求体

 易 —query-string search— 于用命令行 行点 点（ad-hoc） 是非常有用的。然而， 了充分利用 的 大功能， 使用 求体 <code>search</code> API， 之所以称之 求体 (Full-Body Search)，因 大部分参数是通 Http 求体而非 字符串来 的。

求体 —下文 称 —不 可以 理自身的 求， 允 果 行片段 （高亮）、 所有或部分 果 行聚合分析，同 可以 出 是不是想 的建 ， 些建 可以引 使用者快速 到他想要的 果。

空

我 以最 的 search API 的形式 我 的旅程，空 将返回所有索引 （indices)中的所有文 ：

```
GET /_search
{} ①
```

① 是一个空的 求体。

只用一个 字符串， 就可以在一个、多个或者 _all 索引 （indices）和一个、多个或者所有types 中 ：

```
GET /index_2014*/type1,type2/_search
{}
```

同 可以使用 from 和 size 参数来分 ：

```
GET /_search
{
  "from": 30,
  "size": 10
}
```

一个 求体的 GET 求？

某些特定 言（特 是 JavaScript）的 HTTP 是不允 GET 求 有 求体的。事 上，一些使用者 于 GET 求可以 求体感到非常的吃 。

而事 是 个RFC文 RFC 7231— 一个 理 HTTP 和内容的文 — 并没有 定一个 有 求体的 <code>GET</code> 求 如何 理！ 果是，一些 HTTP 服 器允 子，而有一些 — 特 是一些用于 存和代理的服 器 — 不允 。

于一个 求，Elasticsearch 的工程 偏向于使用 GET 方式，因 他 得它比 POST 能更好的描述信息 索（retrieving information）的行 。然而，因 求体的 GET 求并不被广泛支持，所以 search API同 支持 POST 求：

```
POST /_search
{
  "from": 30,
  "size": 10
}
```

似的 可以 用于任何需要 求体的 GET API。

我 将在聚合 [aggregations] 章 深入介 聚合（aggregations），而 在，我 将聚焦在 。

相 于使用晦 的 字符串的方式，一个 求体的 允 我 使用 域特定 言（*query domain-specific language*）或者 Query DSL 来写 句。

表 式

表 式(Query DSL)是一 非常 活又富有表 力的 言。 Elasticsearch 使用它可以以 的 JSON 接口来展 Lucene 功能的 大部分。在 的 用中， 用它来 写 的 句。它可以使 的 句更 活、更精 、易 和易 。

要使用 表 式，只需将 句 query 参数：

```
GET /_search
{
    "query": YOUR_QUERY_HERE
}
```

空 （empty search） —`{}`— 在功能上等 于使用 <code>match_all</code> ， 正如其名字一 ，匹配所有文 ：

```
GET /_search
{
    "query": {
        "match_all": {}
    }
}
```

句的

一个 句的典型 ：

```
{
    QUERY_NAME: {
        ARGUMENT: VALUE,
        ARGUMENT: VALUE,...
    }
}
```

如果是 某个字段，那 它的 如下：

```
{
    QUERY_NAME: {
        FIELD_NAME: {
            ARGUMENT: VALUE,
            ARGUMENT: VALUE,...
        }
    }
}
```

个例子， 可以使用 `match` 句 来 `tweet` 字段中包含 `elasticsearch` 的 tweet：

```
{
    "match": {
        "tweet": "elasticsearch"
    }
}
```

完整的 求如下：

```
GET /_search
{
    "query": {
        "match": {
            "tweet": "elasticsearch"
        }
    }
}
```

合并　　句

　　句*(Query　clauses)*　就像一些　　的　合　，　些　合　可以彼此之　合并　成更　　的　　。　些　句可以是如下形式：

- 叶子　句（*Leaf clauses*）(就像 `match`　句) 被用于将　　字符串和一个字段（或者多个字段）　比。
- 　合*(Compound)*　　句　主要用于　合并其它　　句。　比如，一个 `bool`　句　允　在　需要的　候　合其它　句，无　是 `must` 匹配、 `must_not` 匹配　是 `should` 匹配，同　它可以包含不　分的　器（filters）：

```
{
    "bool": {
        "must":     { "match": { "tweet": "elasticsearch" }},
        "must_not": { "match": { "name":  "mary" }},
        "should":   { "match": { "tweet": "full text" }},
        "filter":   { "range": { "age" : { "gt" : 30 }} }
    }
}
```

