[N3xtChen's Blog](https://n3xtchen.github.io/n3xtchen) [时间线](https://n3xtchen.github.io/n3xtchen/archive) [文章分类](https://n3xtchen.github.io/n3xtchen/categories) [标签](https://n3xtchen.github.io/n3xtchen/tags)

1. N3xt-Tech 博客

Sharing Funny Tech With You

1. 19 个很有用的 ElasticSearch 查询语句

05 July 2017

为了演示不同类型的 **ElasticSearch** 的查询，我们将使用书文档信息的集合（有以下字段：**title**（标题）, **authors**（作者）, **summary**（摘要）, **publish\_date**（发布日期）和 **num\_reviews**（浏览数））。

在这之前，首先我们应该先创建一个新的索引（index），并批量导入一些文档：

创建索引：

PUT /bookdb\_index

{ "settings": { "number\_of\_shards": 1 }}

批量上传文档：

POST /bookdb\_index/book/\_bulk

{ "index": { "\_id": 1 }}

{ "title": "Elasticsearch: The Definitive Guide", "authors": ["clinton gormley", "zachary tong"], "summary" : "A distibuted real-time search and analytics engine", "publish\_date" : "2015-02-07", "num\_reviews": 20, "publisher": "oreilly" }

{ "index": { "\_id": 2 }}

{ "title": "Taming Text: How to Find, Organize, and Manipulate It", "authors": ["grant ingersoll", "thomas morton", "drew farris"], "summary" : "organize text using approaches such as full-text search, proper name recognition, clustering, tagging, information extraction, and summarization", "publish\_date" : "2013-01-24", "num\_reviews": 12, "publisher": "manning" }

{ "index": { "\_id": 3 }}

{ "title": "Elasticsearch in Action", "authors": ["radu gheorge", "matthew lee hinman", "roy russo"], "summary" : "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms", "publish\_date" : "2015-12-03", "num\_reviews": 18, "publisher": "manning" }

{ "index": { "\_id": 4 }}

{ "title": "Solr in Action", "authors": ["trey grainger", "timothy potter"], "summary" : "Comprehensive guide to implementing a scalable search engine using Apache Solr", "publish\_date" : "2014-04-05", "num\_reviews": 23, "publisher": "manning" }

1. 栗子：
2. 1. 基本的匹配（Query）查询

有两种方式来执行一个全文匹配查询：

* 使用 **Search Lite API**，它从 url 中读取所有的查询参数
* 使用完整 **JSON** 作为请求体，这样你可以使用完整的 **Elasticsearch DSL**

下面是一个基本的匹配查询，查询任一字段包含 Guide 的记录

GET /bookdb\_index/book/\_search?q=guide

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.28168046,

"\_source": {

"title": "Elasticsearch: The Definitive Guide",

"authors": ["clinton gormley", "zachary tong"],

"summary": "A distibuted real-time search and analytics engine",

"publish\_date": "2015-02-07",

"num\_reviews": 20,

"publisher": "manning"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.24144039,

"\_source": {

"title": "Solr in Action",

"authors": ["trey grainger", "timothy potter"],

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"publish\_date": "2014-04-05",

"num\_reviews": 23,

"publisher": "manning"

}

}

]

下面是完整 Body 版本的查询，生成相同的内容：

{

"query": {

"multi\_match" : {

"query" : "guide",

"fields" : ["\_all"]

}

}

}

multi\_match 是 match 的作为在多个字段运行相同操作的一个速记法。fields 属性用来指定查询针对的字段，在这个例子中，我们想要对文档的所有字段进行匹配。两个 **API** 都允许你指定要查询的字段。例如，查询 title 字段中包含 **in Action** 的书：

GET /bookdb\_index/book/\_search?q=title:in action

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.6259885,

"\_source": {

"title": "Solr in Action",

"authors": [

"trey grainger",

"timothy potter"

],

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"publish\_date": "2014-04-05",

"num\_reviews": 23,

"publisher": "manning"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 0.5975345,

