

ElasticSearch 课程讲义

第 1 次课 ElasticSearch 入门

1.1. 教学目标

1.1.1. 前言

1.1.2. 掌握 ElasticSearch 的是什么

1.1.3. 掌握 Rest 接口

1.1.4. 理解 ElasticSearch 的安装配置

1.2. 课程内容

1.2.1. 前言

百事不决问百度，万事不决问谷歌！随着信息量，数据量越来越大的今天，如何从海量的数据中聚焦关注点，提高我们的工作、生活效率便显得非常的重要。而我们经常使用的百度等搜索引擎就能帮助我们从小数据中挖掘提取出自己的感兴趣的地方。是怎么做到的呢，就是需要全文索引的技术来完成，我们知道可以查询的有数据库，但是数据库中的数据量一大就会变的非常的缓慢，而且都是模糊查询，数据量有非常的巨大，二者比较矛盾，怎么办，提高数据库的查询我们可以建立索引。这样的就慢慢的诞生了一门技术叫做全文索引，以帮助我们检索的体验，早期的在 java 领域里面 lucene, 后期有 compass, 还有 solr, 到最新的 ElasticSearch (以下都称之为 ES)。ES 作为后来者，在全文检索领域里面有非常大的优势，本身就是一个分布式的玩意，和大数据联系紧密，所以咱们也就来学习新的技术。

1.2.2. ElasticSearch 是什么



elastic

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。

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。

ES 的发展异常迅猛，下面是发展以及和 solr 简单的对比情况。



之前官网为 elasticsearch，太长了不便记忆了，就改为 elastic.co，官网地址

www.elastic.co

- ES 和 SOLR 对比

接口

类似 webservice 的接口

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 xml json

es json

近实时搜索

- ES 和 MySQL 的对比

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)

1.2.3. 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。

资源	一组资源的 URI，比如：	单个资源的 URI，比如：
----	---------------	---------------

	http://example.com/res/	http://example.com/res/123
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

ES 内置的 REST 接口

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

1.2.4. ElasticSearch 的安装配置

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

1.2.4.1. ES 安装配置一

- 下载地址

<https://www.elastic.co/downloads/past-releases/elasticsearch>

[h-2-3-0](#)

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

```
[root@service opt]# java -version
java version "1.7.0_55"
Java(TM) SE Runtime Environment (build 1.7.0_55-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.55-b03, mixed mode)
```

- 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。

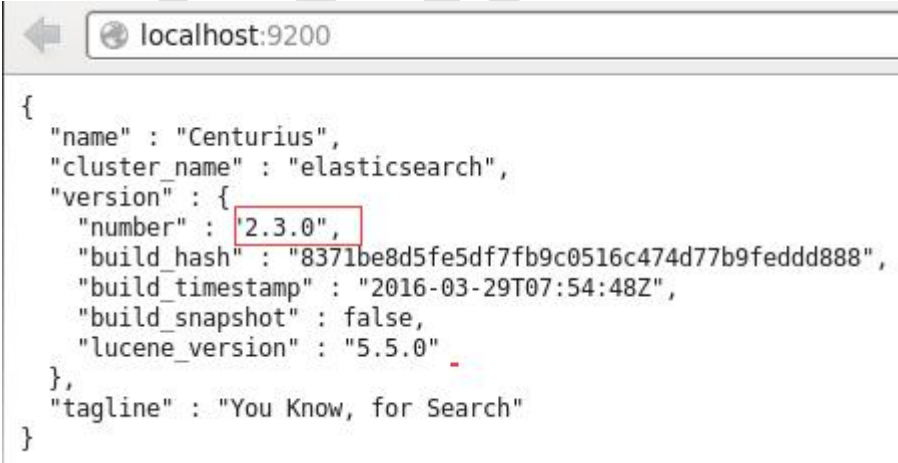
```
opt]# pwd
/opt
opt]# unzip soft/elasticsearch-2.3.0.zip
```

- 启动

```
opt]# cd elasticsearch-2.3.0/
opt]# bin/elasticsearch ---->前台启动
opt]# bin/elasticsearch -d ---->后台启动
```

- 验证

访问 es 的安装服务器，http://<es_ip>:9200



```
{
  "name" : "Centurius",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "2.3.0",
    "build_hash" : "8371be8d5fe5df7fb9c0516c474d77b9feddd888",
    "build_timestamp" : "2016-03-29T07:54:48Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

当页面出现这个状态的时候，就说明 es 安装成功了。

1.2.4.2. 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

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

{
  "name" : "Lament",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "2.3.0",
    "build_hash" : "8371be8d5fe5df7fb9c0516c474d77b9feddd888",
    "build_timestamp" : "2016-03-29T07:54:48Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

修改 cluster.name

```
17 # cluster.name: my-application
18 cluster.name: bigdata

{
  "name" : "hadoop",
  "cluster_name" : "bigdata",
  "version" : {
    "number" : "2.3.0",
    "build_hash" : "8371be8d5fe5df7fb9c0516c474d77b9feddd888",
    "build_timestamp" : "2016-03-29T07:54:48Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

修改 node.name

```
# node.name: node-1
node.name: hadoop
```



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

1.2.5. 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 可以实现常见的 get/post 请求。简单的认为是可以在命令行下面访问 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

- curl 创建索引库

```
curl -XPUT http://<ip>:9200/index_name/
```

PUT 或 POST 都可以创建

举例：

```
curl -XPUT 'http://localhost:9200/bigdata'
```

```
[bigdata@service ~]$ pwd
/home/bigdata
[bigdata@service ~]$ curl -XPUT 'http://localhost:9200/bigdata'
{"acknowledged":true}[bigdata@service ~]$
```

- 创建索引

```
curl -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 ~]$
```

➤ PUT 和 POST 的用户区别

PUT 是幂等方法，POST 不是。所以 **PUT** 用户更新，**POST** 用于新增比较合适。

PUT 和 DELETE 操作是幂等的。所谓幂等是值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

-d '{"name" : "hbase"}'

```
[bigdata@service ~]$ curl -XPOST http://localhost:9200/bigdata/product/2?op_type=create -d '{"name" : "hbase"}'
{"_index":"bigdata","_type":"product","_id":"2","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"created":true}[bigdata@service ~]$
```

curl -XPOST

http://localhost:9200/bigdata/product/3/_create -d

'{"name" : "hive"}'

```
[bigdata@service ~]$ curl -XPOST http://localhost:9200/bigdata/product/3/_create -d '{"name" : "hive"}'
{"_index":"bigdata","_type":"product","_id":"3","_version":1,"_shards":{"total":2,"successful":1,"failed":0},"created":true}[bigdata@service ~]$
```

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

- 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。

```
[bigdata@service ~]$ curl -XGET http://localhost:9200/bigdata/product/1
{"_index":"bigdata","_type":"product","_id":"1","_version":1,"found":true,"_source":{"name" : "hadoop","author" : "Doug Cutting","core" : ["hdfs","mr","yarn"],"latest_version": 3.0}}[bigdata@service ~]$
```

```
[bigdata@service ~]$ curl -XGET http://localhost:9200/bigdata/product/1?pretty
{
  "_index" : "bigdata",
  "_type" : "product",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : {
    "name" : "hadoop",
    "author" : "Doug Cutting",
    "core" : [ "hdfs", "mr", "yarn" ],
    "latest_version" : 3.0
  }
}
```

- 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'

```
[bigdata@service ~]$ curl -XGET 'http://localhost:9200/bigdata/product/1?_source=name,author&pretty'
{
  "_index" : "bigdata",
  "_type" : "product",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : {
    "author" : "Doug Cutting",
    "name" : "hadoop"
  }
}
```

➤ 获取 source 的数据

curl -XGET 'http://localhost:9200/bigdata/product/1/_source?pretty'

```
[bigdata@service ~]$ curl -XGET 'http://localhost:9200/bigdata/product/1/_source?pretty'
{
  "name" : "hadoop",
  "author" : "Doug Cutting",
  "core" : [ "hdfs", "mr", "yarn" ],
  "latest_version" : 3.0
}
[bigdata@service ~]$
```

➤ 查询所有

curl -XGET 'http://localhost:9200/bigdata/product/_search?pretty'

```
[bigdata@service ~]$ curl -XGET 'http://localhost:9200/bigdata/product/_search?pretty'
{
  "took" : 70,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 4,
    "max_score" : 1.0,
    "hits" : [ {

```

➤ 根据条件进行查询

```
curl -XGET
'http://localhost:9200/bigdata/product/_search?q=name:hbase&pretty'
```

```
[bigdata@service ~]$ curl -XGET 'http://localhost:9200/bigdata/product/_search?q=name:hbase&pretty'
{
  "took" : 57,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.30685282,
    "hits" : [ {
      "_index" : "bigdata",
      "_type" : "product",
      "_id" : "2",
      "_score" : 0.30685282,
      "_source" : {
        "name" : "hbase"
      }
    } ]
  }
}
```

- 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"}}'
```

```
[bigdata@service ~]$ curl -XPOST http://localhost:9200/bigdata/product/1/_update -d '{"doc":{"name" :
"apache-hadoop"}}'
{"_index":"bigdata","_type":"product","_id":"1","_version":2,"_shards":{"total":2,"successful":1,"failed":0}}[bigdata@service ~]$
```