一条　合　句可以合并　任何　其它　　句，包括　合　句，了解　一点是很重要的。　就意味着，　合　句之　可以互相嵌套，可以表　非常　　的　　。

例如，以下　　是　了　出信件正文包含 `business　opportunity` 的星　　件，或者在收件箱正文包含 `business opportunity` 的非　　件：

```
{
    "bool": {
        "must": { "match":   { "email": "business opportunity" }},
        "should": [
            { "match":       { "starred": true }},
            { "bool": {
                "must":      { "match": { "folder": "inbox" }},
                "must_not":  { "match": { "spam": true }}
            }}
        ],
        "minimum_should_match": 1
    }
}
```

到目前 止， 不必太在意 个例子的 ，我 会在后面 解 。最重要的是 要理解到，一条 合句可以将多条 句—叶子 句和其它 合 句—合并成一个 一的 句。

与

Elasticsearch 使用的 言（DSL） 有一套 件， 些 件可以以无限 合的方式 行搭配。套 件可以在以下 情况下使用： 情况（filtering context）和 情况（query context）。

当使用于 情况 ， 被 置成一个“不 分”或者“ ” 。即， 个 只是 的 一个 ：“ 篇文 是否匹配？”。回答也是非常的 ，yes 或者 no，二者必居其一。

- `created` 是否在 `2013` 与 `2014` 个区 ？
- `status` 字段是否包含 `published` 个 ？
- `lat_lon` 字段表示的位置是否在指定点的 `10km` 内？

当使用于 情况 ， 就 成了一个“ 分”的 。和不 分的 似，也要去判断 个文 是否匹配，同 它 需要判断 个文 匹配的有 多好（匹配程度如何）。 此 的典型用法是用于 以下文 ：

- 与 `full text search` 个 最佳匹配的文
- 包含 `run` 个 ，也能匹配 `runs`、`running`、`jog` 或者 `sprint`
- 包含 <code>quick</code>、<code>brown</code> 和 <code>fox</code> 几个 — 之 的越近，文 相 性越高
- 有 <code>lucene</code>、<code>search</code> 或者 <code>java</code> — 越多，相 性越高

一个 分 算 一个文 与此 的 相 程度，同 将 个相 程度分配 表示相 性的字段 `_score`，并且按照相 性 匹配到的文 行排序。 相 性的概念是非常 合全文搜索的情况，因 全文搜索几乎没有完全``正 "的答案。

NOTE	自 Elasticsearch 世以来， 与 （queries and filters）就独自成 Elasticsearch 的 件。但从 Elasticsearch 2.0 始， （filters）已 从技 上被排除了，同 所有的 （queries） 有 成不 分 的能力。 然而， 了明 和 ，我 用 "filter" 个 表示不 分、只 情况下的 。 可以把 "filter" 、 "filtering query" 和 "non-scoring query" 几个 相同的。 相似的，如果 独地不加任何修 地使用 "query" 个 ，我 指的是 "scoring query" 。

性能差

（Filtering queries）只是 的 包含或者排除， 就使得 算起来非常快。考 到至少有一个 （filtering query）的 果是 “稀少的”（很少匹配的文 ），并且 常使用不 分 （non-scoring queries）， 果会被 存到内存中以便快速 取，所以有各 各 的手段来 化 果。

相反， 分 （scoring queries）不 要 出匹配的文 ， 要 算 个匹配文 的相 性， 算相 性使得它 比不 分 力的多。同 ， 果并不 存。

多 倒排索引（inverted index），一个 的 分 在匹配少量文 可能与一个涵 百万文 的filter表 的一 好，甚至会更好。但是在一般情况下，一个filter 会比一个 分的query性能更 ，并且 次都表 的很 定。

（filtering）的目 是 少那些需要通 分 （scoring queries） 行 的文 。

如何 与

通常的 是，使用 （query） 句来 行 全文 搜索或者其它任何需要影 相 性得分的搜索。除此以外的情况都使用 （filters)。

最重要的

然 Elasticsearch 自 了很多的 ，但 常用到的也就那 几个。我 将在 [search-in-depth] 章 那些 的 ，接下来我 最重要的几个 行 介 。

match_all

`match_all` 的匹配所有文 。在没有指定 方式 ，它是 的 ：

```
{ "match_all": {}}
```

它 常与 filter 合使用—例如， 索收件箱里的所有 件。所有 件被 具有相同的相性，所以都将 得分 `1` 的中性 `_score`。

match

无 在任何字段上 行的是全文搜索 是精 ，`match` 是 可用的 准 。

如果 在一个全文字段上使用 match ，在 行 前，它将用正 的分析器去分析 字符串：

```
{ "match": { "tweet": "About Search" }}
```

