**ES使用文档**

V1.0.0

吴志彬 @ Enlink 数据中心

**目录**

[1. 概述 3](#_Toc7048)

[1.1. 文档内容 3](#_Toc11211)

[1.2. ES的简介 3](#_Toc19661)

[2. 基本概念 4](#_Toc27995)

[2.1. Node（节点）与Cluster（集群） 4](#_Toc22054)

[2.2. Index（索引） 4](#_Toc31956)

[2.3. Type（类型） 4](#_Toc25674)

[2.4. Doceument（文档） 4](#_Toc24407)

[2.5. Mapping（映射） 4](#_Toc19986)

[2.6. Primary Shards&Replicas Shards（主分片和副本分片） 4](#_Toc17834)

[主分片： 4](#_Toc10549)

[副本分片： 4](#_Toc7820)

[3. 简单使用 5](#_Toc8657)

[3.1. 增加和删除索引 5](#_Toc25745)

[3.2. 查看所有索引 6](#_Toc4947)

[3.3. 查看集群上的所有节点 6](#_Toc19449)

[3.4. 查看集群的健康状况 6](#_Toc5496)

[3.5. 新增记录 7](#_Toc22556)

[3.6. 查看记录 9](#_Toc1661)

[3.7. 删除记录 9](#_Toc23908)

[3.8. 更新记录 10](#_Toc9150)

[3.9. 查询所有记录 11](#_Toc23535)

[3.10. 全文搜索 12](#_Toc3152)

[4. 映射（Mapping） 15](#_Toc16451)

[4.1. 为什么需要映射： 15](#_Toc9557)

[4.2. 映射类型 15](#_Toc17096)

[4.3. 字段数据类型 15](#_Toc22053)

[4.4. 确切值（Exact values）与全文文本（Full text）举例 16](#_Toc9181)

[4.5. 时间格式映射举例 19](#_Toc635)

[5. 动态模板（Dynamic templates） 21](#_Toc12737)

[5.1. 模板简述 21](#_Toc6979)

[5.2. 动态模板基本规则 21](#_Toc11949)

[5.3. 匹配条件 22](#_Toc8908)

[Match\_mapping\_type 22](#_Toc8399)

[Match and unmatch 24](#_Toc10068)

[path\_match and path\_unmatch 26](#_Toc8967)

[5.4. 模板示例 28](#_Toc3930)

# 概述

## 文档内容

本文档介绍ES的基本概念与增删改查的操作，增删改查使用的是cURL指令进行操作。

也可以使用如Postman这类API测试工具进行测试

## ES的简介

ES（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、分析以及将这三者混合使用：

* 维基百科使用Elasticsearch提供全文搜索并高亮关键字，以及输入实时搜索(search-asyou-type)和搜索纠错(did-you-mean)等搜索建议功能。 英国卫报使用Elasticsearch结合用户日志和社交网络数据提供给他们的编辑以实时的反 馈，以便及时了解公众对新发表的文章的回应。
* StackOverflow结合全文搜索与地理位置查询，以及more-like-this功能来找到相关的问 题和答案。
* Github使用Elasticsearch检索1300亿行的代码。

但是Elasticsearch不仅用于大型企业，它还让像DataDog以及Klout这样的创业公司将最初的 想法变成可扩展的解决方案。Elasticsearch可以在你的笔记本上运行，也可以在数以百计的 服务器上处理PB级别的数据。

Elasticsearch所涉及到的每一项技术都不是创新或者革命性的，全文搜索，分析系统以及分 布式数据库这些早就已经存在了。它的革命性在于将这些独立且有用的技术整合成一个一体 化的、实时的应用。它对新用户的门槛很低，它也会跟上你技能和需求增长的步伐。

如果你打算使用ES，说明你已经有数据了，但光有数据是不够的，除非你能对这些数据做些什么事情。

很不幸，现在大部分数据库在提取可用知识方面显得异常无能。的确，它们能够通过时间戳或者精确匹配做过滤，但是它们能够进行全文搜索，处理同义词和根据相关性给文档打分吗？它们能根据同一份数据生成分析和聚合的结果吗？最重要的是，它们在没有大量工作进程（线程）的情况下能做到对数据的实时处理吗？

这就是Elasticsearch存在的理由：Elasticsearch鼓励你浏览并利用你的数据，而不是让它烂在数据库里，因为在数据库里实在太难查询了。

# 基本概念

## Node（节点）与Cluster（集群）

集群是由一个或多个节点（ES实例）组成，这些节点共同保存整个数据。多节点集群是通过配置文件中的集群名称进行联结。在单节点的默认情况下也会形成一个集群（集群默认名称为elasticsearch），只是分片与副本都在同一台机器上，ES的高可用不能发挥优势

