Elastic Search

# 什么是Elastic Search

Elasticsearch，分布式，高性能，高可用，可伸缩的搜索和分析系统

## 功能

### 分布式的搜索引擎和数据分析引擎

搜索：百度，网站的站内搜索，IT系统的检索

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

分布式，搜索，数据分析

### 全文检索，结构化检索，数据分析

全文检索：我想搜索商品名称包含牙膏的商品，select \* from products where product\_name like "%牙膏%"

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'

部分匹配、自动完成、搜索纠错、搜索推荐

数据分析：我们分析每一个商品分类下有多少个商品，select category\_id,count(\*) from products group by category\_id

### 对海量数据进行近实时的处理

分布式：ES自动可以将海量数据分散到多台服务器上去存储和检索

海联数据的处理：分布式以后，就可以采用大量的服务器去存储和检索数据，自然而然就可以实现海量数据的处理了

近实时：检索个数据要花费1小时（这就不要近实时，离线批处理，batch-processing）；在秒级别对数据进行搜索和分析

跟分布式/海量数据相反的：lucene，单机应用，只能在单台服务器上使用，最多只能处理单台服务器可以处理的数据量

## 适用场景

#### 国外

（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）电商网站，检索商品

（6）日志数据分析，logstash采集日志，ES进行复杂的数据分析（ELK技术，elasticsearch+logstash+kibana）

（7）商品价格监控网站，用户设定某商品的价格阈值，当低于该阈值的时候，发送通知消息给用户，比如说订阅牙膏的监控，如果高露洁牙膏的家庭套装低于50块钱，就通知我，我就去买

（8）BI系统，商业智能，Business Intelligence。比如说有个大型商场集团，BI，分析一下某某区域最近3年的用户消费金额的趋势以及用户群体的组成构成，产出相关的数张报表，\*\*区，最近3年，每年消费金额呈现100%的增长，而且用户群体85%是高级白领，开一个新商场。ES执行数据分析和挖掘，Kibana进行数据可视化

#### 国内

（9）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）

## 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，提供了数据库所不不能提供的很多功能

## elasticsearch的核心概念

**（1）Near Realtime（NRT）：**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

**（2）Cluster：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

**（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

**（4）Document&field**：文档，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。一个document里面有多个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"

}

**（5）Index：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。

**（6）Type：**类型，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": "2",

"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": "3",

"product\_name": "基围虾",

"product\_desc": "纯天然，冰岛产",

"category\_id": "4",

"category\_name": "生鲜",

"eat\_period": "7天"

}

**（7）shard：**单台机器无法存储大量数据，es可以将一个索引中的数据切分为多个shard，分布在多台服务器上存储。有了shard就可以横向扩展，存储更多数据，让搜索和分析等操作分布到多台服务器上去执行，提升吞吐量和性能。每个shard都是一个lucene index。

**（8）replica：**任何一个服务器随时可能故障或宕机，此时shard可能就会丢失，因此可以为每个shard创建多个replica副本。replica可以在shard故障时提供备用服务，保证数据不丢失，多个replica还可以提升搜索操作的吞吐量和性能。primary shard（建立索引时一次设置，不能修改，默认5个），replica shard（随时修改数量，默认1个），默认每个索引10个shard，5个primary shard，5个replica shard，最小的高可用配置，是2台服务器。

----------------------------------------------------------------------------------------------------------------------------------------

3、elasticsearch核心概念 vs. 数据库核心概念

Elasticsearch 数据库

Document 行

Type 表

Index 库

## Windows 上安装

1、安装JDK，至少1.8.0\_73以上版本，java -version

2、下载和解压缩Elasticsearch安装包，目录结构

3、启动Elasticsearch：bin\elasticsearch.bat，es本身特点之一就是开箱即用，如果是中小型应用，数据量少，操作不是很复杂，直接启动就可以用了

4、检查ES是否启动成功：http://localhost:9200/?pretty

name: node名称

cluster\_name: 集群名称（默认的集群名称就是elasticsearch）

version.number: 5.2.0，es版本号

{

"name" : "XcFkU8D",

"cluster\_name" : "elasticsearch",

"cluster\_uuid" : "Mr7HydNCQZy3sxBse8nZKw",

"version" : {

"number" : "5.2.0",

"build\_hash" : "24e05b9",

"build\_date" : "2017-01-24T19:52:35.800Z",

"build\_snapshot" : false,

"lucene\_version" : "6.4.0"

},

"tagline" : "You Know, for Search"