如果在一个精 的字段上使用它，例如数字、日期、布 或者一个 not_analyzed 字符串字段，那 它将会精 匹配 定的 ：

```
{ "match": { "age":    26           }}
{ "match": { "date":   "2014-09-01" }}
{ "match": { "public": true         }}
{ "match": { "tag":    "full_text"  }}
```

TIP

于精 的 ， 可能需要使用 filter 句来取代 query，因 filter 将会被存。接下来，我 将看到一些 于 filter 的例子。

不像我 在 [search-lite] 章 介 的字符串 （query-string search）， match 不使用 似 +user_id:2 +tweet:search 的 法。它只是去 定的 。 就意味着将 字段暴露 的用 是安全的； 需要控制那些允 被 字段，不易于 出 法 常。

multi_match

multi_match 可以在多个字段上 行相同的 match ：

```
{
    "multi_match": {
        "query":    "full text search",
        "fields":   [ "title", "body" ]
    }
}
```

range

range 出那些落在指定区 内的数字或者 ：

```
{
    "range": {
        "age": {
            "gte":  20,
            "lt":   30
        }
    }
}
```

被允 的操作符如下：

gt
大于

gte
大于等于

lt
小于

lte
小于等于

term

term 被用于精 匹配， 些精 可能是数字、 、布 或者那些 not_analyzed 的字符串：

```
{ "term": { "age":    26           }}
{ "term": { "date":   "2014-09-01" }}
{ "term": { "public": true         }}
{ "term": { "tag":    "full_text"  }}
```

term 于 入的文本不 分析， 所以它将 定的 行精 。

terms

terms 和 term 一 ，但它允 指定多 行匹配。如果 个字段包含了指定 中的任何一个 ，那 个文 足条件：

```
{ "terms": { "tag": [ "search", "full_text", "nosql" ] }}
```

和 term 一 ，terms 于 入的文本不分析。它 那些精 匹配的 （包括在大小写、重音、空格等方面的差 ）。

exists 和 missing

exists 和 missing 被用于 那些指定字段中有 (exists) 或无 (missing) 的文 。 与SQL中的 IS_NULL (missing) 和 NOT IS_NULL (exists) 在本 上具有共性：

```
{
    "exists":   {
        "field":    "title"
    }
}
```

些 常用于某个字段有 的情况和某个字段 的情况。

合多

的 需求从来都没有那 ；它 需要在多个字段上 多 多 的文本，并且根据一系列的 准来 。 了 建 似的高 ， 需要一 能 将多 合成 一 的 方法。

可以用 bool 来 的需求。 将多 合在一起，成 用 自己想要的布 。它接收以下参数：

`must`
文 必 匹配 些条件才能被包含 来。

`must_not`
文 必 不匹配 些条件才能被包含 来。

`should`
如果 足 些 句中的任意 句，将 加 _score ，否 ，无任何影 。它 主要用于修正 个文 的相 性得分。

`filter`
必 匹配，但它以不 分、 模式来 行。 些 句 分没有 献，只是根据 准来排除或包含文 。

由于 是我 看到的第一个包含多个 的 ，所以有必要 一下相 性得分是如何 合的。 一个子 都独自地 算文 的相 性得分。一旦他 的得分被 算出来， bool 就将 些得分 行合并并且返回一个代表整个布 操作的得分。

下面的 用于 title 字段匹配 how to make millions 并且不被 spam 的文 。那些被 starred 或在2014之后的文 ，将比 外那些文 有更高的排名。如果 者 都 足，那 它排名将更高：

```
{
    "bool": {
        "must":     { "match": { "title": "how to make millions" }},
        "must_not": { "match": { "tag":   "spam" }},
        "should": [
            { "match": { "tag": "starred" }},
            { "range": { "date": { "gte": "2014-01-01" }}}
        ]
    }
}
```

TIP

如果没有 must 句，那 至少需要能 匹配其中的一条 should 句。但，如果存在至少一条 must 句， should 句的匹配没有要求。

加 器（filtering）的

如果我 不想因 文 的 而影 得分，可以用 filter 句来重写前面的例子：

```
{
    "bool": {
        "must":     { "match": { "title": "how to make millions" }},
        "must_not": { "match": { "tag":   "spam" }},
        "should": [
            { "match": { "tag": "starred" }}
        ],
        "filter": {
          "range": { "date": { "gte": "2014-01-01" }} ①
        }
    }
}
```

