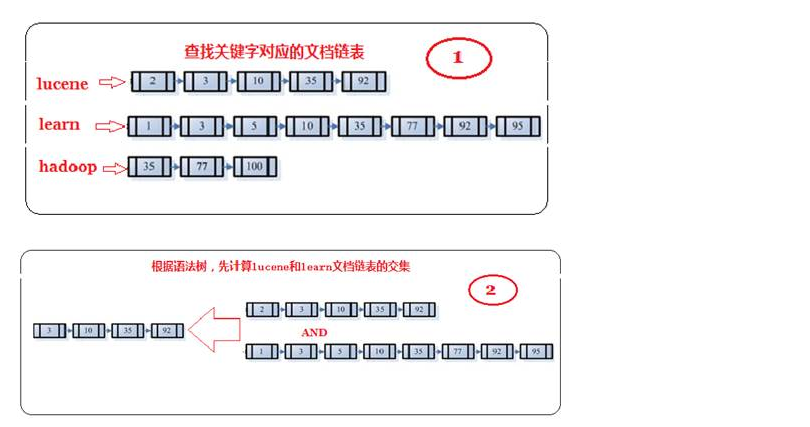
经过第二步，我们得到一棵经过语言处理的语法树。



**第二步：搜索索引，得到符号语法树的文档。**

1、 首先，在反向索引表中，分别找出包含lucene，learn，hadoop的文档链表。

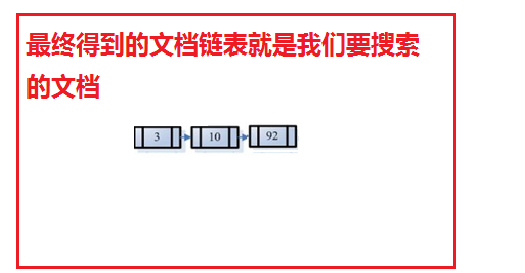
2、 其次，对包含lucene，learn的链表进行合并操作，得到既包含lucene又包含learn的文档链表。



3、 然后，将此链表与hadoop的文档链表进行差操作，去除包含hadoop的文档，从而得到既包含lucene又包含learn而且不包含hadoop的文档链表。



4、 此文档链表就是我们要找的文档。



**第三步：根据得到的文档和查询语句的相关性，对结果进行排序。**

相关度自然打分（权重越高分越高）：

tf越高、权重越高

df越高、权重越低

人为影响分数：

设置Boost值（**加权值**）

## Lucene相关度排序

### 什么是相关度排序

**相关度排序是查询结果按照与查询关键字的相关性进行排序，越相关的越靠前**。比如搜索“Lucene”关键字，与该关键字最相关的文章应该排在前边。

### 相关度打分

Lucene对查询关键字和索引文档的相关度进行打分，得分高的就排在前边。

如何打分呢？Lucene是在用户进行检索时实时根据搜索的关键字计算出来的，分两步：

1. 计算出词（Term）的权重
2. 根据词的权重值，计算文档相关度得分。

**什么是词的权重？**

通过索引部分的学习，明确索引的最小单位是一个Term(索引词典中的一个词)。搜索也是从索引域中查询Term，再根据Term找到文档。**Term对文档的重要性称为权重**，影响Term权重有两个因素：

* **Term Frequency (tf)：**

指此Term在此文档中出现了多少次。**tf 越大说明越重要**。

词(Term)在文档中出现的次数越多，说明此词(Term)对该文档越重要，如“Lucene”这个词，在文档中出现的次数很多，说明该文档主要就是讲Lucene技术的。

* **Document Frequency (df)：**

指有多少文档包含此Term。**df 越大说明越不重要**。

比如，在一篇英语文档中，this出现的次数更多，就说明越重要吗？不是的，有越多的文档包含此词(Term), 说明此词(Term)太普通，不足以区分这些文档，因而重要性越低。

### 设置boost值影响相关度排序

**boost是一个加权值（默认加权值为1.0f），它可以影响权重的计算。**在索引时对某个文档中的field设置加权值，设置越高，在搜索时匹配到这个文档就可能排在前边。

## Lucene的Field域

### Field属性

Field是文档中的域，包括Field名和Field值两部分，一个文档可以包括多个Field，Document只是Field的一个承载体，Field值即为要索引的内容，也是要搜索的内容。

