6映射和分析

**映射(mapping)**机制用于进行字段类型确认，将每个字段匹配为一种确定的数据类型(string, number, booleans, date等)。

**分析(analysis)**机制用于进行**全文文本(Full Text)**的分词，以建立供搜索用的反向索引。

6.1数据类型的差异

**映射及分析**

当在索引中处理数据时，我们注意到一些奇怪的事。有些东西似乎被破坏了：

在索引中有12个tweets，只有一个包含日期2014-09-15，但是我们看看下面查询中的total hits。

GET /\_search?q=2014 # 12 个结果

GET /\_search?q=2014-09-15 # 还是 12 个结果 !

GET /\_search?q=date:2014-09-15 # 1 一个结果

GET /\_search?q=date:2014 # 0 个结果 !

为什么全日期的查询返回所有的tweets，而针对date字段进行年度查询却什么都不返回？ 为什么我们的结果因查询\_all字段(译者注：默认所有字段中进行查询)或date字段而变得不同？

想必是因为我们的数据在\_all字段的索引方式和在date字段的索引方式不同而导致。

让我们看看Elasticsearch在对gb索引中的tweet类型进行*mapping*(也称之为*模式定义*[注：此词有待重新定义(schema definition)])后是如何解读我们的文档结构：

GET /gb/\_mapping/tweet

返回：

{

"gb": {

"mappings": {

"tweet": {

"properties": {

"date": {

"type": "date",

"format": "dateOptionalTime"

},

"name": {

"type": "string"

},

"tweet": {

"type": "string"

},

"user\_id": {

"type": "long"

}

}

}

}

}

}

Elasticsearch为对字段类型进行猜测，动态生成了字段和类型的映射关系。返回的信息显示了date字段被识别为date类型。\_all因为是默认字段所以没有在此显示，不过我们知道它是string类型。

date类型的字段和string类型的字段的索引方式是不同的，因此导致查询结果的不同，这并不会让我们觉得惊讶。

你会期望每一种核心数据类型(strings, numbers, booleans及dates)以不同的方式进行索引，而这点也是现实：在Elasticsearch中他们是被区别对待的。

1

但是更大的区别在于*确切值*(exact values)(比如string类型)及*全文文本*(full text)之间。

这两者的区别才真的很重要 - 这是区分搜索引擎和其他数据库的根本差异。

6.2确切对决全文

**确切值(Exact values) vs. 全文文本(Full text)**

Elasticsearch中的数据可以大致分为两种类型：

*确切值* 及 *全文文本*。

确切值是确定的，正如它的名字一样。比如一个date或用户ID，也可以包含更多的字符串比如username或email地址。

确切值"Foo"和"foo"就并不相同。确切值2014和2014-09-15也不相同。

全文文本，从另一个角度来说是文本化的数据(常常以人类的语言书写)，比如一篇推文(Twitter的文章)或邮件正文。

全文文本常常被称为非结构化数据，其实是一种用词不当的称谓，实际上自然语言是高度结构化的。

问题是自然语言的语法规则是如此的复杂，计算机难以正确解析。例如这个句子：

May is fun but June bores me.

到底是说的月份还是人呢？

确切值是很容易查询的，因为结果是二进制的 -- 要么匹配，要么不匹配。下面的查询很容易以SQL表达：

WHERE name = "John Smith"

AND user\_id = 2

AND date > "2014-09-15"

而对于全文数据的查询来说，却有些微妙。我们不会去询问这篇文档是否匹配查询要求？。 但是，我们会询问这篇文档和查询的匹配程度如何？。换句话说，对于查询条件，这篇文档的*相关性*有多高？

我们很少确切的匹配整个全文文本。我们想在全文中查询*包含*查询文本的部分。不仅如此，我们还期望搜索引擎能理解我们的*意图*：

* 一个针对"UK"的查询将返回涉及"United Kingdom"的文档
* 一个针对"jump"的查询同时能够匹配"jumped"， "jumps"， "jumping"甚至"leap"
* "johnny walker"也能匹配"Johnnie Walker"， "johnnie depp"及"Johnny Depp"
* "fox news hunting"能返回有关hunting on Fox News的故事，而"fox hunting news"也能返回关于fox hunting的新闻故事。