## Index（索引）

索引是具有相似特征的文档集合，索引名称必须全部为小写

## Type（类型）

在ES 6.0.0版本之后，一个索引只能定义一个类型，可以理解为对一类文档的一种分类。

ES官方计划在ES 7.0.0之后完全移除类型这一概念

## Doceument（文档）

文档是可被索引的基础信息单元

## Mapping（映射）

每个索引都有一种映射类型。使用映射来定义将那些字符串视为全文字段，哪些字段包含数字，日期或者地理位置，日期格式的定义等

## Primary Shards&Replicas Shards（主分片和副本分片）

### 主分片：

一个索引可以分为一个或者多个分片，存放在一台或者多台服务器上，每个分片都是独立的索引，提高了水平拓展能力，一旦创建好索引，分片数量便无法改变

### 副本分片：

每个主分片可以拥有多个副本分片，每个副本分片都是主分片的完整拷贝，副本提高了容错率和查询吞吐量，副本数量可以在创建好索引后更改

# 简单使用

格式：

|  |
| --- |
| Cur -H “Content-Type:application/json” -X<VERB> ‘<PROTOCOL>://<HOST>:<PORT>/<INDEX>/<TYPE>?<QUERY\_STRING>’ -d ‘<BODY>’ |

1. VERB HTTP方法：GET, POST, PUT, HEAD, DELETE
2. PROTOCOL http或者https协议（只有在Elasticsearch前面有https代理的时候可用）
3. HOST Elasticsearch集群中的任何一个节点的主机名，如果是在本地的节点，那么就叫localhost
4. PORT Elasticsearch HTTP服务所在的端口，默认为9200
5. PATH API路径（例如\_count将返回集群中文档的数量），PATH可以包含多个组件，例如\_cluster/stats或者\_nodes/stats/jvm
6. QUERY\_STRING 一些可选的查询请求参数，例如?pretty参数将使请求返回更加美观易读的JSON数据
7. BODY 一个JSON格式的请求主体（如果请求需要的话）
8. 其中的 -H "Content-Type:application/json" 指的是请求体为JSON格式

## 增加和删除索引

新建索引：curl -X PUT ‘localhost:9200/weather?pretty=true’。这样，就创建了一个叫weather的Index，服务器返回了一个JSON对象，里面的acknowledge字段表示操作成功。

|  |
| --- |
| curl -XPUT ‘localhost:9200/Index?pretty=true’ |



图 3-1 新增索引的请求体

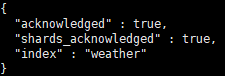


图 3-2 新增索引的返回体

删除索引：curl -X DELETE ‘localhost:9200/weather?pretty=true’。这样，就删除了这个weather索引

|  |
| --- |
| curl -XDELETE ‘localhost:9200/Index?pretty=true’ |



图 3-3 删除索引的请求体



图 3-4 删除索引的返回体

## 查看所有索引

查看集群中所有索引：

|  |
| --- |
| curl -XGET ‘Ip:Port/\_cat/indices?v’ |

最后的 ?v 是将各列所代表的属性展示出来

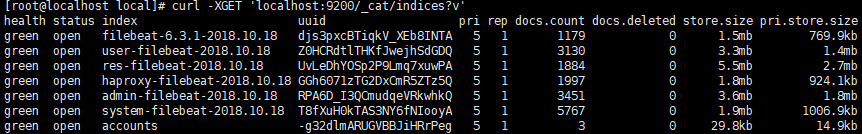


图 3-5 集群中所有节点

## 查看集群上的所有节点

查看集群中所有节点：

|  |
| --- |
| curl -XGET ‘Ip:Port/\_cat/nodes?v’ |

其中master 为 \* ，代表该节点为主节点

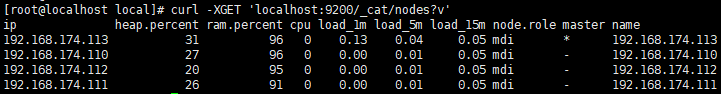


图 3-6 集群中所有节点

## 查看集群的健康状况

查看集群健康状况：

|  |
| --- |
| curl -XGET ‘Ip:Port/\_cluster/health?pretty=true’ |

Cluster\_name为集群的名称

Status为集群的健康状态（有红、黄、绿三种状态，红色代表至少一个主分片不可用，部分数据仍可查到。黄色代表集群至少丢失一个副本分片，搜索结果依旧是完整的，该检查是什么原因。绿色代表集群全部可用）

Time\_out为是否超时

Number\_of\_nodes为节点数

Number\_of\_data\_nodex为数据节点数

Active\_primary\_shards为活动着的主分片

Active\_shards为活动着的分片

Unassigned\_shards为未分配的分片

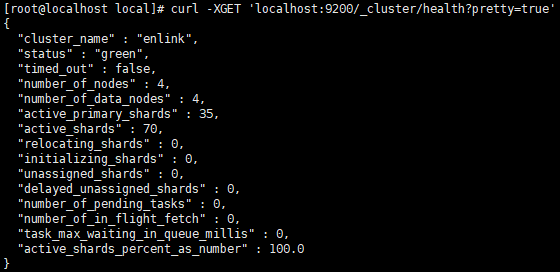


图 3-7查看集群健康状态

## 新增记录

向指定的 /Index/Type发送PUT请求，就可以在Index里面新增一条记录，比如，向

/accounts/person 发送请求，就可以新增一条记录。在没有这个Index的时候，ES会自动创建这个Index，所以，在插入数据时，不要写错Index

|  |
| --- |
| curl -H “Content-Type:application/json” -X PUT ‘IP:port/Index/Type/Id?pretty=true’ -d’  {  “Key”:”Value”,  “Key”:”Value”,  “Key”:”Value”  }’ |

