索引管理

我 已 看到 Elasticsearch 一个新的 用 得 ，不需要任何 先 或 置。 不 ，要不了多久 就会 始想要 化索引和搜索 程，以便更好地 合 的特定用例。 些定制几乎 着索引和 型的方方面面，在本章，我 将介 管理索引和 型映射的 API 以及一些最重要的 置。

建一个索引

到目前 止， 我 已 通 索引一篇文 建了一个新的索引 。 个索引采用的是 的配置，新的字段通 映射的方式被添加到 型映射。 在我 需要 个建立索引的 程做更多的控制：我 想要 保 个索引有数量 中的主分片，并且在我 索引任何数据 之前 ，分析器和映射已 被建立好。

了 到 个目的，我 需要手 建索引，在 求体里面 入 置或 型映射，如下所示：

```
PUT /my_index
{
    "settings": { ... any settings ... },
    "mappings": {
        "type_one": { ... any mappings ... },
        "type_two": { ... any mappings ... },
        ...
    }
}
```

如果 想禁止自 建索引， 可以通 在 `config/elasticsearch.yml` 的 个 点下添加下面的配置：

```
action.auto_create_index: false
```

NOTE 我 会在之后 用 [index-templates] 来 配置 自 建索引。 在索引日志数据的 候尤其有用： 将日志数据索引在一个以日期 尾命名的索引上，子夜 分，一个 配置的新索引将会自 行 建。

除一个索引

用以下的 求来 除索引:

```
DELETE /my_index
```

也可以 除多个索引：

```
DELETE /index_one,index_two
DELETE /index_*
```

甚至可以　　除 全部 索引：

```
DELETE /_all
DELETE /*
```

NOTE

一些人来　，能　用　个命令来　除所有数据可能会　致可怕的后果。如果　想要避免意外的大量　除，　可以在　的 `elasticsearch.yml` 做如下配置：

```
action.destructive_requires_name: true
```

个　置使　除只限于特定名称指向的数据，　而不允　通　指定　`_all`　或通配符来除指定索引　。　同　可以通　Cluster State API　的更新　个　置。

索引　置

可以通　修改配置来自定　索引行　，　配置参照 {ref}/index-modules.html[索引模　]

TIP

Elasticsearch　提供了　化好的　配置。　除非　理解　些配置的作用并且知道　什　要去修改，否　不要随意修改。

下面是　个 最重要的　置：

`number_of_shards`
　个索引的主分片数，　是 `5` 。　个配置在索引　建后不能修改。

`number_of_replicas`
　个主分片的副本数，　是 `1` 。　于活　的索引　，　个配置可以随　修改。

例如，我　可以　建只有 一个主分片，没有副本的小索引：

```
PUT /my_temp_index
{
    "settings": {
        "number_of_shards" :   1,
        "number_of_replicas" : 0
    }
}
```

然后，我　可以用 `update-index-settings` API　修改副本数：

```
PUT /my_temp_index/_settings
{
    "number_of_replicas": 1
}
```

配置分析器

第三个重要的索引 置是 analysis 部分，用来配置已存在的分析器或 的索引 建新的自定 分析器。

在 [analysis-intro]，我 介 了一些内置的分析器，用于将全文字符串 合搜索的倒排索引。

standard 分析器是用于全文字段的 分析器， 于大部分西方 系来 是一个不 的 。它包括了以下几点：

- standard 分 器，通 界分割 入的文本。
- standard 元 器，目的是整理分 器触 的 元（但是目前什 都没做）。
- lowercase 元 器， 所有的 元 小写。
- stop 元 器， 除停用 — 搜索相 性影 不大的常用 ，如 a， the， and， is。

情况下，停用 器是被禁用的。如需 用它， 可以通 建一个基于 standard 分析器的自定分析器并 置 stopwords 参数。 可以 分析器提供一个停用 列表，或者告知使用一个基于特定 言的定 停用 列表。

在下面的例子中，我 建了一个新的分析器，叫做 es_std， 并使用 定 的西班牙 停用 列表：

```
PUT /spanish_docs
{
    "settings": {
        "analysis": {
            "analyzer": {
                "es_std": {
                    "type":      "standard",
                    "stopwords": "_spanish_"
                }
            }
        }
    }
}
```

es_std 分析器不是全局的—它 存在于我 定 的 spanish_docs 索引中。 了使用 analyze API来它 行 ，我 必 使用特定的索引名：

```
GET /spanish_docs/_analyze?analyzer=es_std
El veloz zorro marrón
```

