

# 尚硅谷大数据技术之 Elasticsearch

(作者：大海哥)

版本：V1.0

## 一 概述

### 1.1 什么是搜索？

百度：我们比如说想找寻任何的信息的时候，就会上百度去搜索一下，比如说找一部自己喜欢的电影，或者说找一本喜欢的书，或者找一条感兴趣的新闻（提到搜索的第一印象）。

百度 != 搜索

- 1) 互联网的搜索：电商网站，招聘网站，新闻网站，各种 app
- 2) IT 系统的搜索：OA 软件，办公自动化软件，会议管理，日程管理，项目管理。

搜索，就是在任何场景下，找寻你想要的信息，这个时候，会输入一段你要搜索的关键字，然后就期望找到这个关键字相关的有些信息

### 1.2 如果用数据库做搜索会怎么样？

如果用数据库做搜索会怎么样？

京东商城搜索框

```
select * from products where product_name list “%牙膏%”  
select * from products where product_name list “%生化机%”
```

京东商城后台商品表

商品id	商品名称	商品描述
1	高露洁牙膏	1千字的商品描述
2	中华牙膏	1千字的商品描述
3	佳洁士牙膏	
4	其他牙膏	
5	.....	1万条

逐条遍历

用数据库来实现搜索，是不太靠谱的。通常来说，性能会很差的。

1) 比如说“商品描述”字段的长度，有长达数千个，甚至数万个字符，这个时候，每次都要对每条记录的所有文本进行扫描，判断包不包含我指定的这个关键词（比如说“牙膏”），效率非常低。

2) 还不能将搜索词拆分开来，尽可能去搜索更多的符合你的期望的结果，比如输入“生化机”，就搜索不出来“生化危机”。

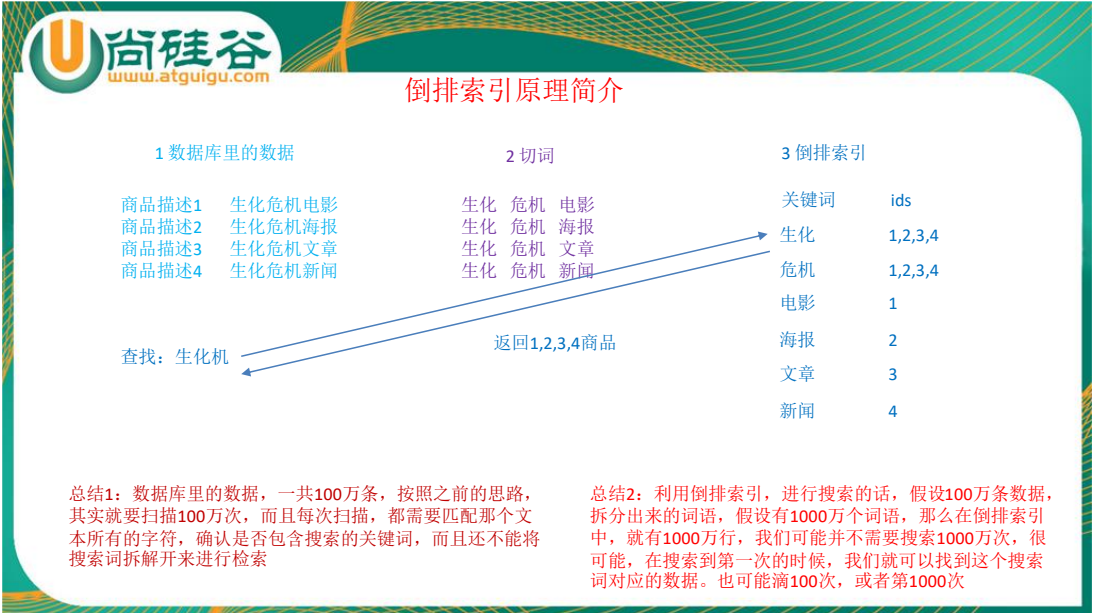
用数据库来实现搜索，是不太靠谱的。通常来说，性能会很差的。

### 1.3 什么是全文检索和 Lucene？

- 1) 全文检索，倒排索引

全文检索是指计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，

指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。这个过程类似于通过字典中的检索字表查字的过程。全文搜索搜索引擎数据库中的数据。



2) lucene，就是一个 jar 包，里面包含了封装好的各种建立倒排索引，以及进行搜索的代码，包括各种算法。我们就用 java 开发的时候，引入 lucene jar，然后基于 lucene 的 api 进行去进行开发就可以了。

## 1.4 什么是 Elasticsearch?

Elasticsearch，基于 lucene，隐藏复杂性，提供简单易用的 restful api 接口、java api 接口（还有其他语言的 api 接口）。

关于 elasticsearch 的一个传说，有一个程序员失业了，陪着自己老婆去英国伦敦学习厨师课程。程序员在失业期间想给老婆写一个菜谱搜索引擎，觉得 lucene 实在太复杂了，就开发了一个封装了 lucene 的开源项目，compass。后来程序员找到了工作，是做分布式的高性能项目的，觉得 compass 不够，就写了 elasticsearch，让 lucene 变成分布式的系统。

**Elasticsearch 是一个实时分布式搜索和分析引擎。它用于全文搜索、结构化搜索、分析。**

**全文检索：**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。

**结构化检索：**我想搜索商品分类为日化用品的商品都有哪些，`select * from products where category_id='日化用品'`

---

数据分析：电商网站，最近 7 天牙膏这种商品销量排名前 10 的商家有哪些；新闻网站，最近 1 个月访问量排名前 3 的新闻版块是哪些

## 1.5 Elasticsearch 的适用场景

- 1) 维基百科，类似百度百科，牙膏，牙膏的维基百科，全文检索，高亮，搜索推荐。
- 2) The Guardian（国外新闻网站），类似搜狐新闻，用户行为日志（点击，浏览，收藏，评论）+ 社交网络数据（对某某新闻的相关看法），数据分析，给到每篇新闻文章的作者，让他知道他的文章的公众反馈（好，坏，热门，垃圾，鄙视，崇拜）。
- 3) Stack Overflow（国外的程序异常讨论论坛），IT 问题，程序的报错，提交上去，有人会跟你讨论和回答，全文检索，搜索相关问题和答案，程序报错了，就会将报错信息粘贴到里面去，搜索有没有对应的答案
- 4) GitHub（开源代码管理），搜索上千亿行代码。
- 5) 国内：站内搜索（电商，招聘，门户，等等），IT 系统搜索（OA，CRM，ERP，等等），数据分析（ES 热门的一个使用场景）。

