始　理各　　言

Elasticsearch　很多世界流行　言提供良好的、　　的、　箱即用的　言分析器集合：

阿拉伯　、　美尼　　、巴斯克　、巴西　、保加利　　、加泰　尼　　、中文、捷克　、丹麦、荷　　、英　、　　、法　、加里西　　、　　、希　　、北印度　、匈牙利　、印度尼西　、　　、意大利　、日　、　国　、　　、　威　、波斯　、葡萄牙　、　　尼　　、俄　、西班牙　、瑞典　、土耳其　和泰　。

些分析器承担以下四　角色：

- 文本拆分　　：

  `The quick brown foxes` → [`The`, `quick`, `brown`, `foxes`]

- 大写　小写：

  `The` → `the`

- 移除常用的 停用　：

  [`The`, `quick`, `brown`, `foxes`] → [`quick`, `brown`, `foxes`]

- 将　型　（例如　数　，　去式）　化　　根：

  `foxes` → `fox`

了更好的搜索性，　个　言的分析器提供了　　言　　的具体　　：

- 英　分析器移除了所有格 `'s`

  `John's` → `john`

- 法　分析器移除了 元音省略 例如 `l'` 和 `qu'` 和　音符号 例如 `¨` 或 `^` ：

  `l'église` → `eglis`

-　　分析器　　化了切　，将切　中的 `ä` 和 `ae` 替　　`a`，或将 `ß` 替　　`ss` ：

  `äußerst` → `ausserst`

使用　言分析器

Elasticsearch　　的内置分析器都是全局可用的，不需要提前配置，它　也可以在字段映射中直接指定在某字段上：

```
PUT /my_index
{
  "mappings": {
    "blog": {
      "properties": {
        "title": {
          "type":     "string",
          "analyzer": "english" ①
        }
      }
    }
  }
}
```

① title 字段将会用 english（英 ）分析器替 的 standard（ 准）分析器

当然，文本 english 分析 理，我 会 失源数据：

```
GET /my_index/_analyze?field=title ①
I'm not happy about the foxes
```

① 切 ：i'm, happi, about, fox

我 无法分 源文 中是包含 数 fox 是 数 foxes ； not 因 是停用 所以被移除了， 所以我 无法分 源文 中是happy about foxes 是not happy about foxes， 然通 使用 english （英 ）分析器，使得匹配 更加 松，我 也因此提高了召回率，但却降低了精准匹配文 的能力。

了 得 方面的 ，我 可以使用multifields（多字段） title 字段建立 次索引： 一次使用 english（英 ）分析器， 一次使用 standard（ 准）分析器:

```
PUT /my_index
{
  "mappings": {
    "blog": {
      "properties": {
        "title": { ①
          "type": "string",
          "fields": {
            "english": { ②
              "type":     "string",
              "analyzer": "english"
            }
          }
        }
      }
    }
  }
}
```

① 主 title 字段使用 standard（ 准）分析器。

② title.english 子字段使用 english（英 ）分析器。

替 字段映射后，我 可以索引一些 文 来展示 在搜索 使用 个字段：

```
PUT /my_index/blog/1
{ "title": "I'm happy for this fox" }

PUT /my_index/blog/2
{ "title": "I'm not happy about my fox problem" }

GET /_search
{
  "query": {
    "multi_match": {
      "type":     "most_fields", ①
      "query":    "not happy foxes",
      "fields": [ "title", "title.english" ]
    }
  }
}
```

① 使用most_fields query type（多字段搜索 法来） 我 可以用多个字段来匹配同一段文本。

感 title.english 字段的切 ，无 我 的文 中是否含有 foxes 都会被搜索到，第二 文 的相性排行要比第一 高， 因 在 title 字段中匹配到了 not 。

配置 言分析器

言分析器都不需要任何配置， 箱即用， 它 中的大多数都允 控制它 的各方面行 ，具体来 ：

干提取排除

想象下某个 景，用 想要搜索 World Health Organization 的 果， 但是却被替 搜索 organ health 的 果。有 个困惑是因 organ 和 organization 有相同的 根： organ 。 通常 不是什 ，但是在一些特殊的文 中就会 致有 的 果，所以我 希望防止 organization 和 organizations 被 干。

自定 停用

英 中 的停用 列表如下：

```
a, an, and, are, as, at, be, but, by, for, if, in, into, is, it,
no, not, of, on, or, such, that, the, their, then, there, these,
they, this, to, was, will, with
```

