多字段搜索

很少是 一句 的 match 匹配 。通常我 需要用相同或不同的字符串 一个或多个字段，也就是 ，需要 多个 句以及它 相 度 分 行合理的合并。

有 候或 我 正 作者 Leo Tolstoy 写的一本名 *War and Peace*（ 争与和平）的 。或 我 正用 “minimum should match” （最少 匹配）的方式在文 中 或 面内容 行搜索，或 我 正在搜索所有名字 John Smith 的用 。

在本章，我 会介 造多 句搜索的工具及在特定 景下 采用的解决方案。

多字符串

最 的多字段 可以将搜索 映射到具体的字段。如果我 知道 *War and Peace* 是 ，Leo Tolstoy 是作者，很容易就能把 个条件用 match 句表示，并将它 用 bool 合起来：

```
GET /_search
{
  "query": {
    "bool": {
      "should": [
        { "match": { "title":  "War and Peace" }},
        { "match": { "author": "Leo Tolstoy"   }}
      ]
    }
  }
}
```

bool 采取 *more-matches-is-better* 匹配越多越好的方式，所以 条 match 句的 分 果会被加在一起，从而 个文 提供最 的分数 _score 。能与 条 句同 匹配的文 比只与一条 句匹配的文 得分要高。

当然，并不是只能使用 match 句：可以用 bool 来包 合任意其他 型的 ，甚至包括其他的 bool 。我 可以在上面的示例中添加一条 句来指定 者版本的偏好：

```
GET /_search
{
  "query": {
    "bool": {
      "should": [
        { "match": { "title":  "War and Peace" }},
        { "match": { "author": "Leo Tolstoy"   }},
        { "bool":  {
          "should": [
            { "match": { "translator": "Constance Garnett" }},
            { "match": { "translator": "Louise Maude"      }}
          ]
        }}
      ]
    }
  }
}
```

什 将 者条件 句放入 一个独立的 bool 中 ？所有的四个 match 都是 should 句，所以什 不将 translator 句与其他如 title 、 author 的 句放在同一 ？

答案在于 分的 算方式。 bool 行 个 match ，再把 分加在一起，然后将果与所有匹配的 句数量相乘，最后除以所有的 句数量。 于同一 的 条 句具有相同的 重。在前面 个例子中，包含 translator 句的 bool ，只占 分的三分之一。如果将 translator 句与 title 和 author 条 句放入同一 ，那 title 和 author 句只 献四分之一 分。

句的 先

前例中 条 句 献三分之一 分的 方式可能并不是我 想要的，我 可能 title 和 author 条 句更感 趣， 就需要 整 ，使 title 和 author 句相 来 更重要。

在武器 中，最容易使用的就是 boost 参数。 了提升 title 和 author 字段的 重， 它 分配的 boost 大于 1 ：

```
GET /_search
{
  "query": {
    "bool": {
      "should": [
        { "match": { ①
            "title":  {
              "query": "War and Peace",
              "boost": 2
        }}},
        { "match": { ①
            "author":  {
              "query": "Leo Tolstoy",
              "boost": 2
        }}},
        { "bool":  { ②
            "should": [
              { "match": { "translator": "Constance Garnett" }},
              { "match": { "translator": "Louise Maude"      }}
            ]
        }}
      ]
    }
  }
}
```

① `title` 和 `author` 句的 `boost` 2。

② 嵌套 `bool` 句 的 `boost` 1。

要 取 `boost` 参数“最佳” ， 的方式就是不断 ： 定 `boost` ， 行 ，如此反 。 `boost` 比 合理的区 于 `1` 到 `10` 之 ，当然也有可能是 `15` 。如果 `boost` 指定比 更高的 ，将不会 最 的 分 果 生更大影 ，因 分是被 一化的（normalized） 。

字符串

`bool` 是多 句 的主干。它的 用 景很多，特 是当需要将不同 字符串映射到不同字段的 候。

在于，目前有些用 期望将所有的搜索 堆 到 个字段中，并期望 用程序能 他 提供正 的 果 。有意思的是多字段搜索的表 通常被称 高 （*Advanced Search*） —— 只是因 它 用 而言是高 的，而多字段搜索的 却非常 。

