同

干提取是通 化他 的 根形式来 大搜索的 ，同 通 相 的 念和概念来 大搜索 。也 没有文 匹配 “英国女王“，但是包含 “英国君主” 的文 可能会被 是很好的匹配。

用 搜索 “美国” 并且期望 到包含 美利 合 国 、 美国 、 美洲 、或者 美国各州 的文 。 然而，他 不希望搜索到 于 国事 或者 政府机 的 果。

个例子提供了宝 的 ，它向我 述了，区分不同的概念 于人 是多 而 于 粹的机器是多棘手的事情。通常我 会 言中的 一个 去 提供同 以 保任何一个文 都是可 的，以保不管文 之 有多 微小的 性都能 被 索出来。

做是不 的。就像我 更喜 不用或少用 根而不是 分使用 根一 ，同 也 只在必要的 候使用。 是因 用 可以理解他 的搜索 果受限于他 的搜索 ，如果搜索 果看上去几乎是随机，他 就会 得无法理解（注：大 模使用同 会 致 果 向于 人 得是随机的）。

同 可以用来合并几乎相同含 的 ，如 跳 、 跳越 或者 脚跳行 ，和 小 子 、 或者 料手 。或者，它 可以用来 一个 得更通用。例如， 可以作 猫 或 子 的通用代名 ， 有， 成人 可以被用于 男人 或者 女人 。

同 似乎是一个 的概念，但是正 的使用它 却是非常困 的。在 一章，我 会介 使用同 的技巧和 它的局限性和陷 。

TIP	同 大了一个匹配文件的 。正如 干提取 或者 部分匹配 ，同 的字段不 被 独使用，而 与一个 主字段的 操作一起使用， 个主字段 包含 格式的原始文本。 在使用同 ，参 [most-fields] 的解 来 相 性。

使用同

同 可以取代 有的 元或 通 使用 {ref}/analysis-synonym-tokenfilter.html[同元 器]，添加到 元流中：

```
PUT /my_index
{
  "settings": {
    "analysis": {
      "filter": {
        "my_synonym_filter": {
          "type": "synonym", ①
          "synonyms": [ ②
            "british,english",
            "queen,monarch"
          ]
        }
      },
      "analyzer": {
        "my_synonyms": {
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "my_synonym_filter" ③
          ]
        }
      }
    }
  }
}
```

① 首先，我 定 了一个 同 型的 元 器。

② 我 在 同 格式 中 同 格式。

③ 然后我 建了一个使用 my_synonym_filter 的自定 分析器。

TIP 同 可以使用 synonym 参数来内嵌指定，或者必 存在于集群 一个 点上的同 文件中。 同 文件路径由 synonyms_path 参数指定， 或相 于 Elasticsearch config 目 。参照 [updating-stopwords] 的技巧，可以用来刷新的同 列表。

通 analyze API 来 我 的分析器， 示如下：

```
GET /my_index/_analyze?analyzer=my_synonyms
Elizabeth is the English queen
```

```
Pos 1: (elizabeth)
Pos 2: (is)
Pos 3: (the)
Pos 4: (british,english) ①
Pos 5: (queen,monarch) ①
```

① 所有同 与原始 占有同一个位置。

的一个文件将匹配任何以下的 ：English queen 、British queen 、 English monarch 或 British monarch 。即使是一个短 也将会工作，因 个 的位置已被保存。

TIP

在索引和搜索中使用相同的同 元 器是多余的。 如果在索引的 候，我 用 english 和 british 个 代替 English ， 然后在搜索的 候，我 只需要搜索 些 中的一个。或者，如果在索引的 候我 不使用同 ，然后在搜索的 候，我 将需要把 English 的 english 或者 british 的 。

是否在搜索或索引的 候做同 展可能是一个困 的 。我 将探索更多的 展或收 。

同 格式

同 最 的表 形式是 逗号分隔：

```
"jump,leap,hop"
```

如果遇到 些 中的任何一 ， 将其替 所有列出的同 。例如：

```
原始 : 取代:
————————————————————————————————————————————
jump            → (jump,leap,hop)
leap            → (jump,leap,hop)
hop             → (jump,leap,hop)
```

或者, 使用 ⇒ 法，可以指定一个 列表（在左 ），和一个或多个替 （右 ）的列表：

```
"u s a,united states,united states of america => usa"
"g b,gb,great britain => britain,england,scotland,wales"
```

```
原始 : 取代:
————————————————————————————————————————————
u s a           → (usa)
united states   → (usa)
great britain   → (britain,england,scotland,wales)
```