为了方便在全文文本字段中进行这些类型的查询，Elasticsearch首先对文本**分析(analyzes)**，然后使用结果建立一个**倒排索引**。我们将在以下两个章节讨论倒排索引及分析过程。

6.3倒排索引

## 倒排索引

Elasticsearch使用一种叫做**倒排索引(inverted index)**的结构来做快速的全文搜索。倒排索引由在文档中出现的唯一的单词列表，以及对于每个单词在文档中的位置组成。

例如，我们有两个文档，每个文档content字段包含：

1. The quick brown fox jumped over the lazy dog
2. Quick brown foxes leap over lazy dogs in summer

为了创建倒排索引，我们首先切分每个文档的content字段为单独的单词（我们把它们叫做**词(terms)**或者**表征(tokens)**）（译者注：关于terms和tokens的翻译比较生硬，只需知道语句分词后的个体叫做这两个。），把所有的唯一词放入列表并排序，结果是这个样子的：

1

| **Term** | **Doc\_1** | **Doc\_2** |
| --- | --- | --- |
| Quick |  | X |
| The | X |  |
| brown | X | X |
| dog | X |  |
| dogs |  | X |
| fox | X |  |
| foxes |  | X |
| in |  | X |
| jumped | X |  |
| lazy | X | X |
| leap |  | X |
| over | X | X |
| quick | X |  |
| summer |  | X |
| the | X |  |

现在，如果我们想搜索"quick brown"，我们只需要找到每个词在哪个文档中出现即可：

| **Term** | **Doc\_1** | **Doc\_2** |
| --- | --- | --- |
| brown | X | X |
| quick | X |  |
| ----- | ------- | ----- |
| Total | 2 | 1 |

两个文档都匹配，但是第一个比第二个有更多的匹配项。 如果我们加入简单的**相似度算法(similarity algorithm)**，计算匹配单词的数目，这样我们就可以说第一个文档比第二个匹配度更高——对于我们的查询具有更多相关性。

但是在我们的倒排索引中还有些问题：

1. "Quick"和"quick"被认为是不同的单词，但是用户可能认为它们是相同的。
2. "fox"和"foxes"很相似，就像"dog"和"dogs"——它们都是同根词。
3. "jumped"和"leap"不是同根词，但意思相似——它们是同义词。

上面的索引中，搜索"+Quick +fox"不会匹配任何文档（记住，前缀+表示单词必须匹配到）。只有"Quick"和"fox"都在同一文档中才可以匹配查询，但是第一个文档包含"quick fox"且第二个文档包含"Quick foxes"。（译者注：这段真啰嗦，说白了就是单复数和同义词没法匹配）

用户可以合理地希望两个文档都能匹配查询，我们也可以做得更好。

如果我们将词为统一为标准格式，这样就可以找到不是确切匹配查询，但是足以相似从而可以关联的文档。例如：

1. "Quick"可以转为小写成为"quick"。
2. "foxes"可以被转为根形式"fox"。同理"dogs"可以被转为"dog"。
3. "jumped"和"leap"同义就可以只索引为单个词"jump"

现在的索引：

| **Term** | **Doc\_1** | **Doc\_2** |
| --- | --- | --- |
| brown | X | X |
| dog | X | X |
| fox | X | X |
| in |  | X |
| jump | X | X |
| lazy | X | X |
| over | X | X |
| quick | X | X |
| summer |  | X |
| the | X | X |

但我们还未成功。我们的搜索"+Quick +fox"依旧失败，因为"Quick"的确切值已经不在索引里，不过，如果我们使用相同的标准化规则处理查询字符串的content字段，查询将变成"+quick +fox"，这样就可以匹配到两个文档。

### IMPORTANT

这很重要。你只可以找到确实存在于索引中的词，所以**索引文本和查询字符串都要标准化为相同的形式**。

这个标记化和标准化的过程叫做**分词(analysis)**，这个在下节中我们讨论。

## 分析和分析器

**分析(analysis)**是这样一个过程：

* 首先，标记化一个文本块为适用于倒排索引单独的**词(term)**
* 然后标准化这些词为标准形式，提高它们的“可搜索性”或“查全率”

这个工作是**分析器(analyzer)**完成的。一个**分析器(analyzer)**只是一个包装用于将三个功能放到一个包里：

1

### 字符过滤器

首先字符串经过**字符过滤器(character filter)**，它们的工作是在标记化前处理字符串。字符过滤器能够去除HTML标记，或者转换"&"为"and"。