于多 （multiword）、多字段（multifield） 来 ，不存在 的 万能 方案。 了 得最好 果，需要 了解我 的数据，并了解如何使用合 的工具。

了解我 的数据

当用 入了 个字符串 的 候，通常会遇到以下三 情形：

最佳字段

当搜索 具体概念的 候，比如 “brown fox”， 比各自独立的 更有意 。像 title 和 body 的字段，尽管它 之 是相 的，但同 又彼此相互 争。文 在 相同字段 中包含的 越多越好， 分也来自于 最匹配字段 。

多数字段

了 相 度 行微 ，常用的一个技 就是将相同的数据索引到不同的字段，它 各自具有独立的分析 。

主字段可能包括它 的 源、同 以及 音 或口音 ，被用来匹配尽可能多的文 。

相同的文本被索引到其他字段，以提供更精 的匹配。一个字段可以包括未 干提取 的原 ， 一个字段包括其他 源、口音， 有一个字段可以提供 相似性 信息的瓦片 （shingles）。

其他字段是作 匹配 个文 提高相 度 分的 信号， 匹配字段越多 越好。

混合字段

于某些 体，我 需要在多个字段中 定其信息， 个字段都只能作 整体的一部分：

- Person： first_name 和 last_name （人：名和姓）
- Book： title 、 author 和 description （ ： 、作者、描述）
- Address： street 、 city 、 country 和 postcode （地址：街道、市、国家和 政 ）

在 情况下，我 希望在 任何 些列出的字段中 到尽可能多的 ， 有如在一个大字段中 行搜索， 个大字段包括了所有列出的字段。

上述所有都是多 、多字段 ，但 个具体 都要求使用不同策略。本章后面的部分，我 会依次介 个策略。

最佳字段

假 有个 站允 用 搜索博客的内容，以下面 篇博客内容文 例：

```
PUT /my_index/my_type/1
{
    "title": "Quick brown rabbits",
    "body":  "Brown rabbits are commonly seen."
}

PUT /my_index/my_type/2
{
    "title": "Keeping pets healthy",
    "body":  "My quick brown fox eats rabbits on a regular basis."
}
```

用 入 “Brown fox” 然后点 搜索按 。事先，我 并不知道用 的搜索 是会在 title 是在 body 字段中被 到，但是，用 很有可能是想搜索相 的 。用肉眼判断，文 2 的匹配度更高，因 它同 包括要 的 个 ：

在 行以下 bool ：

```
{
    "query": {
        "bool": {
            "should": [
                { "match": { "title": "Brown fox" }},
                { "match": { "body":  "Brown fox" }}
            ]
        }
    }
}
```

但是我 的 果是文 1 的 分更高：

```
{
  "hits": [
     {
        "_id":      "1",
        "_score":   0.14809652,
        "_source": {
           "title": "Quick brown rabbits",
           "body":  "Brown rabbits are commonly seen."
        }
     },
     {
        "_id":      "2",
        "_score":   0.09256032,
        "_source": {
           "title": "Keeping pets healthy",
           "body":  "My quick brown fox eats rabbits on a regular basis."
        }
     }
  ]
}
```

了理解 致 的原因，需要回想一下 bool 是如何 算 分的：

1. 它会 行 should 句中的 个 。
2. 加和 个 的 分。
3. 乘以匹配 句的 数。
4. 除以所有 句 数（ 里 ：2）。

文 1 的 个字段都包含 brown 个 ，所以 个 match 句都能成功匹配并且有一个 分。文 2 的 body 字段同 包含 brown 和 fox 个 ，但 title 字段没有包含任何 。 ， body 果中的高分，加上 title 中的 0 分，然后乘以二分之一，就得到比文 1 更低的整体 分。