"\_source": {

"title": "Elasticsearch in Action",

"authors": [

"radu gheorge",

"matthew lee hinman",

"roy russo"

],

"summary": "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms",

"publish\_date": "2015-12-03",

"num\_reviews": 18,

"publisher": "manning"

}

}

]

然而， 完整的 **DSL** 给予你灵活创建更复杂查询和指定返回结果的能力（后面，我们会一一阐述）。在下面例子中，我们指定 size限定返回的结果条数，from 指定起始位子，\_source 指定要返回的字段，以及语法高亮

POST /bookdb\_index/book/\_search

{

"query": {

"match" : {

"title" : "in action"

}

},

"size": 2,

"from": 0,

"\_source": [ "title", "summary", "publish\_date" ],

"highlight": {

"fields" : {

"title" : {}

}

}

}

[Results]

"hits": {

"total": 2,

"max\_score": 0.9105287,

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 0.9105287,

"\_source": {

"summary": "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms",

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

},

"highlight": {

"title": [

"Elasticsearch <em>in</em> <em>Action</em>"

]

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.9105287,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"title": "Solr in Action",

"publish\_date": "2014-04-05"

},

"highlight": {

"title": [

"Solr <em>in</em> <em>Action</em>"

]

}

}

]

}

注意：对于多个词查询，match 允许指定是否使用 and 操作符来取代默认的 or 操作符。你还可以指定 mininum\_should\_match 选项来调整返回结果的相关程度。具体看后面的例子。

1. 2. 多字段（Multi-filed）查询

正如我们已经看到来的，为了根据多个字段检索（e.g. 在 title 和 summary 字段都是相同的查询字符串的结果），你可以使用 multi\_match 语句

POST /bookdb\_index/book/\_search

{

"query": {

"multi\_match" : {

"query" : "elasticsearch guide",

"fields": ["title", "summary"]

}

}

}

[Results]

"hits": {

"total": 3,

"max\_score": 0.9448582,

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.9448582,

"\_source": {

"title": "Elasticsearch: The Definitive Guide",

"authors": [

"clinton gormley",

"zachary tong"

],

"summary": "A distibuted real-time search and analytics engine",

"publish\_date": "2015-02-07",

"num\_reviews": 20,

"publisher": "manning"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 0.17312013,

"\_source": {

"title": "Elasticsearch in Action",

"authors": [

"radu gheorge",

"matthew lee hinman",

"roy russo"

],

"summary": "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms",

"publish\_date": "2015-12-03",

"num\_reviews": 18,

"publisher": "manning"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.14965448,

"\_source": {

"title": "Solr in Action",

"authors": [

"trey grainger",

"timothy potter"

],

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"publish\_date": "2014-04-05",

"num\_reviews": 23,

"publisher": "manning"

}

}

]

}

**注**：第三条被匹配，因为 guide 在 summary 字段中被找到。

1. 3. Boosting

由于我们是多个字段查询，我们可能需要提高某一个字段的分值。在下面的例子中，我们把 summary 字段的分数提高三倍，为了提升 summary 字段的重要度；因此，我们把文档 4 的相关度提高了。

POST /bookdb\_index/book/\_search

{

"query": {

"multi\_match" : {

"query" : "elasticsearch guide",

"fields": ["title", "summary^3"]

}

},

"\_source": ["title", "summary", "publish\_date"]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.31495273,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.14965448,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 0.13094766,

"\_source": {

"summary": "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms",

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

}

}

]

**注**：提升不是简简单单通过提升因子把计算分数加成。实际的 boost 值通过归一化和一些内部优化给出的。相关信息请见 [Elasticsearch guide](https://www.elastic.co/guide/en/elasticsearch/guide/current/query-time-boosting.html)

1. 4. Bool 查询

为了提供更相关或者特定的结果，AND/OR/NOT 操作符可以用来调整我们的查询。它是以 **布尔查询** 的方式来实现的。**布尔查询** 接受如下参数：

* must 等同于 AND
* must\_not 等同于 NOT
* should 等同于 OR

打比方，如果我想要查询这样类型的书：书名包含 **ElasticSearch** 或者（OR） **Solr**，并且（AND）它的作者是 **Clinton Gormley**不是（NOT）**Radu Gheorge**