## 1.6 Elasticsearch 的特点

- 1) 可以作为一个大型分布式集群（数百台服务器）技术，处理 PB 级数据，服务大公司；也可以运行在单机上，服务小公司
- 2) Elasticsearch 不是什么新技术，主要是将全文检索、数据分析以及分布式技术，合并在了一起，才形成了独一无二的 ES；lucene（全文检索），商用的数据分析软件（也是有的），分布式数据库（mycat）
- 3) 对用户而言，是开箱即用的，非常简单，作为中小型的应用，直接 3 分钟部署一下 ES，就可以作为生产环境的系统来使用了，数据量不大，操作不是太复杂
- 4) 数据库的功能面对很多领域是不够用的（事务，还有各种联机事务型的操作）；特殊的功能，比如全文检索，同义词处理，相关度排名，复杂数据分析，海量数据的近实时处理；Elasticsearch 作为传统数据库的一个补充，提供了数据库所不能提供的很多功能

## 1.7 Elasticsearch 的核心概念

### 1.7.1 近实时

近实时，两个意思，从写入数据到数据可以被搜索到有一个小延迟（大概 1 秒）；基于 es 执行搜索和分析可以达到秒级。

---

### 1.7.2 Cluster（集群）

集群包含多个节点，每个节点属于哪个集群是通过一个配置（集群名称，默认是 elasticsearch）来决定的，对于中小型应用来说，刚开始一个集群就一个节点很正常

### 1.7.3 Node（节点）

集群中的一个节点，节点也有一个名称（默认是随机分配的），节点名称很重要（在执行运维管理操作的时候），默认节点会去加入一个名称为“elasticsearch”的集群，如果直接启动一堆节点，那么它们会自动组成一个 elasticsearch 集群，当然一个节点也可以组成一个 elasticsearch 集群。

### 1.7.4 Index（索引-数据库）

索引包含一堆有相似结构的文档数据，比如可以有一个客户索引，商品分类索引，订单索引，索引有一个名称。一个 index 包含很多 document，一个 index 就代表了一类类似的或者相同的 document。比如说建立一个 product index，商品索引，里面可能就存放了所有的商品数据，所有的商品 document。

### 1.7.5 Type（类型-表）

每个索引里都可以有一个或多个 type，type 是 index 中的一个逻辑数据分类，一个 type 下的 document，都有相同的 field，比如博客系统，有一个索引，可以定义用户数据 type，博客数据 type，评论数据 type。

商品 index，里面存放了所有的商品数据，商品 document

但是商品分很多种类，每个种类的 document 的 field 可能不太一样，比如说电器商品，可能还包含一些诸如售后时间范围这样的特殊 field；生鲜商品，还包含一些诸如生鲜保质期之类的特殊 field

type，日化商品 type，电器商品 type，生鲜商品 type

日化商品 type：product\_id，product\_name，product\_desc，category\_id，category\_name

电器商品 type：product\_id，product\_name，product\_desc，category\_id，category\_name，service\_period

生鲜商品 type：product\_id，product\_name，product\_desc，category\_id，category\_name，eat\_period

---

每一个 type 里面，都会包含一堆 document

```
{
  "product_id": "1",
  "product_name": "长虹电视机",
  "product_desc": "4k 高清",
  "category_id": "3",
  "category_name": "电器",
  "service_period": "1 年"
}
```

```
{
  "product_id": "2",
  "product_name": "基围虾",
  "product_desc": "纯天然，冰岛产",
  "category_id": "4",
  "category_name": "生鲜",
  "eat_period": "7 天"
}
```

### 1.7.6 Document（文档-行）

文档是 es 中的最小数据单元，一个 document 可以是一条客户数据，一条商品分类数据，一条订单数据，通常用 JSON 数据结构表示，每个 index 下的 type 中，都可以去存储多个 document。

### 1.7.7 Field（字段-列）

Field 是 Elasticsearch 的最小单位。一个 document 里面有多多个 field，每个 field 就是一个数据字段。

```
product document
{
  "product_id": "1",
```

```
    "product_name": "高露洁牙膏",  
    "product_desc": "高效美白",  
    "category_id": "2",  
    "category_name": "日化用品"  
}
```

### 1.7.8 mapping（映射-约束）

数据如何存放到索引对象上，需要有一个映射配置，包括：数据类型、是否存储、是否分词等。

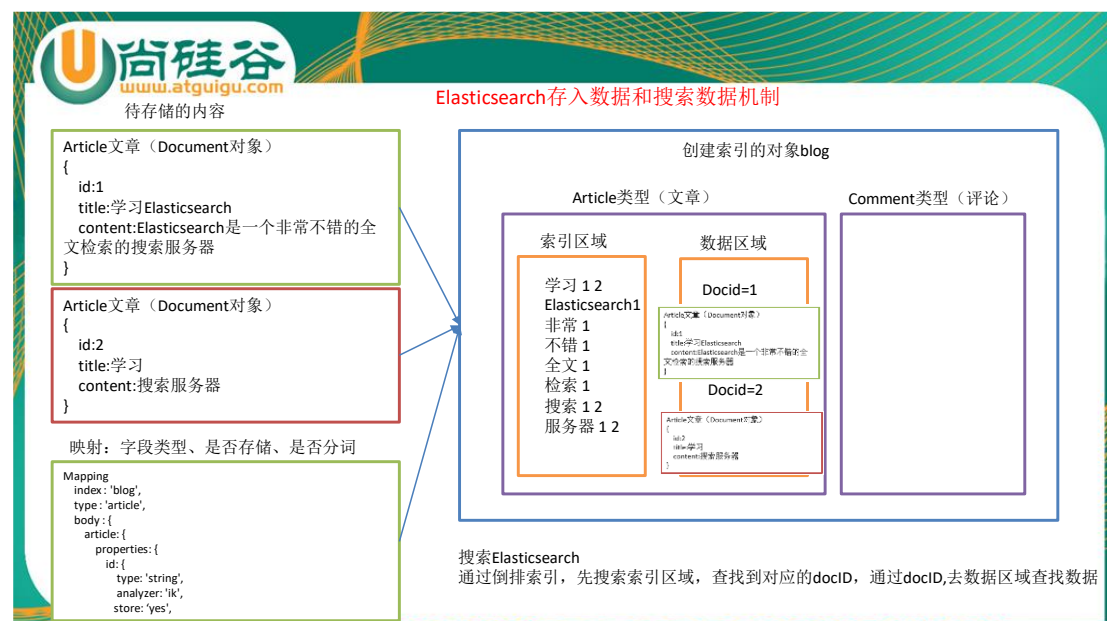
这样就创建了一个名为 `blog` 的 `Index`。`Type` 不用单独创建，在创建 `Mapping` 时指定就可以。`Mapping` 用来定义 `Document` 中每个字段的类型，即所使用的 `analyzer`、是否索引等属性，非常关键等。创建 `Mapping` 的代码示例如下：

```
client.indices.putMapping({  
  index : 'blog',  
  type : 'article',  
  body : {  
    article: {  
      properties: {  
        id: {  
          type: 'string',  
          analyzer: 'ik',  
          store: 'yes',  
        },  
        title: {  
          type: 'string',  
          analyzer: 'ik',  
          store: 'no',  
        },  
        content: {  
          type: 'string',  
          analyzer: 'ik',  
          store: 'yes',  
        }  
      }  
    }  
  }  
});
```

