全文搜索

我 已 介 了搜索 化数据的 用示例， 在来探 全文搜索（*full-text search*） ：在全文字段中搜索到最相 的文 。

全文搜索 个最重要的方面是：

相 性（*Relevance*）

它是 与其 果 的相 程度，并根据 相 程度 果排名的一 能力， 算方式可以是 TF/IDF 方法（参 相 性的介 ）、地理位置 近、模糊相似，或其他的某些算法。

分析（*Analysis*）

它是将文本 有区 的、 化的 token 的一个 程，（参 分析的介 ） 目的是 了（a）建倒排索引以及（b） 倒排索引。

一旦 相 性或分析 个方面的 ，我 所 的 境是 于 的而不是 。

基于 与基于全文

所有 会或多或少的 行相 度 算，但不是所有 都有分析 段。和一些特殊的完全不会 文本 行操作的 （如 `bool` 或 `function_score` ）不同，文本 可以 分成 大家族：

基于 的

如 `term` 或 `fuzzy` 的底 不需要分析 段，它 个 行操作。用 `term` `Foo` 只要在倒排索引中 准 ，并且用 TF/IDF 算法 个包含 的文 算相 度 分 `_score` 。

住 `term` 只 倒排索引的 精 匹配， 点很重要，它不会 的多 性 行 理（如， `foo` 或 `FOO` ）。 里，无 考 是如何存入索引的。如果是将 `["Foo","Bar"]` 索引存入一个不分析的（ `not_analyzed` ）包含精 的字段，或者将 `Foo Bar` 索引到一个 有 `whitespace` 空格分析器的字段， 者的 果都会是在倒排索引中有 `Foo` 和 `Bar` 个 。

基于全文的

像 `match` 或 `query_string` 的 是高 ，它 了解字段映射的信息：

- 如果 日期（`date`）或整数（`integer`）字段，它 会将 字符串分 作 日期或整数 待。
- 如果 一个（ `not_analyzed` ）未分析的精 字符串字段，它 会将整个 字符串作 个 待。
- 但如果要 一个（ `analyzed` ）已分析的全文字段，它 会先将 字符串 到一个合 的分析器，然后生成一个供 的 列表。

一旦 成了 列表， 个 会 个 逐一 行底 的 ，再将 果合并，然后 个文 生成一个最 的相 度 分。

我 将会在随后章 中 个 程。

我 很少直接使用基于 的搜索，通常情况下都是 全文 行 ，而非 个 ， 只需要 的 行一个高 全文 （ 而在高 内部会以基于 的底 完成搜索）。

NOTE

当我 想要 一个具有精 的 not_analyzed 未分析字段之前，需要考 ，是否真的采用 分 ，或者非 分 会更好。

通常可以用是、非 二元 表示，所以更 合用 ，而且 做可以有效利用 存：

```
GET /_search
{
    "query": {
        "constant_score": {
            "filter": {
                "term": { "gender": "female" }
            }
        }
    }
}
```

匹配

匹配 match 是个 核心 。无 需要 什 字段， match 都 会是首 的 方式。它是一个高 全文 ， 表示它既能 理全文字段，又能 理精 字段。

就是 ， match 主要的 用 景就是 行全文搜索，我 以下面一个 例子来明全文搜索是如何工作的：

索引一些数据

首先，我 使用 bulk API 建一些新的文 和索引：

```
DELETE /my_index ①

PUT /my_index
{ "settings": { "number_of_shards": 1 }} ②

POST /my_index/my_type/_bulk
{ "index": { "_id": 1 }}
{ "title": "The quick brown fox" }
{ "index": { "_id": 2 }}
{ "title": "The quick brown fox jumps over the lazy dog" }
{ "index": { "_id": 3 }}
{ "title": "The quick brown fox jumps over the quick dog" }
{ "index": { "_id": 4 }}
{ "title": "Brown fox brown dog" }
```