POST /bookdb\_index/book/\_search

{

"query": {

"bool": {

"must": {

"bool" : { "should": [

{ "match": { "title": "Elasticsearch" }},

{ "match": { "title": "Solr" }} ] }

},

"must": { "match": { "authors": "clinton gormely" }},

"must\_not": { "match": {"authors": "radu gheorge" }}

}

}

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.3672021,

"\_source": {

"title": "Elasticsearch: The Definitive Guide",

"authors": [

"clinton gormley",

"zachary tong"

],

"summary": "A distibuted real-time search and analytics engine",

"publish\_date": "2015-02-07",

"num\_reviews": 20,

"publisher": "oreilly"

}

}

]

**注**：正如你所看到的，**布尔查询** 可以包装任何其他查询类型，包括其他布尔查询，以创建任意复杂或深度嵌套的查询。

1. 5. 模糊（Fuzzy）查询

在进行匹配和多项匹配时，可以启用模糊匹配来捕捉拼写错误，模糊度是基于原始单词的编辑距离来指定的。

POST /bookdb\_index/book/\_search

{

"query": {

"multi\_match" : {

"query" : "comprihensiv guide",

"fields": ["title", "summary"],

"fuzziness": "AUTO"

}

},

"\_source": ["title", "summary", "publish\_date"],

"size": 1

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.5961596,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

}

]

**注**：当术语长度大于 5 个字符时，AUTO 的模糊值等同于指定值 “2”。但是，80％ 拼写错误的编辑距离为 1，所以，将模糊值设置为 1 可能会提高您的整体搜索性能。更多详细信息，请参阅**Elasticsearch指南中的“排版和拼写错误”（Typos and Misspellings）**。

1. 6. 通配符（Wildcard）查询

**通配符查询** 允许你指定匹配的模式，而不是整个术语。

* ？ 匹配任何字符
* \* 匹配零个或多个字符。

例如，要查找名称以字母’t’开头的所有作者的记录：

POST /bookdb\_index/book/\_search

{

"query": {

"wildcard" : {

"authors" : "t\*"

}

},

"\_source": ["title", "authors"],

"highlight": {

"fields" : {

"authors" : {}

}

}

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 1,

"\_source": {

"title": "Elasticsearch: The Definitive Guide",

"authors": [

"clinton gormley",

"zachary tong"

]

},

"highlight": {

"authors": [

"zachary <em>tong</em>"

]

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "2",

"\_score": 1,

"\_source": {

"title": "Taming Text: How to Find, Organize, and Manipulate It",

"authors": [

"grant ingersoll",

"thomas morton",

"drew farris"

]

},

"highlight": {

"authors": [

"<em>thomas</em> morton"

]

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 1,

"\_source": {

"title": "Solr in Action",

"authors": [

"trey grainger",

"timothy potter"

]

},

"highlight": {

"authors": [

"<em>trey</em> grainger",

"<em>timothy</em> potter"

]

}

}

]

1. 7. 正则（Regexp）查询

**正则查询** 让你可以使用比 **通配符查询** 更复杂的模式进行查询：

POST /bookdb\_index/book/\_search

{

"query": {

"regexp" : {

"authors" : "t[a-z]\*y"

}

},

"\_source": ["title", "authors"],

"highlight": {

"fields" : {

"authors" : {}

}

}

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 1,

"\_source": {

"title": "Solr in Action",

"authors": [

"trey grainger",

"timothy potter"

]

},

"highlight": {

"authors": [

"<em>trey</em> grainger",

"<em>timothy</em> potter"

]

}

}

]