### 1.7.9 elasticsearch 与数据库的类比

关系型数据库（比如 Mysql）	非关系型数据库（Elasticsearch）
数据库 Database	索引 Index
表 Table	类型 Type
数据行 Row	文档 Document
数据列 Column	字段 Field
约束 Schema	映射 Mapping

### 1.7.10 ES 存入数据和搜索数据机制




- 1) 索引对象 (blog)：存储数据的表结构，任何搜索数据，存放在索引对象上。
- 2) 映射 (mapping)：数据如何存放到索引对象上，需要有一个映射配置，包括：数据类型、是否存储、是否分词等。
- 3) 文档 (document)：一条数据记录，存在索引对象上。
- 4) 文档类型 (type)：一个索引对象，存放多种类型数据，数据用文档类型进行标识。


## 二 快速入门

### 2.1 安装包下载

- 1) Elasticsearch 官网：<https://www.elastic.co/products/elasticsearch>

 [Products](#) [Cloud](#) [Services](#) [Customers](#) [Learn](#) [downloads](#) [contact](#) [Q](#) [EN](#)

Elasticsearch [Download](#) [As A Service](#)



The Heart of the Elastic Stack

Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data so you can discover the expected and uncover the unexpected.

[GA RELEASE](#) [PREVIEW RELEASE](#)

Version: 5.6.2

Release date: September 26, 2017

Notes: **IMPORTANT:** See [Multi data path bug in Elasticsearch 5.3.0](#) 以前的稳定版本  
View detailed [release notes](#).  
Not the version you're looking for? View [past releases.](#) 以前的稳定版本

Downloads: [ZIP sha](#) [TAR sha](#) [DEB sha](#)  
[RPM sha](#) [MSI sha](#)

## 2.2 安装 Elasticsearch（单节点 Linux 环境）

- 1) 解压 elasticsearch-5.2.2.tar.gz 到/opt/module 目录下

```
[atguigu@hadoop102 software]$ tar -zxvf elasticsearch-5.2.2.tar.gz -C /opt/module/
```

- 2) 在/opt/module/elasticsearch-5.2.2 路径下创建 data 和 logs 文件夹

```
[atguigu@hadoop102 elasticsearch-5.2.2]$ mkdir data
```

```
[atguigu@hadoop102 elasticsearch-5.2.2]$ mkdir logs
```

- 3) 修改配置文件/opt/module/elasticsearch-5.2.2/config/elasticsearch.yml

```
[atguigu@hadoop102 config]$ pwd
```

```
/opt/module/elasticsearch-5.2.2/config
```

```
[atguigu@hadoop102 config]$ vi elasticsearch.yml
```



```
# ----- Cluster -----
cluster.name: my-application
# ----- Node -----
node.name: node-102
# ----- Paths -----
path.data: /opt/module/elasticsearch-5.2.2/data
path.logs: /opt/module/elasticsearch-5.2.2/logs
# ----- Memory -----
bootstrap.memory_lock: false
bootstrap.system_call_filter: false
# ----- Network -----
network.host: 192.168.1.102
# ----- Discovery -----
discovery.zen.ping.unicast.hosts: ["hadoop102"]
```

(1) cluster.name

如果要配置集群需要两个节点上的 elasticsearch 配置的 cluster.name 相同，都启动可以自动组成集群，这里如果不改 cluster.name 则默认是 cluster.name=my-application，

(2) nodename 随意取但是集群内的各节点不能相同

(3) 修改后的每行前面不能有空格，修改后的“:”后面必须有一个空格

5) 配置 linux 系统环境（参考：<http://blog.csdn.net/satiling/article/details/59697916>）

(1) 切换到 root 用户，编辑 limits.conf 添加类似如下内容

```
[root@hadoop102 elasticsearch-5.2.2]# vi /etc/security/limits.conf
```

添加如下内容:

```
* soft nfile 65536
```

```
* hard nfile 131072
```

```
* soft nproc 2048
```

```
* hard nproc 4096
```

(2) 切换到 root 用户，进入 limits.d 目录下修改配置文件。

```
[root@hadoop102 elasticsearch-5.2.2]# vi /etc/security/limits.d/90-nproc.conf
```

修改如下内容:

```
* soft nproc 1024
```

```
#修改为
```

```
* soft nproc 2048
```

(3) 切换到 root 用户修改配置 sysctl.conf

---

```
[root@hadoop102 elasticsearch-5.2.2]# vi /etc/sysctl.conf
```

添加下面配置:

```
vm.max_map_count=655360
```

并执行命令:

```
[root@hadoop102 elasticsearch-5.2.2]# sysctl -p
```

然后, 重新启动 elasticsearch, 即可启动成功。

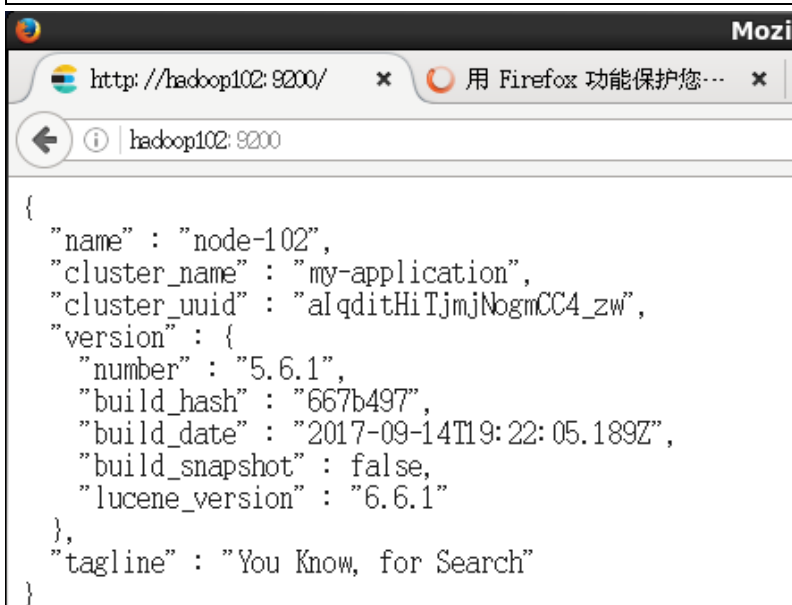
## 6) 启动集群

```
[atguigu@hadoop102 elasticsearch-5.2.2]$ bin/elasticsearch
```