① 除已有的索引。

② 后，我 会在 被破坏的相 性！ 中解 只 个索引分配一个主分片的原因。

个

我 用第一个示例来解 使用 match 搜索全文字段中的 个 ：

```
GET /my_index/my_type/_search
{
    "query": {
        "match": {
            "title": "QUICK!"
        }
    }
}
```

Elasticsearch 行上面 个 match 的 是：

1. 字段 型。

   title 字段是一个 string 型（ analyzed ）已分析的全文字段， 意味着 字符串本身也被分析。

2. 分析 字符串。

   将 的字符串 QUICK! 入 准分析器中， 出的 果是 个 quick 。因 只有一个 ，所以 match 行的是 个底 term 。

3. 匹配文 。

   用 term 在倒排索引中 quick 然后 取一 包含 的文 ，本例的 果是文 ：1、2 和 3 。

4. 个文 分。

   用 term 算 个文 相 度 分 _score ， 是 将 （term frequency，即 quick 在相文 的 title 字段中出 的 率）和反向文 率（inverse document frequency，即 quick 在所有文 的 title 字段中出 的 率），以及字段的 度（即字段越短相 度越高）相 合的 算方式。参 相 性的介 。

个 程 我 以下（ ） 果：

```
"hits": [
 {
    "_id":      "1",
    "_score":   0.5, ①
    "_source": {
       "title": "The quick brown fox"
    }
 },
 {
    "_id":      "3",
    "_score":   0.44194174, ②
    "_source": {
       "title": "The quick brown fox jumps over the quick dog"
    }
 },
 {
    "_id":      "2",
    "_score":   0.3125, ②
    "_source": {
       "title": "The quick brown fox jumps over the lazy dog"
    }
 }
]
```

① 文　1 最相　，因　它的 title 字段更短，即 quick 占据内容的一大部分。

② 文　3 比 文　2 更具相　性，因　在文　2 中 quick 出　了　次。

多

如果我　一次只能搜索一个　，那　全文搜索就会不太　活，幸　的是 match　　多　　得　　：

```
GET /my_index/my_type/_search
{
    "query": {
        "match": {
            "title": "BROWN DOG!"
        }
    }
}
```

上面　个　　返回所有四个文　：

```
{
  "hits": [
     {
        "_id":      "4",
        "_score":   0.73185337, ①
        "_source": {
           "title": "Brown fox brown dog"
        }
     },
     {
        "_id":      "2",
        "_score":   0.47486103, ②
        "_source": {
           "title": "The quick brown fox jumps over the lazy dog"
        }
     },
     {
        "_id":      "3",
        "_score":   0.47486103, ②
        "_source": {
           "title": "The quick brown fox jumps over the quick dog"
        }
     },
     {
        "_id":      "1",
        "_score":   0.11914785, ③
        "_source": {
           "title": "The quick brown fox"
        }
     }
  ]
}
```

① 文 4 最相 ，因 它包含 "brown" 次以及 "dog" 一次。

② 文 2、3 同 包含 brown 和 dog 各一次，而且它 title 字段的 度相同，所以具有相同的 分。

③ 文 1 也能匹配，尽管它只有 brown 没有 dog 。

因 match 必 个 （ ["brown","dog"] ），它在内部 上先 行 次 term ，然后将 次 的 果合并作 最 果 出。 了做到 点，它将 个 term 包入一个 bool 中，信息 布 。

以上示例告 我 一个重要信息：即任何文 只要 title 字段里包含 指定 中的至少一个就能匹配，被匹配的 越多，文 就越相 。

提高精度

用 任意 匹配文 可能会 致 果中出 不相 的 尾。 是 散 式搜索。可能我只想搜索包含 所有 的文 ，也就是 ，不去匹配 brown OR dog ，而通 匹配 brown AND dog 到所有文 。

match 可以接受 operator 操作符作 入参数， 情况下 操作符是 or 。我 可以将它修改成 and 所有指定 都必 匹配：

```
GET /my_index/my_type/_search
{
    "query": {
        "match": {
            "title": {      ①
                "query":    "BROWN DOG!",
                "operator": "and"
            }
        }
    }
}
```