于 no 和 not 有点特 ， 会反 跟在它 后面的 的含 。或 我 个很重要，不 把他 看成停用 。

了自定 english （英 ）分 器的行 ，我 需要基于 english （英 ）分析器 建一个自定

分析器，然后添加一些配置：

```
PUT /my_index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_english": {
          "type": "english",
          "stem_exclusion": [ "organization", "organizations" ], ①
          "stopwords": [ ②
            "a", "an", "and", "are", "as", "at", "be", "but", "by", "for",
            "if", "in", "into", "is", "it", "of", "on", "or", "such", "that",
            "the", "their", "then", "there", "these", "they", "this", "to",
            "was", "will", "with"
          ]
        }
      }
    }
  }
}

GET /my_index/_analyze?analyzer=my_english ③
The World Health Organization does not sell organs.
```

① 防止 organization 和 organizations 被 干

② 指定一个自定 停用 列表

③ 切 world 、 health 、 organization 、 does 、 not 、 sell 、 organ

我 在 [stemming] 和 [stopwords] 中分 了 干提取和停用 。

混合 言的陷

如果 只需要 理一 言，那 很幸 。 到一个正 的策略用于 理多 言文 是一 巨大的挑 。

在索引的 候

多 言文 主要有以下三个 型：

- 一 是 *document* （文 ）有自己的主 言，并包含一些其他 言的片段（参考 文 一言。）
- 一 是 个 *field* （域）有自己的主 言，并包含一些其他 言的片段（参考 个域一 言。）
- 一 是 个 *field* （域）都是混合 言（参考 混合 言域。）

（分 ）目 不 是可以 ，我 当保持将不同 言分隔 。在同一 倒排索引内混合多 言可能造成一些 。

不合理的 干提取

的 干提取 跟英 ，法 ，瑞典 等是不一 的。 不同的 言提供同 的 干提 将会致有的 的 根 的正 ，有的 的 根 的不正 ，有的 根本 不到 根。 甚至是将不同言的不同含 的 切 同一个 根，合并 些 根的搜索 果会 用 来困 。

提供多 的 干提取器 流切分同一 文 的 果很有可能得到一堆 ，因 下一个 干提取器会 切分一个已 被 干的 ， 加 了上面提到的 。

写方式一 干提取器

只有一 情况， *only-one-stemmer* （唯一 干提取器）会 生，就是 言都有自己的写方式。例如，在以色列就有很大的可能一个文 包含希伯来 ， 阿拉伯 ，俄 （古代斯拉夫 ），和英 。

```
- Предупреждение -          - Warning
```

言使用不同的 写方式，所以一 言的 干提取器就不会干 其他 言的，允 同一 文本提供多 干提取器。

不正 的倒排文 率

在 [relevance-intro] （相 性教程）中，一个 term （ ）在一 文 中出 的 率 高， term（ ）的 重就越低。 了精 的 算相 性， 需要精 的 term-frequency（ ）。

一段 文出 在英 主的文本中会 与 更高的 重， 那 高 重是因 相 来 更稀有。但是如果 文 跟以 主的文 混合在一起，那 段 文就会有很低的 重。

在搜索的 候

然而 考 的文 是不 的 。 也需要考 的用 会 搜索 些文 。 通常 能从用 的 言界面来 定用 的主 言，（例如， `mysite.de` 和 `mysite.fr` ） 或者从用 的 器的HTTP header（HTTP 文件） `accept-language` 定。

用 的搜索也注意有三个方面:

- 用 使用他的主 言搜索。
- 用 使用其他的 言搜索，但希望 取主 言的搜索 果。
- 用 使用其他 言搜索，并希望 取 言的搜索 果。（例如，精通双 的人，或者 的外国 者）。

根据 搜索数据的 型，或 会返回 言的合 果（例如，一个用 在西班牙 站搜索商品），也可能是用 主 言的搜索 果和其他 言的搜索 果混合。

通常来 ， 与用 言偏好的搜索很有意 。一个使用英 的用 搜索 更希望看到英 Wikipedia 面而不是法 Wikipedia 面。

言

很可能已 知道 的文 所 用的 言，或者 的文 只是在 自己的 内 写并被翻 成 定的一系列 言。人 的 可能是最可 的将 言正 的方法。

然而，或 的文 来自第三方 源且没 言 ，或者是不正 的 。 情况下， 需要一个学 算法来 文 的主 言。幸 的是，一些 言有 成的工具包可以 解决 个 。

内容是来自 Mike McCandless 的 chromium-compact-language-detector 工具包，使用的是google 的基于 (Apache License 2.0)的 源工具包 Compact Language Detector (CLD) 。 它小巧，快速，且精 ，并能根据短短的 句 就可以 160+ 的 言。 它甚至能 文本 多 言。支持多 言包括 Python，Perl，JavaScript，PHP，C#/.NET，和 R 。