* **是否分词(tokenized)**
  + 是：作分词处理，即将Field值进行分词，分词的目的是为了索引。

比如：商品名称、商品描述等，这些内容用户要输入关键字搜索，由于搜索的内容格式大、内容多需要分词后将语汇单元建立索引

* + 否：不作分词处理

比如：商品id、订单号、身份证号等

* **是否索引(indexed)**
  + 是：进行索引。将Field分词后的词或整个Field值进行索引，存储到索引域，索引的目的是为了搜索。

比如：商品名称、商品描述分析后进行索引，订单号、身份证号不用分词但也要索引，这些将来都要作为查询条件。

* + 否：不索引。

比如：图片路径、文件路径等，不用作为查询条件的不用索引

* **是否存储(stored)**
  + 是：将Field值存储在文档域中，存储在文档域中的Field才可以从Document中获取。

比如：商品名称、订单号，凡是将来要从Document中获取的Field都要存储。

* + 否：不存储Field值

比如：**商品描述**，内容较大不用存储。如果要向用户展示商品描述可以从系统的关系数据库中获取。

### Field常用类型

下边列出了开发中常用 的Filed类型，注意Field的属性，根据需求选择：

| **Field类** | **数据类型** | **Analyzed 是否分词** | **Indexed 是否索引** | **Stored 是否存储** | **说明** |
| --- | --- | --- | --- | --- | --- |
| StringField(FieldName, FieldValue,Store.YES)) | 字符串 | N | Y | Y或N | 这个Field用来构建一个字符串Field，但是不会进行分词，会将整个串存储在索引中，比如(订单号,身份证号等) 是否存储在文档中用Store.YES或Store.NO决定 |
| LongField(FieldName, FieldValue,Store.YES) | Long型 | Y | Y | Y或N | 这个Field用来构建一个Long数字型Field，进行分词和索引，比如(价格) 是否存储在文档中用Store.YES或Store.NO决定 |
| StoredField(FieldName, FieldValue) | 重载方法，支持多种类型 | N | N | Y | 这个Field用来构建不同类型Field 不分析，不索引，但要Field存储在文档中 |
| TextField(FieldName, FieldValue, Store.NO) 或 TextField(FieldName, reader) | 字符串 或 流 | Y | Y | Y或N | 如果是一个Reader, lucene猜测内容比较多,会采用Unstored的策略. |

### Field设计

**Field域如何设计，取决于需求，比如搜索条件有哪些？显示结果有哪些？**

* 商品id：

是否分词：不用分词，因为不会根据商品id来搜索商品

是否索引：不索引，因为不需要根据商品ID进行搜索

是否存储：要存储，因为查询结果页面需要使用id这个值。

* 商品名称：

是否分词：要分词，因为要根据商品名称的关键词搜索。

是否索引：要索引。

是否存储：要存储。

* 商品价格：

是否分词：要分词，lucene对数字型的值只要有搜索需求的都要分词和索引，因为lucene对数字型的内容要特殊分词处理，需要分词和索引。

是否索引：要索引

是否存储：要存储

* 商品图片地址：

是否分词：不分词

是否索引：不索引

是否存储：要存储

* 商品描述：

是否分词：要分词

是否索引：要索引

是否存储：因为商品描述内容量大，不在查询结果页面直接显示，不存储。

**常见问题：**

不存储是指不在lucene的索引域中记录，目的是为了节省lucene的索引文件空间。

**如果要在详情页面显示描述，解决方案：**

从lucene中取出商品的id，根据商品的id查询关系数据库（MySQL）中item表得到描述信息。

# ES基本概念

## ES简介

ES=elaticsearch简写， Elasticsearch是一个开源的高扩展的分布式全文检索引擎，它可以近乎实时的存储、检索数据；本身扩展性很好，可以扩展到上百台服务器，处理PB级别的数据。

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

Elasticsearch 是一个分布式可扩展的实时搜索和分析引擎,一个建立在全文搜索引擎 Apache Lucene(TM) 基础上的搜索引擎.当然 Elasticsearch 并不仅仅是 Lucene 那么简单，它不仅包括了全文搜索功能，还可以进行以下工作:

