化搜索

化搜索（*Structured search*）是指有探那些具有内在数据的程。比如日期、和数字都是化的：它有精的格式，我可以些格式行操作。比常的操作包括比数字或的，或判定个的大小。

文本也可以是化的。如彩色可以有散的色集合：（red）、（green）、（blue）。一个博客可能被了分布式（distributed）和搜索（search）。商站上的商品都有UPCs（通用品 Universal Product Codes）或其他的唯一，它都需要遵从格定的、化的格式。

在化中，我得到的果是非是即否，要存于集合之中，要存在集合之外。化不心文件的相度或分；它的文包括或排除理。

在上是能通的，因一个数字不能比其他数字更合存于某个相同。果只能是：存于之中，抑或反之。同，于化文本来，一个要相等，要不等。没有更似概念。

精

当行精，我会使用器（filters）。器很重要，因它行速度非常快，不会算相度（直接跳了整个分段）而且很容易被存。我会在本章后面的器存中器的性能，不在只要住：尽可能多的使用式。

term 数字

我首先来看最常用的 term，可以用它理数字（numbers）、布（Booleans）、日期（dates）以及文本（text）。

我以下面的例子始介，建并索引一些表示品的文，文里有字段 `price` 和 `productID`（`格`和`品ID`）：

```
POST /my_store/products/_bulk
{ "index": { "_id": 1 }}
{ "price" : 10, "productID" : "XHDK-A-1293-#fJ3" }
{ "index": { "_id": 2 }}
{ "price" : 20, "productID" : "KDKE-B-9947-#kL5" }
{ "index": { "_id": 3 }}
{ "price" : 30, "productID" : "JODL-X-1937-#pV7" }
{ "index": { "_id": 4 }}
{ "price" : 30, "productID" : "QQPX-R-3956-#aD8" }
```

我想要做的是具有某个格的所有品，有系数据背景的人肯定熟悉 SQL，如果我将其用 SQL 形式表，会是下面：

```
SELECT document
FROM   products
WHERE  price = 20
```

在 Elasticsearch 的 表 式（query DSL）中，我 可以使用 term 到相同的目的。 term 会 我 指定的精 。作 其本身， term 是 的。它接受一个字段名以及我 希望 的数 ：

```
{
    "term" : {
        "price" : 20
    }
}
```

通常当 一个精 的 候，我 不希望 行 分 算。只希望 文 行包括或排除的 算，所以我 会使用 constant_score 以非 分模式来 行 term 并以一作 一 分。

最 合的 果是一个 constant_score ，它包含一个 term ：

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : { ①
            "filter" : {
                "term" : { ②
                    "price" : 20
                }
            }
        }
    }
}
```

① 我 用 constant_score 将 term 化成 器

② 我 之前看到 的 term

行后， 个 所搜索到的 果与我 期望的一致：只有文 2 命中并作 果返回（因 只有 2 的格是 20 ）：

```
"hits" : [
    {
        "_index" : "my_store",
        "_type" :  "products",
        "_id" :    "2",
        "_score" : 1.0, ①
        "_source" : {
          "price" :     20,
          "productID" : "KDKE-B-9947-#kL5"
        }
    }
]
```

① 置于 filter 句内不 行 分或相 度的 算，所以所有的 果都会返回一个 分 1 。

term 文本

如本部分 始 提到 的一 ，使用 term 匹配字符串和匹配数字一 容易。如果我 想要 某个具体 UPC ID 的 品，使用 SQL 表 式会是如下 ：

```
SELECT product
FROM   products
WHERE  productID = "XHDK-A-1293-#fJ3"
```

成 表 式（query DSL），同 使用 term ，形式如下：

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "term" : {
                    "productID" : "XHDK-A-1293-#fJ3"
                }
            }
        }
    }
}
```

但 里有个小 ：我 无法 得期望的 果。 什 ？ 不在 term ，而在于索引数据的方式。如果我 使用 analyze API (分析 API)，我 可以看到 里的 UPC 被拆分成多个更小的 token ：

```
GET /my_store/_analyze
{
  "field": "productID",
  "text": "XHDK-A-1293-#fJ3"
}
```

```
{
  "tokens" : [ {
    "token" :        "xhdk",
    "start_offset" : 0,
    "end_offset" :   4,
    "type" :         "<ALPHANUM>",
    "position" :     1
  }, {
    "token" :        "a",
    "start_offset" : 5,
    "end_offset" :   6,
    "type" :         "<ALPHANUM>",
    "position" :     2
  }, {
    "token" :        "1293",
    "start_offset" : 7,
    "end_offset" :   11,
    "type" :         "<NUM>",
    "position" :     3
  }, {
    "token" :        "fj3",
    "start_offset" : 13,
    "end_offset" :   16,
    "type" :         "<ALPHANUM>",
    "position" :     4
  } ]
}
```