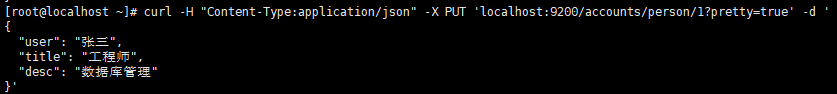


图 3-8 新增记录的PUT请求体

返回体：

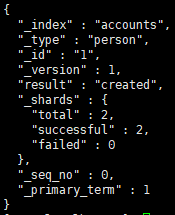


图 3-9 新增记录的返回体

如图 3-9 所示，是指定ID的插入数据，在不指定ID的时候，使用POST请求就可以添加一条不指定Id的记录：

|  |
| --- |
| curl -H “Content-Type:application/json” -X POST ‘IP:port/Index/Type?pretty=true’ -d’  {  “Key”:”Value”,  “Key”:”Value”,  “Key”:”Value”  }’ |

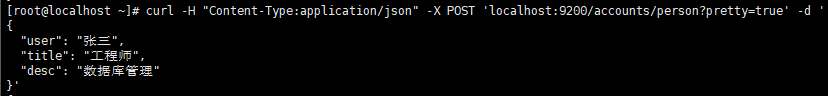


图 3-10 新增记录的POST请求体

返回体：

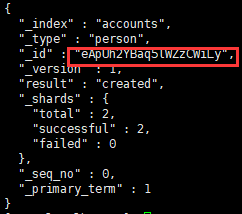


图 3-11 使用POST请求的返回体

如图3-11所示,其中的ID为自动生成

## 查看记录

向 /Index/Type/Id 发出GET请求，查看该Id的记录

|  |
| --- |
| curl -H “Content-Type:application/json” -XGET ‘IP:port/Index/Type/Id?pretty=true’ |

Image

图 3-12查看记录的请求体

返回体：

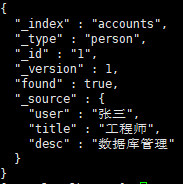


图 3-13 查询记录的返回体

如图3-13所示，“found”:”true”，表示查询成功，”\_source”:”...”字段返回的是原始字段

如果ID 不正确，found 字段就是false，如图3-14所示

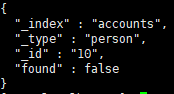


图 3-14 found字段为false示例

## 删除记录

向 /Index/Type/Id 发出DELETE请求，则可以删除该ID 记录

|  |
| --- |
| curl -H “Content-Type:application/json” -X DELETE ‘IP:port/Index/Type/Id?pretty=true’ |

Image

图 3-15 删除记录的请求体

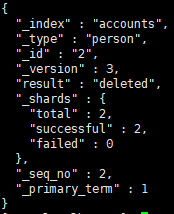


图 3-16删除记录的返回体

## 更新记录

向 /Index/Type/Id 发出PUT请求，更新记录

|  |
| --- |
| curl -H “Content-Type:application/json” -X PUT ‘IP:port/Index/Type/Id?pretty=true’ -d’  {  “Key”:”Value”,  “Key”:”Value”,  “Key”:”Value”  }’ |

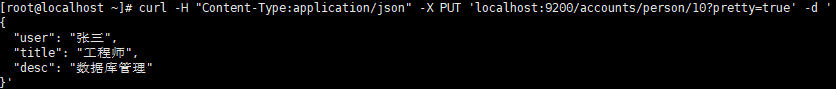


图 3-17更新纪录的请求体

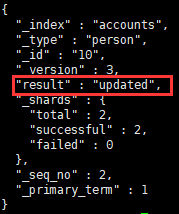


图 3-18 更新记录的返回体

如图 3-18所示。其中的“\_version” 变成3，result是updated

## 查询所有记录

使用GET方法，直接请求 /Index/Type/\_search,就会返回所有记录（ES默认返回10条记录，使用 size 可以改变返回条数）

|  |
| --- |
| curl -H “Content-Type:application/json” -X GET ‘IP:port/Index/Type/\_search?pretty=true’ |

Image

图 3-19查询所有的GET请求体

返回体（因为Index中只有3条记录，所有只返回了3条）：

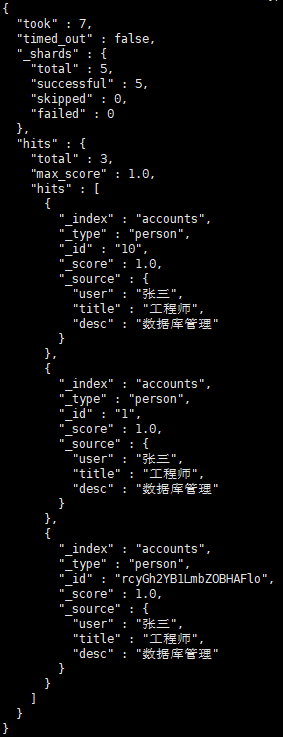


图 3-20查询所有的返回体

返回体中的took字段表示该操作的耗时（单位是毫秒），time\_out表示是否超时，hits表示命中的记录，hits字段中的含义

|  |  |
| --- | --- |
| total | 返回记录数，本例是3条 |
| max\_score | 最高的匹配程度，本例是1.0 |
| hits | 返回的记录组成的数组 |