* 分布式实时文件存储，并将每一个字段都编入索引，使其可以被搜索。
* 实时分析的分布式搜索引擎。
* 可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。

Elasticsearch是面向文档型数据库，一条数据在这里就是一个文档，用JSON作为文档序列化的格式，比如下面这条用户数据：

 {  
    "name" :     "John",  
    "sex" :     "Male",  
    "age" :     25,  
    "birthDate": "1990/05/01",  
    "about" :   "I love to go rock climbing",  
    "interests": [ "sports", "music" ]  
 }

实际项目开发实战中，几乎每个系统都会有一个搜索的功能，当数据达到很大且搜索要做到一定程度时，维护和扩展难度就会越来越高，并且在全文检索的速度上、结果内容的推荐、分析以及统计聚合方面也很难达到我们预期效果。

这时候Elasticsearch就出现了。Elasticsearch能建立全文索引（把文本中的内容拆分成若干关键词，然后根据关键词创建索引。查询时根据关键词查询索引，最终找到包含关键词的文章），它将数据和索引分离，把索引分片，分布式的保存到不同节点，节点可以扩展到上百个，能实时检索、处理PB级别的结构化或非结构化数据。同时分片可以进行副本备份保证数据的可靠性，各分片副本协同工作也大大提高检索性能，且通过简单的RESTful API让全文搜索变得高效简单。

近年ElasticSearch发展迅猛，已经超越了其最初的纯搜索引擎的角色，现在已经增加了数据聚合分析（aggregation）和可视化的特性，如果你有数百万的文档需要通过关键词进行定位、分析统计时，ElasticSearch肯定是最佳选择。

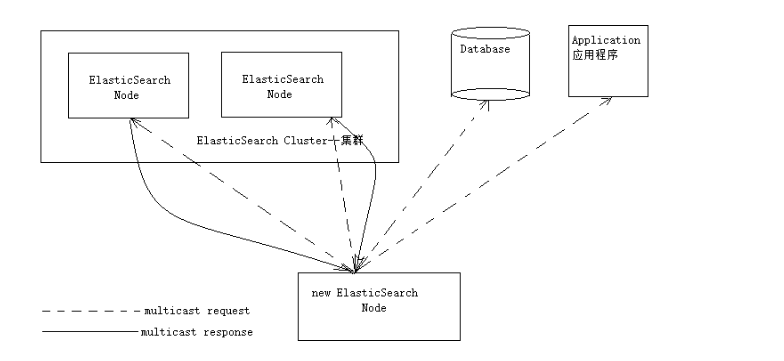
## Lucene和ES关系

1）Lucene只是一个API库。想要使用它，你必须使用Java来作为开发语言并将其直接集成到你的应用中，更糟糕的是，Lucene非常复杂，你需要深入了解检索的相关知识来理解它是如何工作的。

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

## ES工作原理

当ElasticSearch的节点启动后，它会利用多播(multicast)(或者单播，如果用户更改了配置)寻找集群中的其它节点，并与之建立连接。这个过程如下图所示：



## ES核心概念

### Cluster：集群

ES可以作为一个独立的单个搜索服务器。不过，为了处理大型数据集，实现容错和高可用性，ES可以运行在许多互相合作的服务器上。这些服务器的集合称为集群。

### Node：节点

形成集群的每个服务器称为节点。

### Shard：分片

当有大量的文档时，由于内存的限制、磁盘处理能力不足、无法足够快的响应客户端的请求等，一个节点可能不够。这种情况下，数据可以分为较小的分片。每个分片放到不同的服务器上。 当你查询的索引分布在多个分片上时，ES会把查询发送给每个相关的分片，并将结果组合在一起，而应用程序并不知道分片的存在。即：这个过程对用户来说是透明的。

### Replia：副本

为提高查询吞吐量或实现高可用性，可以使用分片副本。 副本是一个分片的精确复制，每个分片可以有零个或多个副本。ES中可以有许多相同的分片，其中之一被选择更改索引操作，这种特殊的分片称为主分片。 当主分片丢失时，如：该分片所在的数据不可用时，集群将副本提升为新的主分片。