① match 的 需要做 整才能使用 operator 操作符参数。

个 可以把文 1 排除在外，因 它只包含 个 中的一个。

控制精度

在 所有 与 任意 二 一有点 于非 即白。如果用 定 5 个 ，想 只包含其中 4 个的文 ， 如何 理？将 operator 操作符参数 置成 and 只会将此文 排除。

有 候 正是我 期望的，但在全文搜索的大多数 用 景下，我 既想包含那些可能相 的文 ，同 又排除那些不太相 的。 句 ，我 想要 于中 某 果。

match 支持 minimum_should_match 最小匹配参数， 我 可以指定必 匹配的 数用来表示一个文 是否相 。我 可以将其 置 某个具体数字，更常用的做法是将其 置 一个百分数 ，因 我 无法控制用 搜索 入的 数量：

```
GET /my_index/my_type/_search
{
  "query": {
    "match": {
      "title": {
        "query":                "quick brown dog",
        "minimum_should_match": "75%"
      }
    }
  }
}
```

当 定百分比的 候， minimum_should_match 会做合 的事情：在之前三 的示例中， 75% 会自 被截断成 66.6% ，即三个里面 个 。无 个 置成什 ，至少包含一个 的文 才会被 是匹配的。

NOTE	参数 minimum_should_match 的 置非常 活，可以根据用 入 的数目用不同的 。完整的信息参考文 {ref}/query-dsl-minimum-should-match.html#query-dsl-minimum-should-match

了完全理解 match 是如何 理多 的，我 就需要 看如何使用 bool 将多个 条件合在一起。

合

在 合 器 中，我 如何使用 bool 器通 and 、 or 和 not 合将多个 器 行合。在 中， bool 有 似的功能，只有一个重要的区 。

器做二元判断：文 是否 出 在 果中？但 更精妙，它除了决定一个文 是否 被包括在果中， 会 算文 的相 程度。

与 器一 ， bool 也可以接受 must 、 must_not 和 should 参数下的多个 句。比如：

```
GET /my_index/my_type/_search
{
  "query": {
    "bool": {
      "must":     { "match": { "title": "quick" }},
      "must_not": { "match": { "title": "lazy"  }},
      "should": [
                  { "match": { "title": "brown" }},
                  { "match": { "title": "dog"   }}
      ]
    }
  }
}
```

以上的 果返回 title 字段包含 quick 但不包含 lazy 的任意文 。目前 止， 与 bool 器的工作方式非常相似。

区 就在于 个 should 句，也就是 ：一个文 不必包含 brown 或 dog 个 ，但如果一旦包含，我 就 它 更相 ：

```
{
  "hits": [
     {
        "_id":      "3",
        "_score":   0.70134366, ①
        "_source": {
           "title": "The quick brown fox jumps over the quick dog"
        }
     },
     {
        "_id":      "1",
        "_score":   0.3312608,
        "_source": {
           "title": "The quick brown fox"
        }
     }
  ]
}
```

① 文　3 会比文　1 有更高　分是因　它同　包含 brown 和 dog 。

分　算

bool　　会　　个文　　算相　度　分 _score ，再将所有匹配的 must 和 should　句的分数 _score 求和，最后除以 must 和 should　句的　数。

must_not　句不会影　　分；它的作用只是将不相　的文　排除。

控制精度

所有　must　　句必　匹配，所有　must_not　　句都必　不匹配，但有多少　should　　句　　匹配　？　　情况下，没有 should　句是必　匹配的，只有一个例外：那就是当没有 must　句的　候，至少有一个 should　句必　匹配。

就像我　能控制 match　　的精度 一　，我　可以通　minimum_should_match 参数控制需要匹配的 should　句的数量，它既可以是一个　　的数字，又可以是个百分比：

```
GET /my_index/my_type/_search
{
  "query": {
    "bool": {
      "should": [
        { "match": { "title": "brown" }},
        { "match": { "title": "fox"   }},
        { "match": { "title": "dog"   }}
      ],
      "minimum_should_match": 2 ①
    }
  }
}
```

