# 【Elasticsearch】

## 主要内容

1. Elasticsearch简介
2. Linux安装Elasticsearch
3. 常用Elasticsearch管理操作
4. 常用元数据简介
5. 分词器和标准化处理
6. Elasticsearch中的mapping问题
7. Search搜索详解

## 学习目标

|  |  |
| --- | --- |
| 知识点 | 要求 |
| Elasticsearch简介 | 掌握 |
| Linx安装Elasticsearch | 掌握 |
| 常用Elasticsearch管理操作 | 掌握 |
| 常用元数据简介 | 掌握 |
| 分词器和标准化处理 | 掌握 |
| Elasticsearch中的mapping问题 | 掌握 |
| Search搜索详解 | 掌握 |

## Elasticsearch简介

Elasticsearch是一个基于Lucene的搜索服务器。它提供了一个分布式的全文搜索引擎，其对外服务是基于RESTful web接口发布的。Elasticsearch是用Java开发的应用，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，能够达到近实时搜索，稳定，可靠，快速，安装使用方便。

### 相关概念

#### cluster

集群。Elasticsearch集群由一或多个节点组成，其中有一个主节点，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。Elasticsearch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，因为从外部看Elasticsearch集群，在逻辑上是个整体，你与集群中的任何一个节点通信和与整个Elasticsearch集群通信是等价的。也就是说，主节点的存在不会产生单点安全隐患、并发访问瓶颈等问题。

#### shards

primary shard：代表索引的主分片，Elasticsearch可以把一个完整的索引分成多个primary shard，这样的好处是可以把一个大的索引拆分成多个分片，分布存储在不同的Elasticsearch节点上，从而形成分布式存储，并为搜索访问提供分布式服务，提高并发处理能。primary shard的数量只能在索引创建时指定，并且索引创建后不能再更改primary shard数量。

#### replicas

replica shard：代表索引主分片的副本，Elasticsearch可以设置多个replica shard。replica shard的作用：一是提高系统的容错性，当某个节点某个primary shard损坏或丢失时可以从副本中恢复。二是提高Elasticsearch的查询效率，Elasticsearch会自动对搜索请求进行负载均衡，将并发的搜索请求发送给合适的节点，增强并发处理能力。

#### Index

索引。相当于关系型数据库中的表。其中存储若干相似结构的Document数据。如：客户索引，订单索引，商品索引等。Elasticsearch中的索引不像数据库表格一样有强制的数据结构约束，在理论上，可以存储任意结构的数据。但了为更好的为业务提供搜索数据支撑，还是要设计合适的索引体系来存储不同的数据。

#### Type

类型。每个索引中都必须有唯一的一个Type，Type是Index中的一个逻辑分类。Elasticsearch中的数据Document是存储在索引下的Type中的。

注意：Elasticsearch5.x及更低版本中，一个Index中可以有多个Type。Elasticsearch6.x版本之后，type概念被弱化，一个index中只能有唯一的一个type。且在7.x版本之后，删除type定义。

#### Document

文档。Elasticsearch中的最小数据单元。一个Document就是一条数据，一般使用JSON数据结构表示。每个Index下的Type中都可以存储多个Document。一个Document中可定义多个field，field就是数据字段。如：学生数据（{"name":"张三", "age":20, "gender":"男"}）。

#### 倒排索引

对数据进行分析，抽取出数据中的词条，以词条作为key，对应数据的存储位置作为value，实现索引的存储。这种索引称为倒排索引。倒排索引是Document写入Elasticsearch时分析维护的。

如：

|  |  |  |
| --- | --- | --- |
| 数据 | | |
| 商品主键 | 商品名 | 商品描述 |
| 1 | 荣耀10 | 更贵的手机 |
| 2 | 荣耀8 | 相对便宜的手机 |
| 3 | IPHONE X | 要卖肾买的手机 |

|  |  |
| --- | --- |
| 分析结果、倒排索引 | |
| 词条 | 数据 |
| 手机 | 1，2，3 |
| 便宜 | 2 |
| 卖肾 | 3 |
| 相对 | 2 |
| 荣耀 | 1，2 |
| IPHONE | 3 |

### Elasticsearch常见使用场景

维基百科：全文检索，高亮显示，搜索推荐

The Guardian（国外的一个新闻网站），此平台可以对用户的行为（点击、浏览、收藏、评论）、社区网络数据（对新闻的评论等）进行数据分析，为新闻的发布者提供相关的公众反馈。

Stack Overflow（国外的程序异常讨论论坛）

Github（开源代码管理），在千亿级别的代码行中搜索信息

电子商务平台等。

### 为什么不用数据库做搜索？

#### 查询语法复杂度高。

如：电商系统中搜索商品数据 - select \* from products where name like '%关键字%' and price bewteen xxx and yyy and ......。不同的用户提供的查询条件不同，需要提供的动态SQL过于复杂。

#### 关键字索引不全面，搜索结果不符合要求

如：电商系统中查询商品数据，条件为商品名包含'笔记本电脑'。那么对此关键字的分析结果为-笔记本、电脑、笔记等。对应的查询语法应该为 - select \* from products where name like '%笔记本%' or name like '%电脑%' .......

#### 效率问题

数据量越大，查询反应效率越低。

## Linux安装Elasticsearch

使用的Elasticsearch的版本是6.8.4。Elasticsearch6.x要求Linux内核必须是3.5+版本以上。

在linux操作系统中，查看内核版本的命令是： uname -a

课堂使用的Linux是CentOS8。内核使用的是4.18。

### 为Elasticsearch提供完善的系统配置

Elasticsearch在Linux中安装部署的时候，需要系统为其提供若干系统配置。如：应用可启动的线程数、应用可以在系统中划分的虚拟内存、应用可以最多创建多少文件等。

#### 修改限制信息

vim /etc/security/limits.conf

是修改系统中允许应用最多创建多少文件等的限制权限。Linux默认来说，一般限制应用最多创建的文件是65535个。但是Elasticsearch至少需要65536的文件创建权限。修改后的内容为：