里有几点需要注意：

- Elasticsearch 用 4 个不同的 token 而不是 个 token 来表示 个 UPC 。
- 所有字母都是小写的。
- 失了 字符和哈希符（ # ）。

所以当我 用 term 精 XHDK-A-1293-#fJ3 的 候， 不到任何文 ，因 它并不在我的倒排索引中，正如前面呈 出的分析 果，索引里有四个 token 。

然 ID 或其他任何精 的 理方式并不是我 想要的。

了避免 ，我 需要告 Elasticsearch 字段具有精 ，要将其 置成 not_analyzed 无需分析的。 我 可以在 自定 字段映射 中 看它的用法。 了修正搜索 果，我 需要首先除旧索引（因 它的映射不再正 ）然后 建一个能正 映射的新索引：

```
DELETE /my_store ①

PUT /my_store ②
{
    "mappings" : {
        "products" : {
            "properties" : {
                "productID" : {
                    "type" : "string",
                    "index" : "not_analyzed" ③
                }
            }
        }
    }

}
```

① 除索引是必 的，因 我 不能更新已存在的映射。

② 在索引被 除后，我 可以 建新的索引并 其指定自定 映射。

③ 里我 告 Elasticsearch，我 不想 `productID` 做任何分析。

在我 可以 文 重建索引：

```
POST /my_store/products/_bulk
{ "index": { "_id": 1 }}
{ "price" : 10, "productID" : "XHDK-A-1293-#fJ3" }
{ "index": { "_id": 2 }}
{ "price" : 20, "productID" : "KDKE-B-9947-#kL5" }
{ "index": { "_id": 3 }}
{ "price" : 30, "productID" : "JODL-X-1937-#pV7" }
{ "index": { "_id": 4 }}
{ "price" : 30, "productID" : "QQPX-R-3956-#aD8" }
```

此 ， `term` 就能搜索到我 想要的 果， 我 再次搜索新索引 的数据（注意， 和 并没有 生任何改 ，改 的是数据映射的方式）：

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "term" : {
                    "productID" : "XHDK-A-1293-#fJ3"
                }
            }
        }
    }
}
```

因 productID 字段是未分析 的， term 不会 其做任何分析， 会 行精 并返回文 1 。成功！

内部 器的操作

在内部，Elasticsearch 会在 行非 分 的 行多个操作：

1. 匹配文 .

   term 在倒排索引中 XHDK-A-1293-#fJ3 然后 取包含 term 的所有文 。本例中，只有文 1 足我 要求。

2. 建 *bitset*.

   器会 建一个 *bitset* （一个包含 0 和 1 的数 ），它描述了 个文 会包含 term 。匹配文 的 志位是 1 。本例中，bitset 的 [1,0,0,0] 。在内部，它表示成一个 "roaring bitmap" ，可以同 稀疏或密集的集合 行高效 。

3. 迭代 *bitset(s)*

   一旦 个 生成了 bitsets ，Elasticsearch 就会循 迭代 bitsets 从而 到 足所有 条件的匹配文 的集合。 行 序是 式的，但一般来 先迭代稀疏的 bitset （因 它可以排除掉大量的文 ）。

4. 量使用 数.

   Elasticsearch 能 存非 分 从而 取更快的 ，但是它也会不太 明地 存一些使用 少的 西。非 分 算因 倒排索引已 足 快了，所以我 只想 存那些我 知道 在将来会被再次使用的 ，以避免 源的浪 。

   了 以上 想，Elasticsearch 会 个索引跟踪保留 使用的 史状 。如果 在最近的 256 次 中会被用到，那 它就会被 存到内存中。当 bitset 被 存后， 存会在那些低于 10,000 个文 （或少于 3% 的 索引数）的段（segment）中被忽略。 些小的段即将会消失，所以 它 分配 存是一 浪 。

情况并非如此（ 行有它的 性， 取决于 是如何重新 的，有些 式的算法是基于 代 的），理 上非 分 先于 分 行。非 分 任 旨在降低那些将 分 算

来更高成本的文 数量，从而 到快速搜索的目的。

从概念上 住非 分 算是首先 行的， 将有助于写出高效又快速的搜索 求。

合 器

前面的 个例子都是 个 器（filter）的使用方式。 在 用中，我 很有可能会 多个 或字段。比方 ， 用 Elasticsearch 来表 下面的 SQL ？