在本例中， title 和 body 字段是相互 争的 系，所以就需要 到 个 最佳匹配 的字段。

如果不是 将 个字段的 分 果加在一起，而是将 最佳匹配 字段的 分作 的整体 分，果会 ？ 返回的 果可能是： 同 包含 brown 和 fox 的 个字段比反 出 相同 的多个不同字段有更高的相 度。

dis_max

不使用 bool ，可以使用 dis_max 即分 最大化 （*Disjunction Max Query*） 。分 （Disjunction）的意思是 或（*or*） ， 与可以把 合（conjunction）理解成 与（*and*） 相 。分 最大化 （Disjunction Max Query）指的是： 将任何与任一 匹配的文 作 果返回，但只将最佳匹配的 分作 的 分 果返回 ：

```
{
    "query": {
        "dis_max": {
            "queries": [
                { "match": { "title": "Brown fox" }},
                { "match": { "body":  "Brown fox" }}
            ]
        }
    }
}
```

得到我 想要的 果 ：

```
{
  "hits": [
     {
        "_id":      "2",
        "_score":   0.21509302,
        "_source": {
           "title": "Keeping pets healthy",
           "body":  "My quick brown fox eats rabbits on a regular basis."
        }
     },
     {
        "_id":      "1",
        "_score":   0.12713557,
        "_source": {
           "title": "Quick brown rabbits",
           "body":  "Brown rabbits are commonly seen."
        }
     }
  ]
}
```

最佳字段

当用 搜索 “quick pets” 会 生什 ？在前面的例子中， 个文 都包含 quick ，但是只有文 2 包含 pets ， 个文 中都不具有同 包含 个 的 相同字段 。

如下，一个 的 dis_max 会采用 个最佳匹配字段，而忽略其他的匹配：

```
{
    "query": {
        "dis_max": {
            "queries": [
                { "match": { "title": "Quick pets" }},
                { "match": { "body":  "Quick pets" }}
            ]
        }
    }
}
```

```
{
  "hits": [
     {
        "_id": "1",
        "_score": 0.12713557, ①
        "_source": {
           "title": "Quick brown rabbits",
           "body": "Brown rabbits are commonly seen."
        }
     },
     {
        "_id": "2",
        "_score": 0.12713557, ①
        "_source": {
           "title": "Keeping pets healthy",
           "body": "My quick brown fox eats rabbits on a regular basis."
        }
     }
   ]
}
```

① 注意 个 分是完全相同的。

我 可能期望同 匹配 title 和 body 字段的文 比只与一个字段匹配的文 的相 度更高，但事 并非如此，因 dis_max 只会 地使用 个 最佳匹配 句的 分 _score 作 整体 分。

tie_breaker 参数

可以通 指定 tie_breaker 个参数将其他匹配 句的 分也考 其中：

```
{
    "query": {
        "dis_max": {
            "queries": [
                { "match": { "title": "Quick pets" }},
                { "match": { "body":  "Quick pets" }}
            ],
            "tie_breaker": 0.3
        }
    }
}
```

果如下：

```
{
  "hits": [
     {
        "_id": "2",
        "_score": 0.14757764, ①
        "_source": {
           "title": "Keeping pets healthy",
           "body": "My quick brown fox eats rabbits on a regular basis."
        }
     },
     {
        "_id": "1",
        "_score": 0.124275915, ①
        "_source": {
           "title": "Quick brown rabbits",
           "body": "Brown rabbits are commonly seen."
        }
     }
   ]
}
```

① 文　2 的相　度比文　1 略高。

tie_breaker 参数提供了一　dis_max 和 bool 之　的折中　，它的　分方式如下：

1. 得最佳匹配　句的　分 _score 。
2. 将其他匹配　句的　分　果与 tie_breaker 相乘。
3. 以上　分求和并　化。

有了 tie_breaker ，会考　所有匹配　句，但最佳匹配　句依然占最　果里的很大一部分。

NOTE	tie_breaker 可以是 0 到 1 之 的浮点数，其中 0 代表使用 dis_max 最佳匹配 句的普通 ， 1 表示所有匹配 句同等重要。最佳的精 需要根据数据与 得出，但是合理 与零接近（ 于 0.1 - 0.4 之 ）， 就不会 覆 dis_max 最佳匹配性 的根本。

multi_match

multi_match 能在多个字段上反 行相同 提供了一 便捷方式。

NOTE	multi_match 多匹配 的 型有多 ，其中的三 恰巧与 了解我 的数据 中介 的三个 景 ，即： best_fields 、 most_fields 和 cross_fields （最佳字段、多数字段、跨字段）。