\* soft nofile 65536

\* hard nofile 65536

\*代表任意用户，soft表示内存中虚拟文件（软文件），hard表示落地到磁盘的具体文件（硬文件）， nofile表示权限，65536表示个数。

#### 修改线程开启限制

在CentOS6.5版本中编辑下述的配置文件

vim /etc/security/limits.d/90-nproc.conf

在CentOS7+版本中编辑配置文件是：

vim /etc/security/limits.conf

是修改系统中允许用户启动的进程开启多少个线程。默认的Linux限制root用户开启的进程可以开启任意数量的线程，其他用户开启的进程可以开启1024个线程。必须修改限制数为4096+。因为Elasticsearch至少需要4096的线程池预备。Elasticsearch在5.x版本之后，强制要求在linux中不能使用root用户启动Elasticsearch进程。所以必须使用其他用户启动Elasticsearch进程才可以。

\* soft nproc 4096

root soft nproc unlimited

\*任何用户 nproc创建线程 数量4096

注意：Linux低版本内核为线程分配的内存是128K。4.x版本的内核分配的内存更大。如果虚拟机的内存是1G，最多只能开启3000+个线程数。**至少为虚拟机分配1.5G以上的内存，保险起见建议2G以上。**

#### 修改系统控制权限

CentOS6.5中的配置文件为：

vim /etc/sysctl.conf

CentOS8中的配置文件为：

vim /etc/sysctl.d/99-sysctl.conf

系统控制文件是管理系统中的各种资源控制的配置文件。Elasticsearch需要开辟一个65536字节以上空间的虚拟内存。Linux默认不允许任何用户和应用直接开辟虚拟内存。

新增内容为：

vm.max\_map\_count=655360

**使用命令： sysctl -p** 让系统控制权限配置生效。

### 安装Elasticsearch

Elasticsearch是java开发的应用。在6.8.4版本中，要求JDK至少是1.8.0\_131版本以上。

Elasticsearch的安装过程非常简单。解压立刻可以使用。

#### 解压缩安装压缩包

tar -zxf Elasticsearch-6.8.4.tar.gz

#### 移动Elasticsearch

mv Elasticsearch-6.8.4 /usr/local/es/

#### 修改Elasticsearch应用的所有者

因为Elasticsearch不允许root用户启动，而课堂案例中，Elasticsearch是root用户解压缩的。所以解压后的Elasticsearch应用属于root用户。所以我们需要将Elasticsearch应用的所有者修改为其他用户。当前课堂案例中虚拟机Linux内有smallming这个用户。

chown -R smallming.smallming /usr/local/es

可以通过：groups 用户名 查看用户所在用户组

#### 切换用户

root向其他用户切换时不是需要写密码，但是其他向root切换需要有密码。

su切换登录。

su smallming

#### 修改配置

修改config/Elasticsearch的配置文件，设置可访问的客户端。0.0.0.0代表任意客户端访问。

注意：如果不小心使用root修改此文件，会在config下生成elasticsearch.keystore文件，通过ll查看该文件是否是root操作权限，如果是一定要删除，否则启动时会报此文件无权限。

vim config/elasticsearch.yml

增加下述内容：

network.host: 0.0.0.0

#### 启动

**注意：**

**如果不小心使用root启动，一定要去logs文件夹中通过ll查看文件权限，所有root的文件都需要删除。**

前台启动

./bin/elasticsearch

关闭： ctrl + c

后台启动

./elasticsearch -d

关闭：

jps 命令查看Elasticsearch线程的编号

kill -9 Elasticsearch线程编号

#### 测试连接

curl http://localhost:9200

返回如下结果：

{

"name" : "L6WdN7y",

"cluster\_name" : "Elasticsearch",

"cluster\_uuid" : "s7\_GSd9YQnaH10VQBKCQ5w",

"version" : {

"number" : "6.3.1",

"build\_flavor" : "default",

"build\_type" : "tar",

"build\_hash" : "eb782d0",

"build\_date" : "2018-06-29T21:59:26.107521Z",

"build\_snapshot" : false,

"lucene\_version" : "7.3.1",

"minimum\_wire\_compatibility\_version" : "5.6.0",

"minimum\_index\_compatibility\_version" : "5.0.0"

},

"tagline" : "You Know, for Search"

}

curl <http://localhost:9200/_cat/nodes> 获取节点信息

curl <http://localhost:9200/_cat/shards> 获取分片信息

curl <http://localhost:9200/_cat/indices> 获取索引信息

curl <http://localhost:9200/_cat/health?v> 获取健康状态信息

### 搭建集群

elasticsearch伪集群特别好安装，只需要把安装的es文件在复制一份（保证smallming用户具有操作权限）后执行运行即可，启动后会自动判断端口是否被占用，如果占用端口递增1.但是需要注意最多递增到9299。

注意：

**复制后的es文件夹要删除data目录，否则无法启动集群。复制后和原文件夹都要删除。**

### 安装Kibana

Kibana是一个基于WEB的Elasticsearch管理控制台。现阶段安装Kibana主要是为了方便学习。

在Linux中安装Kibana很方便。解压，启动即可。Kibana要求的环境配置是小于Elasticsearch的要求的。

tar -zxf kibana-6.3.1-linux-x86\_64.tar.gz

修改config/kibana.yml

vim config/kibana.yml

新增内容： server.host: "0.0.0.0"

bin/kibana

Kibana在启动时会扫描操作系统本地的9200~9299端口。查看是否有ES在运行，如果有则自动连接ES。

访问时，使用浏览器访问http://192.168.8.134:5601/

## 常用Elasticsearch管理操作

### 查看健康状态

GET \_cat/health?v

|  |
| --- |
| Epoch（编号） timestamp（时间戳） cluster（集群名称） status（健康状态） node.total（节点总数） node.data（数据节点数） shards（分片数）  1531290005 14:20:05 Elasticsearch green 1 1 2  Pri（主分片数量） relo（备份节点） init（正在初始化的） unassign（未分配的） pending\_tasks（正在等待执行的任务）  2 0 0 0 0  max\_task\_wait\_time（挂起任务的等待时间） active\_shards\_percent（活动的分片的占有百分比）  - 100.0% |