```
[bigdata@service ~]$ curl -XGET 'http://localhost:9200/bigdata/product/1?_source&pretty'
{
  "_index" : "bigdata",
  "_type" : "product",
  "_id" : "1",
  "_version" : 2,
  "found" : true,
  "_source" : {
    "name" : "apache-hadoop",
    "author" : "Doug Cutting",
    "core" : [ "hdfs", "mr", "yarn" ],
    "latest_version" : 3.0
  }
}
```

已经将 name 更改为 apache-hadoop 了

- ES 删除

- 普通删除, 根据主键删除

```
curl -XDELETE http://localhost:9200/bigdata/product/3/
```

```
[bigdata@service ~]$ curl -XDELETE 'http://localhost:9200/bigdata/product/3/'
{"found":true,"_index":"bigdata","_type":"product","_id":"3","_version":2,"_shards":{"total":2,"successful":1,"failed":0}}[bigdata@service ~]$
[bigdata@service ~]$ curl -XGET 'http://localhost:9200/bigdata/product/3?_source&pretty'
{
  "_index" : "bigdata",
  "_type" : "product",
  "_id" : "3",
  "found" : false
}
```

说明：如果文档存在，es 属性 found: true, successful:1, _version 属性的值+1。

```
[bigdata@service ~]$ curl -XDELETE 'http://localhost:9200/bigdata/product/3/'
{"found":false,"_index":"bigdata","_type":"product","_id":"3","_version":1,"_shards":{"total":2,"successful":1,"failed":0}}[bigdata@service ~]$
```

如果文档不存在，es 属性 found 为 false，但是版本值 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。

● 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"
```

```
[bigdata@service ~]$ curl -XPOST 'http://localhost:9200/bank/accout/_bulk?pretty' --data-binary "@data/accounts.json"
```



accounts.json



accounts数据说明.txt

可以查看一下各个索引库信息

```
curl 'http://localhost:9200/_cat/indices?v'
```

```
[bigdata@service ~]$ curl 'localhost:9200/_cat/indices?v'
health status index      pri rep docs.count docs.deleted store.size pri.store.size
yellow open   bank        5   1    1000           0   518.1kb     518.1kb
yellow open   bigdata     5   1     27           0   148.3kb     148.3kb
yellow open   .kibana     1   1      1           0    3.1kb      3.1kb
```

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
```

● 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/produc/1/_update?version=1 -d
'{"doc":{"name":"apache hadoop","version":2.6.4}}' (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"}'
```

```
[bigdata@service ~]$ curl -XPUT 'http://localhost:9200/bigdata/product/20?version=10&version_type=external' -d '{"name": "flink"}'
{"_index":"bigdata","_type":"product","_id":"20","_version":10,"_shards":{"total":2,"successful":1,"failed":0},"created":true}[bigdata@service ~]$
```

注意: 此处 url 前后的引号不能省略, 否则执行的时候会报错

1.2.6. ES 插件

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。

1.2.6.1. Elasticsearch-servicewrapper

这里就先介绍一个插件用于 ES 的服务端管理—Elasticsearch-servicewrapper (绝大部分的插件都在 github 里面可以找到)。Elasticsearch-servicewrapper 已过时, 但是可以使用它进行简单的启动停止的操作, 在 es2.x 之后就不可以使用了。

- 下载地址:

<https://github.com/elastic/elasticsearch-servicewrapper>

- 安装

把elasticsearch-servicewrapper-master.zip解压后拷贝到ES_HOME/bin目录下面,就可以通过 service bin/service 的脚本来控制 ES 了。

```
[root@service opt]# ll elasticsearch-servicewrapper-master
total 8
-rw-r--r--. 1 root root 2553 Feb 13 2016 README.md
drwxr-xr-x. 4 root root 4096 Feb 13 2016 service
[root@service opt]# cp -r elasticsearch-servicewrapper-master/service/ elasticsearch-1.6.0/bin/
[root@service opt]#
```

- 操作

```
[root@service elasticsearch-1.6.0]# bin/service/elasticsearch
Usage: bin/service/elasticsearch [ console | start | stop | restart | condrestart | status

Commands:
  console      Launch in the current console.
  start        Start in the background as a daemon process.
  stop         Stop if running as a daemon or in another console.
  restart      Stop if running and then start.
  condrestart   Restart only if already running.
  status       Query the current status.
  install      Install to start automatically when system boots.
  remove       Uninstall.
  dump         Request a Java thread dump if running.
```

1.2.6.2. BigDesk Plugin

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。

下载地址: <https://github.com/hlstudio/bigdesk>

在线安装: bin/plugin install hlstudio/bigdesk

```
[bigdata@service elasticsearch-2.3.0]$ bin/plugin install hlstudio/bigdesk
-> Installing hlstudio/bigdesk...
Trying https://github.com/hlstudio/bigdesk/archive/master.zip ...
```

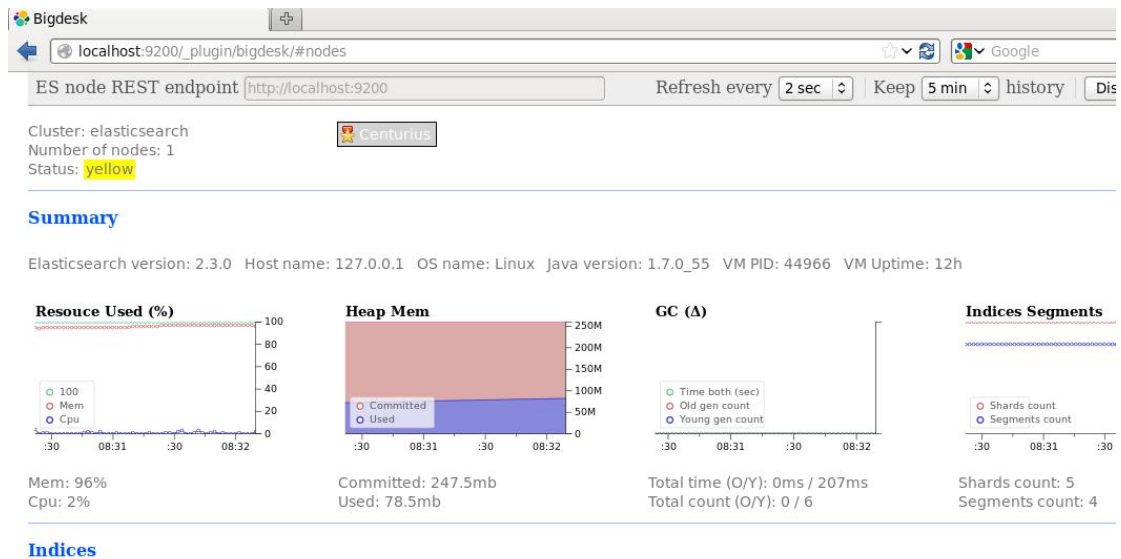
离线安装: bin/plugin install file:/tmp/bigdesk-master.zip (插件比如说在/tmp 目录下面)

```
[bigdata@service elasticsearch-2.3.0]$ bin/plugin install file:/home/bigdata/soft/bigdesk-master.zip
-> Installing from file:/home/bigdata/soft/bigdesk-master.zip...
Trying file:/home/bigdata/soft/bigdesk-master.zip ...
Downloading ...DONE
Verifying file:/home/bigdata/soft/bigdesk-master.zip checksums if available ...
NOTE: Unable to verify checksum for downloaded plugin (unable to find .sha1 or .md5 file to verify)
Installed bigdesk into /home/bigdata/elasticsearch-2.3.0/plugins/bigdesk
```

移除: bin/plugin remove plugin_name


```
[bigdata@service elasticsearch-2.3.0]$ bin/plugin remove bigdesk
-> Removing bigdesk...
Removed bigdesk
```

访问，在浏览器输入 `http://localhost: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。

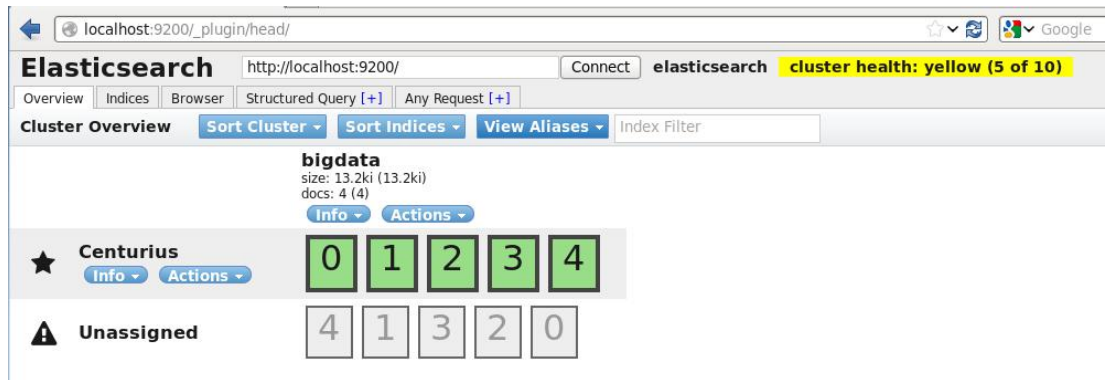
1.2.6.3. Elasticsearch-Head Plugin

方便对 ES 进行各种操作的客户端工具，推荐大家使用

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

[illegible]

