# 第一部分 E

## 1.1 索引创建（数据库）

创建索引类似创建数据库。

### **1.1.1 简单创建**

|  |  |
| --- | --- |
| PUT /my\_index?pretty | {  "acknowledged": true,  "shards\_acknowledged": true  } |

### **1.1.2 带参数创建**

|  |  |
| --- | --- |
| PUT /our\_index?pretty  {  "settings":{  "number\_of\_shards":1,  "number\_of\_replicas":2,  "index":{  "analysis":{  "analyzer":{  "default":{  "tokenizer":"standard",  "filter":[  "asciifolding",  "lowercase",  "ourEnglishFilter"  ]  }  },  "filter":{  "ourEnglishFilter":{  "type":"kstem"  }  }  }  }  }  } | {  "acknowledged": true,  "shards\_acknowledged": true  } |

### **1.1.3 settings**

settings修改索引库默认配置

例如：分片数量，副本数量

　　查看 ：curl -XGET http://192.168.80.10:9200/zhouls/\_settings?pretty

　　操作不存在索引：curl -XPUT '192.168.80.10:9200/liuch/' -d'{"settings":{"number\_of\_shards":3,"number\_of\_replicas":0}}'

　　操作已存在索引：curl -XPUT '192.168.80.10:9200/zhouls/\_settings' -d'{"index":{"number\_of\_replicas":1}}'

总结：就是，不存在索引时,可以指定副本和分片，如果已经存在,则只能修改副本。

在创建新的索引库时，可以指定索引分片的副本数。默认是1，这个很简单

## 1.2 基础

ES的mapping如何用？什么时候需要手动，什么时候需要自动？

Mapping,就是对索引库中索引的字段名称及其数据类型进行定义，类似于mysql中的表结构信息。不过es的mapping比数据库灵活很多，它可以动态识别字段。一般不需要指定mapping都可以，因为es会自动根据数据格式识别它的类型，如果你需要对某些字段添加特殊属性（如：定义使用其它分词器、是否分词、是否存储等），就必须手动添加mapping。

　　我们在es中添加索引数据时不需要指定数据类型，es中有自动影射机制，字符串映射为string，数字映射为long。通过mappings可以指定数据类型是否存储等属性。

### **1.2.1 内置字段**

#### 1.2.1.1 \_uid

每个索引的document会关联一个id和一个type, 内部的\_uid字段将type和id组合起来作为document的唯一标示(这意味着不同的type可以有相同的id， 组合起来仍然是唯一的)。

#### **1.2.1.2 \_id**

每个索引的document会关联一个id和一个type, \_id字段就是用来索引并且存储(可能)id的，默认是不索引(not indexed)并且不存储的(not stored)。

可以设置\_id的path属性来从源文档中提取id， 例如下面的mapping:

注意， 即使\_id是不索引的, 相关的接口仍然起作用(他们会用\_uid字段)， 比如用term, terms或者prefix来根据ids过滤(包括用ids来查询/过滤)。

#### **1.2.1.3 \_type**

每个索引的document会关联一个id和一个type, type在索引时会自动赋给\_type字段, 默认\_type字段是需要索引的(但不analyzed)并且不存储的， 这就意味着\_type字段是可查询的。

#### **1.2.1.4 \_source**

\_source是一个自动生成的字段， 用来存储实际提交的JSON数据， 他是不索引的(不可搜索), 只是用来存储。 在执行"fetch"类的请求时, 比如get或者search, \_source字段默认也会返回。

#### **1.2.1.5 \_all**

\_all字段的设计目的是用来包罗文档的一个或多个字段， 这对一些特定的查询非常有用， 比如我们要查询文档的内容， 但是不确定要具体查询哪一个字段， 这会占用额外的cpu和索引容量。

#### **1.2.1.6 \_analyzer**

#### **1.2.1.7 \_boost**

#### **1.2.1.8 \_parent**

#### **1.2.1.9 \_routing**

路由参数，ELasticsearch通过以下公式计算文档应该分到哪个分片上：

shard\_num = hash(\_routing) % num\_primary\_shards

#### **1.2.1.10 \_index**

#### **1.2.1.11 \_size**

#### **1.2.1.12 \_timestamp**

#### **1.2.1.13 \_ttl**

### **1.2.2 Mappings**

#### **1.2.2.0 type**

##### string类型

ELasticsearch 5.X之后的字段类型不再支持string，由text或keyword取代。 如果仍使用string，会给出警告。

##### text类型

text取代了string，当一个字段是要被全文搜索的，比如Email内容、产品描述，应该使用text类型。设置text类型以后，字段内容会被分析，在生成倒排索引以前，字符串会被分析器分成一个一个词项。text类型的字段不用于排序，很少用于聚合（termsAggregation除外）。