## 全文搜索

ES使用的条件查询是使用GET 请求并带有请求体

|  |
| --- |
| curl -H “Content-Type:application/json” -X GET ‘IP:port/Index/Type/Id?pretty=true’ -d’  {  “query”:{“match”:{“field”:”字段中包含的词”}}  }’ |

Image

图 3-21 全文搜索的请求体

使用的是Match查询，指定的条件是 desc字段中包含 “软件” 这个词

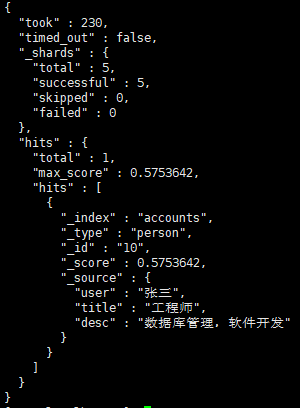


图 3-22包含“软件”查询的返回体

下面是使用 size 进行全文检索的示例，以管理为例：

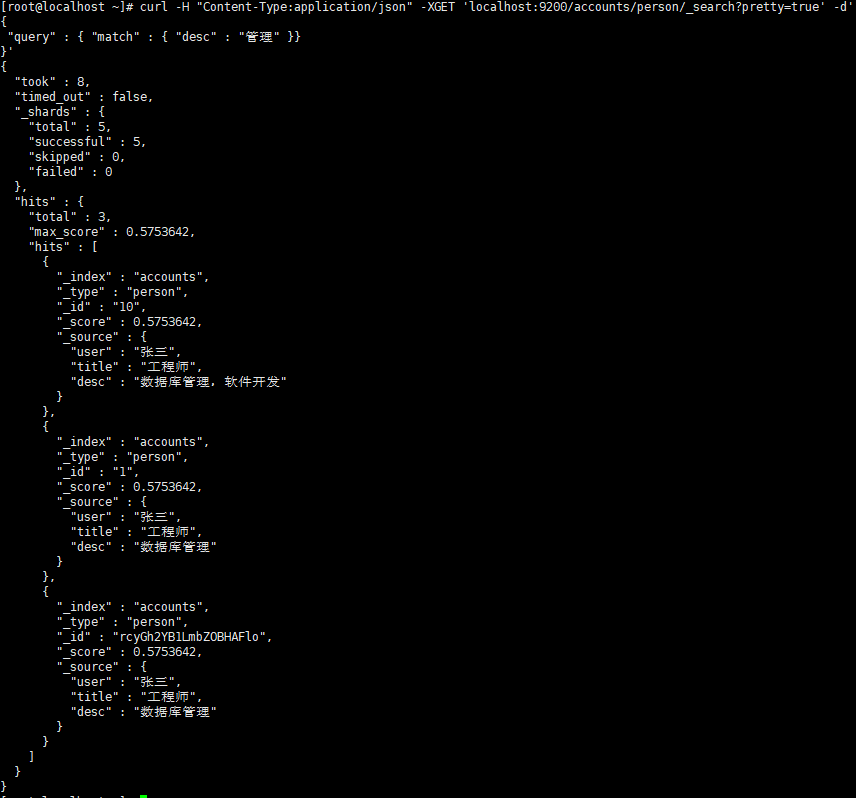


图 3-23查询出所有包含“管理”字段的数据

我们得到了3条数据，如图3-24所示，使用了size改变所展示的条数：

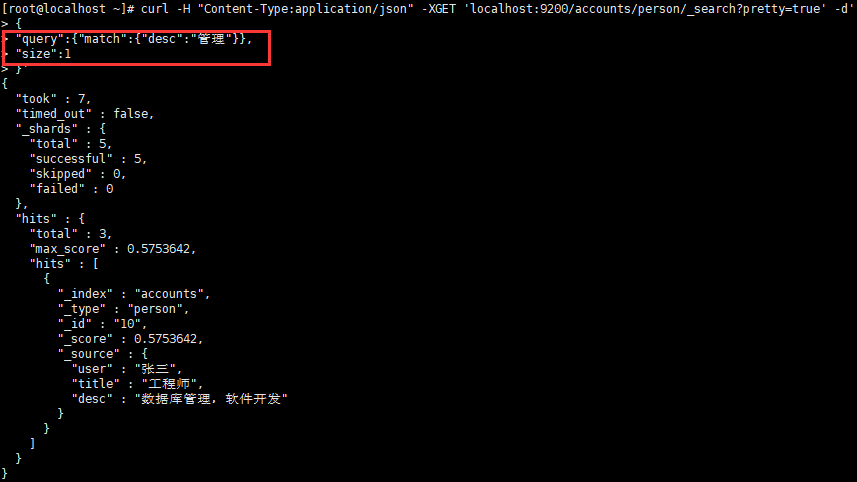


图 3-24使用了size改变所展示条数

也可以使用from与size配合使用，指定从第几条开始（ES默认从0开始）

|  |
| --- |
| curl -H “Content-Type:application/json” -X GET ‘IP:port/Index/Type/Id?pretty=true’ -d’  {  “query”:{“match”:{“field”:”字段中包含的词”}},  “from”:”0”,  “size”:”1”  }’ |