访问: <http://localhost:9200/plugin/head/>



1.2.6.4. Elasticsearch Kibana

kibana 本质上是 elasticsearch web 客户端，是一个分析和可视化 elasticsearch 平台，可通过 kibana 搜索、查看和与存储在 elasticsearch 的索引进行交互。可以很方便的执行先进的数据分析和可视化多种格式的数据，如图表、表格、地图等。

下载地址: <http://www.elastic.co/downloads/kibana>, 这里下载的版本为 kibana-4.5.0-linux-x64.tar.gz, 需要特别注意的就是 kibana 和 ES 之间的版本匹配问题。简单配置即可:

```
# Kibana is served by a back end server. This controls which port to use.
server.port: 5601

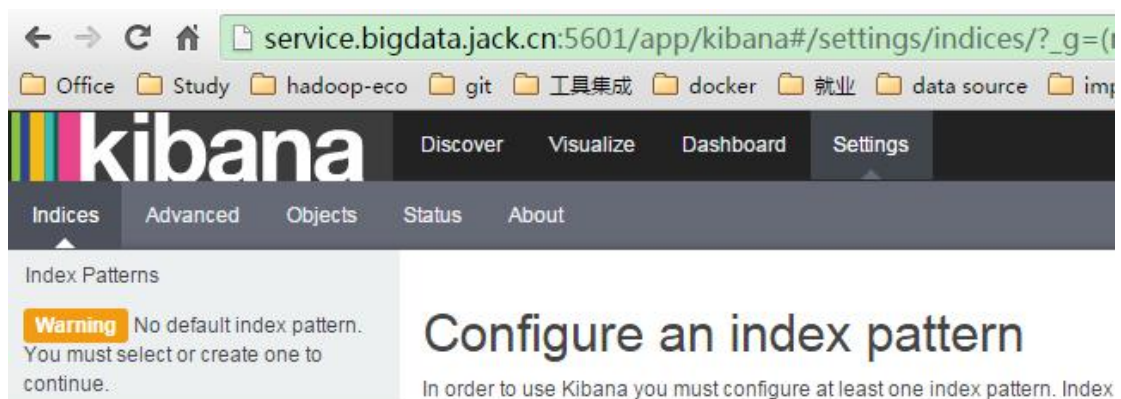
# The host to bind the server to.
server.host: "service.bigdata.jack.cn"

# The Elasticsearch instance to use for all your queries.
elasticsearch.url: "http://service.bigdata.jack.cn:9200"
```

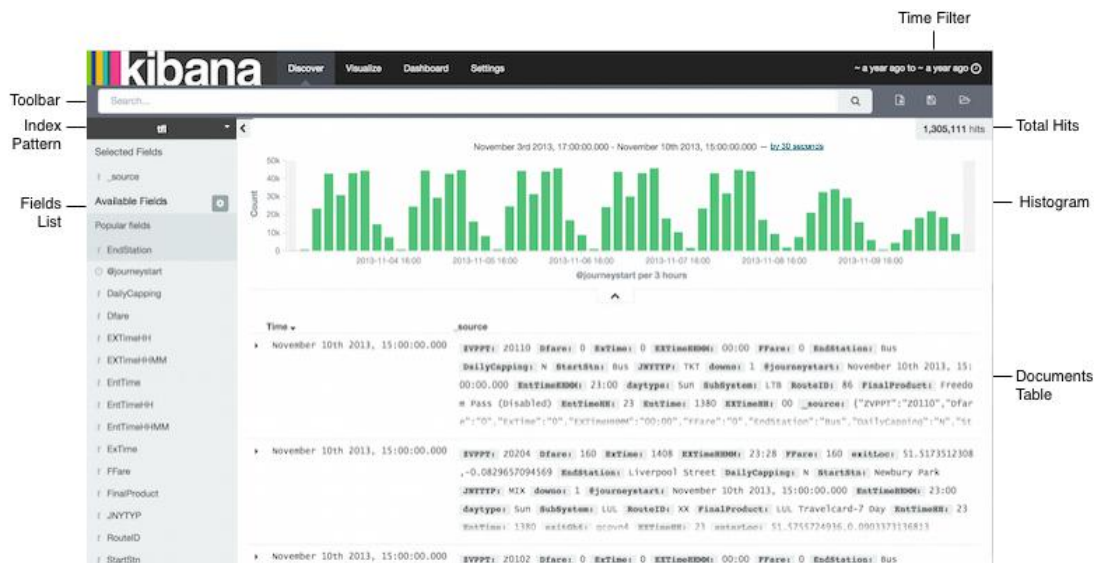
启动: \$KIBANA_HOME/bin/kibana

后台启动: kibana]\$ nohup bin/kibana >logs/kibana.log 2>&1 &

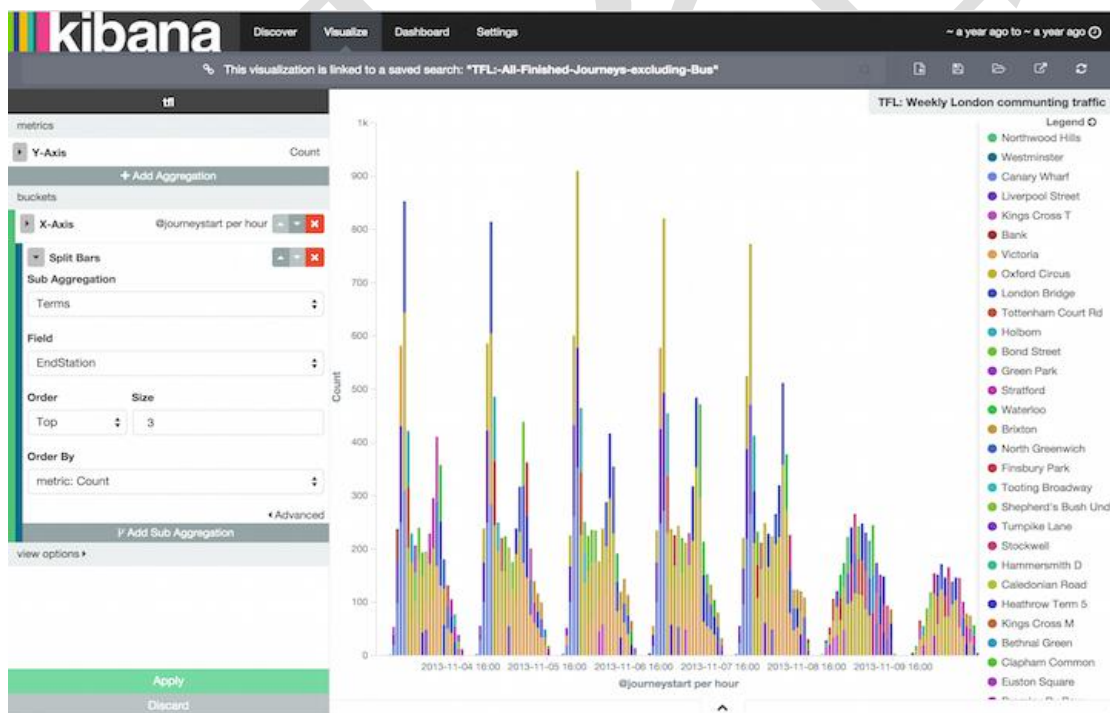
访问: <http://service.bigdata.jack.cn:5601/>



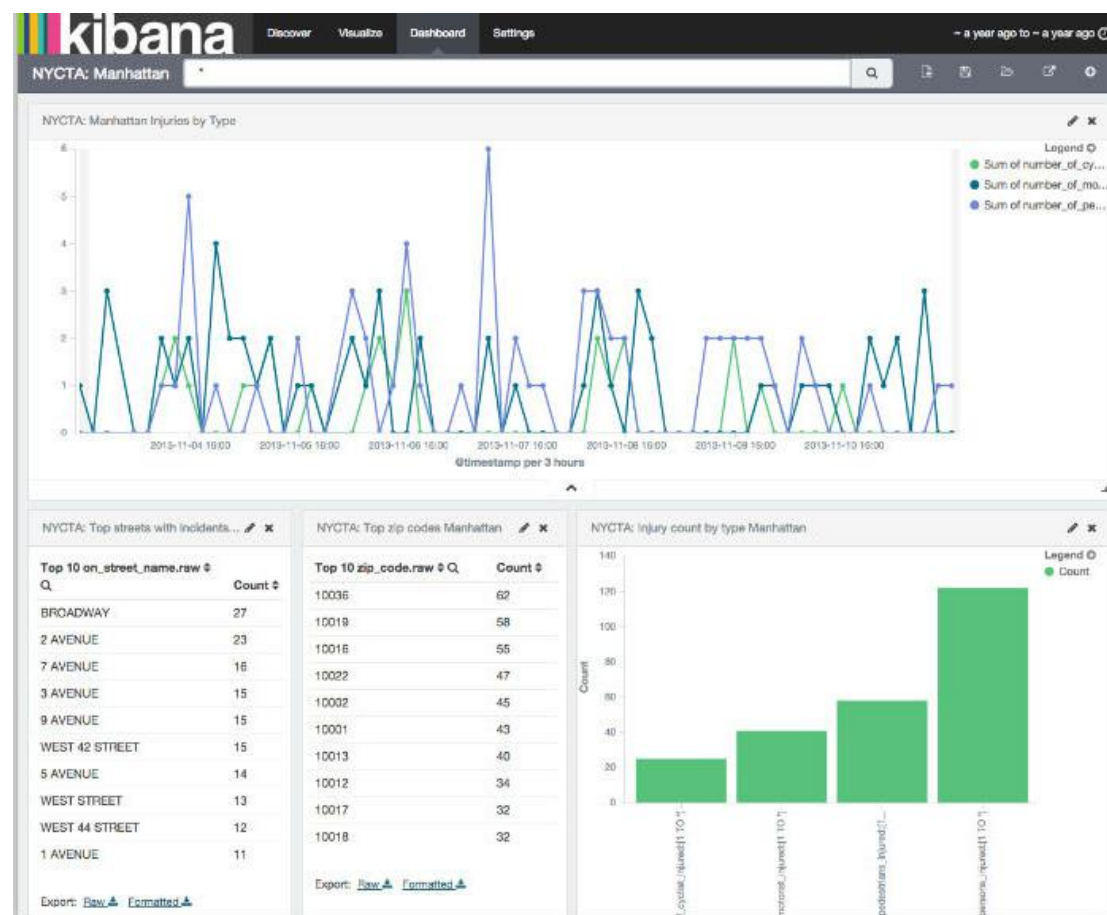
Discover 页面：交互式的浏览数据。可以访问所匹配的索引模式的每个索引的每个文档。可以提交搜索查询，过滤搜索结果和查看文档数据。还可以搜索查询匹配的文档数据和字段值的统计数据。还可以选定时间以及刷新频率。



