注：

?v：表示输出表头

?help：表头字段解释

?pretty=true：表示格式化输出

?level=indices：表示显示索引状态

?level=shards：表示显示分片信息

? refresh：Index，Update，Delete和Bulk API支持设置刷新，以控制此请求所做的更改何时对搜索可见。? refresh=true、? refresh=false、? refresh=wait\_for

# 在Windows上安装CURL及使用方法

## 安装CURL

在https://curl.haxx.se/download.html下载curl-7.52.1-win64-mingw.7z，然后解压，最后配置CURL\_HOME环境变量，并在PATH中加入%CURL\_HOME%\bin即可。

## 使用方法

在命令行中执行相应的语句即可，如果提示没有相关的命令，则直接到CURL\_HOME/bin目录下运行。

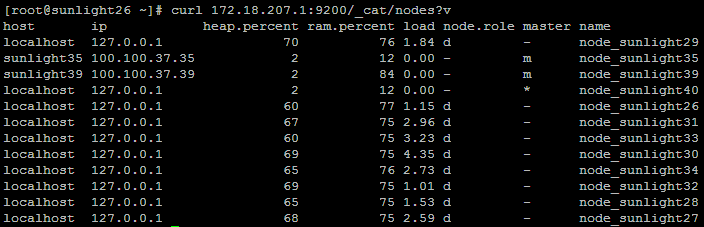
注：由于字符集上有点出入，有些命令无法在windows上执行。

# curl命令\_cat系列

curl -XGET http://192.168.1.101:9200/

## 查看集群节点

curl -XGET 172.18.207.1:9200/\_cat/nodes?v



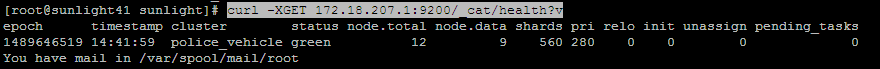
## 查看插件

curl -XGET 172.18.207.1:9200/\_cat/plugins?v



## 查看集群健康

curl -XGET 172.18.207.1:9200/\_cat/health?v



集群三种状态：

Green：表示集群健康。

Yellow：表示有一些备份分片还没有分配。

Red：表示集群不健康，有些数据不可用，此时需要对分片进行修复。

## 查看索引

1. 查看所有索引

GET /\_cat/indices?v

1. 查看某个索引

/\_cat/indices/{index} ---不需要中括号，index为索引名称（多个逗号隔开）

GET /\_cat/indices/kz\_zdry,customer?v

## 其他命令

/\_cat/allocation：查看每个数据节点分配了多少碎片以及它们使用了多少磁盘空间

/\_cat/shards：查看所有分片

/\_cat/shards/{index}：查看指定分片的索引（多个逗号隔开）

/\_cat/master：查找master节点

/\_cat/nodes：查看所有节点

/\_cat/nodeattrs：查看节点属性

/\_cat/segments  
/\_cat/segments/{index}（多个逗号隔开）

/\_cat/count：统计所有索引的文档数

/\_cat/count/{index}：统计单个索引的文档数（多个逗号隔开）

/\_cat/recovery  
/\_cat/recovery/{index}（多个逗号隔开）

/\_cat/health：查看集群健康状态

/\_cat/pending\_tasks  
/\_cat/aliases：查看所有索引别名信息

/\_cat/aliases/{alias}：查看指定的索引别名信息（多个逗号隔开）

/\_cat/thread\_pool：显示每个节点的集群范围的线程池统计信息。默认情况下，将为所有线程池返回活动、队列和拒绝统计信息

/\_cat/plugins：查看插件

/\_cat/repositories：显示在集群中注册的快照储存库

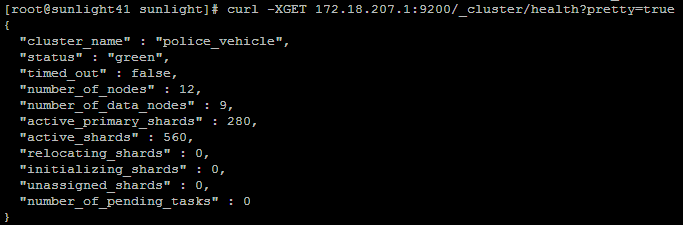
/\_cat/fielddata：查看每个节点每种数据所占的空间大小

/\_cat/fielddata/{fields}：查看每个节点指定类型数据所占的空间大小（多个逗号隔开）

# curl命令\_cluster系列

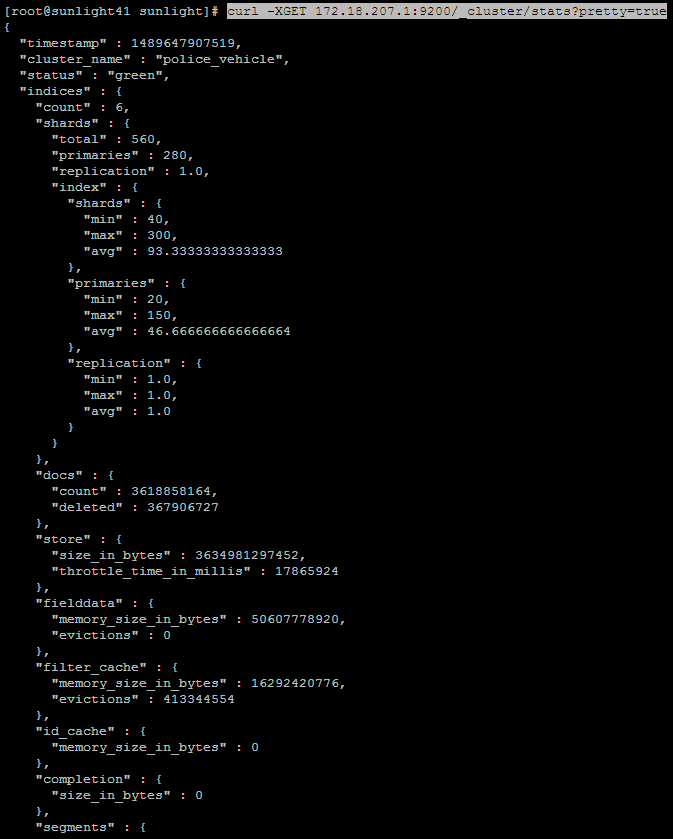
## 查询设置集群状态

curl -XGET 172.18.207.1:9200/\_cluster/health?pretty=true



## 显示集群系统信息，包括CPU、JVM等

curl -XGET 172.18.207.1:9200/\_cluster/stats?pretty=true



## 显示集群的详细信息，包括节点、分片等

curl -XGET 172.18.207.1:9200/\_cluster/state?pretty=true

## 获取集群堆积的任务

curl -XGET 172.18.207.1:9200/\_cluster/pending\_tasks?pretty=true

## 集群设置

集群设置使用GET /\_cluster/settings完成，修改分为persistent（表示永久设置，，将会写入到elasticsearch.yml配置文件中）和transient，（临时设置，一旦elasticsearch重启，这个配置会立即失效）。

注意：并非所有的配置都能通过API修改，仅部分配置支持动态修改，其他只能在elasticsearch.yml配置文件中修改。

### 获取集群设置

GET /\_cluster/settings //仅显示改动后的设置

GET /\_cluster/settings?include\_defaults=true //显示所有设置，包括默认的设置

### 重置设置

可以通过分配null值重置持久或临时设置，如果是临时设置，建议设置回原来的值（永久设置的值、配置文件里的值或默认值）。如下所示：

PUT /\_cluster/settings/

{

"transient": {

"indices.store.throttle.type": null

}

}

支持模糊配置项，如"indices.recovery.\*" : null

显示以下即为重置：

{

……

"persistent": {},

"transient": {}

}

### 设置最小的有资格主节点数

该数不能随便设置，否则一个集群会分裂成多个集群（脑裂），该数值的计算公式：(master\_eligible\_nodes / 2) + 1（master\_eligible\_nodes为可以成为master的节点数）。

修改：

curl -XPUT http://localhost:9200/\_cluster/settings -d '{

"persistent" : {

"discovery.zen.minimum\_master\_nodes":2

}

}'

注：transient 表示临时的，persistent表示永久的

### 修改硬盘的写入速率

curl -XPUT http://localhost:9200/\_cluster/settings/ -d '{

{

"persistent": {

"indices.store.throttle.max\_bytes\_per\_sec": "100mb"

}

}'

注意：适当提高写入速率可以加快索引数据的时间，但不宜太高，因为磁盘IO无法跟得上。

### 关闭与启动段合并merge

关闭：

curl -XPUT http://localhost:9200/\_cluster/settings/ -d '{

"transient": {

"indices.store.throttle.type": "none"