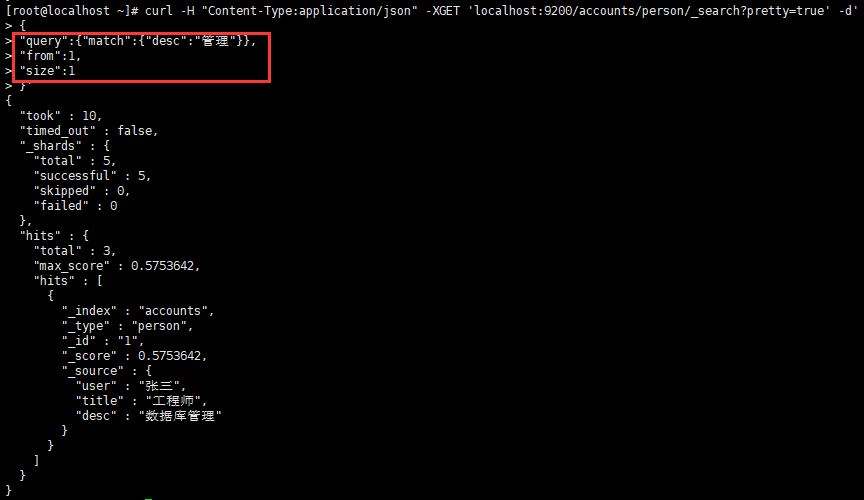


图 3-25使用from与size一起查询

# 映射（Mapping）

## 为什么需要映射：

1. 在不指定映射的时候，ES会动态的生成字段与类型的映射关系，每种核心数据类型（String ,number,boolean,date）也是以不同的方式进行索引
2. 当你的字符串以keyword 存储的时候，不会对字符串进行分词，会创建索引
3. 当你的字符串以text 存储的时候，会进行分词，也会创建索引
4. 当你的时间日期格式不符合ES自动匹配的时间日期格式的时候，就是以keyword 类型存储，这样，按照时间的操作就无法实现

## 映射类型

1. 元字段用于自定义文档的元数据关联的处理方式，元字段包括文档的 \_index,\_type,\_id,\_source 字段
2. 映射类型包含与文档相关的字段或属性列表

## 字段数据类型

1. 每个字段都有一个数据类型，简单类型有text,keyword,date,long,double.boolean,ip
2. 特殊类型：geo\_point,geo\_shape,completion字段自动检测

*表 4-1字段数据类型*

|  |  |
| --- | --- |
| JSON数据 | ES中数据类型 |
| Null | 不添加字段 |
| True or false | Boolean |
| 浮点数 | Double |
| 整数 | Long |
| Object | Object |
| Array | 依赖于第一个非null值 |
| String | 1、如果通过date检测，则为date类型。日期自动检测默认是开启的。  默认日期格式是”strict\_date\_optional\_time||epoch\_millis”，当有其他日期格式时则ES 不会认为是日期，所以我们在创建Index的时候指定日期格式  2、如果通过numeric检测，则为number类型。数字自动检测默认是关闭的 |

## 确切值（Exact values）与全文文本（Full text）举例

我们创建一个索引，指定其字段类型

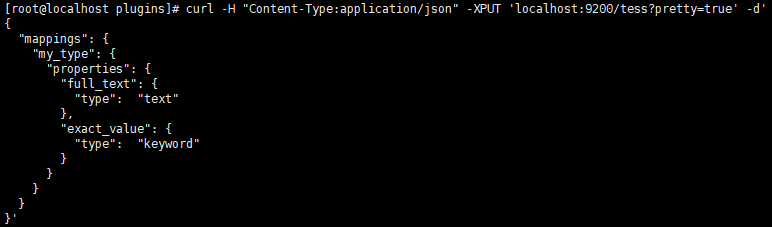


图 4-1创建一个索引，指定字段类型

再插入一条数据

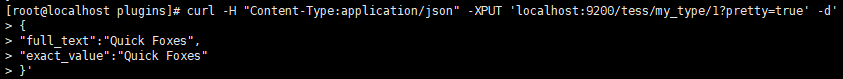


图 4-2 存储一条数据

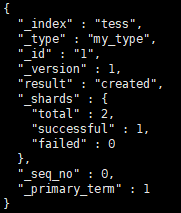


图 4-3存储成功

full\_text: 指定类型为text，是会被分词

exact\_value: 指定类型为keyword，不会被分词

full\_text： 会被standard analyzer分词为如下terms [quick,foxes],存入倒排索引

exact\_value：只有[Quick Foxes]这一个term会被存入倒排索引

接下来做查询，看结果

|  |
| --- |
| curl -H "Content-Type:application/json" -XGET 'localhost:9200/tess/my\_type/\_search?pretty=true' -d'  {  "query": {  "term": {  "exact\_value": "Quick Foxes"  }  }  }' |