情况下， 的 型是 best_fields ， 表示它会 个字段生成一个 match ，然后将它 合到 dis_max 的内部，如下：

```
{
  "dis_max": {
    "queries":  [
      {
        "match": {
          "title": {
            "query": "Quick brown fox",
            "minimum_should_match": "30%"
          }
        }
      },
      {
        "match": {
          "body": {
            "query": "Quick brown fox",
            "minimum_should_match": "30%"
          }
        }
      },
    ],
    "tie_breaker": 0.3
  }
}
```

上面 个 用 multi_match 重写成更 的形式：

```
{
    "multi_match": {
        "query":                "Quick brown fox",
        "type":                 "best_fields", ①
        "fields":               [ "title", "body" ],
        "tie_breaker":          0.3,
        "minimum_should_match": "30%" ②
    }
}
```

① best_fields 型是 ，可以不指定。

② 如 minimum_should_match 或 operator 的参数会被 到生成的 match 中。

字段名称的模糊匹配

字段名称可以用模糊匹配的方式 出：任何与模糊模式正 匹配的字段都会被包括在搜索条件中，例如可以使用以下方式同 匹配 book_title 、 chapter_title 和 section_title （ 名、章名、 名）三个字段：

```
{
    "multi_match": {
        "query":  "Quick brown fox",
        "fields": "*_title"
    }
}
```

提升 个字段的 重

可以使用 ^ 字符 法 个字段提升 重，在字段名称的末尾添加 ^boost ，其中 boost 是一个浮点数：

```
{
    "multi_match": {
        "query":  "Quick brown fox",
        "fields": [ "*_title", "chapter_title^2" ] ①
    }
}
```

① chapter_title 个字段的 boost 2 ，而其他 个字段 book_title 和 section_title 字段的 boost 1 。

多数字段

全文搜索被称作是 召回率（*Recall*） 与 精 率（*Precision*） 的 ： 召回率 ——返回所有的相 文 ；精 率 ——不返回无 文 。目的是在 果的第一 中 用 呈 最 相 的文 。

了提高召回率的效果，我 大搜索 ——不 返回与用 搜索 精 匹配的文 ， 会返回我与 相 的所有文 。如果一个用 搜索 “quick brown box” ，一个包含 fast foxes 的文 被

是非常合理的返回 果。

如果包含 fast foxes 的文 是能 到的唯一相 文 ，那 它会出 在 果列表的最上面，但是，如果有 100 个文 都出 了 quick brown fox ，那 个包含 fast foxes 的文 当然会被 是次相 的，它可能 于返回 果列表更下面的某个地方。当包含了很多潜在匹配之后，我 需要将最匹配的几个置于 果列表的 部。

提高全文相 性精度的常用方式是 同一文本建立多 方式的索引， 方式都提供了一个不同的相 度信号 *signal* 。主字段会以尽可能多的形式的去匹配尽可能多的文 。 个例子，我 可以 行以下操作：

- 使用 干提取来索引 jumps 、 jumping 和 jumped 的 ，将 jump 作 它 的 根形式。 即使用 搜索 jumped ，也 是能 到包含 jumping 的匹配的文 。
- 将同 包括其中，如 jump 、 leap 和 hop 。
- 移除 音或口音 ：如 ésta 、 está 和 esta 都会以无 音形式 esta 来索引。

尽管如此，如果我 有 个文 ，其中一个包含 jumped ， 一个包含 jumping ，用 很可能期望前者能排的更高，因 它正好与 入的搜索条件一致。

了 到目的，我 可以将相同的文本索引到其他字段从而提供更 精 的匹配。一个字段可能是 干未提取 的版本， 一个字段可能是 音 的原始 ，第三个可能使用 *shingles* 提供 相似性 信息。 些附加的字段可以看成提高 个文 的相 度 分的信号 *signals* ，能匹配字段的越多越好。

一个文 如果与广度匹配的主字段相匹配，那 它会出 在 果列表中。如果文 同 又与 *signal* 信号字段匹配，那 它会 得 外加分，系 会提升它在 果列表中的位置。