##### keyword类型

keyword类型适用于索引结构化的字段，比如email地址、主机名、状态码和标签。如果字段需要进行过滤(比如查找已发布博客中status属性为published的文章)、排序、聚合。keyword类型的字段只能通过精确值搜索到。

##### 数字类型

对于数字类型，ELasticsearch支持以下几种：

| **类型** | **取值范围** |
| --- | --- |
| long | -2^63至2^63-1 |
| integer | -2^31至2^31-1 |
| short | -32,768至32768 |
| byte | -128至127 |
| double | 64位双精度IEEE 754浮点类型 |
| float | 32位单精度IEEE 754浮点类型 |
| half\_float | 16位半精度IEEE 754浮点类型 |
| scaled\_float | 缩放类型的的浮点数（比如价格只需要精确到分，price为57.34的字段缩放因子为100，存起来就是5734） |

##### Object类型

##### date类型

##### Array类型

##### binary类型

binary类型接受base64编码的字符串，默认不存储也不可搜索。

##### ip类型

##### range类型

##### nested类型

nested嵌套类型是object中的一个特例，可以让array类型的Object独立索引和查询。

不加nested可能ES会将字段平铺成父节点的数组类型属性。

##### token\_count类型

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "name": {  "type": "text",  "fields": {  "length": {  "type": "token\_count",  "analyzer": "standard"  }  }  }  }  }  } | //  //  //  //  //  //  //  //统计出name的单词个数  //  //  //  // |

##### geo\_point 类型

地理位置信息类型用于存储地理位置信息的经纬度.

#### **1.2.2.[1 analyzer](http://csdnimg.cn/release/phoenix/" \l "31-analyzer" \t "http://blog.csdn.net/napoay/article/details/_blank)**

|  |  |
| --- | --- |
| "mappings":  {  "my\_type": {  "properties": {  "content": {  "type": "text",  "analyzer": "ik\_max\_word",  "search\_analyzer": "ik\_max\_word"  }  }  }  } | //  //  //表名字  //表结构定义开始  //字段名称：context  //字段类型：text  //分词器：创建索引时使用  //分词器：搜索词的分词器  //  //  //  // |

#### **1.2.2.[2 normalizer](http://csdnimg.cn/release/phoenix/" \l "32-normalizer" \t "http://blog.csdn.net/napoay/article/details/_blank)**

normalizer用于解析前的标准化配置，比如把所有的字符转化为小写等。例子：

|  |  |
| --- | --- |
| "mappings":{  "my\_type": {  "properties": {  "content": {  "type": "text",  "normalizer": "my\_normalizer"  }  }  }  } | //  //表名字  //表结构定义开始  //字段名称：context  //字段类型：text  //分词器：创建索引时使用  //标准化器：原始输入的解析前的规格化  //  //  // |

#### **1.2.2.[3 boost](http://csdnimg.cn/release/phoenix/" \l "33-boost" \t "http://blog.csdn.net/napoay/article/details/_blank)**

boost字段用于设置字段的权重，比如，关键字出现在title字段的权重是出现在content字段中权重的2倍，设置mapping如下，其中content字段的默认权重是1。ES依据权重来计算匹配分数。

|  |  |
| --- | --- |
| "mappings":{  "my\_type": {  "properties": {  "content": {  "type": "text",  "boost": 2  },  "title": {  "type": "text",  "boost": 1  }  }  }  } | //  //表名字  //表结构定义开始  //字段名称：context  //字段类型：text  //权重：搜索词在标题的权重为1.  //  //字段名称：title  //字段类型：text  //权重：搜索词在标题的权重为2.  //  //  //  // |

#### **1.2.2.[4 coerce](http://csdnimg.cn/release/phoenix/" \l "34-coerce" \t "http://blog.csdn.net/napoay/article/details/_blank)**

coerce属性用于清除脏数据，coerce的默认值是true。整型数字5有可能会被写成字符串“5”或者浮点数5.0.coerce属性可以用来清除脏数据：

* 字符串会被强制转换为整数
* 浮点数被强制转换为整数

|  |  |
| --- | --- |
| "mappings":{  "my\_type": {  "properties": {  "number\_one": {  "type": "integer"  },  "number\_two": {  "type": "integer",  "coerce": false  }  }  }  } | //  //表名字  //表结构定义开始  //字段名称：number\_one  //字段类型：整型数字  //  //字段名称：number\_two  //字段类型：整型数字  //字段类型：清理脏数据被禁用  //输入“5”时将会返回错误  //  //  // |