化的 果 示西班牙 停用 El 已被正 的移除：

```
{
  "tokens" : [
    { "token" :    "veloz",   "position" : 2 },
    { "token" :    "zorro",   "position" : 3 },
    { "token" :    "marrón",  "position" : 4 }
  ]
}
```

自定　分析器

然Elasticsearch　有一些　成的分析器，然而在分析器上Elasticsearch真正的　大之　在于，　可以通　在一个　合　的特定数据的　置之中　合字符　　器、分　器、　　元　　器来　建自定　的分析器。

在　[analysis-intro]　我　　，一个　分析器　就是在一个包里面　合了三　函数的一个包装器，　三　函数按照　序被　行:

字符　　器

字符　　器　　用来　　<code>整理</code>　　一个尚未被分　的字符串。例如，如果我　的文本是HTML格式的，它会包含像　<code><p></code>　或者　<code><div></code>　的HTML　，　些　是我　不想索引的。我　可以使用　{ref}/analysis-htmlstrip-charfilter.html[<code>html清除</code>　字符　器]　来移除掉所有的HTML　，并且像把<code>Á</code>　相　的Unicode字符 <code>Á</code>　，　HTML　体。

一个分析器可能有0个或者多个字符　　器。

分　器

一个分析器　必　　有一个唯一的分　器。　分　器把字符串分解成　个　条或者　　元。　准　分析器里使用的　{ref}/analysis-standard-tokenizer.html[　准　　分　器]　把一个字符串根据　　界分解成　个　条，并且移除掉大部分的　点符号，然而　有其他不同行　的分　器存在。

例如，　{ref}/analysis-keyword-tokenizer.html[　　分　器]　完整地　出　接收到的同　的字符串，并不做任何分　。　{ref}/analysis-whitespace-tokenizer.html[空格　　分　器]　只根据空格分割文本　。　{ref}/analysis-pattern-tokenizer.html[正　　分　器]　根据匹配正　表　式来分割文本 。

元　　器

分　，作　　果的　　元流 会按照指定的　序通　指定的　　元　　器 。

元　　器可以修改、添加或者移除　　元。我　已　提到　　`lowercase`　和 {ref}/analysis-stop-tokenfilter.html[　`stop`　　器]　，但是在 Elasticsearch　里面　有很多可供　　的　　元　　器。{ref}/analysis-stemmer-tokenfilter.html[　干　　器]　把　　制　　干。　{ref}/analysis-asciifolding-tokenfilter.html[　`ascii_folding`　　器]移除　音符，把一个像 `"très"`　　的　`"tres"`　　。　{ref}/analysis-ngram-tokenfilter.html[`ngram`]　和　{ref}/analysis-edgengram-tokenfilter.html[`edge_ngram`　　元　　器] 可以　生　合用于部分匹配或者自　　全的　　元。

在　　[search-in-depth]，我　　了在　里使用，以及　　使用分　器和　　器。但是首先，我　需要解　一下　　建自定　的分析器。

建一个自定 分析器

和我 之前配置 es_std 分析器一 ，我 可以在 analysis 下的相 位置 置字符 器、分 器和元 器:

```
PUT /my_index
{
    "settings": {
        "analysis": {
            "char_filter": { ... custom character filters ... },
            "tokenizer":   { ...    custom tokenizers     ... },
            "filter":      { ...   custom token filters   ... },
            "analyzer":    { ...    custom analyzers      ... }
        }
    }
}
```

作 示 ， 我 一起来 建一个自定 分析器 ， 个分析器可以做到下面的 些事:

1. 使用 html清除 字符 器移除HTML部分。
2. 使用一个自定 的 映射 字符 器把 & 替 " 和 " ：

   ```
   "char_filter": {
       "&_to_and": {
           "type":       "mapping",
           "mappings": [ "&=> and "]
       }
   }
   ```

3. 使用 准 分 器分 。
4. 小写 条，使用 小写 器 理。
5. 使用自定 停止 器移除自定 的停止 列表中包含的 ：

   ```
   "filter": {
       "my_stopwords": {
           "type":        "stop",
           "stopwords": [ "the", "a" ]
       }
   }
   ```