我 会在本 后 同 、 相似性、部分匹配以及其他潜在的信号 行 ，但 里只使用 干已提取（stemmed）和未提取（unstemmed）的字段作 例子来 明 技 。

多字段映射

首先要做的事情就是 我 的字段索引 次：一次使用 干模式以及一次非 干模式。 了做到 点，采用 *multifields* 来 ，已 在 multifields 有所介 ：

```
DELETE /my_index

PUT /my_index
{
    "settings": { "number_of_shards": 1 }, ①
    "mappings": {
        "my_type": {
            "properties": {
                "title": { ②
                    "type":     "string",
                    "analyzer": "english",
                    "fields": {
                        "std":   { ③
                            "type":     "string",
                            "analyzer": "standard"
                        }
                    }
                }
            }
        }
    }
}
```

① 参考 被破坏的相　度.

② title 字段使用 english 英　分析器来提取　干。

③ title.std 字段使用 standard　准分析器，所以没有　干提取。

接着索引一些文　：

```
PUT /my_index/my_type/1
{ "title": "My rabbit jumps" }

PUT /my_index/my_type/2
{ "title": "Jumping jack rabbits" }
```

里用一个　match　title　字段是否包含 jumping rabbits （跳　的兔子）：

```
GET /my_index/_search
{
   "query": {
        "match": {
            "title": "jumping rabbits"
        }
    }
}
```

因　有了 english 分析器，　个　是在　以 jump 和 rabbit　个被提取　的文　。　个文　的 title

字段都同　包括　　个　，所以　个文　得到的　分也相同：

```
{
  "hits": [
     {
        "_id": "1",
        "_score": 0.42039964,
        "_source": {
           "title": "My rabbit jumps"
        }
     },
     {
        "_id": "2",
        "_score": 0.42039964,
        "_source": {
           "title": "Jumping jack rabbits"
        }
     }
  ]
}
```

如果只是　　title.std 字段，那　只有文　2 是匹配的。尽管如此，如果同　　个字段，然后使用 bool　　将　分　果　合并　，那　　个文　都是匹配的（ title 字段的作用），而且文　2 的相　度　分更高（ title.std 字段的作用）：

```
GET /my_index/_search
{
   "query": {
        "multi_match": {
            "query":  "jumping rabbits",
            "type":   "most_fields", ①
            "fields": [ "title", "title.std" ]
        }
    }
}
```

① 我　希望将所有匹配字段的　分合并起来，所以使用 most_fields　型。　　multi_match　　用 bool　将　个字段　句包在里面，而不是使用 dis_max　　。

```
{
  "hits": [
     {
        "_id": "2",
        "_score": 0.8226396, ①
        "_source": {
           "title": "Jumping jack rabbits"
        }
     },
     {
        "_id": "1",
        "_score": 0.10741998, ①
        "_source": {
           "title": "My rabbit jumps"
        }
     }
  ]
}
```

① 文　2　在的　分要比文　1 高。

用广度匹配字段 `title` 包括尽可能多的文　——以提升召回率——同　又使用字段 `title.std` 作　信号将相　度更高的文　置于　果　部。

个字段　于最　分的　献可以通　自定　`boost` 来控制。比如，使 `title` 字段更　重要，　同也降低了其他信号字段的作用：

```
GET /my_index/_search
{
   "query": {
        "multi_match": {
            "query":       "jumping rabbits",
            "type":        "most_fields",
            "fields":      [ "title^10", "title.std" ] ①
        }
    }
}
```

① `title` 字段的 `boost` 的　`10` 使它比 `title.std` 更重要。

跨字段　体搜索

在　一　普遍的搜索模式：跨字段　体搜索（cross-fields entity search）。在如 `person`、 `product` 或 `address`　（人、　品或地址）　的　体中，需要使用多个字段来唯一　它的信息。　`person` 体可能是　索引的：

```
{
    "firstname":  "Peter",
    "lastname":   "Smith"
}
```

或地址：

```
{
    "street":   "5 Poland Street",
    "city":     "London",
    "country":  "United Kingdom",
    "postcode": "W1V 3DG"
}
```