Visualize 页面：设计数据可视化。可以保存这些可视化，单独或合并成仪表盘。可视化可以基于以下数据源类型 1. 一个新的交互式搜索 2. 一个保存的搜索 3. 现有的可视化。



Dashboard 页面：自由排列已保存的可视化，保存这个仪表盘并可以分享或者重载。



settings 页面：要使用 kibana，得先告诉 kibana 要搜索的 elasticsearch 索引是哪些，可以配置一个或更多索引。

1.2.6.5. More

插件非常非常的丰富，这里就不一一介绍了，主要介绍这几个，安装配置都大同小异。

Paramedic Plugin (作者 Karel Minařík)

简介：es 监控插件

SegmentSpy Plugin (作者 Zachary Tong)

简介：查看 es 索引 segment 状态的插件

Inquisitor Plugin (作者 Zachary Tong)

简介：这个插件主要用来调试你的查询。

1.2.7. ES 集群安装配置

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。

伪分布，只要配置的 http.port 不一致都可以了；完全分布式不作要求。以伪分布为例说明。

拷贝 elasticsearch-2.3.0 到当前目录下：

```
drwxr-xr-x.  9 bigdata bigdata 4096 Oct  8 08:08 elasticsearch-2.3.0
drwxr-xr-x.  9 bigdata bigdata 4096 Oct  8 23:38 elasticsearch-s1
drwxr-xr-x.  9 bigdata bigdata 4096 Oct  8 23:40 elasticsearch-s2
```

主要配置项

elasticsearch-2.3.0 节点一

```
cluster.name: bigdata
http.port: 9200
network.host: 0.0.0.0
```

elasticsearch-s1 节点二

```
cluster.name: bigdata
http.port: 19200
network.host: 0.0.0.0
transport.tcp.port: 19300
```

elasticsearch-s1 节点三

```
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



1.2.8. ES 中的核心概念

1.2.8.1. 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。

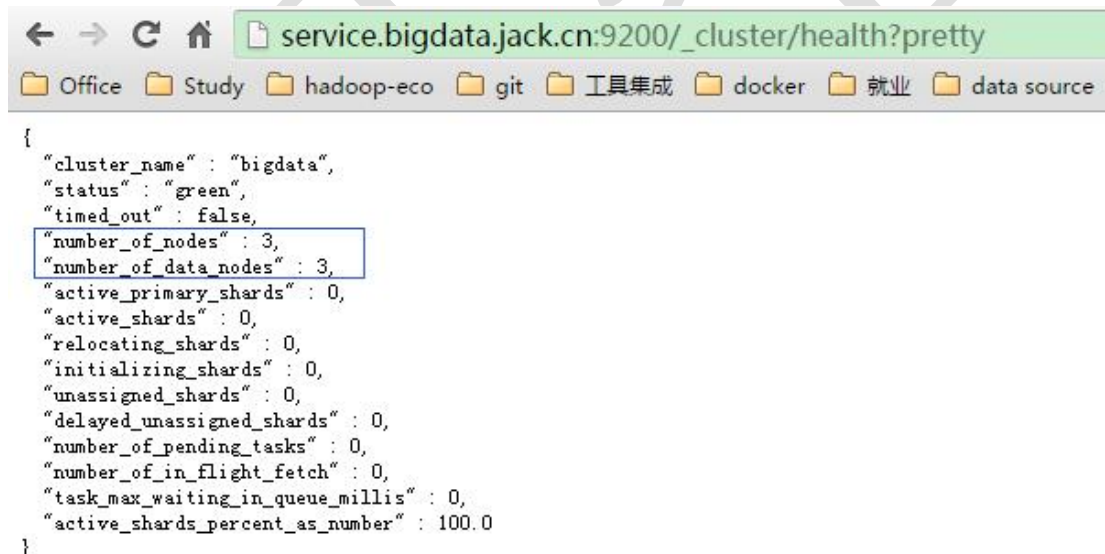
主节点的职责是负责管理集群状态，包括管理分片的状态和副本的状态，以及节点的发现和删除。

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://service.bigdata.jack.cn:9200/_cluster/health?pretty



1.2.8.2. 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
```

1.2.8.3. 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
```

1.2.8.4. recovery

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。

1.2.8.5. gateway

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。

1.2.8.6. **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:["192.8.50.150",  
"192.8.53.124:9300"]
```

1.2.8.7. **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)。

1.2.9. **ES Java 客户端操作**

添加 Maven 依赖:

```
<dependency>  
  <groupId>org.elasticsearch</groupId>  
  <artifactId>elasticsearch</artifactId>  
  <version>2.3.0</version>  
</dependency>  
<dependency>  
  <groupId>com.fasterxml.jackson.core</groupId>  
  <artifactId>jackson-databind</artifactId>  
  <version>2.7.0</version>  
</dependency>  
<dependency>  
  <groupId>org.dom4j</groupId>  
  <artifactId>dom4j</artifactId>  
  <version>2.0.0</version>  
</dependency>
```

1.2.9.1. 建立客户端连接

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 String HOST = "service.bigdata.jack.cn";
    private static final int PORT = 9300;
    private TransportClient client;
    @Before
    public void setUp() {
        client = TransportClient.builder().build();
        InetSocketAddress ista = new
        InetSocketAddress(new InetSocketAddress(HOST, PORT));
        InetSocketAddress ista1 = new
        InetSocketAddress(new InetSocketAddress(HOST, 19300));
        InetSocketAddress ista2 = new
        InetSocketAddress(new InetSocketAddress(HOST, 29300));
        client.addTransportAddresses(ista, ista1, ista2);
        System.out.println("cluster.name = " +
            client.settings().get("cluster.name"));
    }
    @After
    public void cleanUp() {
        client.close();
    }
}

<terminated> ElasticSearchTest [JUnit] D:\Program Files\Java\64bit\jdk1.7.0_7
十月10, 2016 10:15:39 上午 org.elasticsearch.plugins
信息: [Brian Falsworth] modules [], plugins [], sit
cluster.name = elasticsearch
```

- 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();

```

<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));

```

1.2.9.2. 增加索引

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());
}

```

- 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());
}
```

1.2.9.3. 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());
    }
}
```

1.2.9.4. 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();
}
```

结果:

```
version: 2
author=CDH
name=hadoop
version=2.7
```

1.2.9.5. 删除

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

1.2.9.6. 索引文档条数

```
@Test
public void testCount() {
    long count = client.prepareCount("bigdata").get().getCount();
    System.out.println(count);
}
```

1.2.9.7. 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();  
}
```

1.2.9.8. Search 全文检索

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

1.2.9.8.1. 查询 query

对于每个查询项，我们可以通过 must、should、mustNot 方法对 QueryBuilder 进行组合，形成多条件查询。(must=>and,should=>or)

Lucene 支持基于词条的 TermQuery、RangeQuery、PrefixQuery、BolleanQuery、PhraseQuery、WildcardQuery、FuzzyQuery

➤ TermQuery 与 QueryParser

单个单词作为查询表达式时，它相当于一个单独的项。如果表达式是由单个单词构成，QueryParser 的 parse() 方法会返回一个 TermQuery 对象。

如查询表达式为: content:hello, QueryParser 会返回一个域为 content, 值为 hello 的 TermQuery。

```
Query query = new TermQuery("content", "hello").
```