## 7) 测试集群

```
[atguigu@hadoop102 elasticsearch-5.2.2]$ curl http://hadoop102:9200
```

```
{
  "name" : "node-102",
  "cluster_name" : "my-application",
  "cluster_uuid" : "v-nwhc7ITsmVHECpNQYzHw",
  "version" : {
    "number" : "5.2.2",
    "build_hash" : "57e20f3",
    "build_date" : "2017-09-23T13:16:45.703Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```



---

8) 停止集群

kill -9 进程号

## 2.3 安装 Elasticsearch（多节点集群 Linux 环境）

略

## 2.4 Elasticsearch head 插件安装

1) 下载插件

<https://github.com/mobz/elasticsearch-head>

elasticsearch-head-master.zip

2) nodejs 官网下载安装包

<https://nodejs.org/dist/>

node-v6.9.2-linux-x64.tar.xz

3) 将 elasticsearch-head-master.zip 和 node-v6.9.2-linux-x64.tar.xz 都导入到 linux 的 /opt/software 目录。

4) 安装 nodejs

[atguigu@hadoop102 software]\$ tar -zxvf node-v6.9.2-linux-x64.tar.gz -C /opt/module/

5) 配置 nodejs 环境变量

[root@hadoop102 software]# vi /etc/profile

export NODE\_HOME=/opt/module/node-v6.9.2-linux-x64

export PATH=\$PATH:\$NODE\_HOME/bin

[root@hadoop102 software]# source /etc/profile

6) 查看 node 和 npm 版本

[root@hadoop102 software]# node -v

v6.9.2

[root@hadoop102 software]# npm -v

3.10.9

7) 解压 head 插件到 /opt/module 目录下

[atguigu@hadoop102 software]\$ unzip elasticsearch-head-master.zip -d /opt/module/

8) 查看当前 head 插件目录下有无 node\_modules/grunt 目录:

没有: 执行命令创建:

---

```
[atguigu@hadoop102 elasticsearch-head-master]$ npm install grunt --save
```

9) 安装 head 插件:

```
[atguigu@hadoop102 elasticsearch-head-master]$ npm install -g cnpm
--registry=https://registry.npm.taobao.org
```

10) 安装 grunt:

```
[atguigu@hadoop102 elasticsearch-head-master]$ npm install -g grunt-cli
```

11) 编辑 Gruntfile.js

```
[atguigu@hadoop102 elasticsearch-head-master]$ vim Gruntfile.js
```

文件 93 行添加 hostname:'0.0.0.0'

```
options: {
    hostname:'0.0.0.0',
    port: 9100,
    base: '.',
    keepalive: true
}
```

12) 检查 head 根目录下是否存在 base 文件夹

没有: 将 \_site 下的 base 文件夹及其内容复制到 head 根目录下

```
[atguigu@hadoop102 elasticsearch-head-master]$ mkdir base
```

```
[atguigu@hadoop102 _site]$ cp base/* ../base/
```

13) 启动 grunt server:

```
[atguigu@hadoop102 elasticsearch-head-master]$ grunt server -d
```

Running "connect:server" (connect) task

[D]	Task	source:
-----	------	---------

```
/opt/module/elasticsearch-head-master/node_modules/grunt-contrib-connect/tasks/connect.js
```

Waiting forever...

Started connect web server on <http://localhost:9100>

如果提示 grunt 的模块没有安装:

Local Npm module "grunt-contrib-clean" not found. Is it installed?

---

Local Npm module “grunt-contrib-concat” not found. Is it installed?

Local Npm module “grunt-contrib-watch” not found. Is it installed?

Local Npm module “grunt-contrib-connect” not found. Is it installed?

Local Npm module “grunt-contrib-copy” not found. Is it installed?

Local Npm module “grunt-contrib-jasmine” not found. Is it installed?

Warning: Task “connect:server” not found. Use -force to continue.

执行以下命令：

```
npm install grunt-contrib-clean -registry=https://registry.npm.taobao.org
```

```
npm install grunt-contrib-concat -registry=https://registry.npm.taobao.org
```

```
npm install grunt-contrib-watch -registry=https://registry.npm.taobao.org
```

```
npm install grunt-contrib-connect -registry=https://registry.npm.taobao.org
```

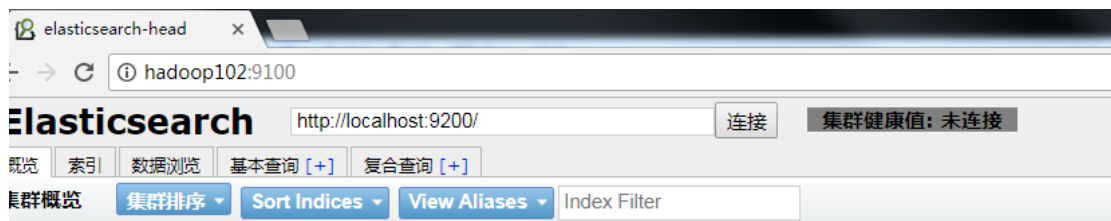
```
npm install grunt-contrib-copy -registry=https://registry.npm.taobao.org
```

```
npm install grunt-contrib-jasmine -registry=https://registry.npm.taobao.org
```

最后一个模块可能安装不成功，但是不影响使用。

14) 浏览器访问 head 插件：

<http://hadoop102:9100>



15) 启动集群插件后发现集群未连接

在/opt/module/elasticsearch-5.2.2/config 路径下修改配置文件 elasticsearch.yml，在文件末尾增加

```
[atguigu@hadoop102 config]$ pwd
```

```
/opt/module/elasticsearch-5.2.2/config
```

```
[atguigu@hadoop102 config]$ vi elasticsearch.yml
```

```
http.cors.enabled: true
```

```
http.cors.allow-origin: "*"

```

---

再重新启动 elasticsearch。

16) 关闭插件服务

ctrl+c

[atguigu@hadoop102 elasticsearch-head-master]\$ netstat -lntp | grep 9100