### 全文检索

全文检索就是对一篇文章进行索引，可以根据关键字搜索，类似于mysql里的like语句。 全文索引就是把内容根据词的意义进行分词，然后分别创建索引，例如”你们的激情是因为什么事情来的” 可能会被分词成：“你们“，”激情“，“什么事情“，”来“ 等token，这样当你搜索“你们” 或者 “激情” 都会把这句搜出来。

## ELK是什么？

ELK=elasticsearch+Logstash+kibana elasticsearch：后台分布式存储以及全文检索 logstash: 日志加工、“搬运工” kibana：数据可视化展示。 ELK架构为数据分布式存储、可视化查询和日志解析创建了一个功能强大的管理链。 三者相互配合，取长补短，共同完成分布式大数据处理工作。

## 特点优势

1）分布式实时文件存储，可将每一个字段存入索引，使其可以被检索到。

2）实时分析的分布式搜索引擎。 分布式：索引分拆成多个分片，每个分片可有零个或多个副本。集群中的每个数据节点都可承载一个或多个分片，并且协调和处理各种操作； 负载再平衡和路由在大多数情况下自动完成。

3）可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。也可以运行在单台PC上（已测试）

4）支持插件机制，分词插件、同步插件、Hadoop插件、可视化插件等。

## 为什么要用ES？

### ES国内外使用优秀案例

1） 2013年初，GitHub抛弃了Solr，采取ElasticSearch 来做PB级的搜索。 “GitHub使用ElasticSearch搜索20TB的数据，包括13亿文件和1300亿行代码”。

2）维基百科：启动以elasticsearch为基础的核心搜索架构。

3）SoundCloud：“SoundCloud使用ElasticSearch为1.8亿用户提供即时而精准的音乐搜索服务”。

4）百度：百度目前广泛使用ElasticSearch作为文本数据分析，采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示，辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线（包括casio、云分析、网盟、预测、文库、直达号、钱包、风控等），单集群最大100台机器，200个ES节点，每天导入30TB+数据。

### 我们也需要

实际项目开发实战中，几乎每个系统都会有一个搜索的功能，当搜索做到一定程度时，维护和扩展起来难度就会慢慢变大，所以很多公司都会把搜索单独独立出一个模块，用ElasticSearch等来实现。

近年ElasticSearch发展迅猛，已经超越了其最初的纯搜索引擎的角色，现在已经增加了数据聚合分析（aggregation）和可视化的特性，如果你有数百万的文档需要通过关键词进行定位时，ElasticSearch肯定是最佳选择。当然，如果你的文档是JSON的，你也可以把ElasticSearch当作一种“NoSQL数据库”， 应用ElasticSearch数据聚合分析（aggregation）的特性，针对数据进行多维度的分析。

【知乎：热酷架构师潘飞】ES在某些场景下替代传统DB 个人以为Elasticsearch作为内部存储来说还是不错的，效率也基本能够满足，在某些方面替代传统DB也是可以的，前提是你的业务不对操作的事性务有特殊要求；而权限管理也不用那么细，因为ES的权限这块还不完善。 由于我们对ES的应用场景仅仅是在于对某段时间内的数据聚合操作，没有大量的单文档请求（比如通过userid来找到一个用户的文档，类似于NoSQL的应用场景），所以能否替代NoSQL还需要各位自己的测试。 如果让我选择的话，我会尝试使用ES来替代传统的NoSQL，因为它的横向扩展机制太方便了