与之前描述的 多字符串 很像，但 存在着巨大的区 。在 多字符串 中，我 个字段使用不同的字符串，在本例中，我 想使用 个 字符串在多个字段中 行搜索。

我 的用 可能想搜索 “Peter Smith” 个人，或 “Poland Street W1V” 个地址， 些 出在不同的字段中，所以如果使用 dis_max 或 best_fields 去 个 最佳匹配字段 然是个 的方式。

的方式

依次 个字段并将 个字段的匹配 分 果相加，听起来真像是 bool ：

```
{
  "query": {
    "bool": {
      "should": [
        { "match": { "street":    "Poland Street W1V" }},
        { "match": { "city":      "Poland Street W1V" }},
        { "match": { "country":   "Poland Street W1V" }},
        { "match": { "postcode":  "Poland Street W1V" }}
      ]
    }
  }
}
```

个字段重 字符串会使 瞬 得冗 ，可以采用 multi_match ，将 type 置成 most_fields 然后告 Elasticsearch 合并所有匹配字段的 分：

```
{
  "query": {
    "multi_match": {
      "query":       "Poland Street W1V",
      "type":        "most_fields",
      "fields":      [ "street", "city", "country", "postcode" ]
    }
  }
}
```

most_fields 方式的

用 most_fields 方式搜索也存在某些 , 些 并不会 上 :

- 它是 多数字段匹配 任意 的，而不是在 所有字段 中 到最匹配的。
- 它不能使用 operator 或 minimum_should_match 参数来降低次相 果造成的 尾效 。
- 于 个字段是不一 的，而且它 之 的相互影 会 致不好的排序 果。

字段中心式

以上三个源于 most_fields 的 都因 它是 字段中心式（*field-centric*） 而不是 中心式（*term-centric*）的：当真正感 趣的是匹配 的 候，它 我 的是最匹配的 字段 。

> **NOTE** best_fields 型也是字段中心式的，它也存在 似的 。

首先 看 些 存在的原因，再想如何解决它 。

1 ：在多个字段中匹配相同的

回想一下 most_fields 是如何 行的：Elasticsearch 个字段生成独立的 match ，再用 bool 将他 包起来。

可以通 validate-query API 看：

```
GET /_validate/query?explain
{
  "query": {
    "multi_match": {
      "query":   "Poland Street W1V",
      "type":    "most_fields",
      "fields":  [ "street", "city", "country", "postcode" ]
    }
  }
}
```

生成 explanation 解 ：

```
(street:poland   street:street   street:w1v)
(city:poland     city:street     city:w1v)
(country:poland  country:street  country:w1v)
(postcode:poland postcode:street postcode:w1v)
```

可以 ， 个 字段都与 poland 匹配的文 要比一个字段同 匹配 poland 与 street 文 的 分高。

2 ：剪掉 尾

在 匹配精度 中，我 使用 and 操作符或 置 minimum_should_match 参数来消除 果中几乎不相的 尾，或 可以 以下方式：

```
{
    "query": {
        "multi_match": {
            "query":       "Poland Street W1V",
            "type":        "most_fields",
            "operator":    "and", ①
            "fields":      [ "street", "city", "country", "postcode" ]
        }
    }
}
```

① 所有 必 呈 。

但是 于 best_fields 或 most_fields 些参数会在 match 生成 被 入， 个 的 explanation 解 如下：

```
(+street:poland   +street:street   +street:w1v)
(+city:poland     +city:street     +city:w1v)
(+country:poland  +country:street  +country:w1v)
(+postcode:poland +postcode:street +postcode:w1v)
```

句 ，使用 and 操作符要求所有 都必 存在于 相同字段 ， 然是不 的！可能就不存在能与个 匹配的文 。

3 ：

在 什 是相 中，我 解 个 使用 TF/IDF 相似度算法 算相 度 分：

一个 在 个文 的某个字段中出 的 率越高， 个文 的相 度就越高。

逆向文 率

一个 在所有文 某个字段索引中出 的 率越高， 个 的相 度就越低。

当搜索多个字段 ，TF/IDF 会 来某些令人意外的 果。

想想用字段 first_name 和 last_name “Peter Smith” 的例子， Peter 是个平常的名 Smith 也是平常的姓， 者都具有 低的 IDF 。但当索引中有 外一个人的名字是 “Smith Williams” ， Smith 作 名来 很不平常，以致它有一个 高的 IDF ！