### 分词器

下一步，**分词器(tokenizer)**被标记化成独立的词。一个简单的**分词器(tokenizer)**可以根据空格或逗号将单词分开（译者注：这个在中文中不适用）。

### 标记过滤

最后，每个词都通过所有**标记过滤(token filters)**，它可以修改词（例如将"Quick"转为小写），去掉词（例如停用词像"a"、"and"、"the"等等），或者增加词（例如同义词像"jump"和"leap"）

Elasticsearch提供很多开箱即用的字符过滤器，分词器和标记过滤器。这些可以组合来创建自定义的分析器以应对不同的需求。我们将在《自定义分析器》章节详细讨论。

### 内建的分析器

不过，Elasticsearch还附带了一些预装的分析器，你可以直接使用它们。下面我们列出了最重要的几个分析器，来演示这个字符串分词后的表现差异：

"Set the shape to semi-transparent by calling set\_trans(5)"

### 标准分析器

标准分析器是Elasticsearch默认使用的分析器。对于文本分析，它对于任何语言都是最佳选择（译者注：就是没啥特殊需求，对于任何一个国家的语言，这个分析器就够用了）。它根据[Unicode Consortium](http://www.unicode.org/reports/tr29/)的定义的**单词边界(word boundaries)**来切分文本，然后去掉大部分标点符号。最后，把所有词转为小写。产生的结果为：

1

set, the, shape, to, semi, transparent, by, calling, set\_trans, 5

### 简单分析器

简单分析器将非单个字母的文本切分，然后把每个词转为小写。产生的结果为：

1

set, the, shape, to, semi, transparent, by, calling, set, trans

### 空格分析器

空格分析器依据空格切分文本。它不转换小写。产生结果为：

1

Set, the, shape, to, semi-transparent, by, calling, set\_trans(5)

### 语言分析器

特定语言分析器适用于很多语言。它们能够考虑到特定语言的特性。例如，english分析器自带一套英语停用词库——像and或the这些与语义无关的通用词。这些词被移除后，因为语法规则的存在，英语单词的主体含义依旧能被理解（译者注：stem English words这句不知道该如何翻译，查了字典，我理解的大概意思应该是将英语语句比作一株植物，去掉无用的枝叶，主干依旧存在，停用词好比枝叶，存在与否并不影响对这句话的理解。）。

3

english分析器将会产生以下结果：

set, shape, semi, transparent, call, set\_tran, 5

注意"transparent"、"calling"和"set\_trans"是如何转为词干的。

### 当分析器被使用

当我们**索引(index)**一个文档，全文字段会被分析为单独的词来创建倒排索引。不过，当我们在全文字段**搜索(search)**时，我们要让查询**字符串**经过**同样的分析流程**处理，以确保这些词在索引中存在。

全文查询我们将在稍后讨论，理解每个字段是如何定义的，这样才可以让它们做正确的事：

* 当你查询**全文(full text)**字段，查询将使用相同的分析器来分析查询字符串，以产生正确的词列表。
* 当你查询一个**确切值(exact value)**字段，查询将不分析查询字符串，但是你可以自己指定。

现在你可以明白为什么《映射和分析》的开头会产生那种结果：

* date字段包含一个确切值：单独的一个词"2014-09-15"。
* \_all字段是一个全文字段，所以分析过程将日期转为三个词："2014"、"09"和"15"。

当我们在\_all字段查询2014，它一个匹配到12条推文，因为这些推文都包含词2014：

1

GET /\_search?q=2014 # 12 results

当我们在\_all字段中查询2014-09-15，首先分析查询字符串，产生匹配**任一**词2014、09或15的查询语句，它依旧匹配12个推文，因为它们都包含词2014。

GET /\_search?q=2014-09-15 # 12 results !

当我们在date字段中查询2014-09-15，它查询一个**确切**的日期，然后只找到一条推文：

GET /\_search?q=date:2014-09-15 # 1 result

当我们在date字段中查询2014，没有找到文档，因为没有文档包含那个确切的日期：

GET /\_search?q=date:2014 # 0 results !