如果多个 指定同一个同 ，它 将被合并在一起，且 序无 ，否 使用最 匹配。以下面的例：

```
"united states            => usa",
"united states of america => usa"
```

如果 些 相互冲突，Elasticsearch 会将 United States of America (usa),(of),(america) 。否 ，会使用最 的序列，即最 得到 (usa) 。

展或收

在 同　格式 中，我　看到了可以通　　展 、　　收　、或_　型　展_ 来指明同　　。 本章　我　将在　三者　做个　衡比　。

TIP	本　　理　同　　。多　同　　又　添了一　　性，在　　多　同　　和短　　中，我　将会　　。

展

通　　展，我　可以把同　　列表中的任意一个　　展成同　　列表 所有 的　：

```
"jump,hop,leap"
```

展可以　用在索引　段或　　段。　者都有　点　（　）　　和　点　（　）　。到底要在　个　段使用，取决于性能与　活性：

	索引	
索引的大小	大索引。因　所有的同　　都会被索引，所以索引的大小相　会　大一些。	正常大小。
	所有同　　都有相同的 IDF（至于什　是 IDF，参 [relevance-intro]），　意味着通用的　和　常用的　都有着相同的　重。	个同　　IDF 都和原来一　。
性能	只需要　到　字符串中指定　个　　。	一个　　的　　重写来　所有的同　　，从而降低性能。
活性	同　　不能改　有的文件。　于有影　的新　，　有的文件都要重建（注：重新索引一次文　）。	同　　可以更新不需要索引文件。

收

收　，把 左　的多个同　　映射到了右　的　个　：

```
"leap,hop => jump"
```

它必　同　　用于索引和　　段，以　保　　映射到索引中存在的同一个　。

相　于　　展方法，　　方法也有一些　点和一些　点：

索引的大小

索引大小是正常的，因 只有 一 被索引。

所有 的 IDF 是一 的，所以 不能区分比 常用的 、不常用的 。

性能

只需要在索引中 到 的出 。

活性

新同 可以添加到 的左 并在 段使用。例如，我 想添加 bound 到先前指定的同 中。那 下面的 将作用于包含 bound 的 或包含 bound 的文 索引：

```
"leap,hop,bound => jump"
```

似乎 旧有的文 不起作用是 ？其 我 可以把上面 个同 改写下，以便 旧有文 同 起作用：

```
"leap,hop,bound => jump,bound"
```

当 重建索引文件， 可以恢 到上面的 （注： leap,hop,bound ⇒ jump ）来 得 个 的性能 （注：因 上面那个 相比 个而言， 段就只要 一个 了）。

型 展

型 展是完全不同于 收 或 ， 并不是平等看待所有的同 ，而是 大了 的意 ，使被拓展的 更 通用。以 些 例：

```
"cat    => cat,pet",
"kitten => kitten,cat,pet",
"dog    => dog,pet"
"puppy  => puppy,dog,pet"
```

通 在索引 段使用 型 展：

- 一个 于 kitten 的 会 于 kittens 的文 。
- 一个 cat 会 到 于 kittens 和 cats 的文 。
- 一个 pet 的 将 有 的 kittens、cats、puppies、dogs 或者 pets 的文 。

或者在 段使用 型 展， kitten 的 果就会被拓展成 及到 kittens、cats、dogs。

也可以有 全其美的 法，通 在索引 段 用 型 展同 ，以 保 型在索引中存在。然后，在 段， 可以 不采用同 （使 kitten 只返回 kittens 的文件）或采用同 ， kitten 的 操作就会返回包括 kittens、cats、pets（也包括 dogs 和 puppies）的相 果。

前面的示例 ， kitten 的 IDF 将是正 的，而 cat 和 pet 的 IDF 将会被 Elasticsearch 降 。然而，是 有利的，当一个 kitten 的 被拓展成了 kitten OR cat OR pet 的 ， 那 kitten

相 的文 就 排在最上方，其次是 cat 的文件， pet 的文件将被排在最底部。

同 和分析

在 同 格式 一章中，我 使用 u s a 来 例 述一些同 相 的知 。那 什 我 使用的不是 U.S.A. ？原因是， 个 同 的 元 器只能接收到在它前面的 元 器或者分 器的 出 果（ 里看不到原始文本）。

假 我 有一个分析器，它由 standard 分 器、 lowercase 的 元 器、 synonym 的 元 器 成。文本 U.S.A. 的分析 程，看起来像 的：

```
original string（原始文本）                    → "U.S.A."
standard          tokenizer（分 器）           → (U),(S),(A)
lowercase         token filter（ 元 器）       → (u),(s),(a)
synonym           token filter（ 元 器）       → (usa)
```