➤ RangeQuery 与 QueryParser

QueryParse 可以使用 [起始 TO 终止] 或 { 起始 TO 终止 } 表达式来构造 RangeQuery。

如查询表达式为: time: [20101010 TO 20101210] , QueryParser 会返回一个域为 time, 下限为 20101010, 上限为 20101210 的 RangeQuery。

```
Term t1 = new Term("time", "20101010");
```

```
Term t2 = new Term("time", "20101210");
```

```
Query query = new RangeQuery(t1, t2, true);
```

➤ PrefixQuery 与 QueryParser

当查询表达式中短语以星号(*)结尾时,QueryParser 会创建一个 PrefixQuery

对象。

如查询表达式为: `content: luc*`, 则 `QueryParser` 会返回一个域为 `content`, 值为 `luc` 的 `PrefixQuery`。

```
Query query = new PrefixQuery(luc);
```

➤ `BooleanQuery` 与 `QueryParser`

当查询表达式中包含多个项时, `QueryParser` 可以方便的构建 `BooleanQuery`。
`QueryParser` 使用圆括号分组, 通过 `-`, `+`, `AND`, `OR` 及 `NOT` 来指定所生成的 `BooleanQuery`。

➤ `PhraseQuery` 与 `QueryParser`

在 `QueryParser` 的分析表达式中双引号的若干项会被转换为一个 `PhraseQuery` 对象, 默认情况下, `Slop` 因子为 0, 可以在表达式中通过 `~n` 来指定 `slop` 因子的值。

如查询表达式为 `content: "hello world"~3`, 则 `QueryParser` 会返回一个域为 `content`, 内容为 `"hello world"`, `slop` 为 3 的短语查询。

```
Query query = new PhraseQuery();
query.setSlop(3);
query.add(new Term("content", "hello"));
query.add(new Term("content", "world"));
```

➤ `Wildcard` 与 `QueryParser`

`Lucene` 使用两个标准的通配符号, `*` 代表 0 或多个字母, `?` 代表 0 或 1 个字母。但查询表达式中包含 `*` 或 `?` 时, 则 `QueryParser` 会返回一个 `WildcardQuery` 对象。但要注意的是, 当 `*` 出现在查询表达式的末尾时, 会被优化为 `PrefixQuery`; 并且查询表达式的首个字符不能是通配符, 防止用户输入以通配符 `*` 为前缀的搜索表达式, 导致 `lucene` 枚举所有的项而耗费巨大的资源。

➤ `FuzzyQuery` 和 `QueryParser`

`QueryParser` 通过在某个项之后添加 `"~"` 来支持 `FuzzyQuery` 类的模糊查询。

1.2.9.8.2. Search Type

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)

总结:

从性能角度考虑:

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 解释:

这个 D 可能是 Distributed, F 可能是 Frequency, 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 的官网我们可以发现, 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。

元素	含义
QUERY_THEN_FETCH	查询是针对所有的块执行的, 但返回的是足够的信息, 而不是文档内容 (Document)。结果会被排序和分级, 基于此, 只有相关的块的文档对象会被返回。由于被取到的仅仅是这些, 故而返回的hit的大小正好等于指定的size。这对于有许多块的index来说是很便利的 (返回结果不会有重复的, 因为块被分组了)。
QUERY_AND_FETCH	最原始 (也可能是最快的) 实现就是简单的在所有相关的shard上执行检索并返回结果。每个shard返回一定尺寸的结果。由于每个shard已经返回了一定尺寸的hit, 这种类型实际上是返回多个shard的一定尺寸的结果给调用者。
DFS_QUERY_THEN_FETCH	与QUERY_THEN_FETCH相同, 预期一个初始的散射相伴用来为更准确的score计算分配了的term频率。
DFS_QUERY_AND_FETCH	与QUERY_AND_FETCH相同, 预期一个初始的散射相伴用来为更准确的score计算分配了的term频率。
SCAN	在执行了没有进行任何排序的检索时执行浏览。此时将会自动的开始滚动结果集。
COUNT	只计算结果的数量, 也会执行facet。

1.2.9.8.3. 代码体现

➤ 准备数据



news_sohusite_x
ml

➤ 解析代码，取其中前 20 条

```
public class XmlParser {  
    public static List<Article> getArticle() {  
        List<Article> list = new ArrayList<Article>();  
        SAXReader reader = new SAXReader();  
        Document document;  
        try {  
            document = reader.read(new File("news_sohusite_xml"));  
            Element root = document.getRootElement();  
            Iterator<Element> iterator =  
root.elementIterator("doc");  
            Article article = null;  
            int count = 0;  
            while(iterator.hasNext()) {  
                Element doc = iterator.next();  
                String url = doc.elementTextTrim("url");  
                String docno = doc.elementTextTrim("docno");  
                String content = doc.elementTextTrim("content");  
                String contenttitle =  
doc.elementTextTrim("contenttitle");  
                article = new Article();  
                article.setContent(content);  
                article.setDocno(docno);  
                article.setContenttitle(contenttitle);  
                article.setUrl(url);  
                if(++count > 20) {  
                    break;  
                }  
                list.add(article);  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        return list;  
    }  
}
```

➤ 批量导入 ES 库

```
@Test  
public void bulkInsert() throws Exception {
```

```
List<Article> list = XmlParser.getArticle();
ObjectMapper oMapper = new ObjectMapper();
BulkRequestBuilder bulkRequestBuilder = client.prepareBulk();
for (int i = 0; i < list.size(); i++) {
    Article article = list.get(i);
    String val = oMapper.writeValueAsString(article);
    bulkRequestBuilder.add(new IndexRequest(index, "news",
        article.getDocno()).source(val));
}
BulkResponse response = bulkRequestBuilder.get();
}
```

➤ 查询

@Test

```
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 color='blue'>")
        .setHighlighterPostTags("</font>");//高亮风格
    builder.setQuery(QueryBuilders.fuzzyQuery("name",
"hadop"));
    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);
}
}
```

与 SQL 使用 LIMIT 来控制单“页”数量类似，Elasticsearch 使用的是 from 以及 size 两个参数：

from: 从哪条结果开始，默认值为 0

size: 每次返回多少个结果，默认值为 10

假设每页显示 5 条结果，那么 1 至 3 页的请求就是：

GET /_search?size=5

GET /_search?size=5&from=5

GET /_search?size=5&from=10

注意：不要一次请求过多或者页码过大的结果，这么会对服务器造成很大的压力。因为它们会在返回前排序。一个请求会经过多个分片。每个分片都会生成自己的排序结果。然后再进行集中整理，以确保最终结果的正确性。

1.2.9.9. 中分分词

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。

集成 IK 分词步骤

1) 下载地址：

<https://github.com/medcl/elasticsearch-analysis-ik>

2) 使用 maven 对源代码进行编译 (mvn clean install -DskipTests)

3) 把编译后的 target/releases 下的 zip 文件拷贝到 ES_HOME/plugins/analysis-ik 目录下面，然后解压

3) 把下载的 ik 插件中的 conf/ik 目录拷贝到 ES_HOME/config 下

4) 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
```

5) 重启 es 服务

```
] [ik-analyzer] [Dict Loading] ik/custom/mydict.dic  
NFO ] [ik-analyzer] [Dict Loading] ik/custom/single_word_low_freq.di  
NFO ] [ik-analyzer] [Dict Loading] ik/custom/ext_stopword.dic  
NFO ] [node] [Ghoul] initialized  
NFO ] [node] [Ghoul] starting ...  
NFO ] [transport] [Ghoul] publish_address {192.168.43.158:9300}, b  
NFO ] [discovery] [Ghoul] elasticsearch/A8V6iLQ-Tfakv-kPuTGPMw  
NFO ] [cluster.service] [Ghoul] new_master {Ghoul}{A8V6iLQ-Tfakv-kPuTGPM  
ted_as_master, [0] joins received)  
NFO ] [http] [Ghoul] publish_address {192.168.43.158:9200}, b  
NFO ] [node] [Ghoul] started
```

6) 测试分词效果

需要说明的是, 数据需要重新插入, 并使用 ik 分词。

```
@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);  
}
```

curl

```
'http://localhost:9200/bigdata/_analyze?analyzer=ik&pretty=true'  
-d '{"text": "我是中国人"}'
```

第 2 次课 ElasticSearch 加强

2.1. 教学目标

2.1.1. ElasticSearch Rest 操作

2.1.2. ElasticSearch 遗留概念

2.1.3. ElasticSearch 优化

2.2. 课程内容

2.2.1. ElasticSearch Rest 操作

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。

2.2.1.1. 通过 REST 请求 URI