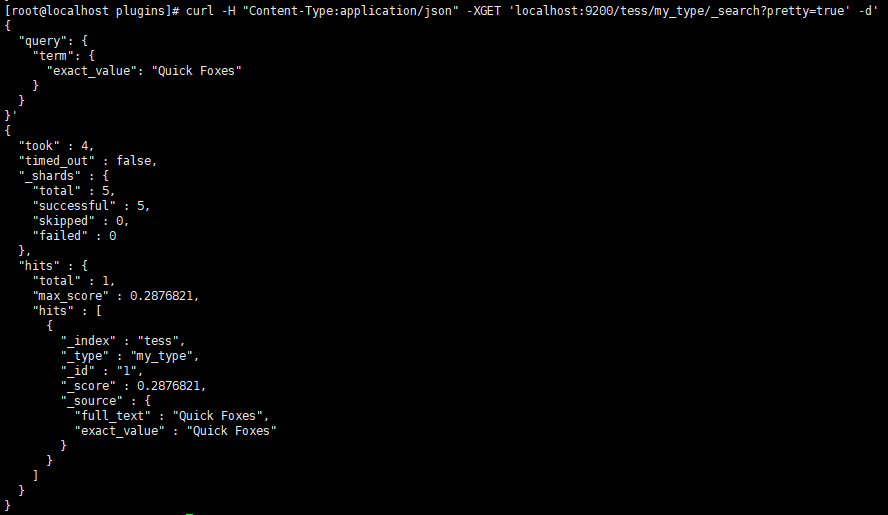


图 4-4 exact\_value 包含了确切的Quick Foxes ，能被查到

|  |
| --- |
| curl -H "Content-Type:application/json" -XGET 'localhost:9200/tess/my\_type/\_search?pretty=true' -d'  {  "query": {  "term": {  "full\_text": "Quick Foxes"  }  }  }' |



图 4-5 full\_text 被分词了，倒序排序中只有quick和foxes，而没有Quick Foxes

单独查询foxes

|  |
| --- |
| curl -H "Content-Type:application/json" -XGET 'localhost:9200/tess/my\_type/\_search?pretty=true' -d'  {  "query": {  "term": {  "full\_text": "foxes"  }  }  }' |