## ES的应用场景是怎样的？

一线公司ES使用场景

1）新浪ES 如何分析处理32亿条实时日志 <http://dockone.io/article/505>

2）阿里ES 构建挖财自己的日志采集和分析体系 <http://afoo.me/columns/tec/logging-platform-spec.html>

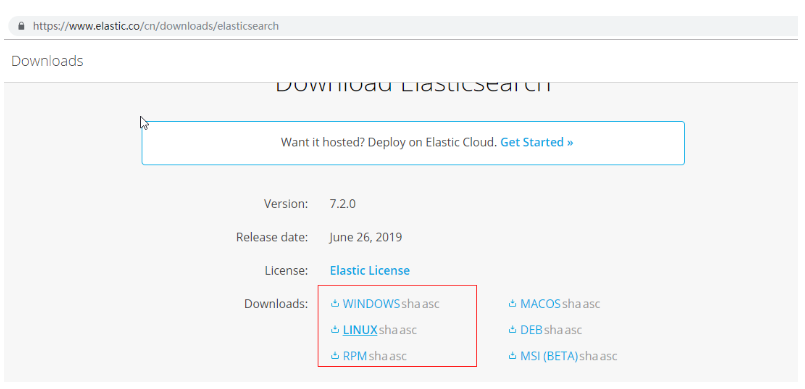
3）有赞ES 业务日志处理 <http://tech.youzan.com/you-zan-tong-ri-zhi-ping-tai-chu-tan/>