status：green、yellow、red

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

yellow：每个索引的primary shard都是active的，但部分的replica shard不是active的

red：不是所有的索引的primary shard都是active状态的。

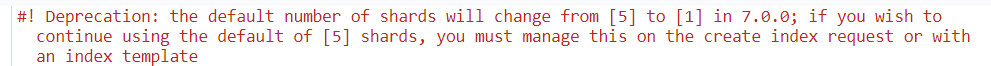
### 创建索引

命令语法：PUT 索引名{索引配置参数}

index名称必须是小写的，且不能以下划线'\_'，'-'，'+'开头。

在Elasticsearch中，默认的创建索引的时候，会分配5个primary shard，并为每个primary shard分配一个replica shard。在Elasticsearch中，默认的限制是：如果磁盘空间不足15%的时候，不分配replica shard。如果磁盘空间不足5%的时候，不再分配任何的primary shard。Elasticsearch中对shard的分布是有要求的。Elasticsearch尽可能保证primary shard平均分布在多个节点上。Replica shard会保证不和他备份的那个primary shard分配在同一个节点上。

创建默认索引。默认索引在7版本之前是5个，到7.x之后改成1个。



PUT test\_index1

创建索引时指定分片。

注意： 编写时大括号要和put命令等不在一行。重点不能把{放在索引名后，kibana会把{当成索引名的一部分

PUT test\_index2

{

"settings":{

"number\_of\_shards" : 2,

"number\_of\_replicas" : 1

}

}

### 修改索引

命令语法：PUT 索引名/\_settings{索引配置参数}

注意：索引一旦创建，primary shard数量不可变化，可以改变replica shard数量。

PUT test\_index2/\_settings

{

"number\_of\_replicas" : 2

}

### 删除索引

命令语法：DELETE 索引名1[, 索引名2 ...]

DELETE test\_index1

### 查看索引信息

GET \_cat/indices?v

|  |
| --- |
| health status index uuid pri rep docs.count  yellow open test\_index 2PJFQBtzTwOUhcy-QjfYmQ 5 1 0  docs.deleted store.size pri.store.size  0 460b 460b |

### 检查分片信息

查看索引的shard信息。

GET \_cat/shards?v

|  |
| --- |
| index shard prirep state docs store ip node  test\_index2 1 p STARTED 0 261b 192.168.89.142 mN\_pylT  test\_index2 1 r UNASSIGNED  test\_index2 1 r UNASSIGNED  test\_index2 0 p STARTED 0 261b 192.168.89.142 mN\_pylT  test\_index2 0 r UNASSIGNED  test\_index2 0 r UNASSIGNED |

### 新增Document

在索引中增加文档。在index中增加document。

Elasticsearch有自动识别机制。如果增加的document对应的index不存在，自动创建index；如果index存在，type不存在，则自动创建type。如果index和type都存在，则使用现有的index和type。

#### PUT语法

此操作为手工指定id的Document新增方式。

语法：PUT 索引名/类型名/唯一ID{字段名:字段值}

如：

|  |
| --- |
| PUT test\_index/test\_type/1  {  "name":"test\_doc\_01",  "remark":"first test elastic search",  "order\_no":1  }  PUT test\_index/test\_type/2  {  "name":"test\_doc\_02",  "remark":"second test elastic search",  "order\_no":2  }  PUT test\_index/test\_type/3  {  "name":"test\_doc\_03",  "remark":"third test elastic search",  "order\_no":3  } |

结果：

|  |
| --- |
| {  "\_index": "test\_index", 新增的document在什么index中，  "\_type": "test\_type", 新增的document在index中的哪一个type中。  "\_id": "1", 指定的id是多少  "\_version": 1, document的版本是多少，版本从1开始递增，每次写操作都会+1  "result": "created", 本次操作的结果，created创建，updated修改，deleted删除  "\_shards": { 分片信息  "total": 2, 分片数量只提示primary shard  "successful": 1, 数据document一定只存放在index中的某一个primary shard中  "failed": 0  },  "\_seq\_no": 0, 执行的序列号  "\_primary\_term": 1 词条比对。  } |

如果使用PUT语法对同id的Document执行多次操作。是一种覆盖操作。如果需要Elasticsearch辅助检查PUT的Document是否已存在，可以使用强制新增语法。使用强制新增语法时，如果Document的id在Elasticsearch中已存在，则会报错。（version conflict, document already exists）

语法：

PUT 索引名/类型名/唯一ID/\_create{字段名:字段值}

或

PUT 索引名/类型名/唯一ID?op\_type=create{字段名:字段值}。

如：

|  |
| --- |
| PUT test\_index/test\_type/1/\_create  {  "name":"new\_test\_doc\_01",  "remark":"first test elastic search",  "order\_no":1  } |

#### POST语法

此操作为Elasticsearch自动生成id的新增Document方式。

语法：POST 索引名/类型名{字段名:字段值}

如：

|  |
| --- |
| POST test\_index/test\_type  {  "name":"test\_doc\_04",  "remark":"forth test elastic search",  "order\_no":4  } |

### 查询Document

#### GET ID单数据查询

语法：GET 索引名/类型名/唯一ID

如：

GET test\_index/test\_type/1

结果：

|  |
| --- |
| {  "\_index": "test\_index",  "\_type": "test\_type",  "\_id": "1",  "\_version": 1,  "found": true,  "\_source": { 找到的document数据内容。  "name": "test\_doc\_01",  "remark": "first test elastic search",  "order\_no":1  }  } |

#### GET \_mget批量查询

批量查询可以提高查询效率。推荐使用（相对于单数据查询来说）。

语法如下：

|  |
| --- |
| GET \_mget  {  "docs" : [  {  "\_index" : "索引名",  "\_type" : "类型名",  "\_id" : "唯一ID值"  }, {}, {}  ]  } |
| GET 索引名/\_mget  {  "docs" : [  {  "\_type" : "类型名",  "\_id" : "唯一ID值"  }, {}, {}  ]  } |
| GET 索引名/类型名/\_mget  {  "docs" : [  {  "\_id" : "唯一ID值"  },  {  "\_id" : "唯一ID值"  }  ]  } |