### 测试分析器

尤其当你是Elasticsearch新手时，对于如何分词以及存储到索引中理解起来比较困难。为了更好的理解如何进行，你可以使用analyze API来查看文本是如何被分析的。在查询字符串参数中指定要使用的分析器，被分析的文本做为请求体：

GET /\_analyze?analyzer=standard&text=Text to analyze

结果中每个节点在代表一个词：

{

"tokens": [

{

"token": "text",

"start\_offset": 0,

"end\_offset": 4,

"type": "<ALPHANUM>",（me:字母数字）

"position": 1

},

{

"token": "to",

"start\_offset": 5,

"end\_offset": 7,

"type": "<ALPHANUM>",

"position": 2

},

{

"token": "analyze",

"start\_offset": 8,

"end\_offset": 15,

"type": "<ALPHANUM>",

"position": 3

}

]

}

token是一个实际被存储在索引中的词。position指明词在原文本中是第几个出现的。start\_offset和end\_offset表示词在原文本中占据的位置。

analyze API 对于理解Elasticsearch索引的内在细节是个非常有用的工具，随着内容的推进，我们将继续讨论它。

### 指定分析器

当Elasticsearch在你的文档中探测到一个新的字符串字段，它将自动设置它为全文string字段并用standard分析器分析。

你不可能总是想要这样做。也许你想使用一个更适合这个数据的语言分析器。或者，你只想把字符串字段当作一个普通的字段——不做任何分析，只存储确切值，就像字符串类型的用户ID或者内部状态字段或者标签。

为了达到这种效果，我们必须通过**映射(mapping)**人工设置这些字段。

## 映射

为了能够把日期字段处理成日期，把数字字段处理成数字，把字符串字段处理成全文本（Full-text）或精确的字符串值，Elasticsearch需要知道每个字段里面都包含了什么类型。这些类型和字段的信息存储（包含）在映射（mapping）中。

正如《数据吞吐》一节所说，索引中每个文档都有一个**类型(type)**。 每个类型拥有自己的**映射(mapping)**或者**模式定义(schema definition)**。一个映射定义了字段类型，每个字段的数据类型，以及字段被Elasticsearch处理的方式。映射还用于设置关联到类型上的元数据。

在《映射》章节我们将探讨映射的细节。这节我们只是带你入门。

### 核心简单字段类型

Elasticsearch支持以下简单字段类型：

| **类型** | **表示的数据类型** |
| --- | --- |
| String | string |
| Whole number | byte, short, integer, long |
| Floating point | float, double |
| Boolean | boolean |
| Date | date |

当你索引一个包含新字段的文档——一个之前没有的字段——Elasticsearch将使用动态映射猜测字段类型，这类型来自于JSON的基本数据类型，使用以下规则：

| **JSON type** | **Field type** |
| --- | --- |
| Boolean: true or false | "boolean" |
| Whole number: 123 | "long" |
| Floating point: 123.45 | "double" |
| String, valid date: "2014-09-15" | "date" |
| String: "foo bar" | "string" |

### 注意

这意味着，如果你索引一个带引号的数字——"123"，它将被映射为"string"类型，而不是"long"类型。然而，如果字段已经被映射为"long"类型，Elasticsearch将尝试转换字符串为long，并在转换失败时会抛出异常。

### 查看映射

我们可以使用\_mapping后缀来查看Elasticsearch中的映射。在本章开始我们已经找到索引gb类型tweet中的映射：

GET /gb/\_mapping/tweet

这展示给了我们字段的映射（叫做**属性(properties)**），这些映射是Elasticsearch在创建索引时动态生成的：

{

"gb": {

"mappings": {

"tweet": {

"properties": {

"date": {

"type": "date",

"format": "strict\_date\_optional\_time||epoch\_millis"

},

"name": {

"type": "string"

},

"tweet": {

"type": "string"

},

"user\_id": {

"type": "long"

}

}

}

}

}

}

### 小提示

错误的映射，例如把age字段映射为string类型而不是integer类型，会造成查询结果混乱。

要检查映射类型，而不是假设它是正确的！

### 自定义字段映射

虽然大多数情况下基本数据类型已经能够满足，但你也会经常需要自定义一些特殊类型（fields），特别是字符串字段类型。 自定义类型可以使你完成以下几点：

* 区分全文（full text）字符串字段和准确字符串字段（译者注：就是分词与不分词，全文的一般要分词，准确的就不需要分词，比如『中国』这个词。全文会分成『中』和『国』，但作为一个国家标识的时候我们是不需要分词的，所以它就应该是一个准确的字符串字段）。
* 使用特定语言的分析器（译者注：例如中文、英文、阿拉伯语，不同文字的断字、断词方式的差异）
* 优化部分匹配字段
* 指定自定义日期格式（译者注：这个比较好理解,例如英文的 Feb,12,2016 和 中文的 2016年2月12日）
* 以及更多