}

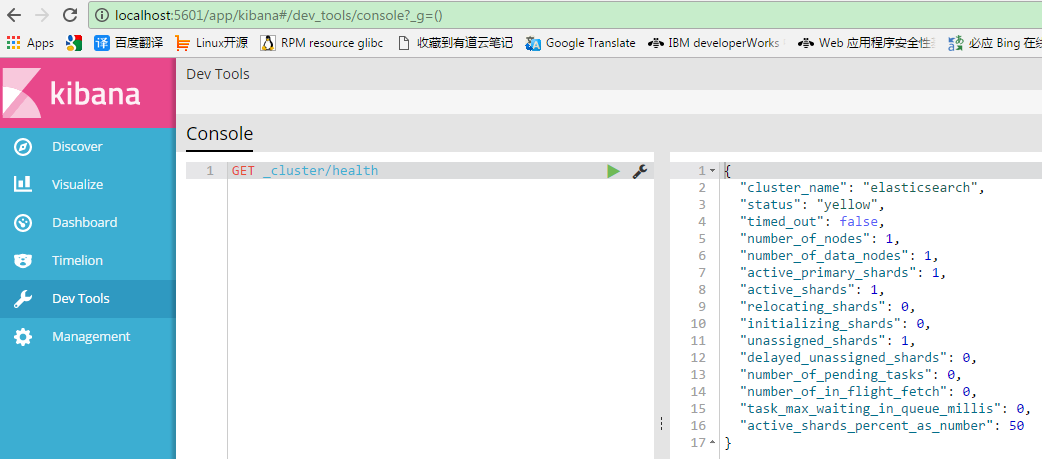
5、修改集群名称：elasticsearch.yml

6、下载和解压缩Kibana安装包，使用里面的开发界面，去操作elasticsearch，作为我们学习es知识点的一个主要的界面入口

7、启动Kibana：bin\kibana.bat

8、进入Dev Tools界面

9、GET \_cluster/health



# 相关知识和功能点

## document数据格式

面向文档的搜索分析引擎

（1）应用系统的数据结构都是面向对象的，复杂的

（2）对象数据存储到数据库中，只能拆解开来，变为扁平的多张表，每次查询的时候还得还原回对象格式，相当麻烦

（3）ES是面向文档的，文档中存储的数据结构，与面向对象的数据结构是一样的，基于这种文档数据结构，es可以提供复杂的索引，全文检索，分析聚合等功能

（4）es的document用json数据格式来表达

public class Employee {

private String email;

private String firstName;

private String lastName;

private EmployeeInfo info;

private Date joinDate;

}

private class EmployeeInfo {

private String bio; // 性格

private Integer age;

private String[] interests; // 兴趣爱好

}

EmployeeInfo info = new EmployeeInfo();

info.setBio("curious and modest");

info.setAge(30);

info.setInterests(new String[]{"bike", "climb"});

Employee employee = new Employee();

employee.setEmail("zhangsan@sina.com");

employee.setFirstName("san");

employee.setLastName("zhang");

employee.setInfo(info);

employee.setJoinDate(new Date());

employee对象：里面包含了Employee类自己的属性，还有一个EmployeeInfo对象

两张表：employee表，employee\_info表，将employee对象的数据重新拆开来，变成Employee数据和EmployeeInfo数据

employee表：email，first\_name，last\_name，join\_date，4个字段

employee\_info表：bio，age，interests，3个字段；此外还有一个外键字段，比如employee\_id，关联着employee表

{

"email": "zhangsan@sina.com",

"first\_name": "san",

"last\_name": "zhang",

"info": {

"bio": "curious and modest",

"age": 30,

"interests": [ "bike", "climb" ]

},

"join\_date": "2017/01/01"

}

我们就明白了es的document数据格式和数据库的关系型数据格式的区别

------------------------------------------------------------------------------------------------------

