**5搜索**

**搜索——基本的工具**

到目前为止，我们已经学会了如何使用elasticsearch作为一个简单的NoSQL风格的分布式文件存储器——我们可以将一个JSON文档扔给Elasticsearch，也可以根据ID检索它们。但Elasticsearch真正强大之处在于可以从混乱的数据中找出有意义的信息——从大数据到全面的信息。

这也是为什么我们使用结构化的JSON文档，而不是无结构的二进制数据。Elasticsearch不只会**存储(store)**文档，也会**索引(indexes)**文档内容来使之可以被搜索。

**每个文档里的字段都会被索引并被查询**。而且不仅如此。在简单查询时，Elasticsearch可以使用**所有**的索引，以非常快的速度返回结果。这让你永远不必考虑传统数据库的一些东西。

A *search* can be: **搜索(search)**可以：

* 在类似于gender或者age这样的字段上使用结构化查询，join\_date这样的字段上使用排序，就像SQL的结构化查询一样。
* 全文检索，可以使用所有字段来匹配关键字，然后按照**关联性(relevance)**排序返回结果。
* 或者结合以上两条。

很多搜索都是开箱即用的，为了充分挖掘Elasticsearch的潜力，你需要理解以下三个概念：

| **概念** | **解释** |
| --- | --- |
| **映射(Mapping)** | 数据在每个字段中的解释说明 |
| **分析(Analysis)** | 全文是如何处理的可以被搜索的 |
| **领域特定语言查询(Query DSL)** | Elasticsearch使用的灵活的、强大的查询语言 |

以上提到的每个点都是一个巨大的话题，我们将在《深入搜索》一章阐述它们。本章节我们将介绍这三点的一些基本概念——仅仅帮助你大致了解搜索是如何工作的。

我们将使用最简单的形式开始介绍search API.

**测试数据**

本章节测试用的数据可以在这里被找到<https://gist.github.com/clintongormley/8579281>

你可以把这些命令复制到终端中执行以便可以实践本章的例子。

# 5.1空搜索

最基本的搜索API表单是**空搜索(empty search)**，它没有指定任何的查询条件，只返回集群索引中的所有文档：

GET /\_search

响应内容（为了编辑简洁）类似于这样：

{

"hits" : {

"total" : 14,

"hits" : [

{

"\_index": "us",

"\_type": "tweet",

"\_id": "7",

"\_score": 1,

"\_source": {

"date": "2014-09-17",

"name": "John Smith",

"tweet": "The Query DSL is really powerful and flexible",

"user\_id": 2

}

},

... 9 RESULTS REMOVED ...

],

"max\_score" :1

},

"took" : 4,

"\_shards" : {

"failed" : 0,

"successful" : 10,

"total" : 10

},

"timed\_out" :false

}

## hits

响应中最重要的部分是hits，它包含了total字段来表示匹配到的文档总数，hits数组还包含了匹配到的前10条数据全文搜索自带了前10条匹配到的数据。

hits数组中的每个结果都包含\_index、\_type和文档的\_id字段，被加入到\_source字段中这意味着在搜索结果中我们将可以直接使用全部文档。这不像其他搜索引擎只返回文档ID，需要你单独去获取文档。

每个节点都有一个\_score字段，这是**相关性得分(relevance score)**，文档按照此字段进行降序排列，\_score不是文档自身具备的，它是搜索过程中根据相关性es计算得来的它衡量了文档与查询的匹配程度。默认的，返回的结果中关联性最大的文档排在首位；这意味着，它是按照\_score降序排列的。这种情况下，我们没有指定任何查询，所以所有文档的相关性是一样的，因此所有结果的\_score都是取得一个中间值1

max\_score指的是所有文档匹配查询中\_score的最大值。

## took

took告诉我们整个搜索请求花费的毫秒数。

## shards

\_shards节点告诉我们参与查询的分片数（total字段），有多少是成功的（successful字段），有多少的是失败的（failed字段）这里将主分片和复制分片看做是一个，都失败的话failed增加1，表明主分片和复制分片都没有命中数据。通常我们不希望分片失败，不过这个有可能发生。如果我们遭受一些重大的故障导致主分片和复制分片都故障，那这个分片的数据将无法响应给搜索请求。这种情况下，Elasticsearch将报告分片failed，但仍将继续返回剩余分片上的结果。