我 的分析器定 用我 之前已 置好的自定 器 合了已 定 好的分 器和 器：

```
"analyzer": {
    "my_analyzer": {
        "type":           "custom",
        "char_filter":  [ "html_strip", "&_to_and" ],
        "tokenizer":      "standard",
        "filter":       [ "lowercase", "my_stopwords" ]
    }
}
```

起来，完整的 建索引 求 看起来 像 ：

```
PUT /my_index
{
    "settings": {
        "analysis": {
            "char_filter": {
                "&_to_and": {
                    "type":       "mapping",
                    "mappings": [ "&=> and "]
            }},
            "filter": {
                "my_stopwords": {
                    "type":       "stop",
                    "stopwords": [ "the", "a" ]
            }},
            "analyzer": {
                "my_analyzer": {
                    "type":         "custom",
                    "char_filter":  [ "html_strip", "&_to_and" ],
                    "tokenizer":    "standard",
                    "filter":       [ "lowercase", "my_stopwords" ]
            }}
}}}
```

索引被 建以后，使用 analyze API 来 个新的分析器：

```
GET /my_index/_analyze?analyzer=my_analyzer
The quick & brown fox
```

下面的 略 果展示出我 的分析器正在正 地 行：

```
{
  "tokens" : [
      { "token" :   "quick",    "position" : 2 },
      { "token" :   "and",      "position" : 3 },
      { "token" :   "brown",    "position" : 4 },
      { "token" :   "fox",      "position" : 5 }
    ]
}
```

个分析器 在是没有多大用 的，除非我 告 Elasticsearch在 里用上它。我 可以像下面 把个分析器 用在一个 `string` 字段上：

```
PUT /my_index/_mapping/my_type
{
    "properties": {
        "title": {
            "type":      "string",
            "analyzer":  "my_analyzer"
        }
    }
}
```

型和映射

 型 在 Elasticsearch 中表示一 相似的文 。 型由 名称 —比如 <code>user</code> 或 <code>blogpost</code> —和 映射 成。

映射, 就像数据 中的 schema ，描述了文 可能具有的字段或 属性 、个字段的数据 型—比如 <code>string</code>, <code>integer</code> 或 <code>date</code> —以及Lucene是如何索引和存 些字段的。

型可以很好的抽象 分相似但不相同的数据。但由于 Lucene 的 理方式， 型的使用有些限制。

Lucene 如何 理文

在 Lucene 中，一个文 由一 的 成。 个字段都可以有多个 ，但至少要有一个 。 似的，一个字符串可以通 分析 程 化 多个 。Lucene 不 心 些 是字符串、数字或日期—所有的 都被当做 不透明字 。

当我 在 Lucene 中索引一个文 ， 个字段的 都被添加到相 字段的倒排索引中。 也可以将未 理的原始数据 存 起来，以便 些原始数据在之后也可以被 索到。

型是如何 的

Elasticsearch 型是以 Lucene 理文 的 个方式 基 来 的。一个索引可以有多个 型， 些 型的文 可以存 在相同的索引中。

Lucene 没有文 型的概念， 个文 的 型名被存 在一个叫 _type 的元数据字段上。 当我 要 索某个 型的文 , Elasticsearch 通 在 _type 字段上使用 器限制只返回 个 型的文 。

Lucene 也没有映射的概念。 映射是 Elasticsearch 将 JSON 文 映射 成 Lucene 需要的扁平化数据的方式。

例如，在 user 型中， name 字段的映射可以声明 个字段是 string 型，并且它的 被索引到名叫 name 的倒排索引之前，需要通 whitespace 分 器分析：

```
"name": {
    "type":     "string",
    "analyzer": "whitespace"
}
```

避免 型陷

致了一个有趣的思想 ： 如果有 个不同的 型， 个 型都有同名的字段，但映射不同（例如：一个是字符串一个是数字），将会出 什 情况？

回答是，Elasticsearch 不会允 定 个映射。当 配置 个映射 ，将会出 常。

回答是， 个 Lucene 索引中的所有字段都包含一个 一的、扁平的模式。一个特定字段可以映射成 string 型也可以是 number 型，但是不能 者兼具。因 型是 Elasticsearch 添加的 于 Lucene 的 外机制（以元数据 _type 字段的形式），在 Elasticsearch 中的所有 型最 都共享相同的映射。

以 data 索引中 型的映射 例：

```
{
   "data": {
      "mappings": {
         "people": {
            "properties": {
               "name": {
                  "type": "string",
               },
               "address": {
                  "type": "string"
               }
            }
         },
         "transactions": {
            "properties": {
               "timestamp": {
                  "type": "date",
                  "format": "strict_date_optional_time"
               },
               "message": {
                  "type": "string"
               }
            }
         }
      }
   }
}
```