### 修改Document

#### 替换Document（全量替换）

和新增的PUT语法是一致。

PUT 索引名/类型名/唯一ID{字段名:字段值}

本操作相当于覆盖操作。全量替换的过程中，Elasticsearch不会真的修改Document中的数据，而是标记Elasticsearch中原有的Document为deleted状态，再创建一个新的Document来存储数据，当Elasticsearch中的数据量过大时，Elasticsearch后台回收deleted状态的Document。

如：

|  |
| --- |
| PUT test\_index/test\_type/1  {  "name":"new\_test\_doc\_01",  "remark":"first test elastic search",  "order\_no":1  } |

结果：

|  |
| --- |
| {  "\_index": "test\_index",  "\_type": "test\_type",  "\_id": "1",  "\_version": 2,  "result": "updated",  "\_shards": {  "total": 2,  "successful": 1,  "failed": 0  },  "\_seq\_no": 1,  "\_primary\_term": 1  } |

#### 更新Document（partial update）

语法：POST 索引名/类型名/唯一ID/\_update{doc:{字段名:字段值}}

只更新某Document中的部分字段。这种更新方式也是标记原有数据为deleted状态，创建一个新的Document数据，将新的字段和未更新的原有字段组成这个新的Document，并创建。对比全量替换而言，只是操作上的方便，在底层执行上几乎没有区别。

如：

|  |
| --- |
| POST test\_index/test\_type/1/\_update  {  "doc":{  "name":" test\_doc\_01\_for\_update"  }  } |

结果：

|  |
| --- |
| {  "\_index": "test\_index",  "\_type": "test\_type",  "\_id": "1",  "\_version": 5,  "result": "updated",  "\_shards": {  "total": 2,  "successful": 1,  "failed": 0  },  "\_seq\_no": 2,  "\_primary\_term": 1  } |

### 删除Document

Elasticsearch中执行删除操作时，Elasticsearch先标记Document为deleted状态，而不是直接物理删除。当Elasticsearch存储空间不足或工作空闲时，才会执行物理删除操作。标记为deleted状态的数据不会被查询搜索到。

语法：DELETE 索引名/类型名/唯一ID

如：

|  |
| --- |
| DELETE test\_index/test\_type/1 |

结果：

|  |
| --- |
| {  "\_index": "test\_index",  "\_type": "test\_type",  "\_id": "1",  "\_version": 6,  "result": "deleted",  "\_shards": {  "total": 2,  "successful": 1,  "failed": 0  },  "\_seq\_no": 5,  "\_primary\_term": 1  } |

### bulk批量增删改

使用bulk语法执行批量增删改。语法格式如下：

注意：\_bulk的两个{}都不要出现换行等操作，两个{}使用回车符进行识别。

POST \_bulk

{ "action\_type" : { "metadata\_name" : "metadata\_value" } }

{ document datas | action datas }

语法中的action\_type可选值为：

create : 强制创建，相当于PUT 索引名/类型名/唯一ID/\_create

index: 普通的PUT操作，相当于创建Document或全量替换

update: 更新操作（partial update）,相当于 POST 索引名/类型名/唯一ID/\_update

delete: 删除操作

案例如下：

|  |
| --- |
| 新增数据：  POST \_bulk  { "create" : { "\_index" : "test\_index" , "\_type" : "test\_type", "\_id" : "1" } }  { "field\_name" : "field value" } |
| PUT操作新增或全量替换  POST \_bulk  { "index" : { "\_index" : "test\_index", "\_type" : "test\_type" , "\_id" : "2" } }  { "field\_name" : "field value 2" } |
| POST更新数据  POST \_bulk  { "update" : { "\_index" : "test\_index", "\_type" : "test\_type" , "\_id" : 2, "\_retry\_on\_conflict" : 3 } }  { "doc" : { "field\_name" : "partial update field value" } } |
| DELETE删除数据  POST \_bulk  { "delete" : { "\_index" : "test\_index", "\_type" : "test\_type", "\_id" : "2" } } |
| 批量写操作  POST \_bulk  { "create" : { "\_index" : "test\_index" , "\_type" : "test\_type", "\_id" : "10" } }  { "field\_name" : "field value" }  { "index" : { "\_index" : "test\_index", "\_type" : "test\_type" , "\_id" : "20" } }  { "field\_name" : "field value 2" }  { "update" : { "\_index" : "test\_index", "\_type" : "test\_type" , "\_id" : 20, "\_retry\_on\_conflict" : 3 } }  { "doc" : { "field\_name" : "partial update field value" } }  { "delete" : { "\_index" : "test\_index", "\_type" : "test\_type", "\_id" : "2" } } |

注意：bulk语法中要求一个完整的json串不能有换行。不同的json串必须使用换行分隔。多个操作中，如果有错误情况，不会影响到其他的操作，只会在批量操作返回结果中标记失败。bulk语法批量操作时，bulk request会一次性加载到内存中，如果请求数据量太大，性能反而下降（内存压力过高），需要反复尝试一个最佳的bulk request size。一般从1000~5000条数据开始尝试，逐渐增加。如果查看bulk request size的话，一般是5~15MB之间为好。

bulk语法要求json格式是为了对内存的方便管理，和尽可能降低内存的压力。如果json格式没有特殊的限制，Elasticsearch在解释bulk请求时，需要对任意格式的json进行解释处理，需要对bulk请求数据做json对象会json array对象的转化，那么内存的占用量至少翻倍，当请求量过大的时候，对内存的压力会直线上升，且需要jvm gc进程对垃圾数据做频繁回收，影响Elasticsearch效率。

生产环境中，bulk api常用。都是使用java代码实现循环操作。一般一次bulk请求，执行一种操作。如：批量新增10000条数据等。

## 常用元数据简介

在ElasticSearch中，除了定义的index，type，和管理的document外，还有若干的元数据。这些元数据用于记录ElasticSearch中需要使用的核心数据。在ElasticSearch中，元数据通常使用下划线’\_’开头。在使用GET查询Document时，返回结果中不仅包括客户端录入的数据，还包括各元数据。

### \_index

代表document存放在哪个index中，\_index就是索引的名字。生产环境中，类似的Document存放在一个index中，非类似的Document存放在不同的index中。一个index中包含若干相似结构的Document。