定用 搜索 求的 言并不是那 。 CLD 是 了至少 200 字符 的文本 的。字符短的文本，例如搜索 字，会 生不精 的 果。 情况下，或 采取一些 的 式算法会更好些，例如 国家的官方 言，用 的 言，和 HTTP `accept-language` headers （HTTP 文件）。

文 一 言

个主 言文 只需要相当 的 置。 不同 言的文 被分 存放在不同的索引中 — <code>blogs-en</code> 、 <code>blogs-fr</code> , 如此等等 — 个索引就可以使用相同的 型和相同的域，只是使用不同的分析器：

```
PUT /blogs-en
{
  "mappings": {
    "post": {
      "properties": {
        "title": {
          "type": "string", ①
          "fields": {
            "stemmed": {
              "type":     "string",
              "analyzer": "english" ②
            }
}}}}}

PUT /blogs-fr
{
  "mappings": {
    "post": {
      "properties": {
        "title": {
          "type": "string", ①
          "fields": {
            "stemmed": {
              "type":     "string",
              "analyzer": "french" ②
            }
}}}}}
```

① 索引 blogs-en 和 blogs-fr 的 post 型都有一个包含 title 域。

② title.stemmed 子域使用了具体 言的分析器。

个方法干 且 活。新 言很容易被添加 — 是 建一个新索引—因 言都是 底的被分 ，我 不用遭受在 混合 言的陷 中描述的 和 干提取的 。

一 言的文 都可被独立 ，或者通 多 索引来 多 言。 我 甚至可以使用 indices_boost 参数 特定的 言添加 先 ：

```
GET /blogs-*/post/_search ①
{
    "query": {
        "multi_match": {
            "query":   "deja vu",
            "fields":  [ "title", "title.stemmed" ] ②
            "type":    "most_fields"
        }
    },
    "indices_boost": { ③
        "blogs-en": 3,
        "blogs-fr": 2
    }
}
```

① 个 会在所有以 blogs- 的索引中 行。

② title.stemmed 字段使用 个索引中指定的分析器 。

③ 也 用 接受 言 表明，更 向于英 ，然后是法 ，所以相 的，我 会 个索引的 果添加 重。任何其他 言会有一个中性的 重 1 。

外

当然，有些文 含有一些其他 言的 或句子，且不幸的是 些 被切 了正 的 根。 于主 言文 ， 通常并不是主要的 。用 常需要搜索很精 的 —例如，一个其他 言的引用—而不是 型 化 的 。召回率 (Recall)可以通 使用 [token-normalization] 中 解的技 提升。

假 有些 例如地名 当能被主 言和原始 言都能 索，例如 *Munich* 和 *München* 。 些 上是我 在 [synonyms] 解 的同 。

> **不要 言使用 型**
>
> 也 很 向于 个 言使用分 的 型，来代替使用分 的索引。 了 到最佳效果， 当避免使用 型。在 [mapping] 解 ，不同 型但有相同域名的域会被索引在相同的倒排索引 中。 意味着不同 型（和不同 言）的 混合在了一起。
>
> 了 保一 言的 不会 染其他 言的 ，在后面的章 中会介 到，无 是 个 言使用 独的索引， 是使用 独的域都可以。

个域一 言

于一些 体 ，例如: 品、 影、法律声明， 通常 的一 文本会被翻 成不同 言的文 。 然 些不同 言的文 可以 独保存在各自的索引中。但 一 更合理的方式是同一 文本的所有翻 一保存在一个索引中。。

```
{
   "title":     "Fight club",
   "title_br":  "Clube de Luta",
   "title_cz":  "Klub rváčů",
   "title_en":  "Fight club",
   "title_es":  "El club de la lucha",
   ...
}
```

翻 存 在不同的域中，根据域的 言决定使用相 的分析器：

```
PUT /movies
{
  "mappings": {
    "movie": {
      "properties": {
        "title": { ①
          "type":       "string"
        },
        "title_br": { ②
            "type":     "string",
            "analyzer": "brazilian"
        },
        "title_cz": { ②
            "type":     "string",
            "analyzer": "czech"
        },
        "title_en": { ②
            "type":     "string",
            "analyzer": "english"
        },
        "title_es": { ②
            "type":     "string",
            "analyzer": "spanish"
        }
      }
    }
  }
}
```