个 型定 个字段 (分 是 "name"/"address" 和 "timestamp"/"message")。它看起来是相互独立的，但在后台 Lucene 将 建一个映射，如:

```
{
   "data": {
      "mappings": {
        "_type": {
          "type": "string",
          "index": "not_analyzed"
        },
        "name": {
          "type": "string"
        }
        "address": {
          "type": "string"
        }
        "timestamp": {
          "type": "long"
        }
        "message": {
          "type": "string"
        }
      }
   }
}
```

注: 不是真 有效的映射 法，只是用于演示

  于整个索引，映射在本 上被 扁平化 成一个 一的、全局的模式。 就是 什 个 型不能定冲突的字段：当映射被扁平化 ，Lucene 不知道如何去 理。

型

那 ， 个 的 是什 ？技 上 ，多个 型可以在相同的索引中存在，只要它 的字段不冲突（要因 字段是互 独占模式，要 因 它 共享相同的字段）。

重要的一点是: 型可以很好的区分同一个集合中的不同 分。在不同的 分中数据的整体模式是相同的（或相似的）。

  型不 合 完全不同 型的数据 。如果 个 型的字段集是互不相同的，就意味着索引中将有一半的数据是空的（字段将是 稀疏的 ），最 将 致性能 。在 情况下，最好是使用 个 独的索引。

  :

- 正 :将 kitchen 和 lawn-care 型放在 products 索引中, 因 型基本上是相同的模式
- : 将 products 和 logs 型放在 data 索引中, 因 型互不相同。 将它 放在不同的索引中。

根　象

映射的最高一　被称　根　象，它可能包含下面几　：

- 一个 *properties*　点，列出了文　中可能包含的　个字段的映射
- 各　元数据字段，它　都以一个下　　，例如 `_type`、 `_id` 和 `_source`
- 　置　，控制如何　　理新的字段，例如 `analyzer`、 `dynamic_date_formats` 和 `dynamic_templates`
- 其他　置，可以同　　用在根　象和其他　`object`　　型的字段上，例如　`enabled`　、　`dynamic`　和 `include_in_all`

属性

我　已　在 [core-fields] 和 [complex-core-fields] 章　中介　　文　字段和属性的三个最重要的　置：

`type`
　字段的数据　型，例如 `string` 或 `date`

`index`
　字段是否　当被当成全文来搜索（　`analyzed`　），或被当成一个准　的　（　`not_analyzed`　），　是完全不可被搜索（ `no` ）

`analyzer`
　　定在索引和搜索　全文字段使用的 `analyzer`

我　将在本　的后　部分　　其他字段　型，例如 `ip`、 `geo_point` 和 `geo_shape`。

元数据: _source 字段

　　地，Elasticsearch　在　`_source`　字段存　代表文　体的JSON字符串。和所有被存　的字段一　，`_source` 字段在被写入磁　之前先会被　　。

　个字段的存　几乎　是我　想要的，因　它意味着下面的　些：

- 搜索　果包括了整个可用的文　——不需要　外的从　一个的数据　　来取文　。
- 如果没有 `_source` 字段，部分 `update`　求不会生效。
- 当　的映射改　　，　需要重新索引　的数据，有了_source字段　可以直接从Elasticsearch　　做，　而不必从　一个（通常是速度更慢的）数据　　取回　的所有文　。
- 当　不需要看到整个文　　，　个字段可以从 `_source` 字段提取和通　`get` 或者 `search`　求返回。
-　　句更加　　，因　　可以直接看到　个文　包括什　，而不是从一列id　　它　的内容。

然而，存　　`_source`　　字段的　要使用磁　空　。如果上面的原因　　来　没有一个是重要的，　可以用下面的映射禁用 `_source` 字段：

```
PUT /my_index
{
    "mappings": {
        "my_type": {
            "_source": {
                "enabled":  false
            }
        }
    }
}
```

在一个搜索 求里， 可以通 在 求体中指定 _source 参数，来 到只 取特定的字段的效果：

```
GET /_search
{
    "query":   { "match_all": {}},
    "_source": [ "title", "created" ]
}
```