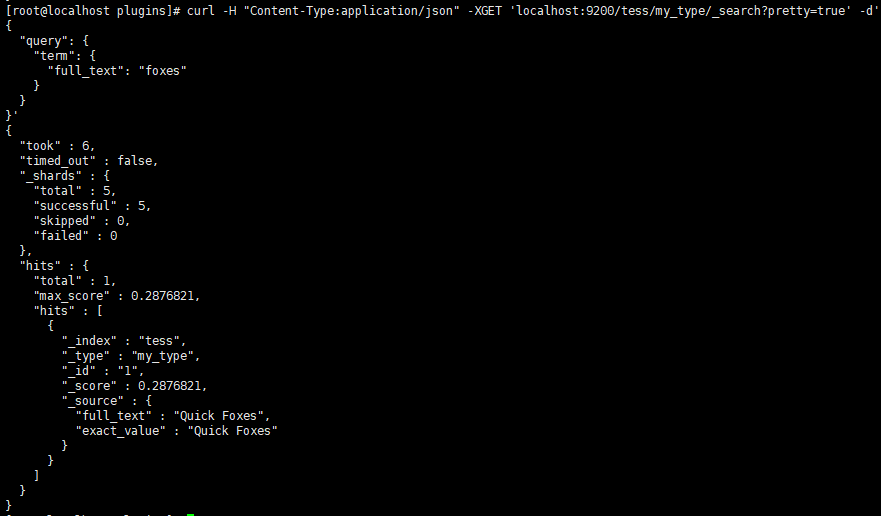


图 4-6索引中有quick和foxes，因此查询成功

## 时间格式映射举例

1. 显示地创建Index，指定字段格式为date 类型

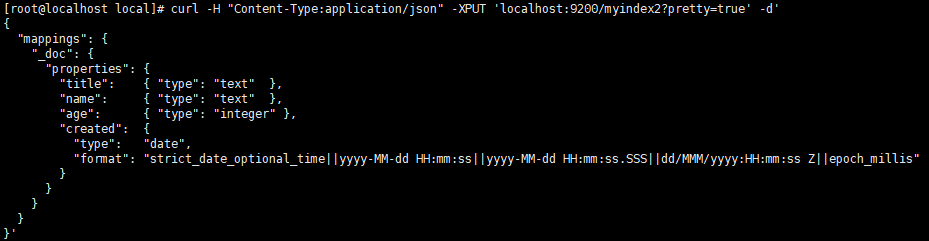


图 4-7 显示地创建索引

这样，指定的 /index/type 中”created” 字段就是date 类型

1. 创建一个索引，改变日期检测格式（非指定哪一个字段）

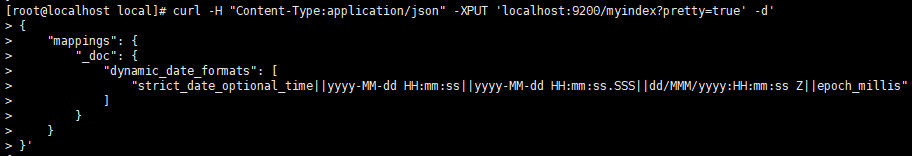


图 4-8 创建索引

现在我们插入数据进行测试

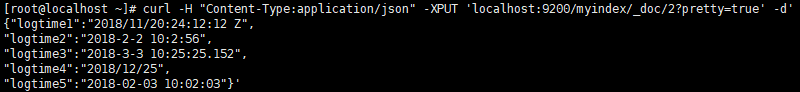


图 4-9 插入数据

我们再看这个索引的mapping ，如图4-10，符合日期时间检测的则为date类型，不符合的则为text类型

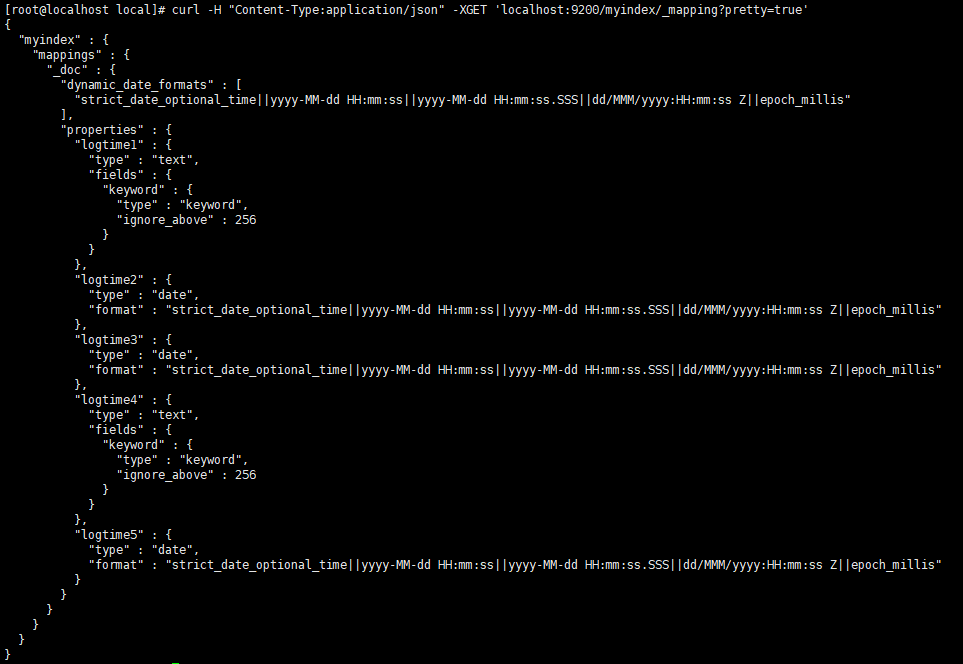


图 4-10 查看映射类型

# 动态模板（Dynamic templates）

在明确字段类型的时候使用Mapping是很好的选择，但是在大多数情况下，字段类型是无法确定的，新加入的字段会被ES自动进行类型推断，按照的是ES的标准，不一定符合我们的使用。所有，我们可以使用动态模板（Dynamic templates）来定义自己的标准。

## 模板简述

Template大致分为setting和mapping两部分，通过模式匹配的方式使得多个索引重用一个模板：

* Setting主要用于index的一些相关配置信息，如分片数、副本数、refresh等
* Mapping主要是一些说明信息，大致分为\_all、\_source、\_properties这三部分
* \_all:主要指的是AllFiled字段，我们可以将一个或多个都包含进来，在进行检索时无需指定字段的情况下检索多个字段。设置”\_all”:{“enabled”:”true”}
* \_source:主要指的是SourceField字段，Source可以理解为ES除了将数据保存在索引文件中，另外还有一份源数据。\_source字段在我们进行索引的时候非常重要，如果在{“enabled”:false}情况下，默认检索只会返回ID，你需要通过Field字段去到索引中取数据，效率不是很高。但是enabled设置为true时，索引会比较大，这时可以通过Compress进行压缩和includes、excludes来在字段级别上进行一些限制，自定义哪些字段允许存储
* Properties：这是最重要的步骤，主要针对索引结构和字段级别上的一些设置

## 动态模板基本规则

可以用于动态添加的字段：

* 使用match\_mapping\_type ，由ES检测到的数据类型
* 字段的名称，具有匹配和不匹配或者 match\_pattern
* 字段的完整路径，使用path\_match和path\_unmatch

动态模板被指定为一个数组对象

表 5-1 模板格式

|  |
| --- |
| "dynamic\_templates": [  {  "my\_template\_name": { 模板名称  ... match conditions ... 匹配条件  "mapping": { ... } 匹配字段用的格式  }  },  ...  ] |

* 匹配条件可以包括:match\_mapping\_type、match、match\_pattern、unmatch、path\_match、path\_unmatch

## 匹配条件

### Match\_mapping\_type

match\_mapping\_type是json解析器检测到的类型，由于json不予许integer转成long，double转成float，所以需要取一个大的范围

下面是可以被自动检测到的类型：

表 5-2自动类型检测

|  |  |
| --- | --- |
| Boolean | True或者false |
| Date | 当日期检测开启且格式匹配中的时候会是date类型 |
| Double | 有小数的时候 |
| Long | 没有小数的时候 |
| Object | 对象，也称作hashes |
| String | 字符串 |
| \* | 可以用作匹配任何数据类型 |

例子：如果我们想把所有整数字段映射成integer而不是long

表 5-3 映射模板

|  |
| --- |
| "dynamic\_templates": [  {  "integers": {  "match\_mapping\_type": "long",  "mapping": {  "type": "integer"  }  }  },  ...  ] |

插入数据：

表 5-4 存入数据

|  |
| --- |
| PUT my\_index/\_doc/1  {  "my\_integer": 5,  "my\_string": "Some string"  } |