```
tcp        0      0 192.168.1.102:9100      0.0.0.0:*               LISTEN
6070/grunt
```

## 三 Java API 操作

Elasticsearch 的 Java 客户端非常强大；它可以建立一个嵌入式实例并在必要时运行管理任务。

运行一个 Java 应用程序和 Elasticsearch 时，有两种操作模式可供使用。该应用程序可在 Elasticsearch 集群中扮演更加主动或更加被动的角色。在更加主动的情况下（称为 **Node Client**），应用程序实例将从集群接收请求，确定哪个节点应处理该请求，就像正常节点所做的一样。（应用程序甚至可以托管索引和处理请求。）另一种模式称为 **Transport Client**，它将所有请求都转发到另一个 Elasticsearch 节点，由后者来确定最终目标。

### 3.1 API 基本操作

#### 3.1.1 操作环境准备

1) 创建 maven 工程

New Maven Project

Specify Archetype parameters

Group Id: com.atguigu

Artifact Id: elasticsearch\_test1

Version: 0.0.1-SNAPSHOT

Package: com.atguigu.elasticsearch\_test1

Properties available from archetype:

Name	Value

▶ Advanced

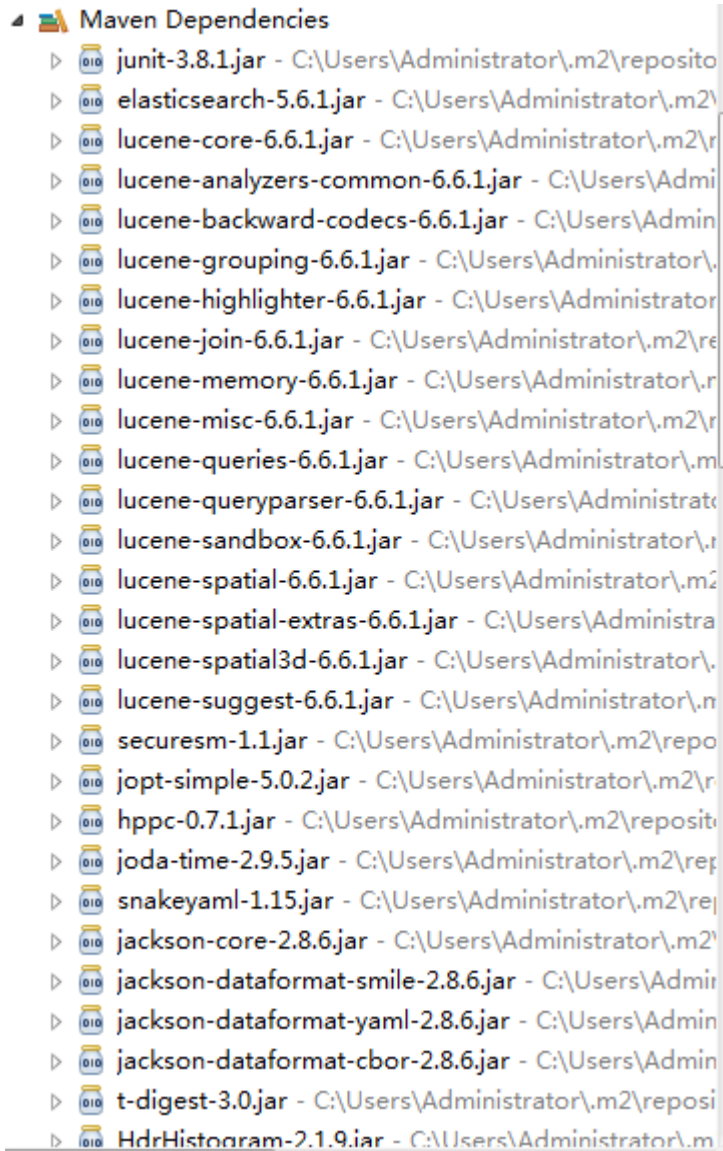
< Back Next > Finish Cancel

## 2) 添加 pom 文件

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>5.2.2</version>
  </dependency>
  <dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>transport</artifactId>
    <version>5.2.2</version>
  </dependency>
  <dependency>
```

```
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-core</artifactId>
<version>2.9.0</version>
</dependency>
</dependencies>
```

3) 等待依赖的 jar 包下载完成



当直接在 ElasticSearch 建立文档对象时，如果索引不存在的，默认会自动创建，映射采用默认方式

### 3.1.2 获取 Transport Client

- (1) ElasticSearch 服务默认端口 9300。
- (2) Web 管理平台端口 9200。

```
private TransportClient client;
```



```

@SuppressWarnings("unchecked")
@Before
public void getClient() throws Exception {

    // 1 设置连接的集群名称
    Settings settings = Settings.builder().put("cluster.name", "my-application").build();

    // 2 连接集群
    client = new PreBuiltTransportClient(settings);
    client.addTransportAddress(new
InetSocketAddress(InetAddress.getByName("hadoop102"), 9300));

    // 3 打印集群名称
    System.out.println(client.toString());
}

```

### 3.1.3 创建索引

#### 1) 源代码

```

@Test
public void createIndex_blog(){
    // 1 创建索引
    client.admin().indices().prepareCreate("blog2").get();

    // 2 关闭连接
    client.close();
}

```

#### 2) 查看结果

```

{"blog2":{"aliases":{},"mappings":{},"settings":{"index":{"creation_date":"1507466730030","number_of_shards":"5","number_of_replicas":"1","uuid":"lec0xYiBSmStspGVa6c80Q","version":{"created":"5060299"},"provided_name":"blog2"}}}}

```

### 3.1.4 删除索引

#### 1) 源代码

```

@Test
public void deleteIndex(){
    // 1 删除索引
    client.admin().indices().prepareDelete("blog2").get();

    // 2 关闭连接
    client.close();
}

```

```
}
```

## 2) 查看结果

浏览器查看 <http://hadoop102:9200/blog2>

没有 blog2 索引了。

```
{"error":{"root_cause":[{"type":"index_not_found_exception","reason":"no such index","resource.type":"index_or_alias","resource.id":"blog2","index_uuid":"_na_","index":"blog2"}],"type":"index_not_found_exception","reason":"no such index","resource.type":"index_or_alias","resource.id":"blog2","index_uuid":"_na_","index":"blog2"},"status":404}
```