① `title` 域含有title的原文，并使用 `standard` （ 准）分析器。

② 其他字段使用 合自己 言的分析器。

在 持干 的 方面， 然 *index-per-language* （一 言一 索引的方法），不像 *field-per-language* （一 言一个域的方法）分 索引那 活。但是使用 `update-mapping` API 添加一个新域也很 ，那些新域需要新的自定 分析器， 些新分析器只能在索引 建 被装配。有一个 通的方案， 可以先 个索引 {ref}/indices-open-close.html[close] ，然后使用 {ref}/indices-update-settings.html[`update-settings` API] ，重新打 个索引，但是 掉 个索引意味着得停止服 一段

。

文 的一 言可以 独 ，也可以通 多个域来 多 言。我 甚至可以通 特定 言 置偏好来提高字段 先 ：

```
GET /movies/movie/_search
{
    "query": {
        "multi_match": {
            "query":    "club de la lucha",
            "fields": [ "title*", "title_es^2" ], ①
            "type":     "most_fields"
        }
    }
}
```

① 个搜索 所有以 title 前 的域，但是 title_es 域加 重 2 。其他的所有域是中性 重 1 。

混合 言域

通常,那些从源数据中 得的多 言混合在一个域中的文 会超出 的控制， 例如从 上爬取的 面：

```
{ "body": "Page not found / Seite nicht gefunden / Page non trouvée" }
```

正 的 理多 言 型文 是非常困 的。即使 所有的域使用 standard （ 准）分析器， 但 的文 会 得不利于搜索，除非 使用了合 的 干提取器。当然， 不可能只 一个 干提取器。 干提取器是由 言具体决定的。或者， 干提取器是由 言和脚本所具体决定的。像在写方式一 干提取器 中那 。如果 个 言都使用不同的脚本，那 干提取器就可以合并了。

假 的混合 言使用的是一 的脚本，例如拉丁文， 有三个可用的 ：

- 切分到不同的域
- 行多次分析
- 使用 n-grams

切分到不同的域

在 言 提到 的 的 言 可以告 部分文 属于 言。 可以用 个域一 言中用 的一 的方法来根据 言切分文本。

行多次分析

如果 主要 理数量有限的 言， 可以使用多个域， 言都分析文本一次。

```
PUT /movies
{
  "mappings": {
    "title": {
      "properties": {
        "title": { ①
          "type": "string",
          "fields": {
            "de": { ②
              "type":     "string",
              "analyzer": "german"
            },
            "en": { ②
              "type":     "string",
              "analyzer": "english"
            },
            "fr": { ②
              "type":     "string",
              "analyzer": "french"
            },
            "es": { ②
              "type":     "string",
              "analyzer": "spanish"
            }
          }
        }
      }
    }
  }
}
```

① 主域 `title` 使用 `standard` （ 准）分析器

② 个子域提供不同的 言分析器来 `title` 域文本 行分析。

使用 n-grams

可以使用 [ngrams-compound-words] 中描述的方法索引所有的 n-grams。 大多数 型化包含 添加一个后 （或在一些 言中添加前 ），所以通 将 拆成 n-grams，有很大的机会匹配到相似但不完全一 的 。 个可以 合 *analyze-multiple times*（多次分析）方法 不支持的 言提供全域 取：

```
PUT /movies
{
  "settings": {
    "analysis": {...} ①
  },
  "mappings": {
    "title": {
      "properties": {
        "title": {
          "type": "string",
          "fields": {
            "de": {
              "type":     "string",
              "analyzer": "german"
            },
            "en": {
              "type":     "string",
              "analyzer": "english"
            },
            "fr": {
              "type":     "string",
              "analyzer": "french"
            },
            "es": {
              "type":     "string",
              "analyzer": "spanish"
            },
            "general": { ②
              "type":     "string",
              "analyzer": "trigrams"
            }
          }
        }
      }
    }
  }
}
```

① 在 analysis 章 ，我 按照 [ngrams-compound-words] 中描述的定 了同 的 trigrams 分析器。

② 在 title.general 域使用 trigrams 分析器索引所有的 言。

当 取所有 general 域 ， 可以使用 minimum_should_match （最少 当匹配数）来 少低 量的匹配。 或 也需要 其他字段 行 微的加 ， 与主 言域的 重要高于其他的在 general 上的域：

```
GET /movies/movie/_search
{
    "query": {
        "multi_match": {
            "query":    "club de la lucha",
            "fields": [ "title*^1.5", "title.general" ], ①
            "type":     "most_fields",
            "minimum_should_match": "75%" ②
        }
    }
}
```

① 所有 title 或 title.* 域 与了比 title.general 域 微高的加 。

② minimum_should_match（最少 当匹配数） 参数 少了低 量匹配的返回数, title.general 域尤其重要。