}

}'

启动：

curl -XPUT http://localhost:9200/\_cluster/settings/ -d '{

"persistent": {

"indices.store.throttle.type": "merge",

"indices.store.throttle.max\_bytes\_per\_sec": "15mb"

}

}'

默认合并速率indices.store.throttle.max\_bytes\_per\_sec为20mb

## 手动控制路由，分片分配（移动或强制分配分片）

elasticsearch可以通过reroute api来手动进行索引分片的分配。在处理任何重路由命令之后，Elasticsearch将像往常一样执行重新平衡(与cluster.routing.rebalance.enable设置有关)，以便保持平衡状态。例如，如果请求的分配包括将碎片从node1移动到node2，那么这可能会导致碎片从node2移动回node1，从而使数据更加均衡。

可以使用cluster.routing.allocation.enable设置控制是否自动分配，如果禁止自动分配，则只能通过reroute命令或重新平衡来实现分片分配。

分片分配的三种操作：

1. 移动（move）：把分片从一节点移动到另一个节点。可以指定索引名和分片号。

POST /\_cluster/reroute

{

"commands" : [

{

"move" : {

"index" : "customer",

"shard" : 0,

"from\_node" : "slave1.htdata.com",

"to\_node" : "slave2.htdata.com"

}

}

]

}

上述完成将索引customer的0号分片从slave1.htdata.com移动slave2.htdata.com

1. 取消（cancel）：取消分配一个分片。可以指定索引名和分片号。node参数可以指定在那个节点取消正在分配的分片。allow\_primary参数支持取消分配主分片。

POST /\_cluster/reroute

{

"commands" : [

{

"cancel" : {

"index" : "customer",

"shard" : 0,

"node" : "master.htdata.com",

"allow\_primary" : true

}

}

]

}

上述完成取消索引customer在master.htdata.com节点上的0号分片（如果该分片是主分片，则需要添加"allow\_primary" : true属性，允许取消主分片），如果开启自动平衡，则该分片会自动分配到其他节点上。

1. 分配（allocate\_replica）：分配一个未分配的分片到指定节点。可以指定索引名和分片号。node参数指定分配到那个节点。allow\_primary参数可以强制分配主分片，不过这样可能导致数据丢失。

POST /\_cluster/reroute

{

"commands" : [

{

"allocate\_replica" : {

"index" : "customer",

"shard" : 0,

"node" : "slave2.htdata.com",

"allow\_primary" : true

}

}

]

}

上述完成将索引customer上未分配的0号分片手动分配到slave2.htdata.com节点上（如果需要将该分片作为主分片，则需要添加"allow\_primary" : true属性，允许该分片做主分片）。

注意：执行手动分配分片时，需要确定分配到哪个节点上，确保每个节点只有一个同一号码的分配（因为分配有一个主分片和多个备份分片）。

## 关闭节点

curl -XPOST 172.18.207.1:9200/\_cluster/nodes/172.18.207.1/\_shutdown

## 关闭主节点

curl -XPOST 172.18.207.1:9200/\_cluster/nodes/\_master/\_shutdown

## 关闭集群

curl -XPOST 172.18.207.1:9200/\_shutdown?delay=10s

curl -XPOST 172.18.207.1:9200/\_cluster/nodes/\_shutdown

curl -XPOST 172.18.207.1:9200/\_cluster/nodes/all/\_shutdown

# curl命令\_nodes系列

## 查询节点状态

curl -XGET 172.18.207.1:9200/\_nodes/status?pretty=true

curl -XGET 172.18.207.1:9200/?pretty=true

## 查询节点信息

1. 查看所有节点信息

GET /\_nodes

1. 查看一个或多个节点信息

GET /\_nodes/slave2.htdata.com,slave2.htdata.com

1. 查看一项或多项信息

GET /\_nodes/slave2.htdata.com,slave2.htdata.com/jvm,process

## 统计节点各项信息

1. 针对所有节点

GET /\_nodes/stats

1. 针对一个或多个节点

GET /\_nodes/slave1.htdata.com,slave2.htdata.com/stats

1. 统计一项或多项信息

默认统计所有信息项，我们可以指定返回某些信息。比如indices、fs、os、process、jvm、transport、http、discovery、ingest、breaker and thread\_pool。

方法：

GET /\_nodes/slave1.htdata.com,slave2.htdata.com/stats/discovery,http

## 查看节点插件

1. 查看所有节点的插件

GET /\_nodes/plugins

1. 查看一个或多个节点的插件

GET /\_nodes/slave1.htdata.com,slave2.htdata.com/plugins/

1. GET /\_nodes/ingest

## 查看节点使用特性

1. 查看所有节点的使用特性

GET \_nodes/usage

1. 查看一个或多个节点的使用特性

GET /\_nodes/slave1.htdata.com,slave2.htdata.com/usage

# curl命令\_cache系列

## 定期清理cache

为避免fields data占用大量的jvm内存，可以通过定期清理的方式来释放缓存的数据。释放的内容包括field data、filter cache、query cache等。

清理操作：

curl -XPOST http://125.70.163.48:50029/\_cache/clear

# 索引增删改查操作

## 创建索引（空mapping）

curl -XPUT http://192.168.1.101:9200/customer?pretty

（查看所有索引：curl -XGET http://192.168.1.101:9200/\_cat/indices?v

curl -XGET http://192.168.1.101:9200/\_cat/indices/customer?v

）

//设置分片数及备份数

curl -XPUT http://192.168.1.101:9200/customer?pretty -d '{ "index": {"number\_of\_shards": 10, "number\_of\_replicas": 0} }'

或

curl -XPUT http://192.168.1.101:9200/customer?pretty -d '{ "settings": { "index": {"number\_of\_shards": 12, "number\_of\_replicas": 0} } }'

或

curl -XPUT http://192.168.1.101:9200/customer?pretty -d '{ "settings": { "number\_of\_shards": 12, "number\_of\_replicas": 0 } }'

## mapping

mapping不仅告诉ES一个field中是什么类型的值，它还告诉ES如何索引数据以及数据是否能被搜索到。

每一个索引有一种或多种类型（mapping type），一个类型是你的索引的一个逻辑上的分类/分区，其语义完全由你来定。

同一个mapping type是具有相同属性定义的mapping。mapping type具有的属性包括以下两种类型：

* 元属性：如\_index、\_type、\_id、\_score和\_source；
* 自定义属性。

### 查看mapping

curl -XGET "http://192.168.1.102:9200/customer/external/\_mapping?pretty";

### 创建mapping

1. type数据类型

|  |  |  |
| --- | --- | --- |
| **序号** | **类型** | **表示的数据类型** |
| 1 | String | string |
| 2 | Whole number | byte, short, integer, long |
| 3 | Floating point | float, double |
| 4 | Boolean | boolean |
| 5 | Date | date |

如果新增加一个字段，则Elasticsearch会根据数值默认设定type类型，如下所示：

Boolean: true or false "boolean"

Whole number: 123 "long"

Floating point: 123.45 "double"

String, valid date: "2014-09-15" "date"

String: "foo bar" "string"

注：如果你索引一个带引号的数字——"123"，它将被映射为"string"类型，而不是"long"类型。然而，如果字段已经被映射为"long"类型，Elasticsearch将尝试转换字符串为long，并在转换失败时会抛出异常。

curl -XPOST "http://192.168.0.101:9200/customer/external/\_mapping?pretty" -d '

{

"external": {

"properties": {

"name": { "type": "string" },

"age": { "type": "integer" },

"remark": { "type": "text" }

}

}

}'

1. index：这个参数可以控制字段应该怎样建索引，怎样查询。它有以下三个可用值：

**no：**不把此字段添加到索引中，也就是不建索引，此字段不可查询

**not\_analyzed**：将字段的原始值放入索引中，作为一个独立的term，它是除string字段以外的所有字段的默认值。

**analyzed**：string字段的默认值，会先进行分析后，再把分析的term结果存入索引中。（存放分析后的值，非原来的值）

注：对于不参与搜索的字段(fields), 将其index方法设置为no, 如果对分词没有需求，对参与搜索的字段，其index方法设置为not\_analyzed。

curl -XPOST "http://192.168.1.102:9200/customer/external/\_mapping?pretty" -d '

{

"external": {

"properties": {

"name": { "type": "string", "index": "not\_analyzed" },

"age": { "type": "integer" },

"remark": { "type": "text" }

}

}

}'