1. 8. 短语匹配(Match Phrase)查询

**短语匹配查询** 要求在请求字符串中的所有查询项必须都在文档中存在，文中顺序也得和请求字符串一致，且彼此相连。默认情况下，查询项之间必须紧密相连，但可以设置 slop 值来指定查询项之间可以分隔多远的距离，结果仍将被当作一次成功的匹配。

POST /bookdb\_index/book/\_search

{

"query": {

"multi\_match" : {

"query": "search engine",

"fields": ["title", "summary"],

"type": "phrase",

"slop": 3

}

},

"\_source": [ "title", "summary", "publish\_date" ]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.22327082,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.16113183,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

}

]

**注**：在上述例子中，对于非整句类型的查询，\_id 为 1 的文档一般会比 \_id 为 4 的文档得分高，结果位置也更靠前，因为它的字段长度较短，但是对于 **短语匹配类型** 查询，由于查询项之间的接近程度是一个计算因素，因此 \_id 为 4 的文档得分更高。

1. 9. 短语前缀（Match Phrase Prefix）查询

**短语前缀式查询** 能够进行 **即时搜索（search-as-you-type）** 类型的匹配，或者说提供一个查询时的初级自动补全功能，无需以任何方式准备你的数据。和 match\_phrase 查询类似，它接收slop 参数（用来调整单词顺序和不太严格的相对位置）和 max\_expansions 参数（用来限制查询项的数量，降低对资源需求的强度）。

POST /bookdb\_index/book/\_search

{

"query": {

"match\_phrase\_prefix" : {

"summary": {

"query": "search en",

"slop": 3,

"max\_expansions": 10

}

}

},

"\_source": [ "title", "summary", "publish\_date" ]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.5161346,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.37248808,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

}

]

**注**：采用 **查询时即时搜索** 具有较大的性能成本。更好的解决方案是采用 **索引时即时搜索**。更多信息，请查看 **自动补齐接口（Completion Suggester API）** 或 **边缘分词器（Edge-Ngram filters）的用法**。

1. 10. 查询字符串（Query String）

**查询字符串** 类型（**query\_string**）的查询提供了一个方法，用简洁的简写语法来执行 **多匹配查询**、 **布尔查询** 、 **提权查询**、 **模糊查询**、 **通配符查询**、 **正则查询** 和**范围查询**。下面的例子中，我们在那些作者是 **“grant ingersoll”** 或 **“tom morton”** 的某本书当中，使用查询项 **“search algorithm”** 进行一次模糊查询，搜索全部字段，但给 summary 的权重提升 2 倍。

POST /bookdb\_index/book/\_search

{

"query": {

"query\_string" : {

"query": "(saerch~1 algorithm~1) AND (grant ingersoll) OR (tom morton)",

"fields": ["\_all", "summary^2"]

}

},

"\_source": [ "title", "summary", "authors" ],

"highlight": {

"fields" : {

"summary" : {}

}

}

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "2",

"\_score": 0.14558059,

"\_source": {

"summary": "organize text using approaches such as full-text search, proper name recognition, clustering, tagging, information extraction, and summarization",

"title": "Taming Text: How to Find, Organize, and Manipulate It",

"authors": [

"grant ingersoll",

"thomas morton",

"drew farris"

]

},

"highlight": {

"summary": [

"organize text using approaches such as full-text <em>search</em>, proper name recognition, clustering, tagging, information extraction, and summarization"

]

}

}

]

1. 11. 简单查询字符串（Simple Query String）

**简单请求字符串** 类型（**simple\_query\_string**）的查询是请求**字符串类型**（**query\_string**）查询的一个版本，它更适合那种仅暴露给用户一个简单搜索框的场景；因为它用 +/\|/- 分别替换了 AND/OR/NOT，并且自动丢弃了请求中无效的部分，不会在用户出错时，抛出异常。

POST /bookdb\_index/book/\_search

{

"query": {

"simple\_query\_string" : {

"query": "(saerch~1 algorithm~1) + (grant ingersoll) | (tom morton)",

"fields": ["\_all", "summary^2"]