### \_type

代表document属于index中的哪个type（类别），就是type的名字。ElasticSearch6.x版本中，一个index只能定义一个type。结构类似的document保存在一个index中。Type命名要求：字符大小写无要求，不能下划线开头，不能包含逗号。（ElasticSearch低版本，5.x或更低版本。一般一个索引会划分若干type，逻辑上对index中的document进行细致的划分。在命名上，可以全大写或者全小写，不能下划线开头，不能包含逗号。）

### \_id

代表document的唯一标识。使用index、type和id可以定位唯一的一个document。id可以在新增document时手工指定，也可以由ElasticSearch自动创建。

#### 手动指定id

如：

|  |
| --- |
| PUT test\_index/test\_type/10  {  "field\_name" : "field\_value"  } |

使用这种方式，需要考虑是否满足手动指定id的条件。如果数据是从其他数据源中读取并新增到ElasticSearch中的时候，使用手动指定id。如：数据是从Database中读取并新增到ElasticSearch中的，那么使用Database中的PK作为ElasticSearch中的id比较合适。建议，不要把不同表的数据新增到同一个index中，可能有id冲突。

#### 自动生成id

语法：

|  |
| --- |
| POST test\_index/test\_type  {  "field\_name" : "field\_value"  } |

自动生成的ID特点：长度为20的字符串；URL安全（经过base64编码的）；GUID生成策略，支持分布式高并发（在分布式系统中，并发生成ID也不会有重复可能，参考https://baike.baidu.com/item/GUID/3352285?fr=aladdin）。适合用于手工录入的数据。数据没有一个数据源，且未经过任何的管理和存储。这种数据，是没有唯一标识，如果使用手工指定id的方式，容易出现id冲突，导致数据丢失。

### \_source元数据

就是查询的document中的field值。也就是document的json字符串。此元数据可以定义显示结果（field）。

语法：GET 索引名/类型名/唯一ID?\_source=字段名1,字段名2

如：

GET test\_index/test\_type/1?\_source=field\_name

### \_version元数据

代表的是document的版本。在ElasticSearch中，为document定义了版本信息，document数据每次变化，代表一次版本的变更。版本变更可以避免数据并发冲突，同时提高ElasticSearch的搜索效率。

第一次创建Document时，\_version版本号为1，默认情况下，后续每次对Document执行修改或删除操作都会对\_version数据自增1。

删除Document也会\_version自增1。

当使用PUT命令再次增加同id的Document，\_version会继续之前的版本继续自增。

## 分词器（analyzer）和标准化处理（normalization）

### 什么是分词器

分词器是一个字符串解析拆分工具。其作用是分析写入的Document中的文本数据field，并将field数据拆分成一个个有完整含义的、不可拆分的单词。

如：I think dogs is human’s best friend.在写入此数据的时候，Elasticsearch会使用分词器分析并拆分数据，将上述的语句切分成若干的单词，分别是：i、 think、 dogs、 human's、 best、 friend。

### 什么是标准化处理

标准化处理是用于完善分词器结果的。

分词器处理的文本结果，通常会有一些不需要的、有异议的、包含时态转化等情况的数据。在上述案例中的分词结果是：i、 think、 dogs、 human's、 best、 friend。其中i是很少应用在搜索条件中的单词；dogs是dog单词的复数形式，通常在搜索过程中使用dog作为搜索条件更频繁一些；human's是特殊的标记方式，通常不会在搜索中作为条件出现。那么Elasticsearch维护这些单词是没有太大必要的。这个时候就需要标准化处理了。

如：china 搜索时，如果条件为cn是否可搜索到。如：dogs，搜索时，条件为dog是否可搜索到数据。如果可以使用简写（cn）或者单复数（dog&dogs）搜索到想要的结果，那么称为搜索引擎人性化。

normalization是为了提升召回率的（recall），就是提升搜索能力的。

normalization是配合分词器(analyzer)完成其功能的。

### Elasticsearch默认提供的常见分词器

要切分的语句：Set the shape to semi-transparent by calling set\_trans(5)

standard analyzer - 是Elasticsearch中的默认分词器。标准分词器，处理英语语法的分词器。切分后的key\_words：set, the, shape, to, semi, transparent, by, calling, set\_trans, 5。这种分词器也是Elasticsearch中默认的分词器。切分过程中不会忽略停止词（如：the、a、an等）。会进行单词的大小写转换、过滤连接符（-）或括号等常见符号。

GET \_analyze

{

"text": "Set the shape to semi-transparent by calling set\_trans(5)",

"analyzer": "standard"

}

simple analyzer - 简单分词器。切分后的key\_words：set, the, shape, to, semi, transparent, by, calling, set, trans。就是将数据切分成一个个的单词。使用较少，经常会破坏英语语法。

GET \_analyze

{

"text": "Set the shape to semi-transparent by calling set\_trans(5)",

"analyzer": "simple"

}

whitespace analyzer - 空白符分词器。切分后的key\_words：Set, the, shape, to, semi-transparent, by, calling, set\_trans(5)。就是根据空白符号切分数据。如：空格、制表符等。使用较少，经常会破坏英语语法。

GET \_analyze

{

"text": "Set the shape to semi-transparent by calling set\_trans(5)",

"analyzer": "whitespace"

}

language analyzer - 语言分词器，如英语分词器（english）等。切分后的key\_words：set, shape, semi, transpar, call, set\_tran, 5。根据英语语法分词，会忽略停止词、转换大小写、单复数转换、时态转换等，应用分词器分词功能类似standard analyzer。

GET \_analyze

{

"text": "Set the shape to semi-transparent by calling set\_trans(5)",

"analyzer": "english"

}

注意：Elasticsearch中提供的常用分词器都是英语相关的分词器，对中文的分词都是一字一词。

### 安装中文分词器

IK中文分词器，可以直接下载：注意：要下载elasticsearch-analysis-ik-xxx.zip，不要下载source

<https://github.com/medcl/elasticsearch-analysis-ik/releases/tag/v6.8.4>

也可以通过github下载源码，本地编译打包。

就是maven工程中的package能力。

github上提供的源码不是伴随ES的每个版本提供，一般只有分词器无效后，才提供新的版本。通常都是伴随ES的次版本号提供IK分词器版本。下载对应的IK分词器源码，本地package打包，生成zip压缩包，既是IK在ES中的分词器安装包。