```
SELECT product
FROM   products
WHERE  (price = 20 OR productID = "XHDK-A-1293-#fJ3")
  AND  (price != 30)
```

情况下，我 需要 bool （布 ） 器。 是个 合 器（*compound filter*），它可以接受多个其他 器作 参数，并将 些 器 合成各式各 的布 （ ） 合。

布 器

一个 bool 器由三部分 成：

```
{
   "bool" : {
      "must" :     [],
      "should" :   [],
      "must_not" : [],
   }
}
```

`must`
所有的 句都必 （*must*）匹配，与 AND 等 。

`must_not`
所有的 句都不能（*must not*）匹配，与 NOT 等 。

`should`
至少有一个 句要匹配，与 OR 等 。

就 ！当我 需要多个 器 ，只 将它 置入 bool 器的不同部分即可。

NOTE

一个 bool 器的 个部分都是可 的（例如，我 可以只有一个 must 句），而且 个部分内部可以只有一个或一 器。

用 Elasticsearch 来表示本部分 始 的 SQL 例子，将 个 term 器置入 bool 器的 should 句内，再 加一个 句 理 NOT 非的条件：

```
GET /my_store/products/_search
{
   "query" : {
      "filtered" : { ①
         "filter" : {
            "bool" : {
              "should" : [
                 { "term" : {"price" : 20}}, ②
                 { "term" : {"productID" : "XHDK-A-1293-#fJ3"}} ②
              ],
              "must_not" : {
                 "term" : {"price" : 30} ③
              }
            }
         }
      }
   }
}
```

① 注意，我　然需要一个 filtered　将所有的　西包起来。

② 在 should　句　里面的　个 term　器与 bool　器是父子　系，　个 term 条件需要匹配其一。

③ 如果一个　品的　格是 30，那　它会自　被排除，因　它　于 must_not　句里面。

我　搜索的　果返回了 2 个命中　果，　个文　分　匹配了 bool　器其中的一个条件：

```
"hits" : [
    {
        "_id" :     "1",
        "_score" :  1.0,
        "_source" : {
          "price" :     10,
          "productID" : "XHDK-A-1293-#fJ3" ①
        }
    },
    {
        "_id" :     "2",
        "_score" :  1.0,
        "_source" : {
          "price" :     20, ②
          "productID" : "KDKE-B-9947-#kL5"
        }
    }
]
```

① 与 term　器中 productID = "XHDK-A-1293-#fJ3" 条件匹配

② 与 term　器中 price = 20 条件匹配

嵌套布　　器

尽管　bool　是一个　合的　　器，可以接受多个子　　器，需要注意的是　bool　　器本身　然只是一个　　器。　　意味着我　可以将一个 bool　　器置于其他 bool　　器内部，　　我　提供了任意　　布　　行　理的能力。

　于以下　个 SQL　句：

```
SELECT document
FROM   products
WHERE  productID      = "KDKE-B-9947-#kL5"
  OR (     productID = "JODL-X-1937-#pV7"
       AND price     = 30 )
```

我　将其　　成一　嵌套的 bool　　器：

```
GET /my_store/products/_search
{
   "query" : {
      "filtered" : {
         "filter" : {
            "bool" : {
              "should" : [
                { "term" : {"productID" : "KDKE-B-9947-#kL5"}}, ①
                { "bool" : { ①
                  "must" : [
                    { "term" : {"productID" : "JODL-X-1937-#pV7"}}, ②
                    { "term" : {"price" : 30}} ②
                  ]
                }}
              ]
            }
         }
      }
   }
}
```

① 因　term 和 bool　　器是兄弟　系，他　都　于外　的布　　should 的内部，返回的命中文至少　匹配其中一个　　器的条件。

②　　个　term　　句作　兄弟　系，同　　于　must　　句之中，所以返回的命中文　要必　都能同匹配　　个条件。

得到的　果有　个文　，它　各匹配 should　句中的一个条件：