```
curl 'http://localhost:9200/bank/_search?q=*&pretty'
```



```
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",
```

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

2.2.1.2. 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": {}}}'
```

```
[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
  },
  "hits" : {
    "total" : 10,
    "max_score" : 1.0,
    "hits" : [
      {
        "_type" : "bank",
        "_id" : 1,
        "name" : "John Doe",
        "age" : 30,
        "address" : "123 Main St, New York, NY 10001",
        "balance" : 1000000
      },
      {
        "_type" : "bank",
        "_id" : 2,
        "name" : "Jane Smith",
        "age" : 25,
        "address" : "456 Main St, New York, NY 10001",
        "balance" : 500000
      },
      {
        "_type" : "bank",
        "_id" : 3,
        "name" : "Bob Johnson",
        "age" : 35,
        "address" : "789 Main St, New York, NY 10001",
        "balance" : 200000
      },
      {
        "_type" : "bank",
        "_id" : 4,
        "name" : "Alice Brown",
        "age" : 28,
        "address" : "101 Main St, New York, NY 10001",
        "balance" : 750000
      },
      {
        "_type" : "bank",
        "_id" : 5,
        "name" : "Charlie Davis",
        "age" : 32,
        "address" : "202 Main St, New York, NY 10001",
        "balance" : 300000
      },
      {
        "_type" : "bank",
        "_id" : 6,
        "name" : "Diana Prince",
        "age" : 27,
        "address" : "303 Main St, New York, NY 10001",
        "balance" : 600000
      },
      {
        "_type" : "bank",
        "_id" : 7,
        "name" : "Ethan Hunt",
        "age" : 31,
        "address" : "404 Main St, New York, NY 10001",
        "balance" : 400000
      },
      {
        "_type" : "bank",
        "_id" : 8,
        "name" : "Fiona Glenanne",
        "age" : 29,
        "address" : "505 Main St, New York, NY 10001",
        "balance" : 800000
      },
      {
        "_type" : "bank",
        "_id" : 9,
        "name" : "Giles Kane",
        "age" : 33,
        "address" : "606 Main St, New York, NY 10001",
        "balance" : 250000
      },
      {
        "_type" : "bank",
        "_id" : 10,
        "name" : "Harvey Dent",
        "age" : 34,
        "address" : "707 Main St, New York, NY 10001",
        "balance" : 150000
      }
    ]
  }
}
```

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。

2.2.1.3. 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。

- 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"}}
}'
```

2.2.1.4. 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。

返回 account_number and balance:

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d
'{
```

```
    "query": {"match_all": {}},
    "_source": ["balance", "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"}},
}'
```

2.2.1.5. 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&&）

➤ 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"}}
      ]
    }
  }
}
```

2.2.1.6. 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"

2.2.1.7. 执行聚合

聚合提供从你的数据中分组和提取统计能力。类似于关系型数据中的 SQL GROUP BY 和 SQL 聚合函数。在 ES, 你有能力执行搜索返回命中结果, 同时拆分命中结果, 然后统一返回结果。当你使用简单的 API 运行搜索和多个聚合, 然后返回所有结果避免网络带宽过大的情况是高效的。

- 根据 state 分组, 降序统计 top 10 state

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {"field": "state"}
    }
  }
}'
```

类似于关系型数据库

```
SELECT state, COUNT(*) FROM bank GROUP BY state ORDER BY COUNT(*)
DESC;
```

size=0 不是展示搜索结果命中数, 因为我只是想要看聚合结果

- 根据 state 计算账户平均 balance, 降序统计 top 10 state

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {"field": "state"}
    },
    "aggs": {
      "average_balance": {
        "avg": {
          "field": "balance"
        }
      }
    }
  }
}'
```

注意嵌套 **average_balance** 聚合 **group_by_state** 内聚合。这是一个常见的模式，所有的聚合。您可以嵌套内聚合聚合任意提取旋转汇总时，你需要从你的数据。

➤ 降序排序平均 balance

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state",
        "order": {
          "average_balance": "desc"
        }
      }
    },
    "aggs": {
      "average_balance": {
        "avg": {
          "field": "balance"
        }
      }
    }
  }
}'
```

➤ 聚合年龄分区间 (ages 20-29, 30-39 and 40-49), 聚合性别, 最后平均 balance 展示最终结果

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '{
  "size": 0,
  "aggs": {
    "group_by_age" : {
      "range" : {
        "field": "age",
        "ranges": [
          {"from":20,"to":30},
          {"from":30,"to":40},
          {"from":40,"to":50}
        ]
      }
    },
    "aggs": {
      "group_by_gender": {
        "terms": {
          "field": "gender"
        },
        "aggs": {
```

```
        "average_balance": {
          "avg": {"field": "balance"}
        }
      }
    }
  }
}
```

```
}, {
  "key" : "30.0-40.0",
  "from" : 30.0,
  "from_as_string" : "30.0",
  "to" : 40.0,
  "to_as_string" : "40.0",
  "doc_count" : 504,
  "group_by_gender" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [ {
      "key" : "f",
      "doc_count" : 253,
      "average_balance" : {
        "value" : 25670.869565217392
      }
    }, {
      "key" : "m",
      "doc_count" : 251,
      "average_balance" : {
        "value" : 24288.239043824702
      }
    } ]
  }
}, {
```

2.2.2. Settings 和 Mappings

2.2.2.1. Settings

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`

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}}'
```

2.2.2.2. Mapping

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"}}}'
```


2.2.3. Elasticsearch 优化

2.2.3.1. 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 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。而分片数过少会导致单个分片索引过大, 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'
```

java 代 码 :

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

2.2.3.2. 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'

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。

2.2.3.3. 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

第3次课 附件说明:

1、不能使用 root 用户进行操作

```
elasticsearch-2.3.0]# bin/elasticsearch
thread "main" java.lang.RuntimeException: don't run elasticsearch as root.
g.elasticsearch.bootstrap.Bootstrap.initializeNatives(Bootstrap.java:93)
g.elasticsearch.bootstrap.Bootstrap.setup(Bootstrap.java:144)
g.elasticsearch.bootstrap.Bootstrap.init(Bootstrap.java:270)
g.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:35)
log for complete error details.
```

所以只能在普通用户下进行操作,将 root 用户切换到普通用户下即可,如遇权限问题,修改相应权限即可。

2、CentOS 安装 Node.js

```
[bigdata@service kibana-4.5.0]$ bin/kibana
which: no node in (/opt/jdk1.7.0_55/bin:/usr/local/zookeeper-3.4.6/bin:/usr/local/sqoop-1.4.5/bin:/usr/local/spark-1.6.1/bin:/opt/hadoop-2.6.0/sbin:/opt/maven-3.1.1/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin)
unable to find usable node.js executable.
[bigdata@service kibana-4.5.0]$ jps
```

准备命令:

```
yum -y install gcc make gcc-c++ openssl-devel wget
```

下载源码及解压:

```
wget http://nodejs.org/dist/v0.10.26/node-v0.10.26.tar.gz
```

```
tar -zxvf node-v0.10.26.tar.gz
```

编译及安装:

```
make && make install
```

验证是否安装配置成功:

node -v

```
[bigdata@service ~]$ node -v
v0.10.26
[bigdata@service ~]$
```

3、ELK 中无法启动 kibana

Kibana 出现无法启动的问题，5601 端口未连接，但是进程存在。

```
ge": "No living connections"}
{"type": "log", "@timestamp": "2016-10-10T09:29:24+00:00", "tags": ["warning", "elasticsearch"], "pid": 30791, "message": "Unable to revive connection: http://service.bigdata.jack.cn:9200/"}
{"type": "log", "@timestamp": "2016-10-10T09:29:24+00:00", "tags": ["warning", "elasticsearch"], "pid": 30791, "message": "No living connections"}
```

解决办法需要删除 .kibana 的索引

```
curl -XDELETE http://localhost:9200/.kibana
```

4、安装 elasticsearch 5.x 以上遇到的问题

- 1) requires kernel 3.5+ with CONFIG_SECCOMP and CONFIG_SECCOMP_FILTER

```
2018-10-30T07:15:32,901][WARN ][o.e.b.JNANatives] unable to install syscall filter:
ava.lang.UnsupportedOperationException: seccomp unavailable: requires kernel 3.5+ with CONFIG_SECCOMP and CONFIG_SECCOMP
FILTER compiled in
at org.elasticsearch.bootstrap.SystemCallFilter.linuxImpl(SystemCallFilter.java:328) ~[elasticsearch-6.2.0.jar:6.
.0]
at org.elasticsearch.bootstrap.SystemCallFilter.init(SystemCallFilter.java:616) ~[elasticsearch-6.2.0.jar:6.2.0]
at org.elasticsearch.bootstrap.JNANatives.tryInstallSystemCallFilter(JNANatives.java:258) [elasticsearch-6.2.0.jar:
6.2.0]
at org.elasticsearch.bootstrap.Natives.tryInstallSystemCallFilter(Natives.java:113) [elasticsearch-6.2.0.jar:6.2.
0]
at org.elasticsearch.bootstrap.Bootstrap.initializeNatives(Bootstrap.java:110) [elasticsearch-6.2.0.jar:6.2.0]
```

centos6.x 的内核太低，需要 centos7 或者升级 centos6.x 对应的内核至 3.5 以上。这里选择升级 centos6.x 对应的内核。

A. 查看内核：

执行 `more /etc/issue` 和 `uname -a` 查看 linux 内核信息

```
[bigdata@bigdata01 logs]$ more /etc/issue
CentOS release 6.5 (Final)
Kernel \r on an \m

[bigdata@bigdata01 logs]$ uname -a
Linux bigdata01 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux
```

可以看到内核信息太低，只有 2.6 左右，需要做升级。

B. 升级

- a. 导入 public key

`sudo rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org`
有如下错误：

```
[root@bigdata01 logs]# rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
curl: (35) SSL connect error
error: https://www.elrepo.org/RPM-GPG-KEY-elrepo.org: import read failed(2).
```

无法在服务器使用 `curl` 命令访问 `https` 域名，原因是 `nss` 版本有点旧了，`yum -y update nss` 更新一下，重新 `curl` 即可！

```
[root@bigdata01 logs]# yum -y update nss
Loaded plugins: fastestmirror, refresh-packagekit, security
Loading mirror speeds from cached hostfile
* base: mirrors.tuna.tsinghua.edu.cn
* extras: mirrors.tuna.tsinghua.edu.cn
* updates: mirrors.tuna.tsinghua.edu.cn
base
```

b. 安装 ELRepo 到 CentOS

可以去 <http://elrepo.org/tiki/tiki-index.php> 选择要安装的 ELRepo。

★ Get started

Import the public key:

```
rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
```

Detailed info on the GPG key used by the ELRepo Project can be found on <https://www.elrepo.org/tiki/keys>
If you have a system with Secure Boot enabled, please see the [SecureBootKey](#) page for more information.