1. store："store":false/true

store就是把这个字段单独存储，默认是不存储的，默认存储的是\_source ，只有设置了才会存储，比如，有一个mapping有三个字段，title，subject，content，其中title，subject比较小，而content非常非常大，如果，你的查询结果只需要title，subject而不需要content时，把title，subject单独存储，可以节省很多时间。

curl -XPOST "http://192.168.1.102:9200/customer/external/\_mapping?pretty" -d '

{

"external": {

"properties": {

"name": { "type": "string", "store": false,"index": "not\_analyzed" },

"age": { "type": "integer", "store": false },

"remark": { "type": "text", "store": false}

}

}

}'

1. format：时间格式

"format":"YYYY-MM-dd" 或 yyyy-MM-dd HH:mm:ss

curl -XPOST "http://192.168.1.102:9200/customer/external/\_mapping?pretty" -d '

{

"external": {

"properties": {

"name": { "type": "string", "store": false,"index": "not\_analyzed" },

"age": { "type": "integer", "store": false },

"insertTime": { "type": "date", "store": false, "format": "yyy-MM-dd HH:mm:ss" },

"remark": { "type": "text", "store": false}

}

}

}'

1. dynamic：true/false/"strict"（如果碰到陌生字段，抛出异常），是否关闭自动添加字段，关闭后索引数据中如果有多余字段不会修改mapping,默认true（不关闭）。配置参数 dynamic 可以用在根object 或任何 object 类型的字段上。你可以将 dynamic 的默认值设置为strict , 而只在指定的内部对象中开启它。
2. \_source：是否存放原始文档。

"\_source": { "enabled": false}：表示不存放原始文档。

"\_source": { "enabled": true}：表示存放原始文档。（默认为true）

"\_source": { "includes": ["name","age"]}：表示仅存放name和age两个字段在文档里。

"\_source": { "excludes": ["age"]}：表示原始文档里不包含age字段。

curl -XPOST "http://192.168.1.102:9200/customer/external/\_mapping?pretty" -d '

{

"external": {

"\_source": { "enabled": false },

"properties": {

"name": { "type": "string", "store": false,"index": "not\_analyzed" },

"age": { "type": "integer", "store": false },

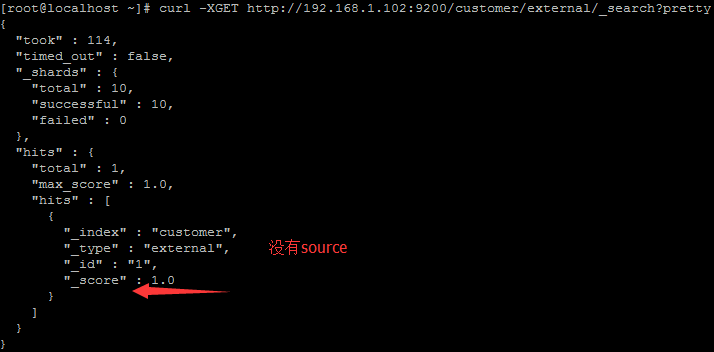
"insertTime": { "type": "date", "store": false, "format": "yyy-MM-dd HH:mm:ss" },

"remark": { "type": "text", "store": false}

}

}

}'



### 修改mapping

mapping创建完成后，不能修改之前的字段，只能新增一个字段。如增加一个sex性别字段。

curl -XPOST "http://192.168.1.102:9200/customer/external/\_mapping?pretty" -d '

{

"external": {

"properties": {

"sex": { "type": "string" }

}

}

}'

## 创建索引（非空mapping）

### 创建简单索引

curl -XPUT "http://192.168.1.102:9200/customer?pretty" -d '

{

"mappings": {

"external": {

"properties": {

"name": { "type": "string", "store": false,"index": "not\_analyzed" },

"age": { "type": "integer", "store": false },

"insertTime": { "type": "date", "store": false, "format": "yyy-MM-dd HH:mm:ss" },

"remark": { "type": "text", "store": false}

}

}

}

}'

### 创建索引同时设置分片数和备份

curl -XPUT "http://192.168.1.102:9200/customer?pretty" -d '

{

"settings": {

"number\_of\_shards": 12,

"number\_of\_replicas": 0

},

"mappings": {

"external": {

"properties": {

"name": { "type": "string", "store": false,"index": "not\_analyzed" },

"age": { "type": "integer", "store": false },

"insertTime": { "type": "date", "store": false, "format": "yyy-MM-dd HH:mm:ss" },

"remark": { "type": "text", "store": false}

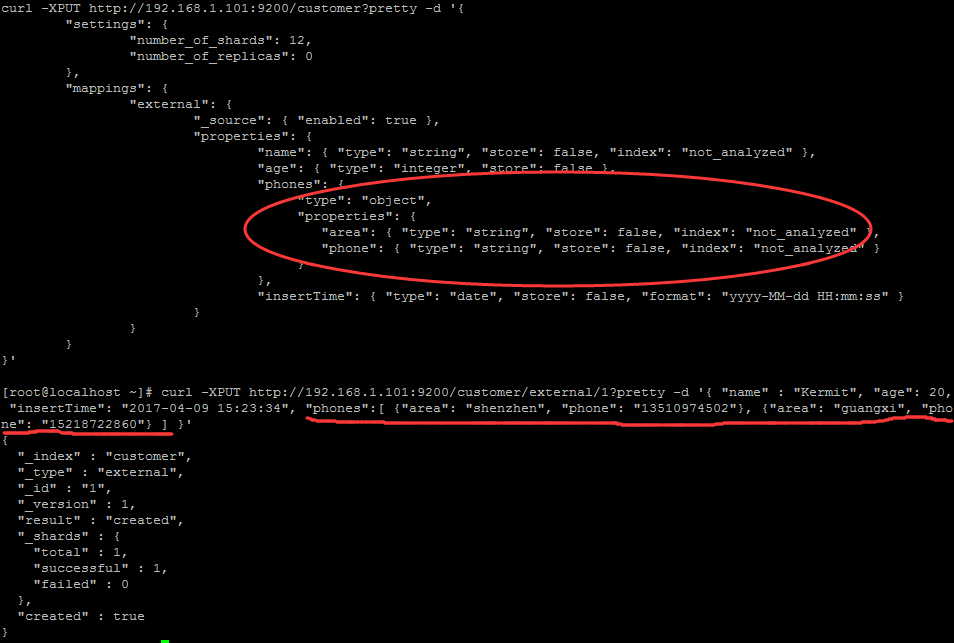
}

}

}

}'