① range 已 从 should 句中移到 filter 句

通 将 range 移到 filter 句中，我 将它 成不 分的 ，将不再影 文 的相 性排名。由于它 在是一个不 分的 ，可以使用各 filter 有效的 化手段来提升性能。

所有 都可以借 方式。将 移到 bool 的 filter 句中， 它就自 的 成一个不 分的 filter 了。

如果 需要通 多个不同的 准来 的文 ，bool 本身也可以被用做不 分的 。 地将它放置到 filter 句中并在内部 建布 ：

```
{
    "bool": {
        "must":     { "match": { "title": "how to make millions" }},
        "must_not": { "match": { "tag":   "spam" }},
        "should": [
            { "match": { "tag": "starred" }}
        ],
        "filter": {
          "bool": { ①
              "must": [
                  { "range": { "date": { "gte": "2014-01-01" }}},
                  { "range": { "price": { "lte": 29.99 }}}
              ],
              "must_not": [
                  { "term": { "category": "ebooks" }}
              ]
          }
        }
    }
}
```

① 将 bool 包 在 filter 句中，我 可以在 准中 加布

通 混合布 ，我 可以在我 的 求中 活地 写 scoring 和 filtering 。

constant_score

尽管没有 bool 使用 繁，constant_score 也是 工具箱里有用的 工具。它将一个不 的常量 分 用于所有匹配的文 。它被 常用于 只需要 行一个 filter 而没有其它 （例如，分 ）的情况下。

可以使用它来取代只有 filter 句的 bool 。在性能上是完全相同的，但 于提高 性和清晰度有很大 助。

```
{
    "constant_score":   {
        "filter": {
            "term": { "category": "ebooks" } ①
        }
    }
}
```

① term 被放置在 constant_score 中， 成不 分的 filter。 方式可以用来取代只有 filter 句的 bool 。

可以 得非常的 ，尤其和不同的分析器与不同的字段映射 合 ，理解起来就有点困 了。不 validate-query API 可以用来 是否合法。

```
GET /gb/tweet/_validate/query
{
   "query": {
      "tweet" : {
         "match" : "really powerful"
      }
   }
}
```

以上 validate 求的 答告 我 个 是不合法的：

```
{
  "valid" :         false,
  "_shards" : {
    "total" :       1,
    "successful" :  1,
    "failed" :      0
  }
}
```

理解　　信息

　了　出　　不合法的原因，可以将 explain 参数 加到　　字符串中：

```
GET /gb/tweet/_validate/query?explain ①
{
   "query": {
      "tweet" : {
         "match" : "really powerful"
      }
   }
}
```

① explain 参数可以提供更多　于　　不合法的信息。

很明　，我　将　　型(match)与字段名称 (tweet)　混了：

```
{
  "valid" :     false,
  "_shards" :   { ... },
  "explanations" : [ {
    "index" :   "gb",
    "valid" :   false,
    "error" :   "org.elasticsearch.index.query.QueryParsingException:
                 [gb] No query registered for [tweet]"
  } ]
}
```

理解　　句

　于合法　　，使用 explain 参数将返回可　的描述，　　准　理解 Elasticsearch 是如何解析　的 query 是非常有用的：

```
GET /_validate/query?explain
{
   "query": {
      "match" : {
         "tweet" : "really powerful"
      }
   }
}
```

我　　的　一个 index 都会返回　　的 explanation，因　　一个 index 都有自己的映射和分析器：

```
{
  "valid" :         true,
  "_shards" :       { ... },
  "explanations" : [ {
    "index" :       "us",
    "valid" :       true,
    "explanation" : "tweet:really tweet:powerful"
  }, {
    "index" :       "gb",
    "valid" :       true,
    "explanation" : "tweet:realli tweet:power"
  } ]
}
```

从 explanation 中可以看出，匹配 really powerful 的 match 被重写 个 tweet 字段的 single-term ，一个single-term 字符串分出来的一个term。

当然， 于索引 us ， 个 term 分 是 really 和 powerful ，而 于索引 gb ，term 分 是 realli 和 power 。之所以出 个情况，是由于我 将索引 gb 中 tweet 字段的分析器修改 english 分析器。