To install ELRepo for RHEL-7, SL-7 or CentOS-7:

```
rpm -Uvh https://www.elrepo.org/elrepo-release-7.0-3.el7.elrepo.noarch.rpm
```

To make use of our mirror system, please also install yum-plugin-fastestmirror.

To install ELRepo for RHEL-6, SL-6 or CentOS-6:

```
rpm -Uvh https://www.elrepo.org/elrepo-release-6-8.el6.elrepo.noarch.rpm
```

To make use of our mirror system, please also install yum-plugin-fastestmirror.

一般会安装到最新版内核

```
~]$ sudo rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
[bigdata@bigdata01 ~]$ sudo rpm -Uvh https://www.elrepo.org/elrepo-release-6-8.el6.elrepo.noarch.rpm
Retrieving https://www.elrepo.org/elrepo-release-6-8.el6.elrepo.noarch.rpm
Preparing...
1:elrepo-release
~]$ sudo yum --enablerepo=elrepo-kernel install kernel-lt -y
[bigdata@bigdata01 ~]$ sudo yum --enablerepo=elrepo-kernel install kernel-lt -y
Loaded plugins: fastestmirror, refresh-packagekit, security
Loading mirror speeds from cached hostfile
* base: mirrors.tuna.tsinghua.edu.cn
* elrepo: mirrors.tuna.tsinghua.edu.cn
* elrepo-kernel: mirrors.tuna.tsinghua.edu.cn
```

c. 编辑 grub.conf 文件, 修改 Grub 引导顺序

sudo vim /etc/grub.conf

```
[bigdata@bigdata01 ~]$ sudo vim /etc/grub.conf
[sudo] password for bigdata:
```

```
# grub.conf generated by anaconda
#
```

因为一般新安装的内核在第一个位置, 所以设置 default=0, 表示启动新内核


```
#
# Note that you do not have to rerun grub after making
# NOTICE: You have a /boot partition. This means that
#           all kernel and initrd paths are relative to
#           root (hd0,0)
#           kernel /vmlinuz-version ro root=/dev/sda3
#           initrd /initrd-[generic-]version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
```

重启之后查看 linux 内核

```
[bigdata@bigdata01 ~]$ uname -r
4.4.162-1.el6.elrepo.x86_64
```

搞定！

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

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

编辑 limits.conf 添加类似如下内容

```
sudo vim /etc/security/limits.conf
* soft nofile 65536
* hard nofile 131072
* soft nproc 2048
* hard nproc 4096
```

```
* soft nofile 65536
* hard nofile 131072
* soft nproc 2048
* hard nproc 4096
```

3) max number of threads [1024] for user [bigdata] is too low, increase to at least [4096]

[2]: max number of threads [1024] for user [bigdata] is too low, increase to at least [4096]

进入/etc/security/limits.d/目录下修改配置文件 90-nproc.conf

```
[bigdata@bigdata01 ~]$ sudo vim /etc/security/limits.d/90-nproc.conf

# Default limit for number of user's processes to prevent
# accidental fork bombs.
# See rhbz #432903 for reasoning.

* soft nproc 1024
root soft nproc unlimited
```


- 4) max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144]

修改配置文件/etc/sysctl.conf

添加如下内容: vm.max_map_count = 262144

```
vm.max_map_count = 262144
```

生效: ~]\$ sudo sysctl -p

```
[bigdata@bigdata01 ~]$ sudo sysctl -p
net.ipv4.ip_forward = 0
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
kernel.sysrq = 0
kernel.core_uses_pid = 1
net.ipv4.tcp_syncookies = 1
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.shmmax = 68719476736
kernel.shmall = 4294967296
vm.max_map_count = 262144
```

[3]: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144]

- 5) system call filters failed to install; check the logs and fix your configuration or disable system call filters at your own risk

[4]: system call filters failed to install; check the logs and fix your configuration or disable system call filters at your own risk

这是在因为 Centos6 不支持 SecComp, 而 ES5.2.0 默认 bootstrap.system_call_filter 为 true 进行检测, 所以导致检测失败, 失败后直接导致 ES 不能启动。

在 elasticsearch.yml 中配置 bootstrap.system_call_filter 为 false, 注意要在 Memory 下面:

```
bootstrap.memory_lock: false
bootstrap.system_call_filter: false
```

```

# ----- Memory -----
#
# Lock the memory on startup:
#
#bootstrap.memory_lock: true
#
bootstrap.memory_lock: false
bootstrap.system_call_filter: false

```

重启电脑！

```

, enforcing bootstrap checks
[2018-10-30T08:18:30,151][INFO ][o.e.c.s.MasterService      ] [elasticsearch] zen-disco-elected-as-master
), reason: new_master {elasticsearch}{c-XVUOzftG6eq8ySAv3UhQ}{-ZsRtDCNTwC6h19IlnJBjA}{bigdata01}{192.168.43.111:9300}, reason: apply cluster state (from master)
[2018-10-30T08:18:30,175][INFO ][o.e.c.s.ClusterApplierService] [elasticsearch] new_master {elasticsearch}{c-XVUOzftG6eq8ySAv3UhQ}{-ZsRtDCNTwC6h19IlnJBjA}{bigdata01}{192.168.43.111:9300}, reason: apply cluster state (from master)
[2018-10-30T08:18:30,271][INFO ][o.e.g.GatewayService       ] [elasticsearch] recovered [0] indices into state: GREEN
[2018-10-30T08:18:30,279][INFO ][o.e.h.n.Netty4HttpServerTransport] [elasticsearch] publish_address {192.168.43.111:9200}, bound_addresses {192.168.43.111:9200}
[2018-10-30T08:18:30,280][INFO ][o.e.n.Node               ] [elasticsearch] started

```

启动成功：



```

{
  "name" : "elasticsearch",
  "cluster_name" : "bigdata",
  "cluster_uuid" : "hJOuC5otQgyfkiJUKooErg",
  "version" : {
    "number" : "6.2.0",
    "build_hash" : "37cdac1",
    "build_date" : "2018-02-01T17:31:12.527918Z",
    "build_snapshot" : false,
    "lucene_version" : "7.2.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}

```

集群配置：

```

node.master: true
node.data: true
discovery.zen.ping.unicast.hosts: ["bigdata01", "bigdata02", "bigdata03"]

```

5、elasticsearch6.2.0 与 kibana6.2.0 整合

下载地址：同 elasticsearch 下载地址，在安装整合的时候，只需要将 kibana 整合在 master 上面即可。

解压:

```
~]$ tar -zxvf soft/kibana-6.2.0-linux-x86_64.tar.gz -C app/
```

重命名:

```
app]$ mv kibana-6.2.0-linux-x86_64/ kibana
```

修改配置文件:

```
$KIBANA_HOME/config/kibana.yml
```

```
# Kibana is served by a back end server. This set
server.port: 5601

# Specifies the address to which the Kibana serve
# The default is 'localhost', which usually means
# To allow connections from remote users, set thi
server.host: "bigdata01"

# Enables you to specify a path to mount Kibana a
# the URLs generated by Kibana, your proxy is exp
# to Kibana. This setting cannot end in a slash.
#server.basePath: ""

# The maximum payload size in bytes for incoming
#server.maxPayloadBytes: 1048576

# The Kibana server's name. This is used for dis
server.name: "kibana"

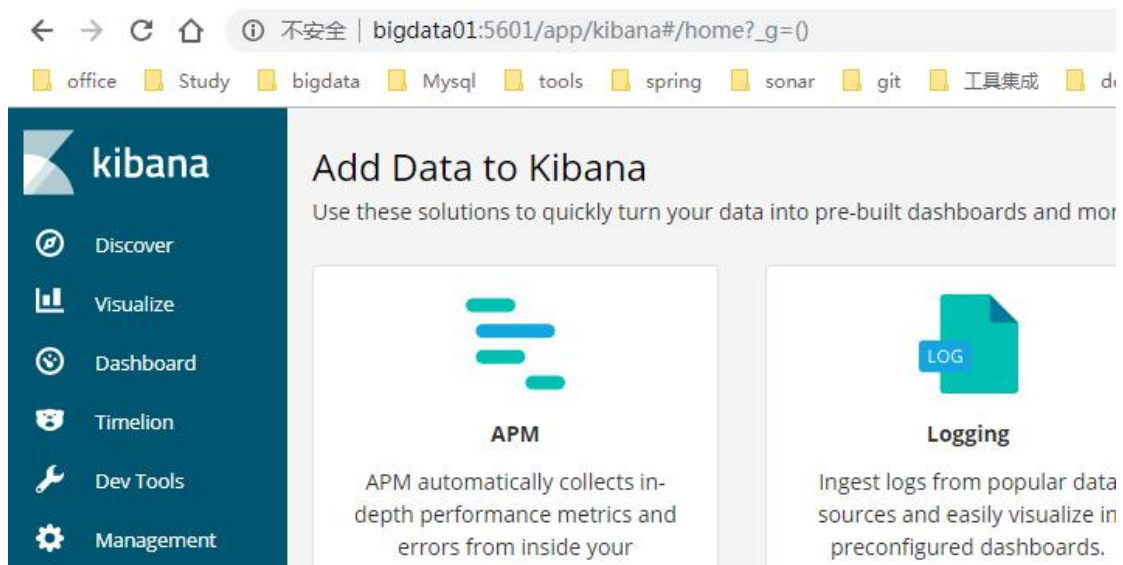
# The URL of the Elasticsearch instance to use fo
elasticsearch.url: "http://bigdata01:9200"
```