```
"hits" : [
    {
        "_id" :     "2",
        "_score" :  1.0,
        "_source" : {
          "price" :     20,
          "productID" : "KDKE-B-9947-#kL5" ①
        }
    },
    {
        "_id" :     "3",
        "_score" :  1.0,
        "_source" : {
          "price" :      30, ②
          "productID" : "JODL-X-1937-#pV7" ②
        }
    }
]
```

① 个 productID 与外 的 bool 器 should 里的唯一一个 term 匹配。

② 个字段与嵌套的 bool 器 must 里的 个 term 匹配。

只是个 的例子，但足以展示布 器可以用来作 造 条件的基本 建模 。

多个精

term 于 个 非常有用，但通常我 可能想搜索多个 。 如果我 想要 格字段 $20 或 $30 的文 如何 理 ？

不需要使用多个 term ，我 只要用 个 terms （注意末尾的 *s* ）， terms 好比是 term 的 数形式（以英 名 的 数做比）。

它几乎与 term 的使用方式一模一 ，与指定 个 格不同，我 只要将 term 字段的 改 数 即可：

```
{
    "terms" : {
        "price" : [20, 30]
    }
}
```

与 term 一 ，也需要将其置入 filter 句的常量 分 中使用：

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "terms" : { ①
                    "price" : [20, 30]
                }
            }
        }
    }
}
```

① 个 terms 被置于 constant_score 中

行 果返回第二、第三和第四个文 ：

```
"hits" : [
    {
        "_id" :    "2",
        "_score" : 1.0,
        "_source" : {
          "price" :     20,
          "productID" : "KDKE-B-9947-#kL5"
        }
    },
    {
        "_id" :    "3",
        "_score" : 1.0,
        "_source" : {
          "price" :     30,
          "productID" : "JODL-X-1937-#pV7"
        }
    },
    {
        "_id":     "4",
        "_score":  1.0,
        "_source": {
           "price":     30,
           "productID": "QQPX-R-3956-#aD8"
        }
     }
]
```

包含，而不是相等

一定要了解 term 和 terms 是 包含（*contains*） 操作，而非 等 （*equals*） （判断）。 如何理解 句 ？

如果我 有一个 term（ ） 器 `{ "term" : { "tags" : "search" } }`，它会与以下 个文 同匹配：

```
{ "tags" : ["search"] }
{ "tags" : ["search", "open_source"] } ①
```

① 尽管第二个文 包含除 `search` 以外的其他 ，它 是被匹配并作 果返回。

回 一下 `term` 是如何工作的？ Elasticsearch 会在倒排索引中 包括某 term 的所有文 ，然后 造一个 bitset 。在我 的例子中，倒排索引表如下：

Token	DocIDs
`open_source`	2
`search`	1,2

当 `term` 匹配 `search` ，它直接在倒排索引中 到 并 取相 的文 ID，如倒排索引所示，里文 1 和文 2 均包含 ，所以 个文 会同 作 果返回。

NOTE

由于倒排索引表自身的特性，整个字段是否相等会 以 算，如果 定某个特定文 是否只（*only*） 包含我 想要 的 ？首先我 需要在倒排索引中 到相 的 并 取文 ID，然后再 描 倒排索引中的 行 ， 看它 是否包含其他的 terms 。

可以想象， 不 低效，而且代 高昂。正因如此， `term` 和 `terms` 是 必 包含（*must contain*）操作，而不是 必 精 相等（*must equal exactly*）。

精 相等

如果一定期望得到我 前面 的那 行 （即整个字段完全相等），最好的方式是 加并索引 一个字段，个字段用以存 字段包含 的数量，同 以上面提到的 个文 例， 在我 包括了一个 数的新字段：

```
{ "tags" : ["search"], "tag_count" : 1 }
{ "tags" : ["search", "open_source"], "tag_count" : 2 }
```

一旦 加 个用来索引 term 数目信息的字段，我 就可以 造一个 `constant_score` ，来 保果中的文 所包含的 数量与要求是一致的：

```
GET /my_index/my_type/_search
{
    "query": {
        "constant_score" : {
            "filter" : {
                 "bool" : {
                   "must" : [
                       { "term" : { "tags" : "search" } }, ①
                       { "term" : { "tag_count" : 1 } } ②
                   ]
                }
            }
        }
    }
}
```