## 电商网站商品管理案例：背景介绍

有一个电商网站，需要为其基于ES构建一个后台系统，提供以下功能：

（1）对商品信息进行CRUD（增删改查）操作

（2）执行简单的结构化查询

（3）可以执行简单的全文检索，以及复杂的phrase（短语）检索

（4）对于全文检索的结果，可以进行高亮显示

（5）对数据进行简单的聚合分析

## 简单的集群管理

### 快速检查集群的健康状况

es提供了一套api，叫做cat api，可以查看es中各种各样的数据

GET /\_cat/health?v

epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent

1488006741 15:12:21 elasticsearch yellow 1 1 1 1 0 0 1 0 - 50.0%

epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent

1488007113 15:18:33 elasticsearch green 2 2 2 1 0 0 0 0 - 100.0%

epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent

1488007216 15:20:16 elasticsearch yellow 1 1 1 1 0 0 1 0 - 50.0%

如何快速了解集群的健康状况？green、yellow、red？

green：每个索引的primary shard和replica shard都是active状态的

yellow：每个索引的primary shard都是active状态的，但是部分replica shard不是active状态，处于不可用的状态

red：不是所有索引的primary shard都是active状态的，部分索引有数据丢失了

#### 为什么现在会处于一个yellow状态？

我们现在就一个笔记本电脑，就启动了一个es进程，相当于就只有一个node。现在es中有一个index，就是kibana自己内置建立的index。由于默认的配置是给每个index分配5个primary shard和5个replica shard，而且primary shard和replica shard不能在同一台机器上（为了容错）。现在kibana自己建立的index是1个primary shard和1个replica shard。当前就一个node，所以只有1个primary shard被分配了和启动了，但是一个replica shard没有第二台机器去启动。

*🌢♉做一个小实验：此时只要启动第二个es进程，就会在es集群中有2个node，然后那1个replica shard就会自动分配过去，然后cluster status就会变成green状态*。

### 快速查看集群中有哪些索引

GET /\_cat/indices?v

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open .kibana rUm9n9wMRQCCrRDEhqneBg 1 1 1 0 3.1kb 3.1kb

（3）简单的索引操作

创建索引：PUT /test\_index?pretty

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open test\_index XmS9DTAtSkSZSwWhhGEKkQ 5 1 0 0 650b 650b

yellow open .kibana rUm9n9wMRQCCrRDEhqneBg 1 1 1 0 3.1kb 3.1kb

### 删除索引：DELETE /test\_index?pretty

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open .kibana rUm9n9wMRQCCrRDEhqneBg 1 1 1 0 3.1kb 3.1kb

## 商品的CRUD操作（document CRUD操作）

### 新增商品：新增文档，建立索引

PUT /index/type/id

{

"json数据"

}

PUT /ecommerce/product/1

{

"name" : "gaolujie yagao",

"desc" : "gaoxiao meibai",

"price" : 30,

"producer" : "gaolujie producer",

"tags": [ "meibai", "fangzhu" ]

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 1,

"result": "created",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": true

}

PUT /ecommerce/product/2

{

"name" : "jiajieshi yagao",

"desc" : "youxiao fangzhu",

"price" : 25,

"producer" : "jiajieshi producer",

"tags": [ "fangzhu" ]

}

PUT /ecommerce/product/3

{

"name" : "zhonghua yagao",

"desc" : "caoben zhiwu",

"price" : 40,

"producer" : "zhonghua producer",

"tags": [ "qingxin" ]

}

es会自动建立index和type，不需要提前创建，而且es默认会对document每个field都建立倒排索引，让其可以被搜索

### 查询商品：检索文档

GET /index/type/id

GET /ecommerce/product/1

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 1,

"found": true,

"\_source": {

"name": "gaolujie yagao",

"desc": "gaoxiao meibai",

"price": 30,

"producer": "gaolujie producer",

"tags": [

"meibai",

"fangzhu"

]

}

}

### 修改商品：替换文档

PUT /ecommerce/product/1

{

"name" : "jiaqiangban gaolujie yagao",

"desc" : "gaoxiao meibai",

"price" : 30,

"producer" : "gaolujie producer",

"tags": [ "meibai", "fangzhu" ]

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 1,

"result": "created",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": true

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 2,

"result": "updated",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": false

}

PUT /ecommerce/product/1

{

"name" : "jiaqiangban gaolujie yagao"

}

替换方式有一个不好，即使必须带上所有的field，才能去进行信息的修改

### 修改商品：更新文档

POST /ecommerce/product/1/\_update

{

"doc": {

"name": "jiaqiangban gaolujie yagao"

}

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 8,

"result": "updated",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

}

}

### 删除商品：删除文档

DELETE /ecommerce/product/1

{

"found": true,

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 9,

"result": "deleted",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

}

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"found": false

}

## 搜索方式

### query string search

搜索全部商品：GET /ecommerce/product/\_search

took：耗费了几毫秒

timed\_out：是否超时，这里是没有

\_shards：数据拆成了5个分片，所以对于搜索请求，会打到所有的primary shard（或者是它的某个replica shard也可以）

hits.total：查询结果的数量，3个document

hits.max\_score：score的含义，就是document对于一个search的相关度的匹配分数，越相关，就越匹配，分数也高

hits.hits：包含了匹配搜索的document的详细数据

{

"took": 2,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 3,

"max\_score": 1,

"hits": [

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "2",

"\_score": 1,

"\_source": {

"name": "jiajieshi yagao",

"desc": "youxiao fangzhu",

"price": 25,

"producer": "jiajieshi producer",

"tags": [

"fangzhu"