## timeout

time\_out值告诉我们查询超时与否。一般的，搜索请求不会超时。如果响应速度比完整的结果更重要，你可以定义timeout参数为10或者10ms（10毫秒），或者1s（1秒）

GET /\_search?timeout=10ms

Elasticsearch将返回在请求超时前收集到的结果。

超时不是一个断路器（circuit breaker）（译者注：关于断路器的理解请看警告）。timeout只是告诉查询请求的节点目前返回所有收集到的数据并关闭与会其他的节点之间的连接，其他的连接可能仍然在查询。

## 警告

需要注意的是timeout不会停止执行查询，它仅仅告诉你**目前**顺利返回结果的节点然后关闭连接。在后台，其他分片可能依旧执行查询，尽管结果已经被发送。

使用超时是因为对于你的业务需求（译者注：SLA，Service-Level Agreement服务等级协议，在此我翻译为业务需求）来说非常重要，而不是因为你想中断执行长时间运行的查询。

## 5.2多索引和多类别

你注意到空搜索的结果中不同类型的文档——user和tweet——来自于不同的索引——us和gb。

通过限制搜索的不同索引或类型，我们可以在集群中跨**所有**文档搜索。Elasticsearch转发搜索请求到集群中平行的主分片或每个分片的复制分片上，收集结果后选择顶部十个返回给我们，这就是es搜索的流程。

通常，当然，你可能想搜索一个或几个自定的索引或类型，我们能通过定义URL中的索引或类型达到这个目的，像这样：

/\_search

在所有索引的所有类型中搜索

/gb/\_search

在索引gb的所有类型中搜索

/gb,us/\_search

在索引gb和us的所有类型中搜索

/g\*,u\*/\_search

在以g或u开头的索引的所有类型中搜索

/gb/user/\_search

在索引gb的类型user中搜索

/gb,us/user,tweet/\_search

在索引gb和us的类型为user和tweet中搜索

/\_all/user,tweet/\_search

在所有索引的user和tweet中搜索-search types user and tweet in all indices

当你搜索包含单一索引时，Elasticsearch转发搜索请求到这个索引的主分片或每个分片的复制分片上，然后聚集每个分片的结果。搜索包含多个索引也是同样的方式——只不过或有更多的分片被关联。

## 重要

搜索一个索引有5个主分片和5个索引各有一个分片**事实上是一样的（n:都是要到5个分片去查询数据病返回请求节点）**

接下来，你将看到这些简单的情况如何灵活的扩展以适应你需求的变更。

**5.3分页**

《空搜索》一节告诉我们在集群中有14个文档匹配我们的（空）搜索语句。但是只有10个文档在hits数组中。我们如何看到其他文档？

和SQL使用LIMIT关键字返回只有一页的结果一样，Elasticsearch接受from和size参数n:作为url的Get参数：

size: 结果数，默认10

from: 跳过开始的结果数，默认0

如果你想每页显示5个结果，页码从1到3，那请求如下：

GET /\_search?size=5

GET /\_search?size=5&from=5

GET /\_search?size=5&from=10

应该当心分页太深或者一次请求太多的结果。结果在返回前会被排序。但是记住一个搜索请求常常涉及多个分片。每个分片生成自己排好序的结果，它们接着需要集中起来排序以确保整体排序正确。

**在集群系统中深度分页**

为了理解为什么深度分页是有问题的，让我们假设在一个有5个主分片的索引中搜索。当我们请求结果的第一页（结果1到10）时，每个分片产生自己最顶端10个结果然后返回它们给**请求节点(requesting node)**，它再排序这所有的50个结果以选出顶端的10个结果。

现在假设我们请求第1000页——结果10001到10010。工作方式都相同，不同的是每个分片都必须产生顶端的10010个结果。然后请求节点排序这50050个结果并丢弃50040个！