① 所有包含 term search 的文 。

② 保文 只有一个 。

个 在只会匹配具有 个 search 的文 ，而不是任意一个包含 search 的文 。

本章到目前 止， 于数字，只介 如何 理精 。 上， 数字 行 有 会更有用。例如，我 可能想要 所有 格大于 $20 且小于 $40 美元的 品。

在 SQL 中， 可以表示 ：

```
SELECT document
FROM   products
WHERE  price BETWEEN 20 AND 40
```

Elasticsearch 有 range ，不出所料地，可以用它来 于某个 内的文 ：

```
"range" : {
    "price" : {
        "gte" : 20,
        "lte" : 40
    }
}
```

range 可同 提供包含（inclusive）和不包含（exclusive） 表 式，可供 合的 如下：

- gt: > 大于（greater than）
- lt: < 小于（less than）
- gte: >= 大于或等于（greater than or equal to）

- `lte`: ⇐ 小于或等于（less than or equal to）

下面是一个　　的例子：

```
GET /my_store/products/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "range" : {
                    "price" : {
                        "gte" : 20,
                        "lt"  : 40
                    }
                }
            }
        }
    }
}
```

如果想要　　无界（比方　>20 ），只　省略其中一　的限制：

```
"range" : {
    "price" : {
        "gt" : 20
    }
}
```

日期

`range`　　同　可以　用在日期字段上：

```
"range" : {
    "timestamp" : {
        "gt" : "2014-01-01 00:00:00",
        "lt" : "2014-01-07 00:00:00"
    }
}
```

当使用它　理日期字段　，　`range`　　支持　　日期　算（*date　math*）　　行操作，比方　，如果我想　　在　去一小　内的所有文　：

```
"range" : {
    "timestamp" : {
        "gt" : "now-1h"
    }
}
```

个 器会一直 在 去一个小 内的所有文 ， 器作 一个 滑 口（*sliding window*）来 文 。

日期 算 可以被 用到某个具体的 ，并非只能是一个像 now 的占位符。只要在某个日期后加上一个双管符号 (||) 并 跟一个日期数学表 式就能做到：

```
"range" : {
    "timestamp" : {
        "gt" : "2014-01-01 00:00:00",
        "lt" : "2014-01-01 00:00:00||+1M" ①
    }
}
```

① 早于 2014 年 1 月 1 日加 1 月（2014 年 2 月 1 日 零 ）

日期 算是 日 相 （*calendar aware*） 的，所以它不 知道 月的具体天数， 知道某年的 天数（ 年）等信息。更 的内容可以参考： {ref}/mapping-date-format.html[格式参考文] 。

字符串

`range` 同 可以 理字符串字段，字符串 可采用 字典 序（*lexicographically*） 或字母序（alphabetically）。例如，下面 些字符串是采用字典序（lexicographically）排序的：

- 5, 50, 6, B, C, a, ab, abb, abc, b

NOTE	在倒排索引中的 就是采取字典 序（lexicographically）排列的， 也是字符串 可以使用 个 序来 定的原因。

如果我 想 从 `a` 到 `b` （不包含）的字符串，同 可以使用 `range` 法：

```
"range" : {
    "title" : {
        "gte" : "a",
        "lt" :  "b"
    }
}
```

注意基数

数字和日期字段的索引方式使高效地　　算成　可能。但字符串却并非如此，要想　其使用　　，Elasticsearch　　上是在　　内的　个　　都　行　　term　　器，会比日期或数字的　　慢　多。

字符串　　在　　低基数（*low　　cardinality*）　　字段（即只有少量唯一　　）　可以正常工作，但是唯一　　越多，字符串　　的　算会越慢。

理 Null

回想在之前例子中，有的文　有名　　tags　　（　　）的字段，它是个多　字段，一个文可能有一个或多个　　，也可能根本就没有　　。如果一个字段没有　，那　如何将它存入倒排索引中的？

　是个有欺　性的　　，因　答案是：什　都不存。　我　看看之前内容里提到　的倒排索引：

Token	DocIDs
open_source	2
search	1,2

如何将某个不存在的字段存　在　个数据　　中　？无法做到！　　的　，一个倒排索引只是一个　token 列表和与之相　的文　信息，如果字段不存在，那　它也不会持有任何　　token，也就无法在倒排索引　　中表　。