## 创建对象索引

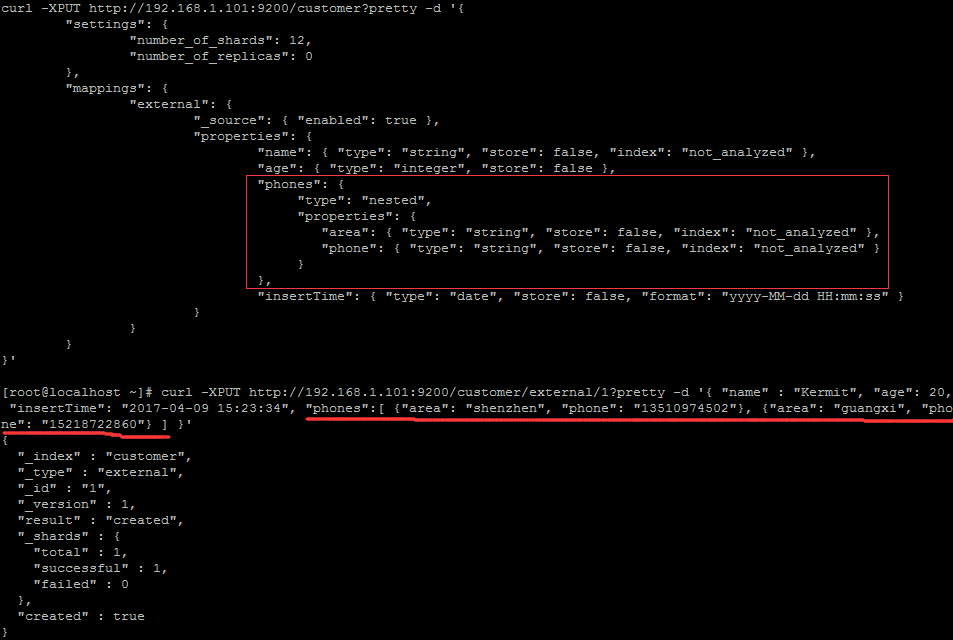


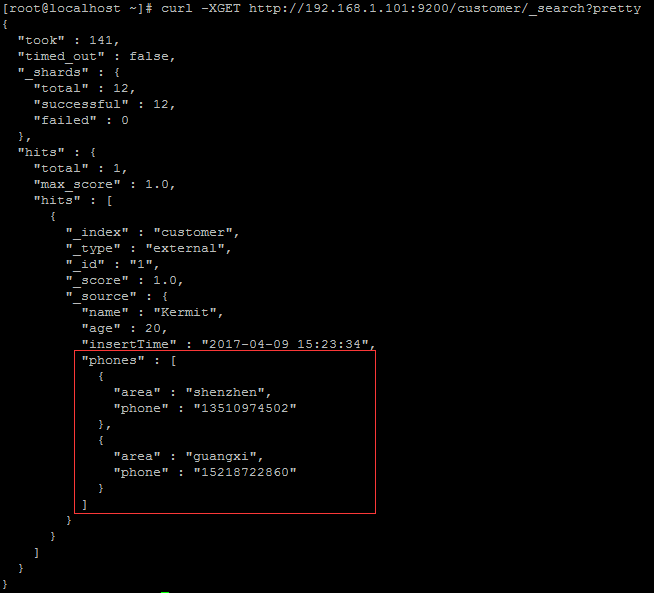


## 创建嵌套索引

嵌套对象允许我们连接一个主文档和多个附属文档，将信息分多个文档存放。例如现在有一种名字为”cloth”的服装，这件服装现有两件存货，一件XXL的红色和一件XL的黑色。如果当普通对象存放，即不将两件服装分开存放，当查询XXL的黑色时，将有结果返回，而实际上是没有的。

我们通过”type”:”nested”将一个对象指定为嵌套对象。





## 查询索引

curl -XGET http://192.168.1.101:9200/\_cat/indices?v

## 删除索引

curl -XDELETE <http://192.168.1.101:9200/customer?pretty>

## 修改索引配置

及时生效。

### 修改备份数

curl -XPUT http://192.168.1.101:9200/customer/\_settings -d { "index" : { "number\_of\_replicas" : 4 } }

//单个索引

PUT /customer/\_settings

{ "index" : { "number\_of\_replicas" : 0 } }

//全部索引

PUT /\_settings

{ "index" : { "number\_of\_replicas" : 0 } }

### 修改刷新间隔时间

索引数据的过程直接将数据写入磁盘（fsync）必然会影响性能，所以es中设计了一种机制，即：先将index-buffer中文档（document）解析完成的segment写到filesystem cache之中，这样避免了比较损耗性能io操作，又可以使document可以被搜索。以上从index-buffer中取数据到filesystem cache中的过程叫做refresh。es默认的refresh间隔时间是1s，这样使得索引进去的数据能够实时地查询出来。

当大量索引数据时，可以适当加大刷新间隔时间或关闭refresh（取值“-1”）。修改刷新间隔时间如下所示：

PUT /customer/\_settings

{

"index" : {

"refresh\_interval": "10s"

}

}

关闭刷新

PUT /customer/\_settings

{

"index" : {

"refresh\_interval": "-1"

}

}

### 设置数据flush方式及修改translog文件大小

如果数据在filesystem cache之中是很有可能在意外的故障中丢失。这个时候就需要一种机制，可以将对es的操作记录下来，来确保当出现故障的时候，保留在filesystem的数据不会丢失，并在重启的时候可以从这个记录中将数据恢复过来。elasticsearch提供了translog来记录这些操作。

当向elasticsearch发送创建document索引请求的时候，document数据会先进入到index buffer之后，与此同时会将操作记录在translog之中，当发生refresh时（数据从index buffer中进入filesystem cache的过程）translog中的操作记录并不会被清除，而是当数据从filesystem cache中被写入磁盘之后才会将translog中清空。而从filesystem cache写入磁盘的过程就是flush（每个分片都会执行flush操作）。修改相应的参数，可适当地降低数据flush到磁盘的频率，提高索引数据的吞吐量。

有关于translog和flush的一些配置项：

* index.translog.flush\_threshold\_ops:当发生多少次操作时进行一次flush。默认是 unlimited。
* index.translog.flush\_threshold\_size:当translog的大小达到此值时会进行一次flush操作。默认是512mb。
* index.translog.flush\_threshold\_period:在指定的时间间隔内如果没有进行flush操作，会进行一次强制flush操作。默认是30m。
* index.translog.interval:多少时间间隔内会检查一次translog，来进行一次flush操作。es会随机的在这个值到这个值的2倍大小之间进行一次操作，默认是5s。
* index.translog.durability：flush方式，有两种方式：async（异步）和request（同步），默认是request。async异步存在数据丢失的情况。

修改：

curl -XPUT http://localhost:9200/test\_index/\_settings/ -d '{

"index.translog.durability": "async",

"index.translog.flush\_threshold\_size": "1024mb"

}'

## 重建索引（Reindex）

5.X版本后新增Reindex，Reindex可以直接在Elasticsearch集群对数据进行重建，如果mapping因为修改而需要重建，又或者索引设置修改需要重建的时候，借助Reindex可以很方便的异步进行重建，并且支持跨集群间的数据迁移。

\_reindex的最基本形式只是将文档从一个索引复制到另一个索引。

注意：

1. 索引里面要启用\_source；
2. 重建索引时，不会拷贝源索引的设置，在运行\_reindex操作之前设置目标索引，包括设置映射，分片计数，副本等。

### 简单的重建索引

举例，现有一个customer索引，但日期类型为text，需要将其改成date。customer索引结构如下所示：

{

"customer" : {

"mappings" : {

"external" : {

"properties" : {

"age" : {

"type" : "long"

},

"createTime" : {

"type" : "text",

"fields" : {

"keyword" : {

"type" : "keyword",

"ignore\_above" : 256

}

}

},

"name" : {

"type" : "text",

"fields" : {

"keyword" : {

"type" : "keyword",

"ignore\_above" : 256

}

}

}

}

}

}

}

}

1. 首先，创建一个新的索引customer\_01,包括设置映射，分片计数，副本等；

PUT /customer\_01

{

"settings": {

"number\_of\_shards": 12,

"number\_of\_replicas": 0

},

"mappings": {

"external": {

"properties": {

"name": { "type": "keyword", "store": false },

"age": { "type": "integer", "store": false },

"createTime": { "type": "date", "store": false, "format": "yyy-MM-dd HH:mm:ss" },

"remark": { "type": "text", "store": false }

}

}

}

}

1. 其次，将源索引（customer）的数据拷贝到目标索引（customer\_01）。

POST \_reindex

{

"source": {

"index": "customer"

},

"dest": {

"index": "customer\_01"

}

}

### 数据版本冲突处理

6.9.1中的重建索引是将源索引的快照复制到新的目标索引，因此不太可能发生版本冲突，这是因为省略version\_type或将其设置为internal将导致Elasticsearch盲目地将文档转储到目标中，覆盖任何碰巧具有相同类型和id的文档。

POST \_reindex

{

"source": {

"index": "customer"

},

"dest": {

"index": "customer\_01",

"version\_type": "internal"

}

}

如果设置op\_type 为 create，将导致\_reindex仅在目标索引中创建缺少的文档，所有已存在于目标索引中的文档都会导致版本冲突。

POST \_reindex

{

"source": {

"index": "customer"

},

"dest": {

"index": "customer\_01",

"op\_type": "create"

}

}

目标索引已存在的版本号，将会导致出错。如下所示：



默认情况下，版本冲突会中止\_reindex进程，但可以通过设置（"conflicts": "proceed"）来计算它们，让操作继续执行下去。

POST \_reindex

{

"conflicts": "proceed",

"source": {

"index": "customer"

},

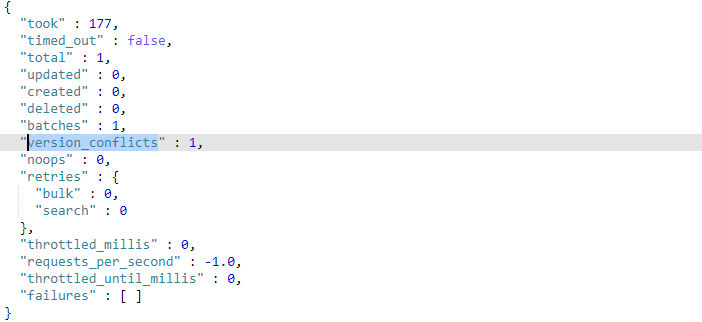
"dest": {

"index": "customer\_01",

"op\_type": "create"

}

}



### 同步时指定源索引类型或相应的文档

POST \_reindex

{

"conflicts": "proceed",

"source": {

"index": "customer",

"type": "external",

"query": {

"term": {

"name": "kermit"