如果我 有指定的同 U.S.A. ，它永 不会匹配任何 西。因 ， my_synonym_filter 看到 的 候，句号已 被移除了，并且字母已 被小写了。

其 是一个非常需要注意的地方。如果我 想同 使用同 特性与 根提取特性，那 jumps 、 jumped 、 jump 、 leaps 、 leaped 和 leap 些 是否都会被索引成一个 jump ？ 我 可以把同 器放置在 根提取之前，然后把所有同 以及 形 化都列 出来：

```
"jumps,jumped,leap,leaps,leaped => jump"
```

但更 的方式将同 器放置在 根 器之后，然后把 根形式的同 列 出来：

```
"leap => jump"
```

大小写敏感的同

通常，我 把同 器放置在 lowercase 元 器之后，因此，所有的同 都是小写。但有 会 致奇怪的合并。例如， CAT 描和一只 cat 有很大的不同，或者 PET （正 子 射断 描）和 pet 。 就此而言，姓 Little 也是不同于形容 little 的 (尽管当一个句子以它 ，首字母会被大写)。

如果根据使用情况来区分 ， 需要将同 器放置在 lowercase 器之前。当然，意味着同 需要列出所有想匹配的 化（例如， Little、LITTLE、little ）。

相反,可以有 个同 器：一个匹配大小写敏感的同 ，一个匹配大小写不敏感的同 。例如，大小写敏感的同 可以是 个 子：

```
"CAT,CAT scan           => cat_scan"
"PET,PET scan           => pet_scan"
"Johnny Little,J Little => johnny_little"
"Johnny Small,J Small   => johnny_small"
```

大小不敏感的同　　可以是　个　子：

```
"cat                   => cat,pet"
"dog                   => dog,pet"
"cat scan,cat_scan scan => cat_scan"
"pet scan,pet_scan scan => pet_scan"
"little,small"
```

大小写敏感的同　　不　会　理　CAT　scan　，而且有　候也可能会匹配到　CAT　scan　中的　CAT（注：从而　致　CAT　scan　被　化成了同　　cat_scan　scan　）。出于　个原因，在大小写敏感的同　　列表中会有一个　　坏替　情况的特　　cat_scan scan 。

提示：　　可以看到它　可以多　　易地　得　　。同平　一　，　　analyze　　API　　是　手，用它来　　分析器是否正　配置。参　[analyze-api]。

多　同　　和短

至此，同　　看上去　挺　　的。然而不幸的是，　　的部分才　　始。　了能使 短　　正常工作，Elasticsearch　　需要知道　个　在初始文本中的位置。多　同　　会　重破坏　的位置信息，尤其当新　的同　　度各不相同的　候。

我　　建一个同　　元　　器，然后使用下面　　的同　　：

```
"usa,united states,u s a,united states of america"
```

```
PUT /my_index
{
  "settings": {
    "analysis": {
      "filter": {
        "my_synonym_filter": {
          "type": "synonym",
          "synonyms": [
            "usa,united states,u s a,united states of america"
          ]
        }
      },
      "analyzer": {
        "my_synonyms": {
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "my_synonym_filter"
          ]
        }
      }
    }
  }
}

GET /my_index/_analyze?analyzer=my_synonyms&text=
The United States is wealthy
```

解析器 会 出下面 的 果：

```
Pos 1:  (the)
Pos 2:  (usa,united,u,united)
Pos 3:  (states,s,states)
Pos 4:  (is,a,of)
Pos 5:  (wealthy,america)
```

如果 用上面 个同 元 器索引一个文 ，然后 行一个短 ，那 就会得到 人的 果 ，下面 些短 都不会匹配成功：

- The usa is wealthy
- The united states of america is wealthy
- The U.S.A. is wealthy

但是 些短 会：

- United states is wealthy
- Usa states of wealthy

- The U.S. of wealthy
- U.S. is america

如果 是在 段使同 ，那 就会看到更加 的匹配 果。看下 个 `validate-query` ：

```
GET /my_index/_validate/query?explain
{
  "query": {
    "match_phrase": {
      "text": {
        "query": "usa is wealthy",
        "analyzer": "my_synonyms"
      }
    }
  }
}
```

字会被同 元 器 理成 似 的信息：

```
"(usa united u united) (is states s states) (wealthy a of) america"
```