### 3.1.5 新建文档（源数据 json 串）

当直接在 Elasticsearch 建立文档对象时，如果索引不存在的，默认会自动创建，映射采用默认方式。

ElasticSearch 服务默认端口 9300

Web 管理平台端口 9200

## 1) 源代码

```
@Test
public void createIndexByJson() throws UnknownHostException {

    // 1 文档数据准备
    String json = "{" + "\"id\": \"1\", " + "\"title\": \"基于 Lucene 的搜索服务器\", "
        + "\"content\": \"它提供了一个分布式多用户能力的全文搜索引擎, 基于 RESTful web 接口\"" + "}";

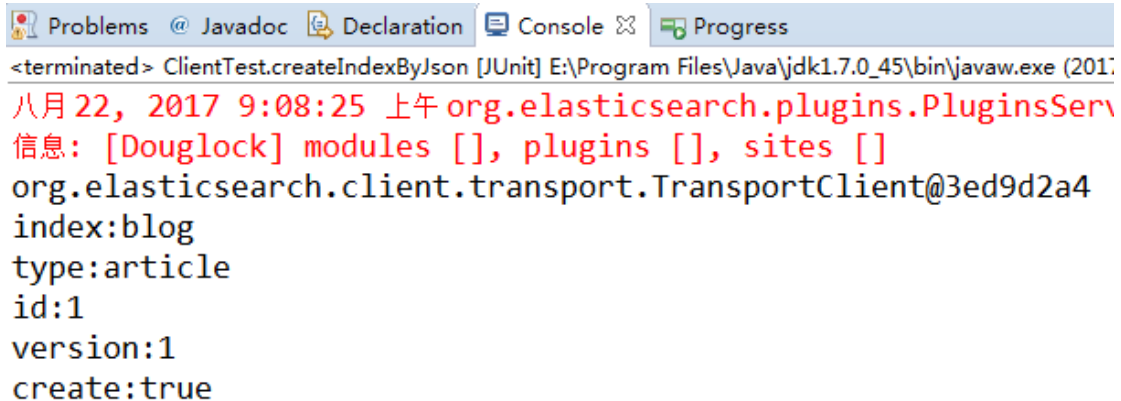
    // 2 创建文档
    IndexResponse indexResponse = client.prepareIndex("blog", "article", "1").setSource(json).execute().actionGet();

    // 3 打印返回的结果
    System.out.println("index:" + indexResponse.getIndex());
    System.out.println("type:" + indexResponse.getType());
    System.out.println("id:" + indexResponse.getId());
    System.out.println("version:" + indexResponse.getVersion());
    System.out.println("result:" + indexResponse.getResult());

    // 4 关闭连接
    client.close();
}
```

```
}
```

## 2) 结果查看



```
<terminated> ClientTest.createIndexByJson [JUnit] E:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2017年八月 22, 2017 9:08:25 上午 org.elasticsearch.plugins.PluginsService: info: [Douglock] modules [], plugins [], sites [] org.elasticsearch.client.transport.TransportClient@3ed9d2a4 index:blog type:article id:1 version:1 create:true
```

## 3.1.6 新建文档（源数据 map 方式添加 json）

### 1) 源代码

```
@Test
public void createIndexByMap() {

    // 1 文档数据准备
    Map<String, Object> json = new HashMap<String, Object>();
    json.put("id", "2");
    json.put("title", "基于 Lucene 的搜索服务器");
    json.put("content", "它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口");

    // 2 创建文档
    IndexResponse indexResponse = client.prepareIndex("blog", "article", "2").setSource(json).execute().actionGet();

    // 3 打印返回的结果
    System.out.println("index:" + indexResponse.getIndex());
    System.out.println("type:" + indexResponse.getType());
    System.out.println("id:" + indexResponse.getId());
    System.out.println("version:" + indexResponse.getVersion());
    System.out.println("result:" + indexResponse.getResult());

    // 4 关闭连接
    client.close();
}
```

## 2) 结果查看

Problems @ Javadoc Declaration Console Progress  
<terminated> ClientTest.createIndexByMap [JUnit] E:\Program Files\Java\jdk1.7.0\_45\bin\javaw.exe (2017年8月22日 上午9:4  
八月 22, 2017 9:40:56 上午 org.elasticsearch.plugins.PluginsService <init>  
信息: [Allison Blaire] modules [], plugins [], sites []  
org.elasticsearch.client.transport.TransportClient@369df65b  
index:blog  
type:article  
id:2  
version:1  
create:true

### 3.1.7 新建文档（源数据 es 构建器添加 json）

#### 1) 源代码

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

    // 1 通过 es 自带的帮助类，构建 json 数据
    XContentBuilder builder = XContentFactory.jsonBuilder().startObject().field("id",
3)
        .field("title", "基于 Lucene 的搜索服务器").field("content", "它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口。")
        .endObject();

    // 2 创建文档
    IndexResponse indexResponse = client.prepareIndex("blog", "article",
"3").setSource(builder).get();

    // 3 打印返回的结果
    System.out.println("index:" + indexResponse.getIndex());
    System.out.println("type:" + indexResponse.getType());
    System.out.println("id:" + indexResponse.getId());
    System.out.println("version:" + indexResponse.getVersion());
    System.out.println("result:" + indexResponse.getResult());

    // 4 关闭连接
    client.close();
}
```

#### 2) 结果查看

```
Problems @ Javadoc Declaration Console Progress
<terminated> ClientTest.createIndex [JUnit] E:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2017年8月22日 上午9:44:26)
八月 22, 2017 9:44:27 上午 org.elasticsearch.plugins.PluginsService <init>
信息: [Hugh Jones] modules [], plugins [], sites []
org.elasticsearch.client.transport.TransportClient@250bfff2d
index:blog
type:article
id:3
version:1
create:true
```

### 3.1.8 搜索文档数据（单个索引）

#### 1) 源代码

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

    // 1 查询文档
    GetResponse response = client.prepareGet("blog", "article", "1").get();

    // 2 打印搜索的结果
    System.out.println(response.getSourceAsString());

    // 3 关闭连接
    client.close();
}
```

#### 2) 结果查看

```
Problems @ Javadoc Declaration Console Progress
<terminated> ClientTest.getData [JUnit] E:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2017年8月22日 上午10:00:39)
八月 22, 2017 10:00:40 上午 org.elasticsearch.plugins.PluginsService <init>
信息: [Sigyn] modules [], plugins [], sites []
{"id": "1", "title": "基于Lucene的搜索服务器", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口"}
```

### 3.1.9 搜索文档数据（多个索引）

#### 1) 源代码

```
@Test
public void getMultiData() {

    // 1 查询多个文档
    MultiGetResponse response = client.prepareMultiGet().add("blog", "article", "1").add("blog", "article", "2", "3")
        .add("blog", "article", "2").get();

    // 2 遍历返回的结果
    for(MultiGetItemResponse itemResponse:response){
        GetResponse getResponse = itemResponse.getResponse();
    }
}
```

```

        // 如果获取到查询结果
        if (getResponse.isExists()) {
            String sourceAsString = getResponse.getSourceAsString();
            System.out.println(sourceAsString);
        }
    }

    // 3 关闭资源
    client.close();
}