}

},

"\_source": [ "title", "summary", "authors" ],

"highlight": {

"fields" : {

"summary" : {}

}

}

}

1. 12. 词条（Term）/多词条（Terms）查询

以上例子均为 full-text(全文检索) 的示例。有时我们对结构化查询更感兴趣，希望得到更准确的匹配并返回结果，**词条查询** 和 **多词条查询** 可帮我们实现。在下面的例子中，我们要在索引中找到所有由 **Manning** 出版的图书。

POST /bookdb\_index/book/\_search

{

"query": {

"term" : {

"publisher": "manning"

}

},

"\_source" : ["title","publish\_date","publisher"]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "2",

"\_score": 1.2231436,

"\_source": {

"publisher": "manning",

"title": "Taming Text: How to Find, Organize, and Manipulate It",

"publish\_date": "2013-01-24"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 1.2231436,

"\_source": {

"publisher": "manning",

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 1.2231436,

"\_source": {

"publisher": "manning",

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

}

]

可使用词条关键字来指定多个词条，将搜索项用数组传入。

{

"query": {

"terms" : {

"publisher": ["oreilly", "packt"]

}

}

}

1. 13. 词条（Term）查询 - 排序（Sorted）

**词条查询** 的结果（和其他查询结果一样）可以被轻易排序，多级排序也被允许：

POST /bookdb\_index/book/\_search

{

"query": {

"term" : {

"publisher": "manning"

}

},

"\_source" : ["title","publish\_date","publisher"],

"sort": [

{ "publish\_date": {"order":"desc"}},

{ "title": { "order": "desc" }}

]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": null,

"\_source": {

"publisher": "manning",

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

},

"sort": [

1449100800000,

"in"

]

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": null,

"\_source": {

"publisher": "manning",

"title": "Solr in Action",

"publish\_date": "2014-04-05"

},

"sort": [

1396656000000,

"solr"

]

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "2",

"\_score": null,

"\_source": {

"publisher": "manning",

"title": "Taming Text: How to Find, Organize, and Manipulate It",

"publish\_date": "2013-01-24"

},

"sort": [

1358985600000,

"to"

]

}

]

1. 14. 范围查询

另一个结构化查询的例子是 **范围查询**。在这个例子中，我们要查找 2015 年出版的书。

POST /bookdb\_index/book/\_search

{

"query": {

"range" : {

"publish\_date": {

"gte": "2015-01-01",

"lte": "2015-12-31"

}

}

},

"\_source" : ["title","publish\_date","publisher"]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 1,

"\_source": {

"publisher": "oreilly",

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 1,

"\_source": {

"publisher": "manning",

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

}

}

]

**注**：**范围查询** 用于日期、数字和字符串类型的字段。

1. 15. 过滤(Filtered)查询

过滤查询允许你可以过滤查询结果。对于我们的例子中，要在标题或摘要中检索一些书，查询项为 **Elasticsearch**，但我们又想筛出那些仅有 20 个以上评论的。

POST /bookdb\_index/book/\_search

{

"query": {

"filtered": {

"query" : {

"multi\_match": {

"query": "elasticsearch",

"fields": ["title","summary"]

}

},

"filter": {

"range" : {

"num\_reviews": {

"gte": 20

}

}

}

}

},

"\_source" : ["title","summary","publisher", "num\_reviews"]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.5955761,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"publisher": "oreilly",

"num\_reviews": 20,

"title": "Elasticsearch: The Definitive Guide"

}

}

]

**注**：**过滤查询** 并不强制它作用于其上的查询必须存在。如果未指定查询，match\_all 基本上会返回索引内的全部文档。实际上，过滤只在第一次运行，以减少所需的查询面积，并且，在第一次使用后过滤会被缓存，大大提高了性能。

**更新**：**过滤查询** 将在 ElasticSearch 5 中移除，使用 **布尔查询** 替代。 下面有个例子使用 **布尔查询** 重写上面的例子：

POST /bookdb\_index/book/\_search

{

"query": {

"bool": {

"must" : {

"multi\_match": {

"query": "elasticsearch",

"fields": ["title","summary"]

}

},

"filter": {

"range" : {

"num\_reviews": {

"gte": 20

}

}

}

}

},

"\_source" : ["title","summary","publisher", "num\_reviews"]