#### **1.2.2.[5 copy\_to](http://csdnimg.cn/release/phoenix/" \l "35-copyto" \t "http://blog.csdn.net/napoay/article/details/_blank)**

copy\_to属性用于配置自定义的\_all字段。换言之，就是多个字段可以合并成一个超级字段。比如，first\_name和last\_name可以合并为full\_name字段。

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "first\_name": {  "type": "text",  "copy\_to": "full\_name"  },  "last\_name": {  "type": "text",  "copy\_to": "full\_name"  },  "full\_name": {  "type": "text"  }  }  }  } | //  //表名字  //表结构定义开始  //字段名称：  //字段类型：整型数字  //合并到full\_name  //  //字段名称：  //字段类型：整型数字  //合并到full\_name  //  //接受合并的字段  //  //  //  //  // |

#### **1.2.2.[6 doc\_values](http://csdnimg.cn/release/phoenix/" \l "36-docvalues" \t "http://blog.csdn.net/napoay/article/details/_blank)**

doc\_values是为了加快排序、聚合操作，在建立倒排索引的时候，额外增加一个列式存储映射，是一个空间换时间的做法。默认是开启的，对于确定不需要聚合或者排序的字段可以关闭。

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "status\_code": {  "type": "keyword"  },  "session\_id": {  "type": "keyword",  "doc\_values": false  }  }  }  } |  |

注:text类型不支持doc\_values。

#### **1.2.2.[7 dynamic](http://csdnimg.cn/release/phoenix/" \l "37-dynamic" \t "http://blog.csdn.net/napoay/article/details/_blank)**

dynamic属性用于检测新发现的字段，有三个取值：

* true:新发现的字段添加到映射中。（默认）
* flase:新检测的字段被忽略。必须显式添加新字段。
* strict:如果检测到新字段，就会引发异常并拒绝文档。

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "dynamic": false,  "properties": {  "user": {  "properties": {  "name": {  "type": "text"  },  "social\_networks": {  "dynamic": true,  "properties": {}  }  }  }  }  }  } |  |

#### **1.2.2.[8 enabled](http://csdnimg.cn/release/phoenix/" \l "38-enabled" \t "http://blog.csdn.net/napoay/article/details/_blank)**

ELasticseaech默认会索引所有的字段，enabled设为false的字段，es会跳过字段内容，该字段只能从\_source中获取，但是不可搜。而且字段可以是任意类型。

|  |  |
| --- | --- |
| "mappings": {  "session": {  "properties": {  "user\_id": {  "type": "keyword"  },  "last\_updated": {  "type": "date"  },  "session\_data": {  "enabled": false  }  }  }  } |  |

#### **1.2.2.[9 fielddata](http://csdnimg.cn/release/phoenix/" \l "39-fielddata" \t "http://blog.csdn.net/napoay/article/details/_blank)**

搜索要解决的问题是“包含查询关键词的文档有哪些？”，聚合恰恰相反，聚合要解决的问题是“文档包含哪些词项”，大多数字段再索引时生成doc\_values，但是text字段不支持doc\_values。

取而代之，text字段在查询时会生成一个fielddata的数据结构，fielddata在字段首次被聚合、排序、或者使用脚本的时候生成。ELasticsearch通过读取磁盘上的倒排记录表重新生成文档词项关系，最后在Java堆内存中排序。

text字段的fielddata属性默认是关闭的，开启fielddata非常消耗内存。在你开启text字段以前，想清楚为什么要在text类型的字段上做聚合、排序操作。大多数情况下这么做是没有意义的。

“New York”会被分析成“new”和“york”，在text类型上聚合会分成“new”和“york”2个桶，也许你需要的是一个“New York”。这是可以加一个不分析的keyword字段：

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "my\_field": {  "type": "text",  "fields": {  "keyword": {  "type": "keyword"  }  }  }  }  }  } |  |

上面的mapping中实现了通过my\_field字段做全文搜索，my\_field.keyword做聚合、排序和使用脚本。

#### **1.2.2.[10 format](http://csdnimg.cn/release/phoenix/" \l "310-format" \t "http://blog.csdn.net/napoay/article/details/_blank)**

format属性主要用于格式化日期：

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "date": {  "type": "date",  "format": "yyyy-MM-dd"  }  }  }  } |  |