最　，　也就意味着，`null`, `[]`（空数　）和 `[null]` 所有　些都是等　的，它　无法存于倒排索引中。

　然，世界并不　　，数据往往会有　失字段，或有　式的空　或空数　。　了　　些状况，Elasticsearch 提供了一些工具来　理空或　失　。

存在

第一件武器就是　　exists　　存在　　。　个　　会返回那些在指定字段有任何　的文　，　我　索引一些示例文　并用　　的例子来　明：

```
POST /my_index/posts/_bulk
{ "index": { "_id": "1"              }}
{ "tags" : ["search"]                } ①
{ "index": { "_id": "2"              }}
{ "tags" : ["search", "open_source"] } ②
{ "index": { "_id": "3"              }}
{ "other_field" : "some data"        } ③
{ "index": { "_id": "4"              }}
{ "tags" : null                      } ④
{ "index": { "_id": "5"              }}
{ "tags" : ["search", null]          } ⑤
```

① tags 字段有 1 个　。

② tags 字段有 2 个　。

③ tags 字段　失。

④ tags 字段被置　null 。

⑤ tags 字段有 1 个　和 1 个 null 。

以上文　集合中 tags 字段　　的倒排索引如下：

Token	DocIDs
open_source	2
search	1,2,5

我　的目　是　到那些被　置　　字段的文　，并不　心　　的具体内容。只要它存在于文　中即可，用 SQL 的　就是用 IS NOT NULL 非空　行　　：

```
SELECT tags
FROM   posts
WHERE  tags IS NOT NULL
```

在 Elasticsearch 中，使用 exists　　的方式如下：

```
GET /my_index/posts/_search
{
    "query" : {
        "constant_score" : {
            "filter" : {
                "exists" : { "field" : "tags" }
            }
        }
    }
}
```

个　　返回 3 个文　：

```
"hits" : [
    {
      "_id" :     "1",
      "_score" :  1.0,
      "_source" : { "tags" : ["search"] }
    },
    {
      "_id" :     "5",
      "_score" :  1.0,
      "_source" : { "tags" : ["search", null] } ①
    },
    {
      "_id" :     "2",
      "_score" :  1.0,
      "_source" : { "tags" : ["search", "open source"] }
    }
]
```

① 尽管文　5　有　null　，但它　会被命中返回。字段之所以存在，是因　有　（　search　）可以被索引，所以 null　不会　生任何影　。

而易　，只要 tags 字段存在　（term）的文　都会命中并作　果返回，只有 3 和 4　个文　被排除。

失

个 missing　本　上与 exists 恰好相反：它返回某个特定 无　字段的文　，与以下 SQL 表　的意思似：

```
SELECT tags
FROM   posts
WHERE  tags IS NULL
```

我　将前面例子中 exists　成 missing　：

```
GET /my_index/posts/_search
{
    "query" : {
        "constant_score" : {
            "filter": {
                "missing" : { "field" : "tags" }
            }
        }
    }
}
```

按照期望的那　，我　得到 3 和 4　个文　（　个文　的 tags 字段没有　）：

```
"hits" : [
    {
      "_id" :     "3",
      "_score" :  1.0,
      "_source" : { "other_field" : "some data" }
    },
    {
      "_id" :     "4",
      "_score" :  1.0,
      "_source" : { "tags" : null }
    }
]
```

当 null 的意思是 null

有 候我 需要区分一个字段是没有 ， 是它已被 式的 置成了 null 。在之前例子中，我 看到的 的行 是无法做到 点的；数据被 失了。不 幸 的是，我 可以 将 式的 null 替 成我 指定 占位符（*placeholder*）。

在 字符串（string）、数字（numeric）、布 （Boolean）或日期（date）字段指定映射 ， 同 可以 之 置 null_value 空 ，用以 理 式 null 的情况。不 即使如此， 是会将一个没有 的字段从倒排索引中排除。

当 合 的 null_value 空 的 候，需要保 以下几点：

- 它会匹配字段的 型，我 不能 一个 date 日期字段 置字符串 型的 null_value 。
- 它必 与普通 不一 ， 可以避免把 当成 null 空的情况。

象上的存在与 失

不 可以 核心 型， exists and missing 可以 理一个 象的内部字段。以下面文 例：

```
{
   "name" : {
      "first" : "John",
      "last" :  "Smith"
   }
}
```