4）ES实现站内搜索 <http://www.wtoutiao.com/p/13bkqiZ.html>

# 三、ES安装

## 1、下载ES

下载：（文件比较大，建议手动下载）



下载网址：



## 2、安装

**解压：**

 tar -zxvf elasticsearch-6.2.4.tar.gz

注意：把elasticsearch软件必须放入/home/es（es是新建用户）的目录下，并把elasticsearch设置为es用户所属

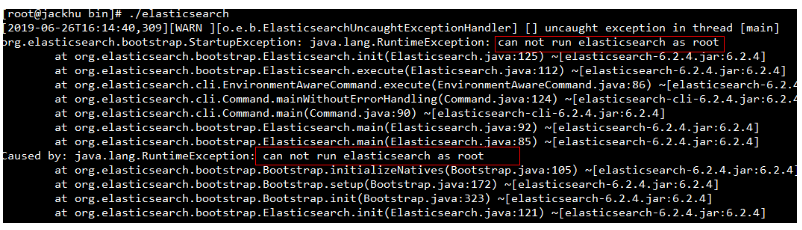
**创建日志、数据存储目录：（留作备用，初次先创建）**

 mkdir -p /data/logs/es  
 mkdir -p /data/es/{data,work,plugins,scripts}

**创建用户**

 useradd es -s /bin/bash #es不能在root用户下启动，必须创建新的用户，用来启动es

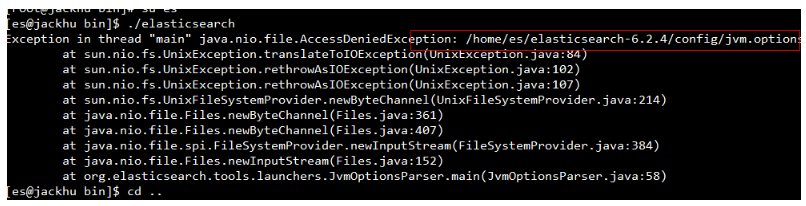
**启动：./elasticsearch**



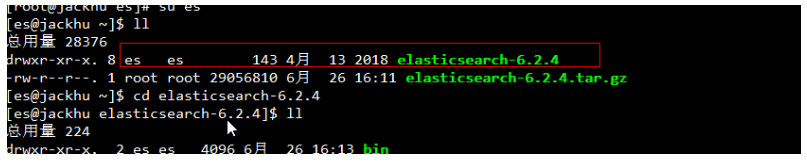
注意：es不能在root用户下启动，必须创建新的用户，用来启动es

**切换用户： su es**

再次启动，发现还是报错，原因：当前用户没有执行权限

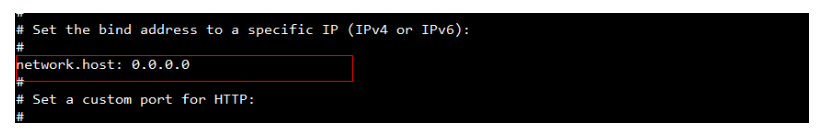


**授权：chown -R es:es elasticsearch-6.2.4**

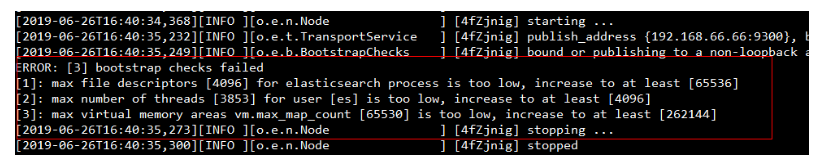


授权成功，发现elasticsearch已经在es用户下面了，可以启动了，但是启动成功，浏览器不能访问，因此还需要做如下配置：

配置修改：\*\*



再次启动：报如下错误



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

每个进程最大同时打开文件数太小，可通过下面2个命令查看当前数量

 ulimit -Hn  
 ulimit -Sn



修改/etc/security/limits.conf文件，增加配置，用户退出后重新登录生效

 \*               soft   nofile         65536  
 \*               hard   nofile         65536



2）max number of threads [3818] for user [es] is too low, increase to at least [4096]

可通过命令查看

 ulimit -Hu  
 ulimit -Su



问题同上，最大线程个数太低。修改配置文件/etc/security/limits.conf，增加配置

 `\*               soft   nproc           4096`

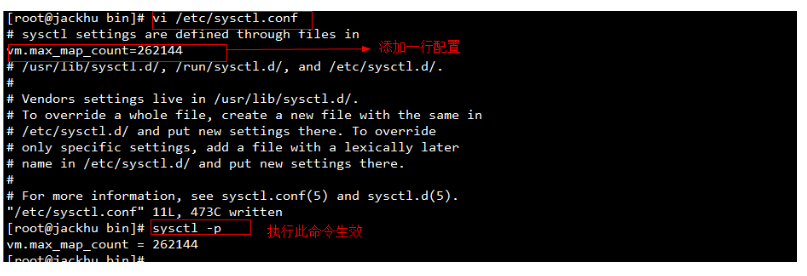
`\*               hard   nproc           4096`



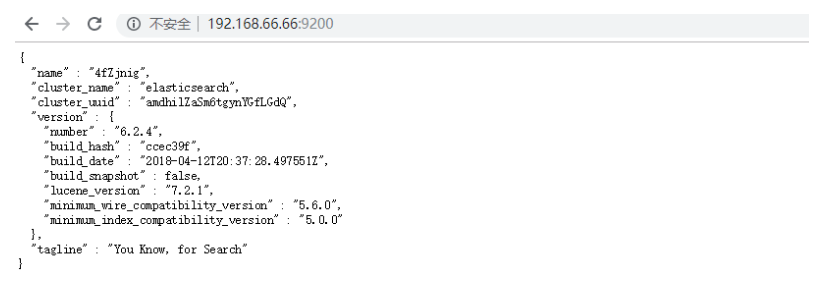
3）、max virtual memory areas vm.max\_map\_count [65530] is too low, increase to at least [262144]

　　修改/etc/sysctl.conf文件

 vi /etc/sysctl.conf  
 sysctl -p #执行命令sysctl -p生效  
 #增加配置vm.max\_map\_count=262144



错误解决完毕：重新启动



后台启动：

 ./elasticsearch -d

# 四、head插件安装

## 1、head插件主要用途

elasticsearch-head是一个用来浏览、与Elastic Search簇进行交互的web前端展示插件。 elasticsearch-head是一个用来监控Elastic Search状态的客户端插件。

elasticsearch主要有以下三个主要操作—— 1）簇浏览，显示簇的拓扑并允许你执行索引（index)和节点层面的操作。 2）查询接口，允许你查询簇并以原始json格式或表格的形式显示检索结果。 3）显示簇状态，有许多快速访问的tabs用来显示簇的状态。 4）支持Restful API接口，包含了许多选项产生感兴趣的结果，包括： 第一，请求方式:get,put,post,delete; json请求数据，节点node， 路径path。 第二，JSON验证器。 第三，定时请求的能力。 第四，用javascript表达式传输结果的能力。 第五，统计一段时间的结果或该段时间结果比对的能力。

