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。