图 5-1 match\_mapping\_type示例中索引的mapping

可以看到my\_integer 字段映射的是integer类型

### Match and unmatch

match参数使用的是匹配字段名，unmatch参数使用的是除了match的字段，即不匹配的字段

例子：匹配名称以long\_开头的所有字符串，排除以\_text结尾的字段，并映射为long类型

表 5-5 映射模板

|  |
| --- |
| "dynamic\_templates": [  {  "longs\_as\_strings": {  "match\_mapping\_type": "string",  "match": "long\_\*",  "unmatch": "\*\_text",  "mapping": {  "type": "long"  }  }  },  ...  ] |

插入数据测试

表 5-6 存入数据

|  |
| --- |
| PUT my\_index/\_doc/1  {  "long\_num": "5",  "long\_text": "foo"  } |

如图5-7所示，”long\_num”字段映射为long类型

“long\_text”字段使用的是默认的 string类型



图 5-2match和unmatch 示例中索引的mapping

### path\_match and path\_unmatch

Path\_match 和path\_unmatch用法与match、unmatch相似，只是匹配的不再是字段名称，而是路径名称

例子：示例将name对象中的任何字段的值复制到顶级full\_name字段，中间字段除外

表 5-8

|  |
| --- |
| "dynamic\_templates": [  {  "full\_name": {  "path\_match": "name.\*",  "path\_unmatch": "\*.middle",  "mapping": {  "type": "text",  "copy\_to": "full\_name"  }  }  },  ...  ] |

插入测试数据

表 5-9

|  |
| --- |
| PUT my\_index/\_doc/1  {  "name": {  "first": "Alice",  "middle": "Mary",  "last": "White"  }  } |

查看到的mapping格式



图 5-3 path\_match与path\_unmatch案例中索引mapping

## 模板示例

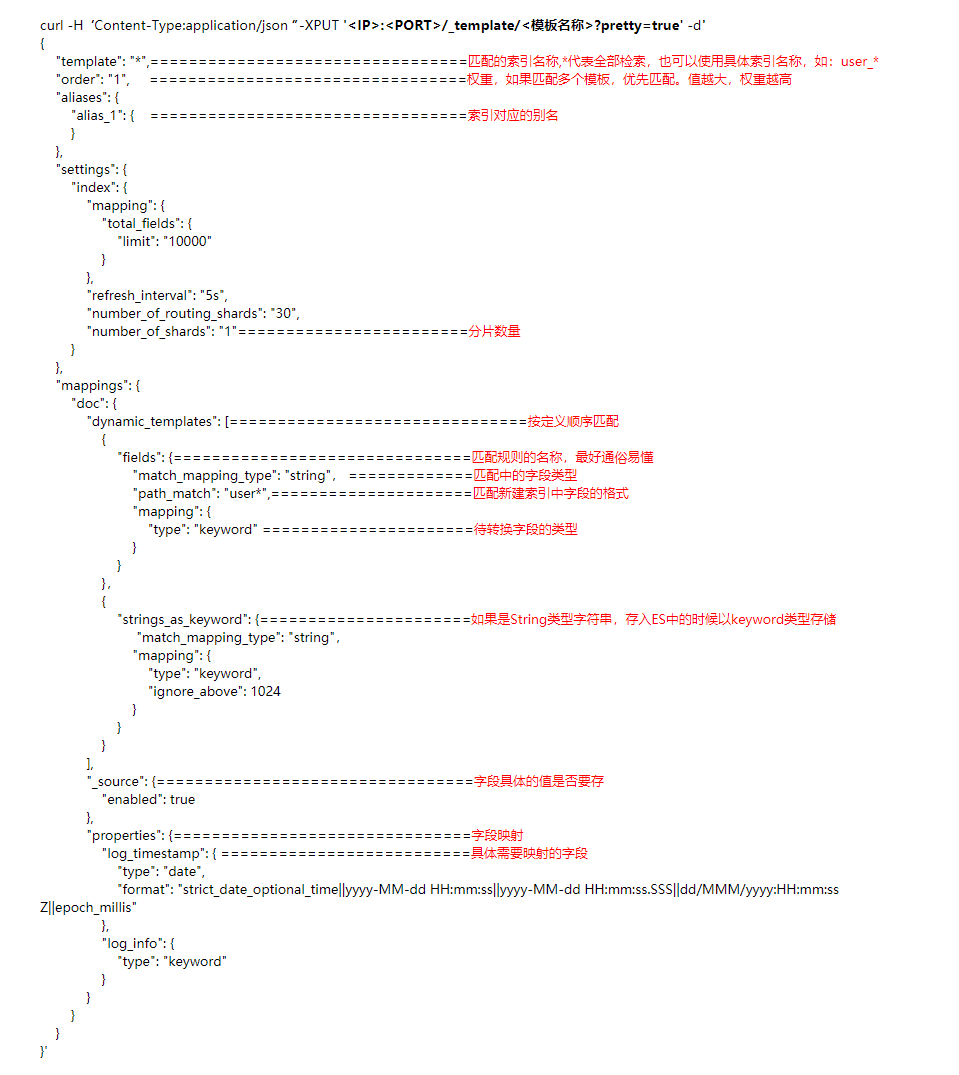


图 5-4模板示例