第六，以简单图标的形式绘制传输结果

## 2、安装

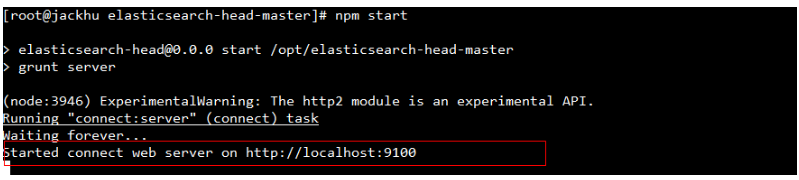
安装步骤：

 #下载nodejs,head插件运行依赖node  
 wget https://nodejs.org/dist/v9.9.0/node-v9.9.0-linux-x64.tar.xz  
 #解压  
 tar -xf node-v9.9.0-linux-x64.tar.xz  
 #重命名  
 mv node-v9.9.0-linux-x64 nodeJs  
 #配置文件  
 vim /etc/profile  
 #刷新配置  
 source /etc/profile  
 #查询node版本，同时查看是否安装成功  
 node -v  
 #下载head插件  
 wget https://github.com/mobz/elasticsearch-head/archive/master.zip  
 #解压  
 unzip master.zip  
 #使用淘宝的镜像库进行下载，速度很快  
 npm install -g cnpm --registry=https://registry.npm.taobao.org  
 #进入head插件解压目录，执行安装命令  
 cnpm install

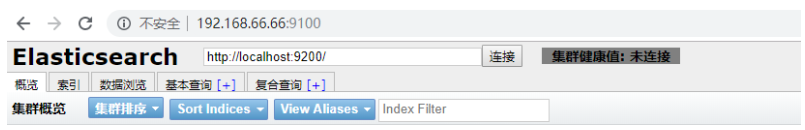
## 3、运行

 npm start #启动head插件

启动运行端口为：9100

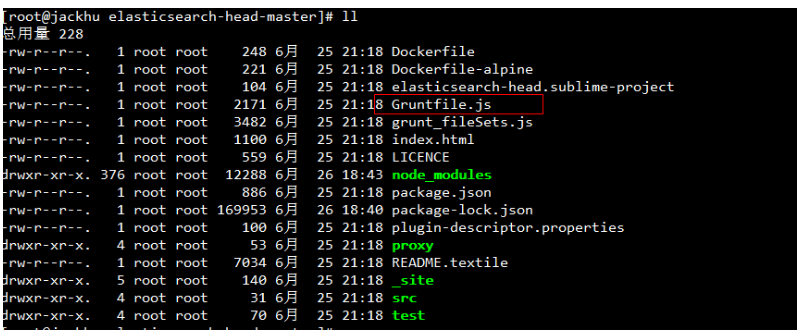


访问：

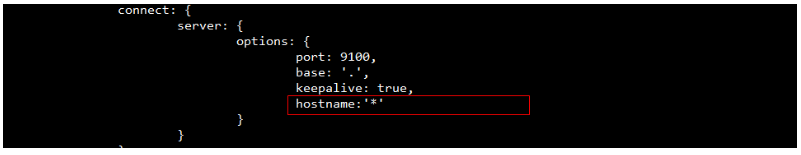


此时未连接，需要配置才能连接：

**修改 Gruntfile.js文件：**

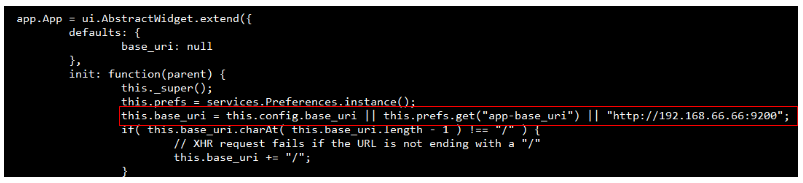


修改如下：



**修改\_site/app.js**

修改IP地址，连接elasticsearch



**启用CORS**:

当head插件访问es时，您必须在elasticsearch中启用CORS，否则您的浏览器将拒绝跨域。

在elasticsearch配置中：

 http.cors.enabled: true

您还必须设置，http.cors.allow-origin因为默认情况下不允许跨域。http.cors.allow-origin: "\*" 是允许配置的，但由于这样配置的任何地方都可以访问，所以有安全风险。 我在集群安装的时候已经配好了、如果你刚配置、需要重启ElasticSearch服务

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

访问head**插件**