}

}

},

"dest": {

"index": "customer\_01",

"op\_type": "create"

}

}

### 同步时指定多个源索引或多个类型

POST \_reindex

{

"source": {

"index": ["twitter", "blog"],

"type": ["external", "post"]

},

"dest": {

"index": "all\_together",

"type": " external"

}

}

### 同步时限制文档数量

POST \_reindex

{

"size": 10,

"source": {

"index": "customer",

"type": "external"

},

"dest": {

"index": "customer\_01",

"type": "external"

}

}

上述限制仅同步10条记录。

### 同步时对源索引文档进行排序

POST \_reindex

{

"size": 10,

"source": {

"index": "customer",

"type": "external",

"sort": {

"age": "desc"

}

},

"dest": {

"index": "customer\_01",

"type": "external"

}

}

### 同步时限定源索引的字段

POST \_reindex

{

"source": {

"index": "customer",

"type": "external",

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

},

"dest": {

"index": "customer\_01",

"type": "external"

}

}

### 同步时通过脚本修改源索引文档

POST \_reindex

{

"source": {

"index": "customer",

"type": "external"

},

"dest": {

"index": "customer\_01",

"type": "external"

},

"script": {

"source": "if (ctx.\_source.age >= 20) { ctx.\_source.age += 5 }",

"lang": "painless"

}

}

### 同步时指定滚动批次的量

默认情况下，\_reindex使用1000进行批量操作，可以使用source元素中的size字段更改批量大小。

POST \_reindex

{

"source": {

"index": "customer",

"type": "external",

"size": 100

},

"dest": {

"index": "customer\_01",

"type": "external"

}

}

### 同步时指定远程源索引

POST \_reindex

{

"source": {

"remote": {

"host": "http://otherhost:9200",

"username": "user",

"password": "pass",

"socket\_timeout": "1m",

"connect\_timeout": "10s"

},

"index": "source",

"query": {

"match": {

"test": "data"

}

}

},

"dest": {

"index": "dest"

}

}

该host参数必须包含方案，主机和端口（例如 https://otherhost:9200）。username、password、socket\_timeout、connect\_timeout参数是可选的，并且当它们存在时\_reindex将连接到使用基本认证远程Elasticsearch节点。https使用基本身份验证时一定要使用，否则密码将以纯文本形式发送。

注：必须将远程集群配置在elasticsearch.yml文件中（每个节点都要配置），使用reindex.remote.whitelist属性指定。例如：

reindex.remote.whitelist: "otherhost:9200, another:9200, 127.0.10.\*:9200, localhost:\*"

# 数据增删改操作

es提供两种搜索方式：

1. 通过url地址直接搜索
2. 封装请求体搜索（封装参数在请求体中）

## 插入数据

curl -XPUT http://192.168.1.101:9200/customer/external/1?pretty -d '{ "name" : "Kermit", "age": 20 }'

（查看数据：curl -XGET http://192.168.1.101:9200/customer/external/1?pretty）

索引（\_index）：customer

类型（\_type）：external

ID（\_id）：1（ID为主键，如果不给值，则es会自动生成，由\_id通过hash方法确定文档存放在哪个主分片上。）

如果ID不给定值，则使用以下方式插入数据：（XPOST代替XPUT）

curl -XPOST http://192.168.1.101:9200/customer/external?pretty -d '{ "name" : "Lang"}'

如果没有创建索引，es将会自动创建索引，并生成mapping。

## 插入数据并指定路由（给指定路由搜索）

执行搜索时，它将会广播到所有索引/索引碎片(副本之间的轮询)。可以通过提供路由参数来控制要搜索哪些碎片，这样可以避免搜索所有的索引/索引碎片。如下所示：

curl -XPOST http://192.168.1.101:9200/customer/external?routing=Lily -d '{ "name" : "Lily", "age" : 12}'

上述指定了一个以姓名（可以是其他值）作为路由，这样搜索的时候，可以指定路由进行查询。

curl -XPOST http://192.168.1.101:9200/customer/\_search?routing=Lily

## 更新数据

### 替换更新方式

curl -XPUT http://192.168.224.242:9200/customer/external/1?pretty -d '{ "name" : "Kermit", "age": 26 }'

如果所指定的\_id不存在，则会索引数据到es；如果所指定的\_id存在，则会替换已存在的文档。

### doc更新方式

这种方式支持更新部分文档，使用doc可以实现简单的递归合并、内部合并、替换KV以及数组。这种方法不会替换原文档，只会添加或修改key-value值。例如：

curl -XPOST http://192.168.224.242:9200/customer/external/1/\_update?pretty -d '{ "doc": { "name" : "Kermit", "age" : 20 } }'

如果使用doc，那么会自动合并到现有的文档中。如果doc中定义的部分与现在的文档相同，则默认不会执行任何动作。设置detect\_noop=false，就会无视是否修改，强制合并到现有的文档。例如：

curl -XPOST http://192.168.224.242:9200/customer/external/1/\_update?pretty -d '{ "doc": { "name" : "Kermit", "age" : 20 }, "detect\_noop": false }'

### script更新方式

script更新操作允许ES获得某个指定的文档，可以通过脚本等操作对该文档进行更新。

1. 更新数据

POST /customer/external/1/\_update?pretty

{

"script": "ctx.\_source.age += 5"

}

POST customer/external/1/\_update

{

"script" : {

"source": "ctx.\_source.age += params.num",

"params" : {

"num" : 4

}

}

}

1. 移除字段

POST /customer/external/1/\_update?pretty

{

"script": "ctx.\_source.remove(\"age\")"

}

1. 添加字段

POST /customer/external/1/\_update?pretty

{

"script": "ctx.\_source.age = 20"

}

除了\_source字段，可以通过ctx来获得\_index、\_type、\_id、\_version、\_parent、\_timestamp、\_ttl等字段信息。

## 删除数据

### 根据id删除

如果索引不存在，该删除语句将会自动创建相应的索引。根据id删除数据，仅标记，不删除文档。

DELETE /customer/external/1?pretty

### 根据指定条件删除

根据条件删除数据，仅标记，不删除文档。

POST /customer/external/\_delete\_by\_query?pretty

{

"query": {

"bool": {

"filter": {

"range": {

"age": {

"gte": 29,

"lte": 30

}

}

}

}

}

}

curl -XPOST 'http://192.168.1.101:9200/bank/account/\_delete\_by\_query?pretty' -d '{

"query": {

"range": {

"balance": {

"gte": 29000,

"lte": 30000

}

}

}

}'

### 根据指定条件批量删除

默认每1000条滚动批量删除，可以使用scroll\_size参数来设置每次批量删除的条数。

curl -XPOST 'http://192.168.1.101:9200/bank/account/\_delete\_by\_query?pretty&scroll\_size=2000' -d '{

"query": {

"range": {

"balance": {

"gte": 29000,

"lte": 30000

}

}

}

}'

还有其他参数：refresh, wait\_for\_completion, wait\_for\_active\_shards, and timeout

## 批量操作Batch Processing

除了能够索引、更新和删除单个文档外，Elasticsearch还提供了使用\_bulk API批量执行上述任何操作的能力。这个功能非常重要，因为它提供了一种非常有效的机制，可以以尽可能少的网络往返尽可能快地执行多个操作。