启动 kibana:

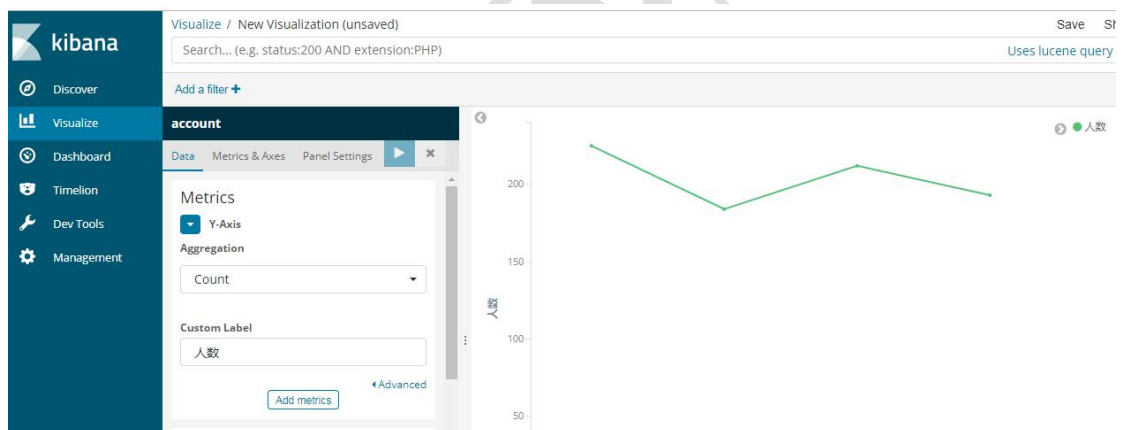
```
kibana]$ nohup bin/kibana serve >/dev/null 2>&1 &
```

```
[bigdata@bigdata01 kibana]$ bin/kibana serve -q
^C
[bigdata@bigdata01 kibana]$ bin/kibana serve
log [14:50:54.090] [info][status][plugin:kibana@6.2.0] Status changed from uninitialized to green - Ready
log [14:50:54.172] [info][status][plugin:elasticsearch@6.2.0] Status changed from uninitialized to yellow - 
or Elasticsearch
log [14:50:54.404] [info][status][plugin:timelion@6.2.0] Status changed from uninitialized to green - Ready
log [14:50:54.419] [info][status][plugin:console@6.2.0] Status changed from uninitialized to green - Ready
log [14:50:54.430] [info][status][plugin:metrics@6.2.0] Status changed from uninitialized to green - Ready
log [14:50:54.472] [info][listening] Server running at http://bigdata01:5601
log [14:50:54.583] [info][status][plugin:elasticsearch@6.2.0] Status changed from yellow to green - Ready
```

浏览器查看:



简单操作：



6、elasticsearch6.2.0 与 head 整合

ElasticSearch-head 是一个 H5 编写的 ElasticSearch 集群操作和管理工具,可以对集群进行傻瓜式操作。

显示集群的拓扑,并且能够执行索引和节点级别操作,搜索接口能够查询集群中原始 json 或表格格式的检索数据,能够快速访问并显示集群的状态,有一个输入窗口,允许任意调用 RESTful API。这个接口包含几个选项,可以组合在一起以产生有趣的结果;。5.0 版本之前可以通过 plugin 名安装,5.0 之后可以独立运行。

(一) Head 插件安装

ElasticSearch-head 插件有各种安装方式,除了 chrome 浏览器集成版本意外,其余各个版本的安装都是非常非常麻烦,所以这里建议使用 chrome 插件版,非

常简单，一分钟即可完成完成，前提需要一款开发神器 chrome 浏览器！

Running as a Chrome extension

- Install [ElasticSearch Head](#) from the Chrome Web Store.
- Click the extension icon in the toolbar of your web browser.

Running as a plugin of Elasticsearch (deprecated)

- for Elasticsearch 5.x: site plugins are not supported. Run [as a standalone server](#)
- for Elasticsearch 2.x: `sudo elasticsearch/bin/plugin install mobz/elasticsearch-head`
- for Elasticsearch 1.x: `sudo elasticsearch/bin/plugin -install mobz/elasticsearch-head/1.x`
- for Elasticsearch 0.x: `sudo elasticsearch/bin/plugin -install mobz/elasticsearch-head/0.9`
- open `http://localhost:9200/_plugin/head/`

This will automatically download the appropriate version of elasticsearch-head from github and run it as a plugin within t elasticsearch cluster. In this mode elasticsearch-head automatically connects to the node that is running it

Running with the local proxy

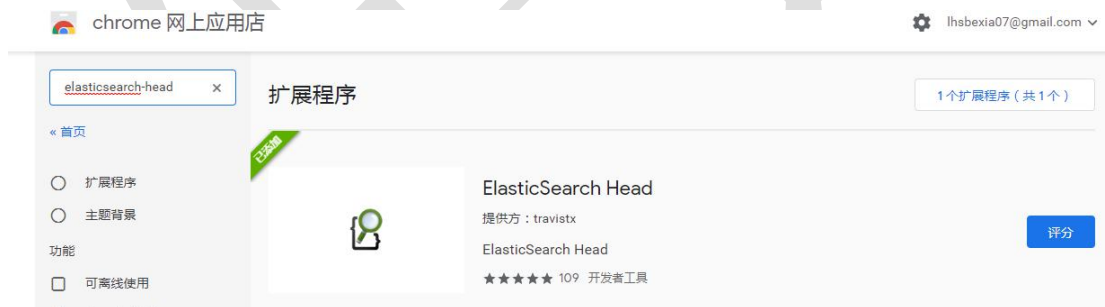
This is an *experimental feature* which creates a local proxy for many remote elasticsearch clusters

使用 chrome 和 es 的插件进行安装 elasticsearch-head，一步到位！

登 录 网 址 :

<https://chrome.google.com/webstore/detail/elasticsearch-head/ffmkiejjmecolpfloofpjologoblkegm>

或者: <https://chrome.google.com/webstore>



=====>

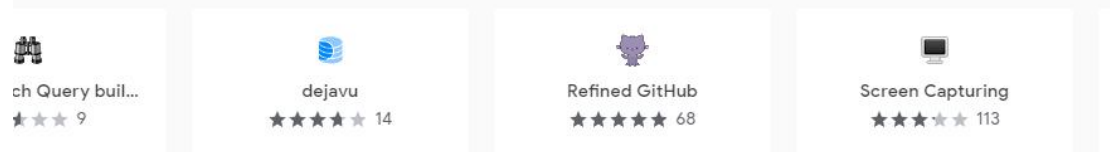
https://chrome.google.com/webstore/detail/elasticsearch-head/ffmkiejjmecolpfloofpjologoblkegm

bigdata Mysql tools spring sonar git 工具集成 docker java groovy dataset 开源项目

ome 网上应用店

English (United States)

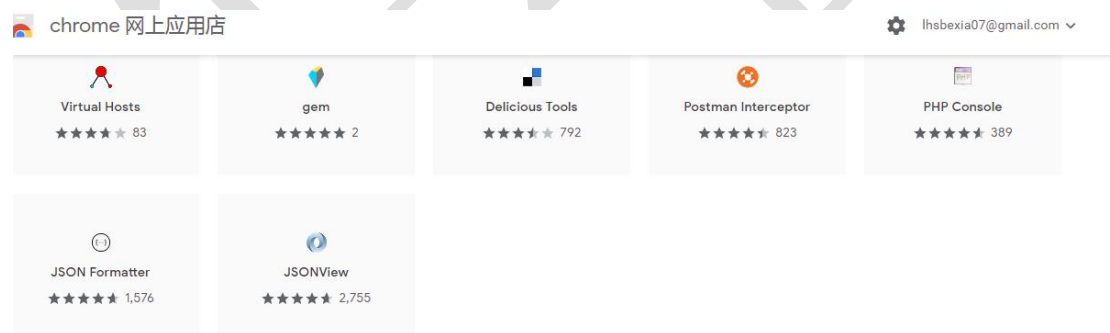
相关内容



ElasticSearch Head

添加至 Chrome

想删除非常简单:



ElasticSearch Head

从 Chrome 中删除

连接 es