}

在后续的例子中，我们将会把它使用在 **多重过滤** 中。

1. 16. 多重过滤（Multiple Filters）

**多重过滤** 可以结合 **布尔查询** 使用，下一个例子中，过滤查询决定只返回那些包含至少20条评论，且必须在 2015 年前出版，且由 O’Reilly 出版的结果。

POST /bookdb\_index/book/\_search

{

"query": {

"filtered": {

"query" : {

"multi\_match": {

"query": "elasticsearch",

"fields": ["title","summary"]

}

},

"filter": {

"bool": {

"must": {

"range" : { "num\_reviews": { "gte": 20 } }

},

"must\_not": {

"range" : { "publish\_date": { "lte": "2014-12-31" } }

},

"should": {

"term": { "publisher": "oreilly" }

}

}

}

}

},

"\_source" : ["title","summary","publisher", "num\_reviews", "publish\_date"]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.5955761,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"publisher": "oreilly",

"num\_reviews": 20,

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

}

]

1. 17. 作用分值: 域值（Field Value）因子

也许在某种情况下，你想把文档中的某个特定域作为计算相关性分值的一个因素，比较典型的场景是你想根据普及程度来提高一个文档的相关性。在我们的示例中，我们想把最受欢迎的书（基于评论数判断）的权重进行提高，可使用 field\_value\_factor 用以影响分值。

POST /bookdb\_index/book/\_search

{

"query": {

"function\_score": {

"query": {

"multi\_match" : {

"query" : "search engine",

"fields": ["title", "summary"]

}

},

"field\_value\_factor": {

"field" : "num\_reviews",

"modifier": "log1p",

"factor" : 2

}

}

},

"\_source": ["title", "summary", "publish\_date", "num\_reviews"]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.44831306,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"num\_reviews": 20,

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.3718407,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"num\_reviews": 23,

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 0.046479136,

"\_source": {

"summary": "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms",

"num\_reviews": 18,

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "2",

"\_score": 0.041432835,

"\_source": {

"summary": "organize text using approaches such as full-text search, proper name recognition, clustering, tagging, information extraction, and summarization",

"num\_reviews": 12,

"title": "Taming Text: How to Find, Organize, and Manipulate It",

"publish\_date": "2013-01-24"

}

}

]

**注1**: 我们可能刚运行了一个常规的 multi\_match (多匹配)查询，并对 num\_reviews 域进行了排序，这让我们失去了评估相关性分值的好处。

**注2**: 有大量的附加参数可用来调整提升原始相关性分值效果的程度，比如 modifier, factor, boost\_mode 等等，至于细节可在 **Elasticsearch** 指南中探索。

1. 18. 作用分值: 衰变（Decay）函数

假设不想使用域值做递增提升，而你有一个理想目标值，并希望用这个加权因子来对这个离你较远的目标值进行衰减。有个典型的用途是基于经纬度、价格或日期等数值域的提升。在如下的例子中，我们查找在2014年6月左右出版的，查询项是 **search engines** 的书。

POST /bookdb\_index/book/\_search

{

"query": {

"function\_score": {

"query": {

"multi\_match" : {

"query" : "search engine",

"fields": ["title", "summary"]

}

},

"functions": [

{

"exp": {

"publish\_date" : {

"origin": "2014-06-15",

"offset": "7d",

"scale" : "30d"

}

}

}

],

"boost\_mode" : "replace"

}

},

"\_source": ["title", "summary", "publish\_date", "num\_reviews"]

}

[Results]

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.27420625,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"num\_reviews": 23,

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.005920768,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"num\_reviews": 20,

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "2",

"\_score": 0.000011564,

"\_source": {

"summary": "organize text using approaches such as full-text search, proper name recognition, clustering, tagging, information extraction, and summarization",

"num\_reviews": 12,

"title": "Taming Text: How to Find, Organize, and Manipulate It",

"publish\_date": "2013-01-24"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 0.0000059171475,