在对大量数据执行批量操作时，建议提高refresh时间。详见6.8.2

### 批量添加

POST /customer/external/\_bulk?pretty

{"index":{"\_id":"110"}}

{"name": "李逵", "age": 45 }

{"create":{"\_id":"112"}}

{"name": "诸葛亮", "age": 75 }

或

POST /\_bulk?pretty

{"index":{"\_index":"customer","\_type":"external", "\_id":"110"}}

{"name": "李逵", "age": 45 }

{"create":{"\_index":"customer","\_type":"external", "\_id":"112"}}

{"name": "诸葛亮", "age": 75 }

### 批量更新

POST /customer/external/\_bulk?pretty

{"update":{"\_id":"110"}}

{"doc": {"name": "李逵", "age": 65}}

{"index":{"\_id":"112"}}

{"name": "诸葛亮", "age": 45}

### 批量删除

POST /customer/external/\_bulk?pretty

{"delete":{"\_id":"110"}}

{"delete":{"\_id":"112"}}

### 添加、更新、删除同时执行

POST /customer/external/\_bulk?pretty

{"index":{"\_id":"110"}}

{"name": "李逵", "age": 45 }

{"index":{"\_id":"112"}}

{"name": "诸葛亮", "age": 75 }

{"update":{"\_id":"110"}}

{"doc": {"name": "李逵", "age": 65}}

{"delete":{"\_id":"AWdY76bTaapDnyJwPpYa"}}

### 从文件中批量导入数据

//批量插入数据集

curl -XPOST 'http://192.168.1.101:9200/bank/account/\_bulk?pretty&refresh' -H "Content-Type: application/json" --data-binary "@accounts.json"

accounts.json：数据集，格式如下：

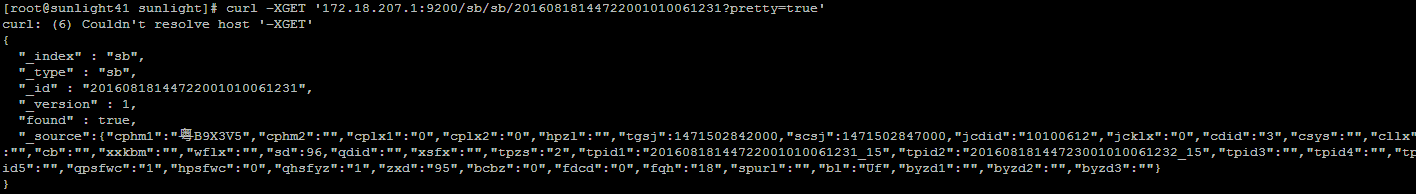
{"index":{"\_id":"1"}}

{"account\_number":1,"balance":39225,"firstname":"Amber","lastname":"Duke","age":32,"gender":"M","address":"880 Holmes Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}

# 数据搜索操作

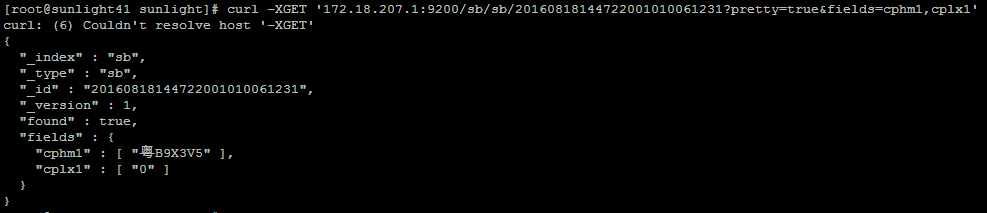
## 根据ID获取数据

curl -XGET http://localhost:9200/index/type/id



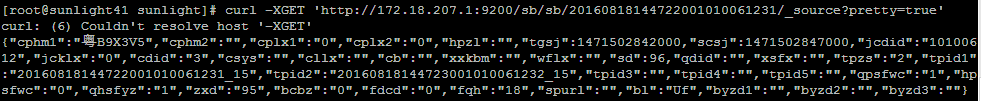
过滤字段：

curl -XGET 172.18.207.1:9200/sb/sb/20160818144722001?pretty=true&fields=cphm1,cplx1



仅获取\_source字段：

curl -XGET http://172.18.207.1:9200/sb/sb/201608181447220/\_source?pretty=true



## 查询数据（\_search）与统计数量（\_count）

curl -XGET http://192.168.1.101:9200/customer/\_search?pretty

curl -XGET http://192.168.1.101:9200/customer/external/\_search?pretty

注：查询数量时，将\_search改为\_count即可。

## 两种基本的REST请求方式

运行搜索有两种基本方法：

1. 一种是通过REST请求URI发送搜索参数（REST request URI），即请求参数封装在URI地址后面；
2. 另一种是通过REST请求体发送参数（REST request body）。request body方法更有表现力，还可以用更可读的JSON格式定义搜索。

## URI Search（URI搜索）

可以通过提供请求参数纯粹使用URI来执行搜索请求。在URI中可以使用的参数如下：

## Request Body Search（封装请求体搜索）

string类型在es5.\*分为text和keyword。text是要被分词的，整个字符串根据一定规则分解成一个个小写的term，keyword类似es2.3中not\_analyzed的情况。

string数据put到elasticsearch中，默认是text，默认分词器为standard analyzer。如果字段要实现分词搜索，则必须是text类型，keyword 、numeric、date等类型字段不支持分词搜索。

### 根据条件查询数据与统计数量

term是精确查询，match是模糊查询。

1. 无条件查询，即查询所有的数据

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"match\_all": {}

}

}'

1. 匹配一个条件（match属于分词搜索，term属于精确搜索）

// account\_number等于20的记录

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"match": { "account\_number": 20 }

}

}'

//地址中包含mill的记录

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"match": { "address": "mill" }

}

}'

//地址中包含mill或者Lane的记录

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"match": { "address": "mill lane" }

}

}'

注：查询数量时，将\_search改为\_count即可。

### term精确搜索

term是代表完全匹配，也就是精确查询，搜索前不会再对搜索词进行分词，所以我们的搜索词必须是文档分词集合中的一个。这种查询适合**keyword** 、**numeric**、**date**。例如：

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "query" : { "term" : { "age": 40 } } }'

注意：有些字段类型无法进行完全匹配。

### match分词搜索

match查询会先对搜索词进行分词,分词完毕后再逐个对分词结果进行匹配。match搜索还有两个相似功能的变种，一个是match\_phrase，一个是multi\_match。

对于最基本的match搜索来说，只要搜索词的分词集合中的一个或多个分词存在于文档中即可，例如，当我们搜索中国杭州，搜索词会先分词为中国和杭州,只要文档中包含搜索和杭州任意一个词，都会被搜索到。

注：如果字段要实现分词搜索，则必须是text类型，keyword 、numeric、date等类型字段不支持分词搜索。

1. 对指定字段进行分词搜索

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "query" : { "match" : { "name": "中国杭州" } } }'

### match\_phrase短语搜索

match\_phrase为按短语搜索，即不会对词语进行拆分。比如中国杭州、love you看作是两个词，不会对中国杭州、love you进行拆分。如下所示：

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "query" : { "match\_phrase" : { "name": "中国杭州" } } }'

注：如果字段要实现分词搜索，则必须是text类型，keyword 、numeric、date等类型字段不支持分词搜索。

### multi\_match多字段分词搜索

如果希望对一个或多个字段进行匹配，其中一个字段有这个文档就满足的话，则可使用multi\_match。

注：如果字段要实现分词搜索，则必须是text类型，keyword 、numeric、date等类型字段不支持分词搜索。

以下示例可以指定一个或多个索引，也可以不指定索引即全部索引。

1. 对指定的一个字段进行分词搜索

curl -XGET http://125.70.163.48:50029/\_search?pretty -d '{ "query" : { "multi\_match" : { "query": "中 杭", "fields": "name" } } }'

1. 对指定的一个或多个字段进行分词搜索

curl -XGET http://125.70.163.48:50029/\_search?pretty -d '{ "query" : { "multi\_match" : { "query": "中 杭 kimchy", "fields": ["name", "user"] } } }'

1. 如果字段中有数字类型，则需要加入"lenient": "true"

curl -XGET http://125.70.163.48:50029/\_search?pretty -d '{ "query" : { "multi\_match" : { "query": "中 杭", "fields": ["name", "age"], "lenient": "true" } } }'

1. 对一个字段名称模糊进行分词搜索

curl -XGET http://125.70.163.48:50029/\_search?pretty -d '{ "query" : { "multi\_match" : { "query": "中 杭", "fields": "nam\*", "lenient": "true"} } }'

1. 对一个或多个字段名称模糊进行分词搜索

curl -XGET http://125.70.163.48:50029/\_search?pretty -d '{ "query" : { "multi\_match" : { "query": "中 杭 kimchy", "fields": ["\*ame\*", "use\*"], "lenient": "true" } } }'

1. 全字段进行分词搜索

GET /\_search?pretty

{

"query" : {

"multi\_match" : {

"query": "中 杭",

"fields": "\_all",

"lenient": "true"

}

}

}

1. 提高字段权重（给相应字段加分）

可以使用 ^ 字符语法为单个字段提升权重，在字段名称的末尾添加 ^boost ， 其中 boost 是一个浮点数。例如：

curl -XGET http://125.70.163.48:50029/\_search?pretty -d '{ "query" : { "multi\_match" : { "query": "中 杭 kimchy", "fields": ["\*ame\*", "use\*^3"], "lenient": "true" } } }'