1. 下载分词器项目
2. 将项目导入到Idea中
3. 修改POM文件制定es版本
4. 打包 package

#### 安装IK分词器

Elasticsearch是一个开箱即用的工具。插件安装方式也非常简单。

将IK分词器的zip压缩文件上传到Linux，并在Elasticsearch安装目录的plugins目录中手工创建子目录，目录命名为ik。将zip压缩文件解压缩到新建目录ik中。重新启动Elasticsearch即可。

创建IK中文分词器的插件子目录：

mkdir /usr/local/es/plugins/ik/

复制中文分词器zip压缩文件到Elasticsearch应用目录中：

cp elasticsearch-analysis-ik-6.8.4.zip /usr/local/es/plugins/ik

解压缩：

unzip elasticsearch-analysis-ik-6.8.4.zip

所有的分词器，都是针对词语的，不是语句的。拆分单元是词语，不是语句。

#### 测试IK分词器

IK分词器提供了两种analyzer，分别是ik\_max\_word和ik\_smart。

ik\_max\_word: 会将文本做最细粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民共和国,中华人民,中华,华人,人民共和国,人民,人,民,共和国,共和,国,国歌”，会穷尽各种可能的组合；

ik\_smart: 会做最粗粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民共和国,国歌”。

|  |
| --- |
| GET \_analyze  {  "text" : "中华人民共和国国歌",  "analyzer": "ik\_max\_word"  }  GET \_analyze  {  "text" : "中华人民共和国国歌",  "analyzer": "ik\_smart"  } |

#### IK配置文件

IK的配置文件在Elasticsearch安装目录/plugins/ik/config/中。

配置文件有：

main.dic ： IK中内置的词典。 main dictionary。记录了IK统计的所有中文单词。一行一词。文件中未记录的单词，IK无法实现有效分词。如：雨女无瓜。不建议修改当前文件中的单词。这个是最核心的中文单词库。就好像，很多的网络词不会收集到辞海中一样。

quantifier.dic ： IK内置的数据单位词典

suffix.dic ：IK内置的后缀词典

surname.dic ：IK内置的姓氏词典

stopword.dic ：IK内置的英文停用词

preposition.dic ：IK内置的中文停用词（介词）

IKAnalyzer.cfg.xml ： 用于配置自定义词库的

自定义词库是用户手工提供的特殊词典，类似网络热词，特定业务用词等。

ext\_dict - 自定义词库，配置方式为相对于IKAnalyzer.cfg.xml文件所在位置的相对路径寻址方式。相当于是用户自定义的一个main.dic文件。是对main.dic文件的扩展。

ext\_stopwords - 自定义停用词，配置方式为相对于IKAnalyzer.cfg.xml文件所在位置的相对路径寻址方式。相当于是preposition.dic的扩展。

注意：IK的所有的dic词库文件，必须使用UTF-8字符集。不建议使用windows自带的文本编辑器编辑。Windows中自带的文本编辑器是使用GBK字符集。IK不识别，是乱码。

## Elasticsearch中的mapping问题

Mapping在Elasticsearch中是非常重要的一个概念。决定了一个index中的field使用什么数据格式存储，使用什么分词器解析，是否有子字段等。

Mapping决定了index中的field的特征。

### mapping核心数据类型

Elasticsearch中的数据类型有很多，在这里只介绍常用的数据类型。

文本（字符串）：text

整数：byte、short、integer、long

浮点型：float、double

布尔类型：boolean

日期类型：date

数组类型：array {a:[]}

对象类型：object {a:{}}

不分词的字符串（关键字）： keyword

### dynamic mapping对字段的类型分配

true or false -> boolean

123 -> long

123.123 -> double

2018-01-01 -> date

hello world -> text

[] -> array

{} -> object

在上述的自动mapping字段类型分配的时候，只有text类型的字段需要分词器。默认分词器是standard分词器。

### 查看索引mapping

可以通过命令查看已有index的mapping具体信息，语法如下：

GET 索引名/\_mapping

如：

|  |
| --- |
| GET test\_index/\_mapping |

结果：

|  |
| --- |
| {  "test\_index": { # 索引名  "mappings": { # 映射列表  "test\_type": { # 类型名  "properties": { # 字段列表  "age": { # 字段名  "type": "long" # 字段类型  },  "gender": {  "type": "text",  "fields": { # 子字段列表  "keyword": { # 子字段名  "type": "keyword", # 子字段类型，keyword不进行分词处理的文本类型  "ignore\_above": 256 # 子字段存储数据长度  }  }  },  "name": {  "type": "text",  "fields": {  "keyword": {  "type": "keyword",  "ignore\_above": 256  }  }  }  }  }  }  }  } |

### custom mapping

可以通过命令，在创建index和type的时候来定制mapping映射，也就是指定字段的类型和字段数据使用的分词器。

手工定制mapping时，只能新增mapping设置，不能对已有的mapping进行修改。

如：有索引a，其中有类型b，增加字段f1的mapping定义。后续可以增加字段f2的mapping定义，但是不能修改f1字段的mapping定义。

通常都是手工创建index，并进行各种定义。如：settings,mapping等。

#### 创建索引时指定mapping

语法：

PUT 索引名称

{

"mappings":{

"类型名称":{

"properties":{

"字段名":{

"type":类型,

["analyer":字段的分词器,]

["fields":{

"子字段名称":{

"type":类型,

"ignore\_above":长度限制

}

}]

}

}

}

}

}

如：

|  |
| --- |
| PUT /test\_index  {  "settings": {  "number\_of\_shards": 2,  "number\_of\_replicas": 1  },  "mappings": {  "test\_type":{  "properties": {  "author\_id" : {  "type": "byte",  "index": false  },  "title" : {  "type": "text",  "analyzer": "ik\_max\_word",  "fields": {  "keyword" : {  "type": "keyword",  "ignore\_above": 256  }  }  },  "content" : {  "type": "text",  "analyzer": "ik\_max\_word"  },  "post\_date" : {  "type": "date"  }  }  }  }  } |