会匹配包含有 `u is of america` 的文 ，但是匹配不出任何含有 `america` 的文 。

TIP 多 同 高亮匹配 果也会造成影 。一个 `USA` 的 ，返回的 果可能却高亮了：The *United States is wealthy* 。

使用 收 行短

避免 混乱的方法是使用 收 ， 用 个 表示所有的同 ， 然后在段，就只需要 个 行 了：

```
PUT /my_index
{
  "settings": {
    "analysis": {
      "filter": {
        "my_synonym_filter": {
          "type": "synonym",
          "synonyms": [
            "united states,u s a,united states of america=>usa"
          ]
        }
      },
      "analyzer": {
        "my_synonyms": {
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "my_synonym_filter"
          ]
        }
      }
    }
  }
}

GET /my_index/_analyze?analyzer=my_synonyms
The United States is wealthy
```

上面那个　信息就会被　理成　似下面　：

```
Pos 1:  (the)
Pos 2:  (usa)
Pos 3:  (is)
Pos 5:  (wealthy)
```

在我　再次　行我　之前做　的那个 validate-query　，就会　出一个　又合理的　果：

```
"usa is wealthy"
```

个方法的　点是，因　把 united states of america　成了同　usa，　就不能使用 united states of america 去搜索出 united 或者 states 。　需要使用一个　外的字段并用　一个解析器　来　到个目的。

同　与 query_string

本　很少　到 query_string　，因　真心不推　用它。　在　一　中有提到，由于 query_string　支持一个精　的　法　，因此，可能　会　致它搜出一些出人意料的

果或者甚至是含有 法 的 果。

方式存在不少 ，而其中之一便与多 同 有 。 了支持它的 法， 必 用指定的、法所能 的操作符号来 示，比如 AND 、 OR 、 + 、 - 、 field: 等等。 (更多相 内容参 {ref}/query-dsl-query-string-query.html#query-string-syntax[query_string 法] 。)

而在 法的解析 程中，解析 作会把 文本在空格符 作切分，然后分 把 个切分出来的相 性解析器。 也即意味着 的同 解析器永 都不可能收到 似 United States 的多个 成的同 。由于不会把 United States 作 一个原子性的文本，所以同 解析器的 入信息永都是 个被切分 的 United 和 States 。

所幸， match 相比而言就可 得多了，因 它不支持上述 法，所以多个字 成的同 不会被切分，而是会完整地交 解析器 理。

符号同

最后一 内容我 来 述下 符号 行同 理， 和我 前面 的同 理不太一 。符号同 是用 名来表示 个符号，以防止它在分 程中被 是不重要的 点符号而被移除。

然 大多数情况下，符号 于全文搜索而言都无 要，但是字符 合而成的表情，或 又会是很有意的 西，甚至有 候会改 整个句子的含 ， 比一下 句 ：

- 我很高 能在星期天工作。
- 我很高 能在星期天工作 :(（注： 的表情）

准 （注：standard）分 器或 会 地消除掉第二个句子里的字符表情，致使 个原本意思相去甚的句子 得相同。

我 可以先使用 {ref}/analysis-mapping-charfilter.html[映射]字符 器，在文本被 交 分 器 理之前， 把字符表情替 成符号同 emoticon_happy 或者 emoticon_sad ：

```
PUT /my_index
{
  "settings": {
    "analysis": {
      "char_filter": {
        "emoticons": {
          "type": "mapping",
          "mappings": [ ①
            ":)=>emoticon_happy",
            ":(=>emoticon_sad"
          ]
        }
      },
      "analyzer": {
        "my_emoticons": {
          "char_filter": "emoticons",
          "tokenizer":   "standard",
          "filter":    [ "lowercase" ]
          ]
        }
      }
    }
  }
}

GET /my_index/_analyze?analyzer=my_emoticons
I am :) not :( ②
```

① 映射　　器把字符从 ⇒ 左　的格式　　成右　的　子。

②　出：i、am、emoticon_happy、not、emoticon_sad。

很少有人会搜　　emoticon_happy　　个　，但是　保　似字符表情的　　重要符号被存　到索引中是非常好的做法，在　行情感分析的　候会很有用。当然，我　也可以用真　的　　来　理符号同　　，比如：happy 或者 sad。

提示：　　映射　　字符　　器是个非常有用的　　器，它可以用来　一些已有的字　　行替　操作，　如果想要采用更　活的正　表　式去替　字　的　，那　可以使用　　{ref}/analysis-pattern-replace-charfilter.html[pattern_replace]字符　　器。