]

}

},

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_score": 1,

"\_source": {

"name": "gaolujie yagao",

"desc": "gaoxiao meibai",

"price": 30,

"producer": "gaolujie producer",

"tags": [

"meibai",

"fangzhu"

]

}

},

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "3",

"\_score": 1,

"\_source": {

"name": "zhonghua yagao",

"desc": "caoben zhiwu",

"price": 40,

"producer": "zhonghua producer",

"tags": [

"qingxin"

]

}

}

]

}

}

query string search的由来，因为search参数都是以http请求的query string来附带的

搜索商品名称中包含yagao的商品，而且按照售价降序排序：GET /ecommerce/product/\_search?q=name:yagao&sort=price:desc

适用于临时的在命令行使用一些工具，比如curl，快速的发出请求，来检索想要的信息；但是如果查询请求很复杂，是很难去构建的

在生产环境中，几乎很少使用query string search

### query DSL

DSL：Domain Specified Language，特定领域的语言

http request body：请求体，可以用json的格式来构建查询语法，比较方便，可以构建各种复杂的语法，比query string search肯定强大多了

查询所有的商品

GET /ecommerce/product/\_search

{

"query": { "match\_all": {} }

}

查询名称包含yagao的商品，同时按照价格降序排序

GET /ecommerce/product/\_search

{

"query" : {

"match" : {

"name" : "yagao"

}

},

"sort": [

{ "price": "desc" }

]

}

分页查询商品，总共3条商品，假设每页就显示1条商品，现在显示第2页，所以就查出来第2个商品

GET /ecommerce/product/\_search

{

"query": { "match\_all": {} },

"from": 1,

"size": 1

}

指定要查询出来商品的名称和价格就可以

GET /ecommerce/product/\_search

{

"query": { "match\_all": {} },

"\_source": ["name", "price"]

}

更加适合生产环境的使用，可以构建复杂的查询

### query filter

搜索商品名称包含yagao，而且售价大于25元的商品

GET /ecommerce/product/\_search

{

"query" : {

"bool" : {

"must" : {

"match" : {

"name" : "yagao"

}

},

"filter" : {

"range" : {

"price" : { "gt" : 25 }

}

}

}

}

}

### full-text search（全文检索）

GET /ecommerce/product/\_search

{

"query" : {

"match" : {

"producer" : "yagao producer"

}

}

}

producer这个字段，会先被拆解，建立倒排索引

special 4

yagao 4

producer 1,2,3,4

gaolujie 1

zhognhua 3

jiajieshi 2

yagao producer ---> yagao和producer

{

"took": 4,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 4,

"max\_score": 0.70293105,

"hits": [

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "4",

"\_score": 0.70293105,

"\_source": {

"name": "special yagao",

"desc": "special meibai",

"price": 50,

"producer": "special yagao producer",

"tags": [

"meibai"

]

}

},

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_score": 0.25811607,

"\_source": {

"name": "gaolujie yagao",

"desc": "gaoxiao meibai",

"price": 30,

"producer": "gaolujie producer",

"tags": [

"meibai",

"fangzhu"

]

}

},

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "3",

"\_score": 0.25811607,

"\_source": {

"name": "zhonghua yagao",

"desc": "caoben zhiwu",

"price": 40,

"producer": "zhonghua producer",

"tags": [

"qingxin"

]

}

},

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "2",

"\_score": 0.1805489,

"\_source": {

"name": "jiajieshi yagao",

"desc": "youxiao fangzhu",

"price": 25,

"producer": "jiajieshi producer",

"tags": [

"fangzhu"

]

}

}

]

}

}

### phrase search（短语搜索）

跟全文检索相对应，相反，全文检索会将输入的搜索串拆解开来，去倒排索引里面去一一匹配，只要能匹配上任意一个拆解后的单词，就可以作为结果返回

phrase search，要求输入的搜索串，必须在指定的字段文本中，完全包含一模一样的，才可以算匹配，才能作为结果返回

GET /ecommerce/product/\_search

{

"query" : {

"match\_phrase" : {

"producer" : "yagao producer"

}

}

}

{

"took": 11,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 1,

"max\_score": 0.70293105,

"hits": [

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "4",

"\_score": 0.70293105,

"\_source": {

"name": "special yagao",

"desc": "special meibai",

"price": 50,

"producer": "special yagao producer",

"tags": [

"meibai"

]

}

}

]

}

}

### highlight search（高亮搜索结果）

GET /ecommerce/product/\_search

{

"query" : {

"match" : {

"producer" : "producer"

}

},

"highlight": {

"fields" : {

"producer" : {}

}

}

}