下面 个 的 可能会在 果中将 “Smith Williams” 置于 “Peter Smith” 之上，尽管事 上是第二个人比第一个人更 匹配。

```
{
    "query": {
        "multi_match": {
            "query":       "Peter Smith",
            "type":        "most_fields",
            "fields":      [ "*_name" ]
        }
    }
}
```

里的 是 smith 在名字段中具有高 IDF ，它会削弱 “Peter” 作 名和 “Smith” 作 姓 低 IDF 的所起作用。

解决方案

存在 些 是因 我 在 理着多个字段，如果将所有 些字段 合成 个字段， 就会消失。可以 person 文 添加 full_name 字段来解决 个 ：

```
{
    "first_name":  "Peter",
    "last_name":   "Smith",
    "full_name":   "Peter Smith"
}
```

当 full_name 字段 ：

- 具有更多匹配 的文 会比只有一个重 匹配 的文 更重要。
- minimum_should_match 和 operator 参数会像期望那 工作。
- 姓和名的逆向文 率被合并，所以 Smith 到底是作 姓 是作 名出 ，都会 得无 要。

做当然是可行的，但我 并不太喜 存 冗余数据。取而代之的是 Elasticsearch 可以提供 个解决方案——一个在索引 ，而 一个是在搜索 ——随后会 它 。

自定 _all 字段

在 all-field 字段中，我 解 _all 字段的索引方式是将所有其他字段的 作 一个大字符串索引的。然而 做并不十分 活， 了 活我 可以 人名添加一个自定 _all 字段，再 地址添加 一个 _all 字段。

Elasticsearch 在字段映射中 我 提供 copy_to 参数来 个功能：

```
PUT /my_index
{
    "mappings": {
        "person": {
            "properties": {
                "first_name": {
                    "type":     "string",
                    "copy_to":  "full_name" ①
                },
                "last_name": {
                    "type":     "string",
                    "copy_to":  "full_name" ①
                },
                "full_name": {
                    "type":     "string"
                }
            }
        }
    }
}
```

① first_name 和 last_name 字段中的 会被 制到 full_name 字段。

有了 个映射，我 可以用 first_name 来 名，用 last_name 来 姓，或者直接使用 full_name 整个姓名。

first_name 和 last_name 的映射并不影 full_name 如何被索引， full_name 将 个字段的内容 制到本地，然后根据 full_name 的映射自行索引。

WARNING

copy_to 置 multi-field无效。如果 配置映射，Elasticsearch 会 常。

什 ？多字段只是以不同方式 索引“主”字段；它 没有自己的数据源。也就是没有可供 copy_to 到 一字段的数据源。

只要 “主”字段 copy_to 就能 而易 的 到相同的效果：

```
PUT /my_index
{
    "mappings": {
        "person": {
            "properties": {
                "first_name": {
                    "type":     "string",
                    "copy_to":  "full_name", ①
                    "fields": {
                        "raw": {
                            "type": "string",
                            "index": "not_analyzed"
                        }
                    }
                },
                "full_name": {
                    "type":     "string"
                }
            }
        }
    }
}
```

① copy_to 是 “主”字段，而不是多字段的

cross-fields 跨字段

自定 all 的方式是一个好的解决方案，只需在索引文 前 其 置好映射。不 ， Elasticsearch 在搜索 提供了相 的解决方案：使用 cross_fields 型 行 multi_match 。 cross_fields 使用 中心式（term-centric）的 方式， 与 best_fields 和 most_fields 使用字段中心式（field-centric）的 方式非常不同，它将所有字段当成一个大字段，并在 _ 个字段 中 个 。