由于是多个字段，则multi\_match会涉及到评分问题，可以根据情况对结果进行评分。如完全匹配的文档占的评分比较高，则需要使用best\_fields（最佳字段）；越多字段匹配的文档评分越高，就要使用most\_fields（多数字段）；希望这个词条的分词词汇是分配到不同字段中的，那么就使用cross\_fields（跨字段）。如下所示：

curl -XGET http://125.70.163.48:50029/\_search?pretty -d '{ "query" : { "multi\_match" : { "query": "中 杭", "fields": ["name"], "type": "best\_fields", "tie\_breaker": 0.3 } } }'

完全匹配"中 杭"的文档评分会比较靠前，如果只匹配一个的文档评分乘以0.3的系数。

### Bool Query

bool查询允许我们根据布尔逻辑将多个条件组合进行查询。

1. “与”组合，必须满足所有条件

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"bool": {

"must": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}'

1. “或”组合，满足其中一个条件即可

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"bool": {

"should": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}'

1. “非”组合，一个条件都不满足

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"bool": {

"must\_not": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}'

1. “与”、“或”、“非”组合

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"bool": {

"must": [

{ "match": { "account\_number": 20 } }

],

"must\_not": [

{ "match": { "address": "Place" } }

]

}

}

}'

//查询姓名为kermit，且年龄为20或40岁

GET /customer/external/\_search?pretty

{

"query": {

"bool": {

"must": [

{ "match": { "name": "kermit" } },

{

"bool": {

"should": [

{ "match": { "age": 20 } },

{ "match": { "age": 40 } }

]

}

}

]

}

}

}

注：查询数量时，将\_search改为\_count即可。

### 通配符查询（**Wildcard Query**）

注意：模糊查询要求字段类型必须是not\_analyzed，即keyword类型（term查询），这样才能使用?和\*，对text类型的字段无效（只能做分词搜索）。

1. 单个字符使用?

GET /\_search?pretty

{

"query": {

"bool": {

"must": [

{ "wildcard": { "XM": "李?龙"} }

]

}

}

}

1. 多个字符使用\*

GET /\_search?pretty

{

"query": {

"bool": {

"must": [

{ "wildcard": { "XM": "李\*"} }

]

}

}

}

### 正则表达式查询（Regexp Query）

regexp允许使用正则表达式进行term查询。注意regexp如果使用不正确，会给服务器带来很严重的性能压力。比如.\*开头的查询，将会匹配所有的倒排索引中的关键字，这几乎相当于全表扫描，会很慢。因此如果可以的话，最好在使用正则前，加上匹配的前缀。在正则中如果使用.\*?或者+都会降低查询的性能。

### 范围查询（Range Query）

range query只能用在数值或日期类型的字段。

\_score字段这个得分是与我们指定的搜索查询匹配程度的一个相对度量。得分越高，文档越相关，得分越低文档的相关度越低。

Elasticsearch中的所有的查询都会触发相关度得分的计算。对于那些我们不需要相关度得分的场景下，Elasticsearch以过滤器的形式提供了另一种查询功能。过滤器在概念上类似于查询，但是它们有非常快的执行速度，这种快的执行速度主要有以下两个原因：

过滤器不会计算相关度的得分，所以它们在计算上更快一些

过滤器可以被缓存到内存中，这使得在重复的搜索查询上，其要比相应的查询快出许多。

filter过滤可以嵌套在bool查询内部使用。

gt：大于

lt：小于

gte：大于或等于

lte：小于或等于

1. 查询balance范围在大于或等于29000，且小于或等于30000的记录

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"bool": {

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

"filter": {

"range": {

"balance": {

"gte": 29000,

"lte": 30000

}

}

}

}

}

}'

注：查询数量时，将\_search改为\_count即可。

### 字段过滤

1. 指定是否返回\_source，即是否返回数据字段与值

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "\_source": false, "query" : { "term" : { "age": 40 } } }'

注：默认为true。

1. 返回一个指定字段

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "\_source": "name", "query" : { "term" : { "age": 40 } } }'

1. 返回一个或多个指定字段

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"match\_all": {}

},

"\_source": ["account\_number", "balance"]

}'

或

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "\_source": ["name", "age"], "query" : { "term" : { "age": 40 } } }'

1. 返回包含指定字符的字段

一个模糊字符：

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "\_source": "\*am\*", "query" : { "term" : { "age": 40 } } }'

一个或多个模糊字符：

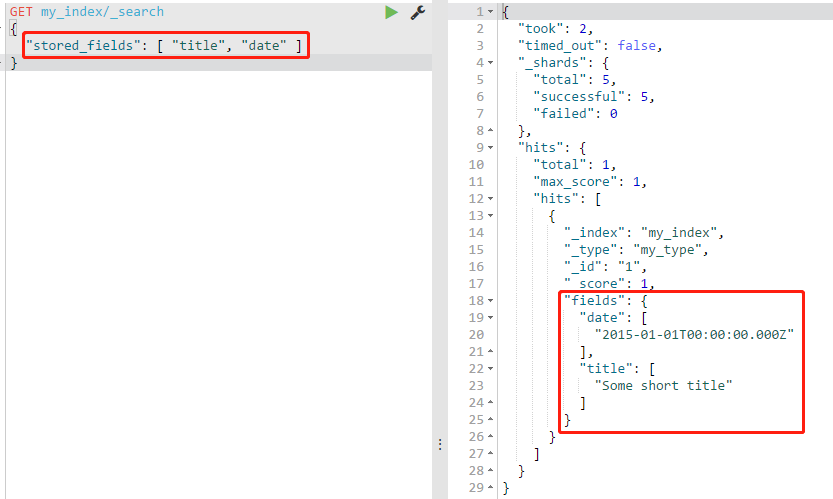
curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "\_source": ["\*am\*", "\*g\*"], "query" : { "term" : { "age": 40 } } }'

1. 包含或排除指定字段（可以是精确字段或模糊字段）

curl -XGET http://125.70.163.48:50029/customer/external/\_search?pretty -d '{ "\_source": { "includes": ["\*am\*", "\*g\*"], "excludes": ["\*g\*"] }, "query" : { "term" : { "age": 40 } } }'

### stored\_fields

stored\_fields默认是关闭的，用于获取单独存储的字段，使用stored\_fields字段将会导致\_source字段失效。例如：



注：title、date字段的"store"属性为true。

#### 返回所有store属性为true的字段

GET my\_index/\_search

{

"stored\_fields": [ "\*" ]

}

返回如下：

{

"took": 1,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 1,

"max\_score": 1,

"hits": [

{

"\_index": "my\_index",

"\_type": "my\_type",

"\_id": "1",

"\_score": 1,

"fields": {

"date": [

"2015-01-01T00:00:00.000Z"

],

"title": [

"Some short title"

]

}

}

]

}

}

#### 不返回store属性为true的字段，除\_id、\_type外

GET my\_index/\_search

{

"stored\_fields": [ ]

}

返回如下：

{

"took": 2,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 1,

"max\_score": 1,

"hits": [

{

"\_index": "my\_index",

"\_type": "my\_type",

"\_id": "1",

"\_score": 1

}

]

}

}

#### 完全禁用store属性为true的字段

GET my\_index/\_search

{

"stored\_fields": "\_none\_"

}

返回如下：

{

"took": 1,

"timed\_out": false,

"\_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {

"total": 1,

"max\_score": 1,

"hits": [

{

"\_index": "my\_index",

"\_score": 1

}

]

}

}

### Script Fields

对一个或多个字段进行计算，得出一个或多个新字段值。如下所示：

GET customer/external/\_search

{

"query" : {

"match\_all": {}

},

"script\_fields" : {

"age\_1" : {

"script" : {

"lang": "painless",

"source": "doc['age'].value \* 2"

}

},

"age\_2" : {

"script" : {

"lang": "painless",

"source": "doc['age'].value \* params.factor",

"params" : {

"factor" : 2.0

}

}

}

}

}

以上是通过doc方式实现，该方式将相关字段的值缓存到内存中进行计算，不需要加载所有\_source并进行分析。下面这种方式需要加载所有\_source并进行分析，相对比较慢。

GET customer/external/\_search

{

"query" : {

"match\_all": {}

},

"script\_fields" : {

"age" : {

"script" : "params['\_source']['age'] \* 2"