些字段的 会从 _source 字段被提取和返回，而不是返回整个 _source 。

> **Stored Fields 被存 字段**
>
> 了之后的 索，除了索引一个字段的 ， 可以 存 原始字段 。有 Lucene 使用背景的用 使用被存 字段来 他 想要在搜索 果里面返回的字段。事 上， _source 字段就是一个被存 的字段。
>
> 在Elasticsearch中， 文 的个 字段 置存 的做法通常不是最 的。整个文 已 被存 _source 字段。使用 _source 参数提取 需要的字段 是更好的。

元数据: _all 字段

在 [search-lite] 中，我 介 了 _all 字段：一个把其它字段 当作一个大字符串来索引的特殊字段。 query_string 子句(搜索 ?q=john)在没有指定字段 使用 _all 字段。

_all 字段在新 用的探索 段，当 不清楚文 的最 是比 有用的。 可以使用 个字段来做任何 ，并且有很大可能 到需要的文 ：

```
GET /_search
{
    "match": {
        "_all": "john smith marketing"
    }
}
```

随着 用的 展，搜索需求 得更加明 ， 会 自己越来越少使用 _all 字段。 _all 字段是搜索的 急之策。通 指定字段， 的 更加 活、 大， 也可以 相 性最高的搜索 果 行更 粒度的控制。

> **NOTE** relevance algorithm 考 的一个最重要的原 是字段的 度：字段越短越重要。 在 短的 title 字段中出 的短 可能比在 的 content 字段中出 的短 更加重要。字段 度的区 在 _all 字段中不会出 。

如果 不再需要 _all 字段， 可以通 下面的映射来禁用：

```
PUT /my_index/_mapping/my_type
{
    "my_type": {
        "_all": { "enabled": false }
    }
}
```

通 include_in_all 置来逐个控制字段是否要包含在 _all 字段中， 是 true。在一个 象(或根 象)上 置 include_in_all 可以修改 个 象中的所有字段的 行 。

可能想要保留 _all 字段作 一个只包含某些特定字段的全文字段，例如只包含 title, overview, summary 和 tags。 相 于完全禁用 _all 字段， 可以 所有字段 禁用 include_in_all ，在 的字段上 用：

```
PUT /my_index/my_type/_mapping
{
    "my_type": {
        "include_in_all": false,
        "properties": {
            "title": {
                "type":           "string",
                "include_in_all": true
            },
            ...
        }
    }
}
```

住，_all 字段 是一个 分 的 string 字段。它使用 分 器来分析它的 ，不管 个原本所在字段指定的分 器。就像所有 string 字段， 可以配置 _all 字段使用的分 器：

```
PUT /my_index/my_type/_mapping
{
    "my_type": {
        "_all": { "analyzer": "whitespace" }
    }
}
```

元数据：文

文　　与四个元数据字段相　：

`_id`
　文　的 ID 字符串

`_type`
　文　的　型名

`_index`
　文　所在的索引

`_uid`
　`_type` 和 `_id`　接在一起　造成 `type#id`

　情况下，`_uid` 字段是被存　（可取回）和索引（可搜索）的。`_type` 字段被索引但是没有存　，`_id` 和 `_index` 字段　既没有被索引也没有被存　，　意味着它　并不是真　存在的。

尽管如此，　然可以像真　字段一　　`_id` 字段。Elasticsearch 使用 `_uid` 字段来派生出 `_id` 。　然　可以修改　些字段的 `index` 和 `store`　置，但是基本上不需要　做。

映射

当 Elasticsearch 遇到文　中以前　未遇到的字段，它用 *dynamic mapping* 来　定字段的数据　型并自　把新的字段添加到　型映射。

有　是想要的行　有　又不希望　。通常没有人知道以后会有什　新字段加到文　，但是又希望　些字段被自　的索引。也　只想忽略它　。如果Elasticsearch是作　重要的数据存　，可能就会期望遇到新字段就会　出　常，　能及　　。

幸　的是可以用 `dynamic` 配置来控制　行　，可接受的　如下：

`true`
　添加新的字段—　省

`false`
　忽略新的字段

`strict`
　如果遇到新字段　出　常

配置参数 `dynamic` 可以用在根 `object` 或任何 `object`　型的字段上。　可以将 `dynamic` 的　置 `strict`，而只在指定的内部　象中　它，例如：

```
PUT /my_index
{
    "mappings": {
        "my_type": {
            "dynamic":      "strict", ①
            "properties": {
                "title":  { "type": "string"},
                "stash":  {
                    "type":     "object",
                    "dynamic":  true ②
                }
            }
        }
    }
}
```