#### **1.2.2.[11 ignore\_above](http://csdnimg.cn/release/phoenix/" \l "311-ignoreabove" \t "http://blog.csdn.net/napoay/article/details/_blank)**

ignore\_above用于指定字段索引和存储的长度最大值，超过最大值的会被忽略：

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "message": {  "type": "keyword",  "ignore\_above": 15  }  }  }  } |  |

#### **1.2.2.[12 ignore\_malformed](http://csdnimg.cn/release/phoenix/" \l "312-ignoremalformed" \t "http://blog.csdn.net/napoay/article/details/_blank)**

ignore\_malformed可以忽略不规则数据，对于login字段，有人可能填写的是date类型，也有人填写的是邮件格式。给一个字段索引不合适的数据类型发生异常，导致整个文档索引失败。如果ignore\_malformed参数设为true，异常会被忽略，出异常的字段不会被索引，其它字段正常索引。

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "number\_one": {  "type": "integer",  "ignore\_malformed": true  },  "number\_two": {  "type": "integer"  }  }  }  } |  |

#### **1.2.2.[13 include\_in\_all](http://csdnimg.cn/release/phoenix/" \l "313-includeinall" \t "http://blog.csdn.net/napoay/article/details/_blank)**

include\_in\_all属性用于指定字段是否包含在\_all字段里面，默认开启，除索引时index属性为no。   
例子如下，title和content字段包含在\_all字段里，date不包含。

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "title": {  "type": "text"  },  "content": {  "type": "text"  },  "date": {  "type": "date",  "include\_in\_all": false  }  }  }  } |  |

include\_in\_all也可用于字段级别，如下my\_type下的所有字段都排除在\_all字段之外，author.first\_name 和author.last\_name 包含在in \_all中：

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "include\_in\_all": false,  "properties": {  "title": { "type": "text" },  "author": {  "include\_in\_all": true,  "properties": {  "first\_name": { "type": "text" },  "last\_name": { "type": "text" }  }  },  "editor": {  "properties": {  "first\_name": { "type": "text" },  "last\_name": { "type": "text", "include\_in\_all": true }  }  }  }  }  } |  |

#### **1.2.2.[14 index](http://csdnimg.cn/release/phoenix/" \l "314-index" \t "http://blog.csdn.net/napoay/article/details/_blank)**

index属性指定字段是否索引，不索引也就不可搜索，取值可以为true或者false。

#### **1.2.2.[15 index\_options](http://csdnimg.cn/release/phoenix/" \l "315-indexoptions" \t "http://blog.csdn.net/napoay/article/details/_blank)**

index\_options控制索引时存储哪些信息到倒排索引中，接受以下配置：

| **参数** | **作用** |
| --- | --- |
| docs | 只存储文档编号 |
| freqs | 存储文档编号和词项频率 |
| positions | 文档编号、词项频率、词项的位置被存储，偏移位置可用于临近搜索和短语查询 |
| offsets | 文档编号、词项频率、词项的位置、词项开始和结束的字符位置都被存储，offsets设为true会使用Postings highlighter |

#### **1.2.2.[16 fields](http://csdnimg.cn/release/phoenix/" \l "316-fields" \t "http://blog.csdn.net/napoay/article/details/_blank)**

fields可以让同一文本有多种不同的索引方式，比如一个String类型的字段，可以使用text类型做全文检索，使用keyword类型做聚合和排序。

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "city": {  "type": "text",  "fields": {  "raw": {  "type": "keyword"  }  }  }  }  }  } |  |

#### **1.2.2.[17 norms](http://csdnimg.cn/release/phoenix/" \l "317-norms" \t "http://blog.csdn.net/napoay/article/details/_blank)**

norms参数用于标准化文档，以便查询时计算文档的相关性。norms虽然对评分有用，但是会消耗较多的磁盘空间，如果不需要对某个字段进行评分，最好不要开启norms。

#### **1.2.2.[18 null\_value](http://csdnimg.cn/release/phoenix/" \l "318-nullvalue" \t "http://blog.csdn.net/napoay/article/details/_blank)**

值为null的字段不索引也不可以搜索，null\_value参数可以让值为null的字段显式的可索引、可搜索。例子：

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "status\_code": {  "type": "keyword",  "null\_value": "NULL"  }  }  }  } |  |

#### **1.2.2.[19 position\_increment\_gap](http://csdnimg.cn/release/phoenix/" \l "319-positionincrementgap" \t "http://blog.csdn.net/napoay/article/details/_blank)**

为了支持近似或者短语查询，text字段被解析的时候会考虑此项的位置信息。举例，一个字段的值为数组类型：

"names": [ "John Abraham", "Lincoln Smith"]