```

## 2) 结果查看

```

{"id": "1", "title": "基于 Lucene 的搜索服务器", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口"}
{"content": "它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口", "id": "2", "title": "基于 Lucene 的搜索服务器"}
{"id": "3", "title": "ElasticSearch 是一个基于 Lucene 的搜索服务器", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口。"}
{"content": "它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口", "id": "2", "title": "基于 Lucene 的搜索服务器"}

```

### 3.1.10 更新文档数据 (update)

#### 1) 源代码

```

@Test
public void updateData() throws Throwable {

    // 1 创建更新数据的请求对象
    UpdateRequest updateRequest = new UpdateRequest();
    updateRequest.index("blog");
    updateRequest.type("article");
    updateRequest.id("3");

    updateRequest.doc(XContentFactory.jsonBuilder().startObject()
        // 对没有的字段添加, 对已有的字段替换
        .field("title", "基于 Lucene 的搜索服务器")
        .field("content",
            "它提供了一个分布式多用户能力的全文搜索引擎，基于
RESTful web 接口。大数据前景无限")
        .field("createDate", "2017-8-22").endObject());

    // 2 获取更新后的值
    UpdateResponse indexResponse = client.update(updateRequest).get();

    // 3 打印返回的结果
}

```

```

System.out.println("index:" + indexResponse.getIndex());
System.out.println("type:" + indexResponse.getType());
System.out.println("id:" + indexResponse.getId());
System.out.println("version:" + indexResponse.getVersion());
System.out.println("create:" + indexResponse.getResult());

// 4 关闭连接
client.close();
}

```

## 2) 结果查看



### 3.1.11 更新文档数据 (upsert)

设置查询条件，查找不到则添加 IndexRequest 内容，查找到则按照 UpdateRequest 更新。

```

@Test
public void testUpsert() throws Exception {

    // 设置查询条件，查找不到则添加
    IndexRequest indexRequest = new IndexRequest("blog", "article", "5")
        .source(XContentFactory.jsonBuilder().startObject().field("title", "搜索服务器").field("content", "它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口。Elasticsearch 是用 Java 开发的，并作为 Apache 许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。").endObject());

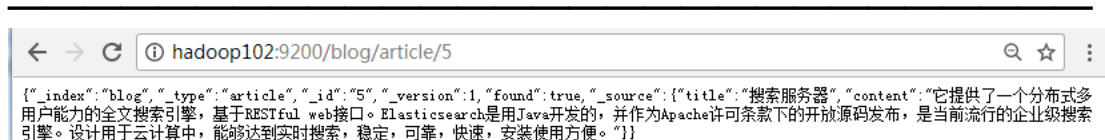
    // 设置更新，查找到更新下面的设置
    UpdateRequest upsert = new UpdateRequest("blog", "article", "5")
        .doc(XContentFactory.jsonBuilder().startObject().field("user", "李四").endObject()).upsert(indexRequest);

    client.update(upsert).get();
    client.close();
}

```

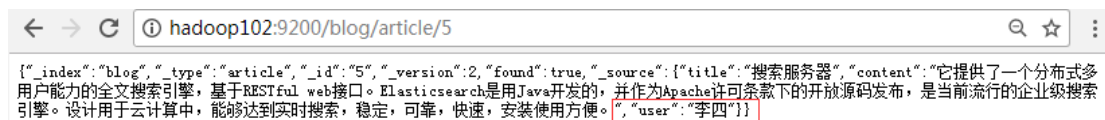
第一次执行

hadoop102:9200/blog/article/5



第二次执行

hadoop102:9200/blog/article/5



### 3.1.12 删除文档数据（prepareDelete）

1) 源代码

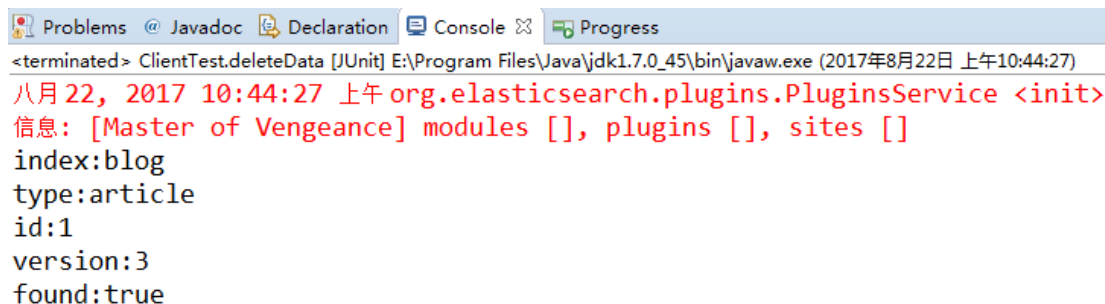
```
@Test
public void deleteData() {

    // 1 删除文档数据
    DeleteResponse indexResponse = client.prepareDelete("blog", "article", "5").get();

    // 2 打印返回的结果
    System.out.println("index:" + indexResponse.getIndex());
    System.out.println("type:" + indexResponse.getType());
    System.out.println("id:" + indexResponse.getId());
    System.out.println("version:" + indexResponse.getVersion());
    System.out.println("found:" + indexResponse.getResult());

    // 3 关闭连接
    client.close();
}
```

2) 结果查看



## 3.2 条件查询 QueryBuilder

### 3.2.1 查询所有（matchAllQuery）

1) 源代码

```
@Test
```



```

public void matchAllQuery() {

    // 1 执行查询
    SearchResponse searchResponse = client.prepareSearch("blog").setTypes("article")
        .setQuery(QueryBuilders.matchAllQuery()).get();

    // 2 打印查询结果
    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
    System.out.println("查询结果有： " + hits.getTotalHits() + "条");

    Iterator<SearchHit> iterator = hits.iterator();


    while (iterator.hasNext()) {
        SearchHit searchHit = iterator.next(); // 每个查询对象

        System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
    }

    // 3 关闭连接
    client.close();
}

```

## 2) 结果查看



```

<terminated> ClientTest.matchAllQuery [Unit] E:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2017年8月22日 上午11:31:37)
八月 22, 2017 11:31:38 上午 org.elasticsearch.plugins.PluginsService <init>
信息: [Chi Demon] modules [], plugins [], sites []
查询结果有： 2条
{"content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口","id": "2", "title": "基于Lucene的搜索服务器"}
{"id": 3, "title": "基于Lucene的搜索服务器", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。大数据前景无限"}