① 也可以用百分比表示。

个 果会将所有 足以下条件的文 返回： title 字段包含 "brown" AND "fox" 、 "brown" AND "dog" 或 "fox" AND "dog" 。如果有文 包含所有三个条件，它会比只包含 个的文 更相 。

如何使用布 匹配

目前 止，可能已 意 到多 match 只是 地将生成的 term 包 在一个 bool 中。如果使用 的 or 操作符， 个 term 都被当作 should 句， 就要求必至少匹配一条 句。以下 个 是等 的：

```
{
    "match": { "title": "brown fox"}
}
```

```
{
  "bool": {
    "should": [
      { "term": { "title": "brown" }},
      { "term": { "title": "fox"   }}
    ]
  }
}
```

如果使用 and 操作符，所有的 term 都被当作 must 句，所以 所有（*all*） 句都必 匹配。以下个 是等 的：

```
{
    "match": {
        "title": {
            "query":    "brown fox",
            "operator": "and"
        }
    }
}
```

```
{
  "bool": {
    "must": [
      { "term": { "title": "brown" }},
      { "term": { "title": "fox"   }}
    ]
  }
}
```

如果指定参数 minimum_should_match，它可以通 bool 直接 ，使以下 个 等 ：

```
{
    "match": {
        "title": {
            "query":                "quick brown fox",
            "minimum_should_match": "75%"
        }
    }
}
```

```
{
  "bool": {
    "should": [
      { "term": { "title": "brown" }},
      { "term": { "title": "fox"   }},
      { "term": { "title": "quick" }}
    ],
    "minimum_should_match": 2 ①
  }
}
```

① 因 只有三条 句，match 的参数 minimum_should_match 75% 会被截断成 2 。即三条 should 句中至少有 条必 匹配。

当然，我 通常将 些 用 match 来表示，但是如果了解 match 内部的工作原理，我 就能根据自己的需要来控制 程。有些 候 个 match 无法 足需求，比如 某些 条件分配更高的 重。我 会在下一小 中看到 个例子。

句提升　重

当然 bool　　不　限于　合　　的　个　　match　　，它可以　合任意其他的　　，以及其他　bool　　。普遍的用法是通　　多个独立　　的分数，从而　到　　个文　微　其相　度　分　　_score 的目的。

假　想要　　于　“full-text　search（全文搜索）”　的文　，但我　希望　提及　“Elasticsearch”　或　“Lucene”的文　　予更高的　　重，　里　更高　重　是指如果文　中出　　“Elasticsearch”或“Lucene”，它　会比没有的出　　些　的文　　得更高的相　度　分　　_score　　，也就是　，它　会出　在　果集的更上面。

一个　　的bool　　允　我　写出如下　　非常　　的　　：

```
GET /_search
{
    "query": {
        "bool": {
            "must": {
                "match": {
                    "content": { ①
                        "query":    "full text search",
                        "operator": "and"
                    }
                }
            },
            "should": [ ②
                { "match": { "content": "Elasticsearch" }},
                { "match": { "content": "Lucene"        }}
            ]
        }
    }
}
```

① content 字段必　包含 full 、 text 和 search 所有三个　。

② 如果 content 字段也包含 Elasticsearch 或 Lucene，文　会　得更高的　分 _score 。

should　句匹配得越多表示文　的相　度越高。目前　止　挺好。

但是如果我　想　包含 Lucene 的有更高的　重，并且包含 Elasticsearch 的　句比 Lucene 的　重更高，　如何　理?

我　可以通　指定 boost 来控制任何　　句的相　的　重，　boost 的　　1 ，大于 1 会提升一个　句的相　　重。所以下面重写之前的　　：

```
GET /_search
{
    "query": {
        "bool": {
            "must": {
                "match": {  ①
                    "content": {
                        "query":    "full text search",
                        "operator": "and"
                    }
                }
            },
            "should": [
                { "match": {
                    "content": {
                        "query": "Elasticsearch",
                        "boost": 3 ②
                    }
                }},
                { "match": {
                    "content": {
                        "query": "Lucene",
                        "boost": 2 ③
                    }
                }}
            ]
        }
    }
}
```