① 如果遇到新字段， 象 my_type 就会 出 常。

② 而内部 象 stash 遇到新字段就会 建新字段。

使用上述 映射， 可以 stash 象添加新的可 索的字段：

```
PUT /my_index/my_type/1
{
    "title":   "This doc adds a new field",
    "stash": { "new_field": "Success!" }
}
```

但是 根 点 象 my_type 行同 的操作会失 ：

```
PUT /my_index/my_type/1
{
    "title":     "This throws a StrictDynamicMappingException",
    "new_field": "Fail!"
}
```

NOTE

把 dynamic 置 false 一点儿也不会改 _source 的字段内容。 _source 然包含被索引的整个JSON文 。只是新的字段不会被加到映射中也不可搜索。

自定 映射

如果 想在 行 加新的字段， 可能会 用 映射。然而，有 候， 映射可能不太智能。幸 的是，我 可以通 置去自定 些 ，以便更好的 用于 的数据。

日期

当 Elasticsearch 遇到一个新的字符串字段 ，它会 个字段是否包含一个可 的日期，比如 2014-

01-01 。如果它像日期， 个字段就会被作 date 型添加。否 ，它会被作 string 型添加。

有些 候 个行 可能 致一些 。想象下， 有如下 的一个文 ：

```
{ "note": "2014-01-01" }
```

假 是第一次 note 字段，它会被添加 date 字段。但是如果下一个文 像 ：

```
{ "note": "Logged out" }
```

然不是一个日期，但 已 。 个字段已 是一个日期 型， 个 不合法的日期 将会造成一个常。

日期 可以通 在根 象上 置 date_detection false 来 ：

```
PUT /my_index
{
    "mappings": {
        "my_type": {
            "date_detection": false
        }
    }
}
```

使用 个映射，字符串将始 作 string 型。如果 需要一个 date 字段， 必 手 添加。

NOTE	Elasticsearch 判断字符串 日期的 可以通 {ref}/dynamic-field-mapping.html#date-detection[dynamic_date_formats setting] 来 置。

模板

使用 dynamic_templates ， 可以完全控制新 生成字段的映射。 甚至可以通 字段名称或数据型来 用不同的映射。

个模板都有一个名称， 可以用来描述 个模板的用途， 一个 mapping 来指定映射 使用，以及至少一个参数 (如 match) 来定 个模板 用于 个字段。

模板按照 序来 ；第一个匹配的模板会被 用。例如，我 string 型字段定 个模板：

- es ：以 _es 尾的字段名需要使用 spanish 分 器。
- en ：所有其他字段使用 english 分 器。

我 将 es 模板放在第一位，因 它比匹配所有字符串字段的 en 模板更特殊：

```
PUT /my_index
{
    "mappings": {
        "my_type": {
            "dynamic_templates": [
                { "es": {
                      "match":              "*_es", ①
                      "match_mapping_type": "string",
                      "mapping": {
                          "type":           "string",
                          "analyzer":       "spanish"
                      }
                }},
                { "en": {
                      "match":              "*", ②
                      "match_mapping_type": "string",
                      "mapping": {
                          "type":           "string",
                          "analyzer":       "english"
                      }
                }}
            ]
}}}
```

① 匹配字段名以 _es 尾的字段。

② 匹配其他所有字符串 型字段。

match_mapping_type 允 用模板到特定 型的字段上，就像有 准 映射 的一 ， (例如 string 或 long)。

match 参数只匹配字段名称， path_match 参数匹配字段在 象上的完整路径，所以 address.*.name 将匹配 的字段：

```
{
    "address": {
        "city": {
            "name": "New York"
        }
    }
}
```

unmatch 和 path_unmatch将被用于未被匹配的字段。

更多的配置 {ref}/dynamic-mapping.html[映射文] 。

省映射

通常，一个索引中的所有 型共享相同的字段和 置。 default 映射更加方便地指定通用 置，而不是

次 建新 型 都要重 置。 default 映射是新 型的模板。在 置 default 映射之后 建的所有 型都将 用 些 省的 置，除非 型在自己的映射中明 覆 些 置。