```

### 3.2.2 对所有字段分词查询（queryStringQuery）

#### 1) 源代码

```

@Test
public void query() {
    // 1 条件查询
    SearchResponse searchResponse = client.prepareSearch("blog").setTypes("article")
        .setQuery(QueryBuilders.queryStringQuery("全文")).get();

    // 2 打印查询结果
    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
    System.out.println("查询结果有： " + hits.getTotalHits() + "条");

    Iterator<SearchHit> iterator = hits.iterator();
}

```

```

        while (iterator.hasNext()) {
            SearchHit searchHit = iterator.next(); // 每个查询对象

            System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
        }

        // 3 关闭连接
        client.close();
    }

```

## 2) 结果查看

```

<terminated> ClientTestQuery [Unit] E:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2017年8月22日 上午11:12:13)
[八月 22, 2017 11:12:14 上午 org.elasticsearch.plugins.PluginsService <init>
信息: [Jack Kirby] modules [], plugins [], sites []
查询结果有: 2条
{"content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口", "id": "2", "title": "基于Lucene的搜索服务器"}
{"id": "3", "title": "基于Lucene的搜索服务器", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。大数据前景无限", "createDate": "2

```

### 3.2.3 通配符查询 (wildcardQuery)

\*: 表示多个字符 (任意的字符)

? : 表示单个字符

#### 1) 源代码

```

@Test
public void wildcardQuery() {

    // 1 通配符查询
    SearchResponse searchResponse = client.prepareSearch("blog").setTypes("article")
        .setQuery(QueryBuilders.wildcardQuery("content", "*全*")).get();

    // 2 打印查询结果
    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对
    System.out.println("查询结果有: " + hits.getTotalHits() + "条");

    Iterator<SearchHit> iterator = hits.iterator();

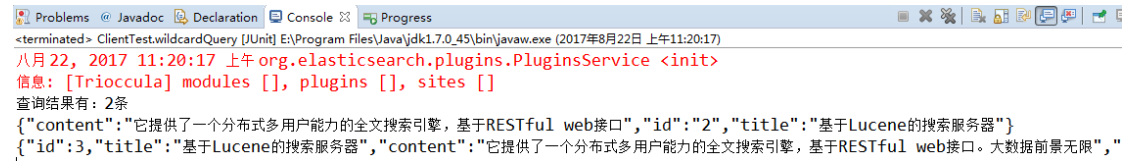
    while (iterator.hasNext()) {
        SearchHit searchHit = iterator.next(); // 每个查询对象

        System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
    }

    // 3 关闭连接
    client.close();
}

```

## 2) 结果查看



```
<terminated> ClientTest.wildcardQuery [JUnit] E:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2017年8月22日 上午11:20:17)
八月 22, 2017 11:20:17 上午 org.elasticsearch.plugins.PluginsService <init>
信息: [Trioocula] modules [], plugins [], sites []
查询结果有: 2条
{"content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口", "id": "2", "title": "基于Lucene的搜索服务器"}
{"id": "3", "title": "基于Lucene的搜索服务器", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。大数据前景无限", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。大数据前景无限", "id": "3", "title": "基于Lucene的搜索服务器"}
```

### 3.2.4 词条查询 (TermQuery)

#### 1) 源代码

```
@Test
public void termQuery() {

    // 1 第一 field 查询
    SearchResponse searchResponse = client.prepareSearch("blog").setTypes("article")
        .setQuery(QueryBuilders.termQuery("content", "全")).get();

    // 2 打印查询结果
    SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对象
    System.out.println("查询结果有: " + hits.getTotalHits() + "条");

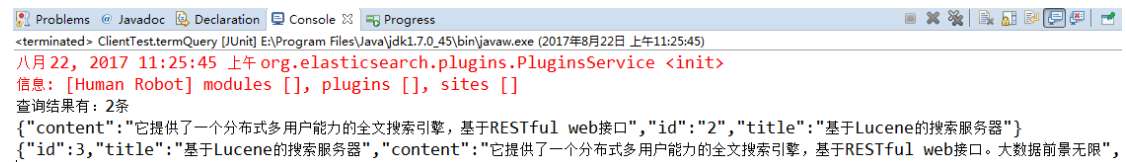
    Iterator<SearchHit> iterator = hits.iterator();

    while (iterator.hasNext()) {
        SearchHit searchHit = iterator.next(); // 每个查询对象

        System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
    }

    // 3 关闭连接
    client.close();
}
```

## 2) 结果查看



```
<terminated> ClientTest.termQuery [JUnit] E:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (2017年8月22日 上午11:25:45)
八月 22, 2017 11:25:45 上午 org.elasticsearch.plugins.PluginsService <init>
信息: [Human Robot] modules [], plugins [], sites []
查询结果有: 2条
{"content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口", "id": "2", "title": "基于Lucene的搜索服务器"}
{"id": "3", "title": "基于Lucene的搜索服务器", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。大数据前景无限", "content": "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。大数据前景无限", "id": "3", "title": "基于Lucene的搜索服务器"}
```

### 3.2.5 模糊查询 (fuzzy)

```
@Test
public void fuzzy() {

    // 1 模糊查询
    SearchResponse searchResponse = client.prepareSearch("blog").setTypes("article")
```

象

```
.setQuery(QueryBuilders.fuzzyQuery("title", "lucene")).get();

// 2 打印查询结果
SearchHits hits = searchResponse.getHits(); // 获取命中次数，查询结果有多少对

System.out.println("查询结果有: " + hits.getTotalHits() + "条");

Iterator<SearchHit> iterator = hits.iterator();

while (iterator.hasNext()) {
    SearchHit searchHit = iterator.next(); // 每个查询对象

    System.out.println(searchHit.getSourceAsString()); // 获取字符串格式打印
}

// 3 关闭连接
client.close();
}
```

### 3.3 映射相关操作

#### 1) 源代码

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

    // 1 设置 mapping
    XContentBuilder builder = XContentFactory.jsonBuilder()
        .startObject()
        .startObject("article")
        .startObject("properties")
        .startObject("id1")
        .field("type", "string")
        .field("store", "yes")
        .endObject()
        .startObject("title2")
        .field("type", "string")
        .field("store", "no")
        .endObject()
        .startObject("content")
        .field("type", "string")
        .field("store", "yes")
        .endObject()
        .endObject()
        .endObject()
}
```

```

        .endObject();

        // 2 添加 mapping
        PutMappingRequest mapping =
Requests.putMappingRequest("blog4").type("article").source(builder);

        client.admin().indices().putMapping(mapping).get();

        // 3 关闭资源
        client.close();
    }
}

```

## 2) 查看结果