"\_source": {

"summary": "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms",

"num\_reviews": 18,

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

}

}

]

1. 19. 函数分值: 脚本评分

当内置的评分函数无法满足你的需求时，还可以用 **Groovy** 脚本。在我们的例子中，想要指定一个脚本，能在决定把 num\_reviews 的因子计算多少之前，先将 publish\_date 考虑在内。因为很新的书也许不会有评论，分值不应该被惩罚。

评分脚本如下：

publish\_date = doc['publish\_date'].value

num\_reviews = doc['num\_reviews'].value

if (publish\_date > Date.parse('yyyy-MM-dd', threshold).getTime()) {

my\_score = Math.log(2.5 + num\_reviews)

} else {

my\_score = Math.log(1 + num\_reviews)

}

return my\_score

在 script\_score 参数内动态调用评分脚本：

POST /bookdb\_index/book/\_search

{

"query": {

"function\_score": {

"query": {

"multi\_match" : {

"query" : "search engine",

"fields": ["title", "summary"]

}

},

"functions": [

{

"script\_score": {

"params" : {

"threshold": "2015-07-30"

},

"script": "publish\_date = doc['publish\_date'].value; num\_reviews = doc['num\_reviews'].value; if (publish\_date > Date.parse('yyyy-MM-dd', threshold).getTime()) { return log(2.5 + num\_reviews) }; return log(1 + num\_reviews);"

}

}

]

}

},

"\_source": ["title", "summary", "publish\_date", "num\_reviews"]

}

[Results]

"hits": {

"total": 4,

"max\_score": 0.8463001,

"hits": [

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "1",

"\_score": 0.8463001,

"\_source": {

"summary": "A distibuted real-time search and analytics engine",

"num\_reviews": 20,

"title": "Elasticsearch: The Definitive Guide",

"publish\_date": "2015-02-07"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "4",

"\_score": 0.7067348,

"\_source": {

"summary": "Comprehensive guide to implementing a scalable search engine using Apache Solr",

"num\_reviews": 23,

"title": "Solr in Action",

"publish\_date": "2014-04-05"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "3",

"\_score": 0.08952084,

"\_source": {

"summary": "build scalable search applications using Elasticsearch without having to do complex low-level programming or understand advanced data science algorithms",

"num\_reviews": 18,

"title": "Elasticsearch in Action",

"publish\_date": "2015-12-03"

}

},

{

"\_index": "bookdb\_index",

"\_type": "book",

"\_id": "2",

"\_score": 0.07602123,

"\_source": {

"summary": "organize text using approaches such as full-text search, proper name recognition, clustering, tagging, information extraction, and summarization",

"num\_reviews": 12,

"title": "Taming Text: How to Find, Organize, and Manipulate It",

"publish\_date": "2013-01-24"

}

}

]

}

**注1**: 要在 **Elasticsearch** 实例中使用动态脚本，必须在 *config/elasticsearch.yaml* 文件中启用它；也可以使用存储在 **Elasticsearch** 服务器上的脚本。建议看看 **Elasticsearch** 指南文档获取更多信息。

**注2**: 因 **JSON** 不能包含嵌入式换行符，请使用分号来分割语句。

引用自：[23 USEFUL ELASTICSEARCH EXAMPLE QUERIES](http://distributedbytes.timojo.com/2016/07/23-useful-elasticsearch-example-queries.html)

* [elasticsearch 1](https://n3xtchen.github.io/n3xtchen/categories.html#elasticsearch-ref)
* [elasticsearch 1](https://n3xtchen.github.io/n3xtchen/tags.html#elasticsearch-ref)
* [← Previous](https://n3xtchen.github.io/n3xtchen/vim/2017/06/07/vim-80-ubuntu-1704)

* [Next →](https://n3xtchen.github.io/n3xtchen/bash/2017/07/14/skip-grep-use-awk)

© 2019 n3xtchen with help from [Jekyll Bootstrap](http://jekyllbootstrap.com/) and [Twitter Bootstrap](http://twitter.github.com/bootstrap/)