你可以看到在分布式系统中，排序结果的花费随着分页的深入而成倍增长。这也是为什么网络搜索引擎中任何语句不能返回多于1000个结果的原因n:因为分布式决定了它必须在每个节点上分别获取到足够多的数据并排序以便汇总成整体再排序，最后还要丢弃很大量的数据。

**TIP**

在《重建索引》章节我们将阐述如何能高效的检索大量文档

**5.4简易搜索**

search API有两种表单，（n:一种定Get方法，一种对应Post方法）：一种是“简易版”的**查询字符串(query string)**将所有参数通过查询字符串定义，另一种版本使用JSON完整的表示**请求体(request body)**，这种富搜索语言叫做结构化查询语句（DSL）

查询字符串搜索对于在命令行下运行**点对点(ad hoc)**查询特别有用。例如这个语句查询所有类型为tweet并在tweet字段中包含elasticsearch字符的文档：

GET /\_all/tweet/\_search?q=tweet:elasticsearch

下一个语句查找name字段中包含"john"和tweet字段包含"mary"的结果。实际的查询只需要：

+name:john +tweet:mary

但是**百分比编码(percent encoding)**（译者注：就是url编码）需要将查询字符串参数变得更加神秘：

GET /\_search?q=%2Bname%3Ajohn+%2Btweet%3Amary

"+"前缀表示语句匹配条件**必须**被满足。类似的"-"前缀表示条件**必须不**被满足。所有条件如果没有+或-表示是可选的——匹配越多，相关的文档就越多。

**\_all字段**

返回包含"mary"字符的所有文档的简单搜索：

GET /\_search?q=mary

在前一个例子中，我们搜索tweet或name字段中包含某个字符的结果。然而，这个语句返回的结果在三个不同的字段中包含"mary"：

* 用户的名字是“Mary”
* “Mary”发的六个推文
* 针对“@mary”的一个推文

Elasticsearch是如何设法找到三个不同字段的结果的？

当你索引一个文档，Elasticsearch把所有字符串字段值连接起来放在一个大字符串中，它被索引为一个特殊的字段\_all。例如，当索引这个文档：

{

"tweet": "However did I manage before Elasticsearch?",

"date": "2014-09-14",

"name": "Mary Jones",

"user\_id": 1

}

这好比我们增加了一个叫做\_all的额外字段值：

"However did I manage before Elasticsearch? 2014-09-14 Mary Jones 1"

若没有指定字段，查询字符串搜索（即q=xxx）使用\_all字段搜索。

**TIP**

\_all字段对于开始一个新应用时是一个有用的特性。之后，如果你定义字段来代替\_all字段，你的搜索结果将更加可控。当\_all字段不再使用，你可以停用它，这个会在《全字段》章节阐述。

**更复杂的语句**

下一个搜索推特的语句：

\_all field

* name字段包含"mary"或"john"
* date晚于2014-09-10
* \_all字段包含"aggregations"或"geo"

+name:(mary john) +date:>2014-09-10 +(aggregations geo)

编码后的查询字符串变得不太容易阅读：

?q=%2Bname%3A(mary+john)+%2Bdate%3A%3E2014-09-10+%2B(aggregations+geo)

就像你上面看到的例子，**简单(lite)**查询字符串搜索惊人的强大。它的查询语法，会在《查询字符串语法》章节阐述。参考文档允许我们简洁明快的表示复杂的查询。这对于命令行下一次性查询或者开发模式下非常有用。

然而，你可以看到简洁带来了隐晦和调试困难。而且它很脆弱——查询字符串中一个细小的语法错误，像-、:、/或"错位就会导致返回错误而不是结果。

最后，查询字符串搜索允许任意用户在索引中任何一个字段上运行潜在的慢查询语句，可能暴露私有信息甚至使你的集群瘫痪。

**TIP**

因为这些原因，我们不建议直接暴露查询字符串搜索给用户，除非这些用户对于你的数据和集群可信。

取而代之的，生产环境我们一般依赖全功能的**请求体**搜索API，它能完成前面所有的事情，甚至更多。在了解它们之前，我们首先需要看看数据是如何在Elasticsearch中被索引的。