}

}

}

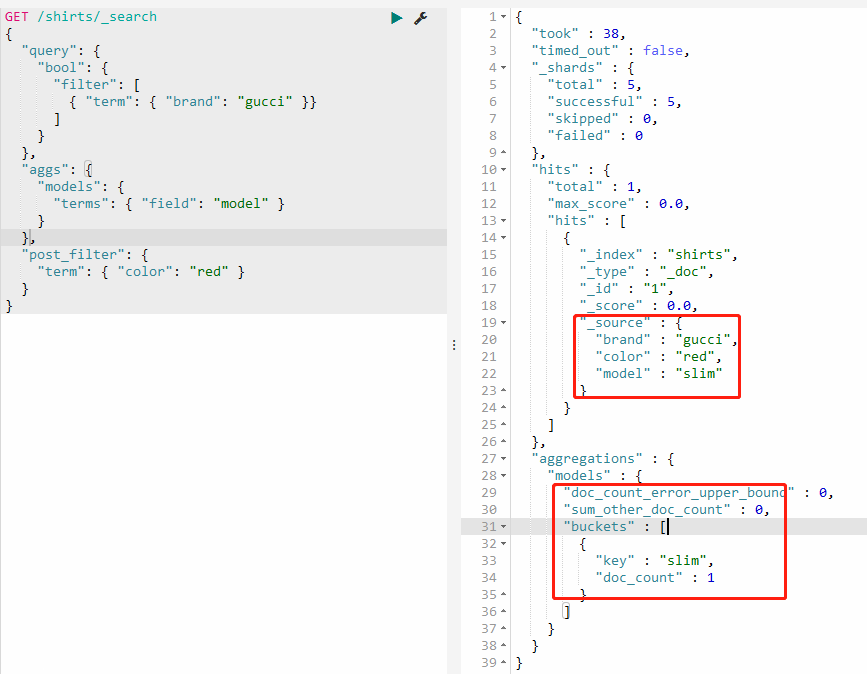
### Doc value Fields

### Post filter（后置过滤）

一般的过滤（filter）是针对数据进行筛选，而Post filter是对搜索结果（对执行完后的结果进行筛选）进行过滤。Post filter一般结合聚合使用，其对聚合结果没有影响，仅对搜索结果有影响。如下比较：









### Highlighting（高亮）

高亮显示使您能够从搜索结果中的一个或多个字段中获得高亮显示的代码片段，以便向用户显示查询匹配的位置。当您请求高亮显示时，响应将为每个搜索命中包含一个额外的高亮元素，其中包括高亮显示的字段和高亮显示的片段。

为了执行高亮显示，该字段必须有实际的内容，并且这个字段必须存储，即在mapping中store设为true,不能只存在于内存中，否则系统会自动加载\_source字段并匹配相关的列。



#### 高亮器类型

1. Unified highlighter

unified高亮器使用Lucene统一高亮器。 这个高亮器将文本分解为句子，并使用BM25算法对单个句子进行评分，就好像它们是文集中的文档一样。 它还支持准确的短语和多项（模糊，前缀，正则表达式）突出显示。 这是默认的高亮器。

1. Plain highlighter

plain高亮器使用标准的Lucene高亮器。它试图在短语查询中理解单词重要性和任何单词定位标准来反映查询匹配逻辑。

1. Fast Vector highlighter

fvh高亮器使用Lucene Fast Vector高亮器。此高亮器可用于在映射中将term\_vector设置为with\_positions\_offsets的字段。 快速向量高亮器：

* 可以使用boundary\_scanner进行自定义。
* 需要将term\_vector设置为with\_positions\_offsets，这会增加索引的大小
* 可以将来自多个字段的匹配组合成一个结果。
* 可以为不同的位置上的匹配分配不同的权重，从而允许在突出显示提高词条匹配的term匹配

#### 设置高亮类型

GET /customer/external/\_search

{

"query": {

"term": { "name": "Lang" }

},

"highlight" : {

"fields" : {

"name" : {},

"comment" : {"type" : "plain"}

}

}

}

#### 自定义高亮标签

默认的标签是<em>和</em>，可以根据需要自定义高亮标签。如下所示：

GET /customer/external/\_search

{

"query": {

"term": { "remark": "are" }

},

"highlight" : {

"pre\_tags": ["<mark>"],

"post\_tags": ["</mark>"],

"fields" : {

"remark" : {}

}

}

}

Fast Vector highlighter类型用到：

GET /customer/external/\_search

{

"query": {

"term": { "remark": "are" }

},

"highlight" : {

"pre\_tags": ["<mark1>", "<mark2>"],

"post\_tags": ["</mark1>", "</mark2>"],

"fields" : {

"remark" : {}

}

}

}

#### 多字段高亮

默认情况下，只有包含查询匹配的字段才会突出显示。 因为默认require\_field\_match值为true，可以设置为false以突出显示所有字段。

GET /customer/external/\_search

{

"query": {

"term": { "name": "are" }

},

"highlight" : {

"pre\_tags": ["<mark1>"],

"post\_tags": ["</mark1>"],

"require\_field\_match": false,

"fields" : {

"name": {},

"remark" : {}

}

}

}

### 分页查询

* 限定记录数

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"match\_all": {}

},

"size": 1

}'

* 分页获取记录

curl -XGET 'http://192.168.1.101:9200/bank/account/\_search?pretty' -d '{

"query": {

"match\_all": {}

},

"from": 2,

"size": 2

}'

注意：from从0开始，size从1开始，from+size不能超过index.max\_result\_window这个配置的值。

### 数据排序

注意：分词等类型字段不能用于排序。

#### 单个字段排序

GET /customer/external/\_search?pretty

{

"query": {

"match\_all": {}

},

"sort": [

{"age": "asc" }

]

}

或

GET /customer/external/\_search?pretty

{

"query": {

"match\_all": {}

},

"sort": [

{"age": {"order": "asc"} }

]

}

#### 按得分排序

GET /customer/external/\_search?pretty

{

"query": {

"match\_all": {}

},

"sort": [

{"\_score": {"order": "desc"} }

]

}

或

GET /customer/external/\_search?pretty

{

"query": {

"match\_all": {}

},

"sort": [

"\_score"

]

}

注意：\_score默认是desc倒序，其他字段默认是升序。

#### 多个字段排序

GET /customer/external/\_search?pretty

{

"query": {

"match\_all": {}

},

"sort": [

{"name": { "order": "desc" }},

{"age": "asc" }

]

}

#### 对数组或多值字段排序

Elasticsearch支持按数组或多值字段进行排序，mode option用来挑选数组的值来对文档排序，mode option取值：

| 取值 | 含义 |
| --- | --- |
| min | 最小值 |
| max | 最大值 |
| sum | 所有值的和（只适用于数组字段） |
| avg | 所有值的平均值（只适用于数组字段） |
| median | 中值（只适用于数组字段） |

例如，下面文档中的价格有多个取值。我们希望按照文件平均价格递增排序：

PUT /my\_index/\_doc/1?refresh

{

"product": "chocolate",

"price": [20, 4]

}

POST /\_search

{

"query" : {

"term" : { "product" : "chocolate" }

},

"sort" : [

{"price" : {"order" : "asc", "mode" : "avg"}}

]

}

#### 对嵌套对象的字段排序

Elasticsearch还支持按一个或多个嵌套对象内的字段进行排序。

### 分组查询

## 多索引搜索

搜索时可以搜索单个索引、多个索引或所有索引进行搜索。

1. 指定几个索引

curl -XPOST http://125.70.163.48:50029/customer,twitter/\_search?pretty

上述指定搜索customer、twitter两个索引，以半角逗号分割，可指定多个索引。

1. 指定所有索引

curl -XPOST http://125.70.163.48:50029/\_all/\_search?pretty

或

curl -XPOST http://125.70.163.48:50029/\_search?pretty

## 聚合搜索

聚合提供了对数据进行分组和提取统计信息的能力。考虑聚合最简单的方法是将其大致等同于SQL GROUP by和SQL聚合函数。

## 词向量（Term Vectors）

# 问题解决

## 修复UNASSIGNED的分片

1. 首先，查看是否有UNASSIGNED分配。

curl -XGET 'http://125.70.163.48:50029/\_cat/shards' | grep UNASSIGNED

1. 其次，如果有，则将相应索引的备份数改为0，等待该索引的分片副本消失。

curl -XPUT "http://125.70.163.48:50029/customer/\_settings" -d '{

"index": {

"number\_of\_replicas": 0

}

}'

1. 然后，开始手动将未分配的分配进行分配

curl -XPOST "http://125.70.163.48:50029/\_cluster/reroute" -d '{

"commands": [{

"allocate\_replica": {

"index": "customer",

"shard": 0,

"node": "master.htdata.com",

"allow\_primary": "true"

}

}]

}'

注：如果报错，红色部分可以去掉。

1. 最后将分片备份数改回去。

curl -XPUT "http://125.70.163.48:50029/customer/\_settings" -d '{

"index": {

"number\_of\_replicas": 1

}

}'

考虑到副本的优化情况，建议在做该操作的同时，做一次段合并。以节约空间和减少文件句柄数。