"index" - 是否可以作为搜索索引。可选值：true | false

"analyzer" - 指定分词器。

"type" - 指定字段类型

#### 为已有索引添加**新的字段mapping**

语法：

PUT 索引名/\_mapping/类型名

{

"properties":{

"新字段名":{

"type":类型,

"analyer":字段的分词器,

"fields":{

"子字段名":{

"type":类型,

"ignore\_above":长度

}

}

}

}

}

如：

|  |
| --- |
| PUT /test\_index/\_mapping/test\_type  {  "properties" : {  "new\_field" : { "type" : "text" , "analyzer" : "standard" }  }  } |

#### 测试不同的字段的分词器

语法：

GET 索引名称/\_analyze

{

"field":"索引中的text类型的字段名",

"text":"要分词处理的文本数据"

}

使用索引中的字段对应的分词器，对文本数据做分词处理。

如：

|  |
| --- |
| GET /test\_index/\_analyze  {  "field": "new\_field",  "text": "中华人民共和国国歌"  }  GET /test\_index/\_analyze  {  "field": "content",  "text": "中华人民共和国国歌"  } |

## Search 搜索详解

### 搜索学习测试数据

|  |
| --- |
| PUT test\_search  {  "mappings": {  "test\_type" : {  "properties": {  "dname" : {  "type" : "text",  "analyzer": "standard"  },  "ename" : {  "type" : "text",  "analyzer": "standard"  },  "eage" : {  "type": "long"  },  "hiredate" : {  "type": "date"  },  "gender" : {  "type" : "keyword"  }  }  }  }  }  POST test\_search/test\_type/\_bulk  { "index": {}}  { "dname" : "Sales Department", "ename" : "张三", "eage":20, "hiredate" : "2019-01-01", "gender" : "男性" }  { "index": {}}  { "dname" : "Sales Department", "ename" : "李四", "eage":21, "hiredate" : "2019-02-01", "gender" : "男性" }  { "index": {}}  { "dname" : "Development Department", "ename" : "王五", "eage":23, "hiredate" : "2019-01-03", "gender" : "男性" }  { "index": {}}  { "dname" : "Development Department", "ename" : "赵六", "eage":26, "hiredate" : "2018-01-01", "gender" : "男性" }  { "index": {}}  { "dname" : "Development Department", "ename" : "韩梅梅", "eage":24, "hiredate" : "2019-03-01", "gender" : "女性" }  { "index": {}}  { "dname" : "Development Department", "ename" : "钱虹", "eage":29, "hiredate" : "2018-03-01", "gender" : "女性" } |

### query string search

search的参数都是类似http请求头中的字符串参数提供搜索条件的。

GET [/index\_name/type\_name/]\_search[?parameter\_name=parameter\_value&...]

#### 全搜索

timeout参数：是超时时长定义。代表每个节点上的每个shard执行搜索时最多耗时多久。不会影响响应的正常返回。只会影响返回响应中的数据数量。

如：索引a中，有10亿数据。存储在5个shard中，假设每个shard中2亿数据，执行全数据搜索的时候，需要耗时1000毫秒。定义timeout为10毫秒，代表的是shard执行10毫秒，搜索出多少数据，直接返回。

在商业项目中，是禁止全数据搜索的。必须指定搜索的索引，类型和关键字。如果没有指定索引或类型，则代表开发目的不明确，需要重新做用例分析。如果没有关键字，称为索引内全搜索，也叫魔鬼搜索。

语法：

|  |
| --- |
| GET [索引名/类型名/]\_search?timeout=10ms |

结果：

|  |
| --- |
| {  "took": 144, #请求耗时多少毫秒  "timed\_out": false, #是否超时。默认情况下没有超时机制，也就是客户端等待Elasticsearch搜索结束（无论执行多久），提供超时机制的话，Elasticsearch则在指定时长内处理搜索，在指定时长结束的时候，将搜索的结果直接返回（无论是否搜索结束）。指定超时的方式是传递参数，参数单位是：毫秒-ms。秒-s。分钟-m。  "\_shards": {  "total": 1, #请求发送到多少个shard上  "successful": 1,#成功返回搜索结果的shard  "skipped": 0, #停止服务的shard  "failed": 0 #失败的shard  },  "hits": {  "total": 1, #返回了多少结果  "max\_score": 1, #搜索结果中，最大的相关度分数，相关度越大分数越高，\_score越大，排位越靠前。  "hits": [ #搜索到的结果集合，默认查询前10条数据。  {  "\_index": "test\_index", #数据所在索引  "\_type": "test\_type", #数据所在类型  "\_id": "1", #数据的id  "\_score": 1, #数据的搜索相关度分数  "\_source": { # 数据的具体内容。  "field": "value"  }  }  ]  }  } |

#### multi index搜索

所谓的multi-index就是从多个index中搜索数据。相对使用较少，只有在复合数据搜索的时候，可能出现。一般来说，如果真使用复合数据搜索，都会使用\_all。

如：搜索引擎中的无条件搜索。（现在的应用中都被屏蔽了。使用的是默认搜索条件，执行数据搜索。 如： 电商中的搜索框默认值， 搜索引擎中的类别）

无条件搜索，在搜索应用中称为“魔鬼搜索”，代表的是，搜索引擎会执行全数据检索，效率极低，且对资源有非常高的压力。

语法：

|  |
| --- |
| GET \_search  GET 索引名1,索引名2/\_search # 搜索多个index中的数据  GET 索引名/类型名/\_search # 所属一个index中type的数据  GET prefix\_\*/\_search # 通配符搜索  GET \*\_suffix/\_search  GET 索引名1,索引名2/类型名/\_search # 搜索多个index中type的数据  GET \_all/\_search # \_all代表所有的索引 |

#### 条件搜索

query string search 搜索是通过HTTP请求的请求头传递参数的，默认的HTTP请求头字符集是ISO-8859-1，**请求头传递中文会有乱码**。

语法：

|  |
| --- |
| GET 索引名/\_search?**q=**字段名:搜索条件 |

#### 分页搜索

默认情况下，Elasticsearch搜索返回结果是10条数据。从第0条开始查询。

语法：

|  |
| --- |
| GET 索引名/\_search?size=10 # size查询数据的行数  GET 索引名/\_search?from=0&size=10 # from 从第几行开始查询，行号从0开始。 |