映射中最重要的字段参数是type。除了string类型的字段，你可能很少需要映射其他的type：

{

"number\_of\_clicks": {

"type": "integer"

}

}

string类型的字段，默认的，考虑到包含全文本，它们的值在索引前要经过分析器分析，并且在全文搜索此字段前要把查询语句做分析处理。

对于string字段，两个最重要的映射参数是index和analyer。

### index

index参数控制字符串以何种方式被索引。它包含以下三个值当中的一个：

| **值** | **解释** |
| --- | --- |
| analyzed | 首先分析这个字符串，然后索引。换言之，以全文形式索引此字段。 |
| not\_analyzed | 索引这个字段，使之可以被搜索，但是索引内容和指定值一样。不分析此字段。 |
| no | 不索引这个字段。这个字段不能为搜索到。 |

string类型字段默认值是analyzed。如果我们想映射字段为确切值，我们需要设置它为not\_analyzed：

{

"tag": {

"type": "string",

"index": "not\_analyzed"

}

}

其他简单类型（long、double、date等等）也接受index参数，但相应的值只能是no和not\_analyzed，它们的值不能被分析。

### 分析

对于analyzed类型的字符串字段，使用analyzer参数来指定哪一种分析器将在搜索和索引的时候使用。默认的，Elasticsearch使用standard分析器，但是你可以通过指定一个内建的分析器来更改它，例如whitespace、simple或english。

{

"tweet": {

"type": "string",

"analyzer": "english"

}

}

在《自定义分析器》章节我们将告诉你如何定义和使用自定义的分析器。

### 更新映射

你可以在第一次创建索引的时候指定映射的类型。此外，你也可以晚些时候为新类型添加映射（或者为已有的类型更新映射）。

### 重要

你可以向已有映射中**增加**字段，但你不能**修改**它。如果一个字段在映射中已经存在，这可能意味着那个字段的数据已经被索引。如果你改变了字段映射，那已经被索引的数据将错误并且不能被正确的搜索到。

我们可以更新一个映射来增加一个新字段，但是不能把已有字段的类型那个从analyzed改到not\_analyzed。

为了演示两个指定的映射方法，让我们首先删除索引gb：

DELETE /gb

然后创建一个新索引，指定tweet字段的分析器为english：

PUT /gb <1>

{

"mappings": {

"tweet" : {

"properties" : {

"tweet" : {

"type" : "string",

"analyzer": "english"

},

"date" : {

"type" : "date"

},

"name" : {

"type" : "string"

},

"user\_id" : {

"type" : "long"

}

}

}

}

}

<1> 这将创建包含mappings的索引，映射在请求体中指定。

再后来，我们决定在tweet的映射中增加一个新的not\_analyzed类型的文本字段，叫做tag，使用\_mapping后缀:

PUT /gb/\_mapping/tweet

{

"properties" : {

"tag" : {

"type" : "string",

"index": "not\_analyzed"

}

}

}

注意到我们不再需要列出所有的已经存在的字段，因为我们没法修改他们。我们的新字段已经被合并至存在的那个映射中。

### 测试映射

你可以通过名字使用analyze API测试字符串字段的映射。对比这两个请求的输出：

GET /gb/\_analyze?field=tweet&text=Black-cats <1>

GET /gb/\_analyze?field=tag&text=Black-cats <2>

<1> <2> 我们想要分析的文本被放在请求体中。

tweet字段产生两个词，"black"和"cat",tag字段产生单独的一个词"Black-cats"。换言之，我们的映射工作正常。

6.6复合类型

## 复合核心字段类型

除了之前提到的简单的标量类型，JSON还有null值，数组和对象，所有这些Elasticsearch都支持：

### 多值字段

我们想让tag字段包含多个字段，这非常有可能发生。我们可以索引一个标签数组来代替单一字符串：

{ "tag": [ "search", "nosql" ]}

对于数组不需要特殊的映射。任何一个字段可以包含零个、一个或多个值，同样对于全文字段将被分析并产生多个词。

言外之意，这意味着**数组中所有值必须为同一类型**。你不能把日期和字符串混合。如果你创建一个新字段，这个字段索引了一个数组，Elasticsearch将使用第一个值的类型来确定这个新字段的类型。