① 些 句使用 的 boost 1 。

② 条 句更 重要，因 它有最高的 boost 。

③ 条 句比使用 的更重要，但它的重要性不及 Elasticsearch 句。

NOTE

boost 参数被用来提升一个 句的相 重（ boost 大于 1 ）或降低相 重（ boost 于 0 到 1 之 ），但是 提升或降低并不是 性的， 句 ，如果一个 boost 2 ，并不能 得 倍的 分 _score 。

相反，新的 分 score 会在 用 重提升之后被 _ 一化 ， 型的 都有自己的 一算法， 超出了本 的 ，所以不作介 。 的 ，更高的 boost 我 来更高的 分 _score 。

如果不基于 TF/IDF 要 自己的 分模型，我 就需要 重提升的 程能有更多控制，可以使用 function_score 操 一个文 的 重提升方式而跳 一化 一 。

更多的 合 方式会在下章多字段搜索中介 ，但在此之前， 我 先看 外一个重要的 特性：文本分析（text analysis）。

控制分析

只能 倒排索引表中真 存在的 ，所以保 文 在索引 与 字符串在搜索 用相同的分析程非常重要， 的 才能 匹配倒排索引中的 。

尽管是在 文 ，不 分析器可以由 个字段决定。 个字段都可以有不同的分析器，既可以通 配置 字段指定分析器，也可以使用更高 的 型（type）、索引（index）或 点（node）的 配置。在索引 ，一个字段 是根据配置或 分析器分析的。

例如 `my_index` 新 一个字段：

```
PUT /my_index/_mapping/my_type
{
    "my_type": {
        "properties": {
            "english_title": {
                "type":     "string",
                "analyzer": "english"
            }
        }
    }
}
```

在我 就可以通 使用 `analyze` API 来分析 `Foxes` ， 而比 `english_title` 字段和 `title` 字段在索引 的分析 果：

```
GET /my_index/_analyze
{
  "field": "my_type.title",   ①
  "text": "Foxes"
}

GET /my_index/_analyze
{
  "field": "my_type.english_title",   ②
  "text": "Foxes"
}
```

① 字段 `title` ，使用 的 `standard` 准分析器，返回 `foxes` 。

② 字段 `english_title` ，使用 `english` 英 分析器，返回 `fox` 。

意味着，如果使用底 `term` 精 `fox` ，`english_title` 字段会匹配但 `title` 字段不会。

如同 `match` 的高 知道字段映射的 系，能 个被 的字段 用正 的分析器。可以使用 `validate-query` API 看 个行 ：

```
GET /my_index/my_type/_validate/query?explain
{
    "query": {
        "bool": {
            "should": [
                { "match": { "title":         "Foxes"}},
                { "match": { "english_title": "Foxes"}}
            ]
        }
    }
}
```

返回 句的 explanation 果：

```
(title:foxes english_title:fox)
```

match 个字段使用合 的分析器，以保 它在 个 都 字段使用正 的格式。

分析器

然我 可以在字段 指定分析器，但是如果 没有指定任何的分析器，那 我 如何能 定 个字段使用的是 个分析器 ？

分析器可以从三个 面 行定 ：按字段（per-field）、按索引（per-index）或全局 省（global default）。Elasticsearch 会按照以下 序依次 理，直到它 到能 使用的分析器。索引 的 序如下：

- 字段映射里定 的 analyzer ，否
- 索引 置中名 default 的分析器，
- standard 准分析器

在搜索 ， 序有些 不同：

- 自己定 的 analyzer ，否
- 字段映射里定 的 analyzer ，否
- 索引 置中名 default 的分析器，
- standard 准分析器

有 ，在索引 和搜索 使用不同的分析器是合理的。我 可能要想 同 建索引（例如，所有 quick 出 的地方，同 也 fast 、 rapid 和 speedy 建索引）。但在搜索 ，我 不需要搜索所有的同 ，取而代之的是 用 入的 是否是 quick 、 fast 、 rapid 或 speedy 。