了 明字段中心式（field-centric）与 中心式（term-centric） 方式的不同，先看看以下字段中心式的 most_fields 的 explanation 解 ：

```
GET /_validate/query?explain
{
    "query": {
        "multi_match": {
            "query":       "peter smith",
            "type":        "most_fields",
            "operator":    "and", ①
            "fields":      [ "first_name", "last_name" ]
        }
    }
}
```

① 所有 都是必 的。

于匹配的文 , peter 和 smith 都必 同 出 在相同字段中，要 是 first_name 字段，要 last_name 字段：

```
(+first_name:peter +first_name:smith)
(+last_name:peter  +last_name:smith)
```

中心式 会使用以下 ：

```
+(first_name:peter last_name:peter)
+(first_name:smith last_name:smith)
```

句 , peter 和 smith 都必 出 ，但是可以出 在任意字段中。

cross_fields 型首先分析 字符串并生成一个 列表，然后它从所有字段中依次搜索 个 。 不同的搜索方式很自然的解决了 字段中心式 三个 中的二个。剩下的 是逆向文率不同。

幸 的是 cross_fields 型也能解决 个 ，通 validate-query 可以看到：

```
GET /_validate/query?explain
{
    "query": {
        "multi_match": {
            "query":       "peter smith",
            "type":        "cross_fields", ①
            "operator":    "and",
            "fields":      [ "first_name", "last_name" ]
        }
    }
}
```

① 用 cross_fields 中心式匹配。

它通　混合 不同字段逆向索引文　　率的方式解决了　　的　　：

```
+blended("peter", fields: [first_name, last_name])
+blended("smith", fields: [first_name, last_name])
```

　句　　，它会同　在 first_name 和 last_name　个字段中　　smith 的 IDF ，然后用　者的最小作　　个字段的 IDF 。　果　　上就是 smith 会被　　既是个平常的姓，也是平常的名。

NOTE

　了　　cross_fields　　以最　方式工作，所有的字段都使用相同的分析器，具有相同分析器的字段会被分　在一起作　混合字段使用。

如果包括了不同分析　的字段，它　会以　　best_fields　　的相同方式被加入到　　果中。例如：我　将　　title　　字段加到之前的　　中（假　它使用的是不同的分析器）， explanation 的解　　果如下：

```
(+title:peter +title:smith)
(
  +blended("peter", fields: [first_name, last_name])
  +blended("smith", fields: [first_name, last_name])
)
```

当在使用 minimum_should_match 和 operator 参数　，　点尤　重要。

按字段提高　重

采用 cross_fields　　与 自定　_all 字段 相比，其中一个　　就是它可以在搜索　　个字段提升　重。

　像 first_name 和 last_name 具有相同　的字段并不是必　的，但如果要用 title 和 description 字段搜索　　，可能希望　title 分配更多的　重，　同　可以使用前面介　　的 ^ 符号　法来　　：

```
GET /books/_search
{
    "query": {
        "multi_match": {
            "query":       "peter smith",
            "type":        "cross_fields",
            "fields":      [ "title^2", "description" ] ①
        }
    }
}
```

① title 字段的　重提升　　2 ， description 字段的　重提升　　1 。

自定　　字段　　是否能　　于多字段　　，取决于在多字段　　与　字段自定　　_all　　之　代　的　衡，即　　解决方案会　来更大的性能　化就　　一　。

Exact-Value 精　　字段

在　束多字段　　个　　之前，我　最后要　　的是精　　not_analyzed 未分析字段。将 not_analyzed 字段与 multi_match 中 analyzed 字段混在一起没有多大用　。

原因可以通　　看　　的 explanation 解　得到，　想将 title 字段　置成 not_analyzed ：

```
GET /_validate/query?explain
{
    "query": {
        "multi_match": {
            "query":       "peter smith",
            "type":        "cross_fields",
            "fields":      [ "title", "first_name", "last_name" ]
        }
    }
}
```

因　title 字段是未分析　的，Elasticsearch 会将 “peter smith”　个完整的字符串作　　条件来搜索！

```
title:peter smith
(
    blended("peter", fields: [first_name, last_name])
    blended("smith", fields: [first_name, last_name])
)
```

　然　个　不在 title 的倒排索引中，所以需要在 multi_match　　中避免使用 not_analyzed 字段。