#### +/-搜索

语法：

|  |
| --- |
| GET 索引名/\_search?q=字段名:条件  GET 索引名/\_search?q=+字段名:条件  GET 索引名/\_search?q=-字段名:条件 |

+ ：和不定义符号含义一样，就是搜索指定的字段中包含key words的数据

- ： 与+符号含义相反，就是搜索指定的字段中不包含key words的数据

#### **排序**

语法：GET 索引名/\_search?sort=字段名:排序规则

排序规则： asc(升序) | desc(降序)

|  |
| --- |
| GET test\_search/\_search?sort=eage:asc  GET test\_search/\_search?sort=eage:desc |

### query DSL

DSL - Domain Specified Language ， 特殊领域的语言。

请求参数是请求体传递的。在Elasticsearch中，请求体的字符集默认为UTF-8。

语法格式：

GET 索引名/\_search

{

"command":{ "parameter\_name" : "parameter\_value"}

}

#### 查询所有数据

|  |
| --- |
| GET 索引名/\_search  {  "query" : { "match\_all" : {} }  } |

#### match search

全文检索。要求查询条件拆分后的任意词条与具体数据匹配就算搜索结果。

|  |
| --- |
| GET 索引名/\_search  {  "query": {  "match": {  "字段名": "搜索条件"  }  }  } |

#### phrase search

短语检索。要求查询条件必须和具体数据完全匹配才算搜索结果。其特征是：1-搜索条件不做任何分词解析；2-在搜索字段对应的倒排索引(正排索引)中进行精确匹配，不再是简单的全文检索。

|  |
| --- |
| GET 索引名/\_search  {  "query": {  "match\_phrase": {  "字段名": "搜索条件"  }  }  } |

#### range

范围比较搜索

|  |
| --- |
| GET 索引名/类型名/\_search  {  "query" : {  "range" : {  "字段名" : {  "gt" : 搜索条件1,  "lte" : 搜索条件2  }  }  }  } |

#### term

词组比较，词组搜索。

忽略搜索条件分词，在Elasticsearch倒排索引中进行精确匹配。

|  |
| --- |
| GET 索引名/类型名/\_search  {  "query" : {  "term" : {  "字段名": "搜索条件"  }  }  }  GET 索引名/类型名/\_search  {  "query" : {  "terms" : {  "字段名": ["搜索条件1", "搜索条件2"]  }  }  } |

#### 多条件复合搜索

在一个请求体中，有多个搜索条件，就是复合搜索。如：搜索数据，条件为部门名称是Sales Department，员工年龄在20到26之间，部门员工姓名叫张三。上述条件中，部门名称为可选条件，员工年龄必须满足要求，部门员工姓名为可选要求。这种多条件搜索就是复合搜索。

|  |
| --- |
| GET 索引名/类型名/\_search  {  "query": {  "bool": {  "must": [ #数组中的多个条件必须同时满足  {  "range": {  "字段名": {  "lt": 条件  }  }  }  ],  "must\_not":[ #数组中的多个条件必须都不满足  {  "match": {  "字段名": "条件"  }  },  {  "range": {  "字段名": {  "gte": "搜索条件"  }  }  }  ]  "should": [# 数组中的多个条件有任意一个满足即可。  {  "match": {  "字段名": "条件"  }  },  {  "range": {  "字段名": {  "gte": "搜索条件"  }  }  }  ]  }  }  } |

#### 排序

在Elasticsearch的搜索中，默认是使用相关度分数实现排序的。可以通过搜索语法实现定制化排序。

|  |
| --- |
| GET 索引名/类型名/\_search  {  "query": {  [搜索条件]  },  "sort": [  {  "字段名1": {  "order": "asc"  }  },  {  "字段名2": {  "order": "desc"  }  }  ]  } |

注意：在Elasticsearch中，如果使用text类型的字段作为排序依据，会有问题。Elasticsearch需要对text类型字段数据做分词处理。如果使用text类型字段做排序，Elasticsearch给出的排序结果未必友好，毕竟分词后，先使用哪一个单词做排序都是不合理的。所以Elasticsearch中默认情况下不允许使用text类型的字段做排序，如果需要使用字符串做结果排序，则可使用keyword类型字段作为排序依据，因为keyword字段不做分词处理。

#### 分页

DSL分页也是使用from和size实现的。

|  |
| --- |
| GET 索引名称/\_search  {  "query":{  "match\_all":{}  },  "from": 起始下标,  "size": 查询记录数  } |

#### highlight display

在搜索中，经常需要对搜索关键字做高亮显示，这个时候就可以使用highlight语法。

语法：

|  |
| --- |
| GET 索引名/\_search  {  "query": {  "match": {  "字段名": "条件"  }  },  "highlight": {  "fields": {  "要高亮显示的字段名": {  "fragment\_size": 5, #每个分段长度，默认20  "number\_of\_fragments": 1 #返回多少个分段，默认3  }  },  "pre\_tags": ["前缀"],  "post\_tags": ["后缀"]  }  } |
| // 演示案例  GET test\_search/\_search  {  "query": {  "bool": {  "should": [  {  "match": {  "dname": "Development department"  }  },  {  "match": {  "gender": "男性"  }  }  ]  }  },  "highlight": {  "fields": {  "dname": {  "fragment\_size": 20,  "number\_of\_fragments": 1  },  "gender": {  "fragment\_size": 20,  "number\_of\_fragments": 1  }  },  "pre\_tags":["<span style='color:red'>"],  "post\_tags":["</span>"]  },  "from": 2,  "size": 2  } |

fragment\_size：代表字段数据如果过长，则分段，每个片段数据长度为多少。长度不是字符数量，是Elasticsearch内部的数据长度计算方式。默认不对字段做分段。

number\_of\_fragments：代表搜索返回的高亮片段数量，默认情况下会将拆分后的所有片段都返回。

pre\_tags：高亮前缀

post\_tags：高亮后缀

很多搜索结果显示页面中都不会显示完整的数据，这样在数据过长的时候会导致页面效果不佳，都会按照某一个固定长度来显示搜索结果，所以fragment\_size和number\_of\_fragments参数还是很常用的。