例如，我 可以使用 default 映射 所有的 型禁用 _all 字段， 而只在 blog 型 用：

```
PUT /my_index
{
    "mappings": {
        "_default_": {
            "_all": { "enabled":  false }
        },
        "blog": {
            "_all": { "enabled":  true  }
        }
    }
}
```

default 映射也是一个指定索引 dynamic templates 的好方法。

重新索引 的数据

尽管可以 加新的 型到索引中，或者 加新的字段到 型中，但是不能添加新的分析器或者 有的字段做改 。如果 那 做的 ， 果就是那些已 被索引的数据就不正 ， 搜索也不能正常工作。

有数据的 改 最 的 法就是重新索引：用新的 置 建新的索引并把文 从旧的索引 制到新的索引。

字段 _source 的一个 点是在Elasticsearch中已 有整个文 。 不必从源数据中重建索引，而且那 通常比 慢。

了有效的重新索引所有在旧的索引中的文 ，用 *scroll* 从旧的索引 索批量文 ， 然后用 bulk API 把文 推送到新的索引中。

从Elasticsearch v2.3.0 始， {ref}/docs-reindex.html[Reindex API] 被引入。它能 文 重建索引而不需要任何 件或外部工具。

批量重新索引

同 并行 行多个重建索引任 ，但是 然不希望 果有重 。正 的做法是按日期或者 的字段作 条件把大的重建索引分成小的任 ：

```
GET /old_index/_search?scroll=1m
{
    "query": {
        "range": {
            "date": {
                "gte":  "2014-01-01",
                "lt":   "2014-02-01"
            }
        }
    },
    "sort": ["_doc"],
    "size":  1000
}
```

如果旧的索引持 会有 化， 希望新的索引中也包括那些新加的文 。那就可以 新加的文 做重新索引，但 是要用日期 字段 来匹配那些新加的文 。

索引 名和零停机

在前面提到的，重建索引的 是必 更新 用中的索引名称。索引 名就是用来解决 个 的！

索引 名 就像一个快捷方式或 接，可以指向一个或多个索引，也可以 任何一个需要索引名的API来使用。 名 我 大的 活性，允 我 做下面 些：

- 在 行的集群中可以无 的从一个索引切 到 一个索引
- 多个索引分 (例如， `last_three_months`)
- 索引的一个子集 建

在后面我 会 更多 于 名的使用。 在，我 将解 使用 名在零停机下从旧索引切 到新索引。

有 方式管理 名： `_alias` 用于 个操作， `_aliases` 用于 行多个原子 操作。

在本章中，我 假 的 用有一个叫 `my_index` 的索引。事 上， `my_index` 是一个指向当前真 索引的 名。真 索引包含一个版本号： `my_index_v1` ， `my_index_v2` 等等。

首先， 建索引 `my_index_v1` ，然后将 名 `my_index` 指向它：

```
PUT /my_index_v1 ①
PUT /my_index_v1/_alias/my_index ②
```

① 建索引 `my_index_v1` 。

② 置 名 my_index 指向 my_index_v1 。

可以 个 名指向 一个索引：

```
GET /*/_alias/my_index
```

或 些 名指向 个索引：

```
GET /my_index_v1/_alias/*
```

者都会返回下面的 果：

```
{
    "my_index_v1" : {
        "aliases" : {
            "my_index" : { }
        }
    }
}
```

然后，我 决定修改索引中一个字段的映射。当然，我 不能修改 存的映射，所以我 必 重新索引数据 。 首先, 我 用新映射 建索引 my_index_v2 ：

```
PUT /my_index_v2
{
    "mappings": {
        "my_type": {
            "properties": {
                "tags": {
                    "type":   "string",
                    "index":  "not_analyzed"
                }
            }
        }
    }
}
```

然后我 将数据从 my_index_v1 索引到 my_index_v2 ，下面的 程在 重新索引 的数据 中已 描述 。一旦我 定文 已 被正 地重索引了，我 就将 名指向新的索引。

一个 名可以指向多个索引，所以我 在添加 名到新索引的同 必 从旧的索引中 除它。 个操作需要原子化， 意味着我 需要使用 _aliases 操作：

```
POST /_aliases
{
    "actions": [
        { "remove": { "index": "my_index_v1", "alias": "my_index" }},
        { "add":    { "index": "my_index_v2", "alias": "my_index" }}
    ]
}
```

的　用已　在零停机的情况下从旧索引　移到新索引了。

TIP

即使　　在的索引　　已　很完美了，在生　　境中，　是有可能需要做一些修改的。

做好准　：在　的　用中使用　名而不是索引名。然后　就可以在任何　候重建索引。　名的　很小，　　广泛使用。