2

当你从Elasticsearch中取回一个文档，任何一个数组的顺序和你索引它们的顺序一致。你取回的\_source字段的顺序同样与索引它们的顺序相同。

1

然而，数组是作为多值字段被**索引**的，它们没有顺序。在搜索阶段你不能指定“第一个值”或者“最后一个值”。倒不如把数组当作一个**值集合(bag of values)**

2

### 空字段

当然数组可以是空的。这等价于有零个值。事实上，Lucene没法存放null值，所以一个null值的字段被认为是空字段。

这四个字段将被识别为空字段而不被索引：

"empty\_string": "",

"null\_value": null,

"empty\_array": [],

"array\_with\_null\_value": [ null ]

### 多层对象

我们需要讨论的最后一个自然JSON数据类型是**对象(object)**——在其它语言中叫做hash、hashmap、dictionary 或者 associative array.

1

**内部对象(inner objects)**经常用于在另一个对象中嵌入一个实体或对象。例如，做为在tweet文档中user\_name和user\_id的替代，我们可以这样写：

{

"tweet": "Elasticsearch is very flexible",

"user": {

"id": "@johnsmith",

"gender": "male",

"age": 26,

"name": {

"full": "John Smith",

"first": "John",

"last": "Smith"

}

}

}

### 内部对象的映射

Elasticsearch 会动态的检测新对象的字段，并且映射它们为 object 类型，将每个字段加到 properties字段下

{

"gb": {

"tweet": { <1>

"properties": {

"tweet": { "type": "string" },

"user": { <2>

"type": "object",

"properties": {

"id": { "type": "string" },

"gender": { "type": "string" },

"age": { "type": "long" },

"name": { <3>

"type": "object",

"properties": {

"full": { "type": "string" },

"first": { "type": "string" },

"last": { "type": "string" }

}

}

}

}

}

}

}

}

<1> 根对象.

<2><3> 内部对象.

对user和name字段的映射与tweet类型自己很相似。事实上，type映射只是object映射的一种特殊类型，我们将 object 称为根对象。它与其他对象一模一样，除非它有一些特殊的顶层字段，比如 \_source, \_all 等等。

1

### 内部对象是怎样被索引的

Lucene 并不了解内部对象。 一个 Lucene 文件包含一个键-值对应的扁平表单。 为了让 Elasticsearch 可以有效的索引内部对象，将文件转换为以下格式：

{

"tweet": [elasticsearch, flexible, very],

"user.id": [@johnsmith],

"user.gender": [male],

"user.age": [26],

"user.name.full": [john, smith],

"user.name.first": [john],

"user.name.last": [smith]

}

内部栏位可被归类至name，例如"first"。 为了区别两个拥有相同名字的栏位，我们可以使用完整路径，例如"user.name.first" 或甚至类型名称加上路径："tweet.user.name.first"。

注意： 在以上扁平化文件中，并没有栏位叫作user也没有栏位叫作user.name。 Lucene 只索引阶层或简单的值，而不会索引复杂的资料结构。

## 对象-数组

### 内部对象数组

最后，一个包含内部对象的数组如何索引。 我们有个数组如下所示：

{

"followers": [

{ "age": 35, "name": "Mary White"},

{ "age": 26, "name": "Alex Jones"},

{ "age": 19, "name": "Lisa Smith"}

]

}

此文件会如我们以上所说的被扁平化，但其结果会像如此：

{

"followers.age": [19, 26, 35],

"followers.name": [alex, jones, lisa, smith, mary, white]

}

{age: 35}与{name: Mary White}之间的关联会消失，因每个多值的栏位会变成一个值集合，而非有序的阵列。 这让我们可以知道：

* 是否有26岁的追随者？

但我们无法取得准确的资料如：

* 是否有26岁的追随者***且名字叫Alex Jones？***

关联内部对象可解决此类问题，我们称之为嵌套对象，我们之後会在嵌套对象中提到它。