了区分，Elasticsearch 也支持一个可 的 search_analyzer 映射，它 会 用于搜索 （ analyzer 用于索引 ）。 有一个等 的 default_search 映射，用以指定索引 的 配置。

如果考 到 些 外参数，一个搜索 的 完整 序会是下面 ：

- 自己定 的 `analyzer`，否
- 字段映射里定 的 `search_analyzer`，否
- 字段映射里定 的 `analyzer`，否
- 索引 置中名 `default_search` 的分析器，
- 索引 置中名 `default` 的分析器，
- `standard` 准分析器

分析器配置 践

就可以配置分析器地方的数量而言是十分 人的，但是 非常 。

保持

多数情况下，会提前知道文 会包括 些字段。最 的途径就是在 建索引或者 加 型映射 ， 个全文字段 置分析器。 方式尽管有点麻 ，但是它 我 可以清楚的看到 个字段 个分析器是如何置的。

通常，多数字符串字段都是 `not_analyzed` 精 字段，比如 （tag）或枚 （enum），而且更多的全文字段会使用 的 `standard` 分析器或 `english` 或其他某 言的分析器。 只需要 少数一 个字段指定自定 分析：或 `title` 字段需要以支持 入即 （*find-as-you-type*）的方式 行索引。

可以在索引 置中， 大部分的字段 置 想指定的 `default` 分析器。然后在字段置中， 某一 个字段配置需要指定的分析器。

NOTE 于和 相 的日志数据，通常的做法是 天自行 建索引，由于 方式不是从 建的索引， 然可以用 {ref}/indices-templates.html[索引模板（Index Template）] 新建的索引指定配置和映射。

被破坏的相 度！

在 更 的 多字段搜索 之前， 我 先快速解 一下 什 只在主分片上 建 索引 。

用 会 不 的抱怨无法按相 度排序并提供 短的重 ：用 索引了一些文 ， 行一个 的 ，然后 明 低相 度的 果出 在高相 度 果之上。

了理解 什 会 ，可以 想，我 在 个主分片上 建了索引和 共 10 个文 ，其中 6 个文有 `foo` 。可能是分片 1 有其中 3 个 `foo` 文 ，而分片 2 有其中 外 3 个文 ， 句 ，所有文是均 分布存 的。

在 什 是相 度？中，我 描述了 Elasticsearch 使用的相似度算法， 个算法叫做 /逆向文 率 或 TF/IDF 。 是 算某个 在当前被 文 里某个字段中出 的 率，出 的 率越高，文越相 。 逆向文 率 将 某个 在索引内所有文 出 的百分数 考 在内，出 的 率越高，它的重就越低。

但是由于性能原因， Elasticsearch 不会 算索引内所有文 的 IDF 。相反， 个分片会根据 分片内的所有文 算一个本地 IDF 。

因 文 是均 分布存 的， 个分片的 IDF 是相同的。相反， 想如果有 5 个 foo 文 存于分片 1 ，而第 6 个文 存于分片 2 ，在 景下， foo 在一个分片里非常普通（所以不那 重要），但是在一个分片里非常出 很少（所以会 得更重要）。 些 IDF 之 的差 会 致不正 的 果。

在 用中， 并不是一个 ，本地和全局的 IDF 的差 会随着索引里文 数的 多 消失，在真世界的数据量下，局部的 IDF 会被迅速均化，所以上述 并不是相 度被破坏所致的，而是由于数据太少。

了 ，我 可以通 方式解决 个 。第一 是只在主分片上 建索引，正如 match 里介的那 ，如果只有一个分片，那 本地的 IDF 就是 全局的 IDF。

第二个方式就是在搜索 求后添加 ?search_type=dfs_query_then_fetch ， dfs 是指 分布式率搜索（*Distributed Frequency Search*） ， 它告 Elasticsearch ，先分 得 个分片本地的 IDF ，然后根据 果再 算整个索引的全局 IDF 。

TIP 不要在生 境上使用 dfs_query_then_fetch 。完全没有必要。只要有足的数据就能保 是均 分布的。没有理由 个 外加上 DFS 。