我 不 可以 name.first 和 name.last 的存在性，也可以 name ，不 在 映射 中，如上 象的内部是个扁平的字段与 （field-value）的 ， 似下面 ：

```
{
   "name.first" : "John",
   "name.last"  : "Smith"
}
```

那 我 如何用 exists 或 missing name 字段 ？ name 字段并不真 存在于倒排索引中。

原因是当我 行下面 个 的 候：

```
{
    "exists" : { "field" : "name" }
}
```

行的是：

```
{
    "bool": {
        "should": [
            { "exists": { "field": "name.first" }},
            { "exists": { "field": "name.last" }}
        ]
    }
}
```

也就意味着，如果 first 和 last 都是空，那 name 个命名空 才会被 不存在。

于 存

在本章前面（ 器的内部操作）中，我 已 介 了 器是如何 算的。其核心 是采用一个 bitset 与 器匹配的文 。Elasticsearch 地把 些 bitset 存起来以 随后使用。一旦 存成功，bitset 可以 用 任何 已使用 的相同 器，而无需再次 算整个 器。

些 bitsets 存是“智能”的：它 以 量方式更新。当我 索引新文 ，只需将那些新文 加入已有 bitset，而不是 整个 存一遍又一遍的重 算。和系 其他部分一 ， 器是 的，我 无需担心 存 期 。

独立的 器 存

属于一个 件的 bitsets 是独立于它所属搜索 求其他部分的。 就意味着，一旦被 存，一个 可以被用作多个搜索 求。bitsets 并不依 于它所存在的 上下文。 使得 存可以加速 中 常使用的部分，从而降低 少、易 的部分所 来的消耗。

同 ，如果 个 求重用相同的非 分 ，它 存的 bitset 可以被 个搜索里的所有 例所重用。

我 看看下面例子中的 ，它 足以下任意一个条件的 子 件：

- 在收件箱中，且没有被 的

- 不在 收件箱中，但被　注重要的

```
GET /inbox/emails/_search
{
  "query": {
     "constant_score": {
        "filter": {
           "bool": {
              "should": [
                 { "bool": {
                       "must": [
                          { "term": { "folder": "inbox" }}, ①
                          { "term": { "read": false }}
                       ]
                 }},
                 { "bool": {
                       "must_not": {
                          "term": { "folder": "inbox" } ①
                       },
                       "must": {
                          "term": { "important": true }
                       }
                 }}
              ]
           }
        }
     }
  }
}
```

①　个　　器是相同的，所以会使用同一 bitset 。

尽管其中一个收件箱的条件是　must　　句，　一个是　must_not　　句，但他　　者是完全相同的。　意味着在第一个　句　行后，　　bitset　　就会被　算然后　存起来供　一个使用。当再次　行　个　　，收件箱的　个　　器已　被　存了，所以　个　句都会使用已　存的 bitset 。

　点与　　表　式（query　　DSL）的可　合性　合得很好。它易被移　到表式的任何地方，或者在同一　　中的多个位置　用。　不　能方便　　者，而且　提升性能有直接的益　。

自　　存行

在 Elasticsearch 的　早版本中，　　的行　是　存一切可以　存的　象。　也通常意味着系　　存 bitsets 太富侵略性，从而因　清理　存　来性能　力。不　如此，尽管很多　　器都很容易被　　，但本　上是慢于　存的（以及从　存中　用）。　存　些　　器的意　不大，因　可以　　地再次　行　　器。

　　一个倒排是非常快的，然后　大多数　　件却很少使用它。例如　term　　字段　"user_id" ：如果有上百万的用　，　个具体的用　　ID　　出　的概率都很小。那　　个　　器　存　　bitsets 就不是很合算，因　　存的　果很可能在重用之前就被　除了。

　　存的　　性能有着　重的影　。更　重的是，它　　者　以区分有良好表　的　存以及无用　存

。

了解决 ，Elasticsearch 会基于使用 次自 存 。如果一个非 分 在最近的 256 中被使用 （次数取决于 型），那 个 就会作 存的候 。但是，并不是所有的片段都能保 存 bitset 。只有那些文 数量超 10,000 （或超 文 数量的 3%)才会 存 bitset 。因 小的片段可以很快的 行搜索和合并， 里 存的意 不大。

一旦 存了，非 分 算的 bitset 会一直 留在 存中直到它被 除。 除 是基于 LRU 的：一旦存 了，最近最少使用的 器会被 除。