为了区别第一个字段和第二个字段，Abraham和Lincoln在索引中有一个间距，默认是100。

|  |  |
| --- | --- |
| "mappings": {  "groups": {  "properties": {  "names": {  "type": "text",  "position\_increment\_gap": 0  }  }  }  } |  |

如果搜索时设定了搜索词的间距，小于mappings中设定的间距，那么"Abraham Lincoln"搜索词会导致命中

#### **1.2.2.[20 properties](http://csdnimg.cn/release/phoenix/" \l "320-properties" \t "http://blog.csdn.net/napoay/article/details/_blank)**

Object或者nested类型，下面还有嵌套类型，可以通过properties参数指定。

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "manager": {  "properties": {  "age": { "type": "integer" },  "name": { "type": "text" }  }  },  "employees": {  "type": "nested",  "properties": {  "age": { "type": "integer" },  "name": { "type": "text" }  }  }  }  }  } |  |

#### **1.2.2.[21 search\_analyzer](http://csdnimg.cn/release/phoenix/" \l "321-searchanalyzer" \t "http://blog.csdn.net/napoay/article/details/_blank)**

大多数情况下索引和搜索的时候应该指定相同的分析器，确保query解析以后和索引中的词项一致。但是有时候也需要指定不同的分析器，例如使用edge\_ngram过滤器实现自动补全。

默认情况下查询会使用analyzer属性指定的分析器，但也可以被search\_analyzer覆盖。例子：

|  |  |
| --- | --- |
| "settings": {  "analysis": {  "filter": {  "autocomplete\_filter": {  "type": "edge\_ngram",  "min\_gram": 1,  "max\_gram": 20  }  },  "analyzer": {  "autocomplete": {  "type": "custom",  "tokenizer": "standard",  "filter": [  "lowercase",  "autocomplete\_filter"  ]  }  }  }  },  "mappings": {  "my\_type": {  "properties": {  "text": {  "type": "text",  "analyzer": "autocomplete",  "search\_analyzer": "standard"  }  }  }  } |  |

#### **1.2.2.[22 similarity](http://csdnimg.cn/release/phoenix/" \l "322-similarity" \t "http://blog.csdn.net/napoay/article/details/_blank)**

similarity参数用于指定文档评分模型，参数有三个：

* BM25 ：ES和Lucene默认的评分模型
* classic ：TF/IDF评分
* boolean：布尔模型评分

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "default\_field": {  "type": "text"  },  "classic\_field": {  "type": "text",  "similarity": "classic"  },  "boolean\_sim\_field": {  "type": "text",  "similarity": "boolean"  }  }  }  } |  |

#### **1.2.2.[23 store](http://csdnimg.cn/release/phoenix/" \l "323-store" \t "http://blog.csdn.net/napoay/article/details/_blank)**

默认情况下，自动是被索引的也可以搜索，但是不存储，这也没关系，因为\_source字段里面保存了一份原始文档。在某些情况下，store参数有意义，比如一个文档里面有title、date和超大的content字段，如果只想获取title和date，可以这样：

|  |  |
| --- | --- |
| "mappings": {  "my\_type": {  "properties": {  "title": {  "type": "text",  "store": true  },  "date": {  "type": "date",  "store": true  },  "content": {  "type": "text"  }  }  }  } | //  //  //  //  //  //  //  //  //  //会被存储于stored\_fields字段  //  //  //  //  //  //  // |

#### **1.2.2.[24 term\_vector](http://csdnimg.cn/release/phoenix/" \l "324-termvector" \t "http://blog.csdn.net/napoay/article/details/_blank)**

词向量包含了文本被解析以后的以下信息：

* 词项集合
* 词项位置
* 词项的起始字符映射到原始文档中的位置。

term\_vector参数有以下取值：

| **参数取值** | **含义** |
| --- | --- |
| no | 默认值，不存储词向量 |
| yes | 只存储词项集合 |
| with\_positions | 存储词项和词项位置 |
| with\_offsets | 词项和字符偏移位置 |
| with\_positions\_offsets | 存储词项、词项位置、字符偏移位置 |

### **1.2.3 动态Mapping**

#### **1.2.3.1 default mapping**

在mapping中使用default字段，那么其它字段会自动继承default中的设置。

#### **1.2.3.2 Dynamic field mapping**

#### **1.2.3.4 Dynamic templates**

#### **1.2.3.5 Override default template**

## 1.3 创建扁平结构[映射](表)

### **1.3.1 简单创建语句**

### **1.3.2 带参数创建语句**

## 1.4 创建非扁平结构[映射](表)