Doc Values and Fielddata

Doc Values

聚合使用一个叫 *doc values* 的数据 （在 [docvalues-intro] 里 介 ）。 Doc values 可以使聚合更快、更高效并且内存友好，所以理解它的工作方式十分有益。

Doc values 的存在是因 倒排索引只 某些操作是高效的。 倒排索引的 在于 包含某个 的文 ，而 于从 外一个方向的相反操作并不高效，即： 定 些 是否存在 个文 里，聚合需要 次的 模式。

于以下倒排索引：

```
Term      Doc_1   Doc_2   Doc_3
-----------------------------------
brown   |   X   |   X   |
dog     |   X   |       |   X
dogs    |       |   X   |   X
fox     |   X   |       |   X
foxes   |       |   X   |
in      |       |   X   |
jumped  |   X   |       |   X
lazy    |   X   |   X   |
leap    |       |   X   |
over    |   X   |   X   |   X
quick   |   X   |   X   |   X
summer  |       |   X   |
the     |   X   |       |   X
-----------------------------------
```

如果我 想要 得所有包含 brown 的文 的 的完整列表，我 会 建如下 ：

```
GET /my_index/_search
{
  "query" : {
    "match" : {
      "body" : "brown"
    }
  },
  "aggs" : {
    "popular_terms": {
      "terms" : {
        "field" : "body"
      }
    }
  }
}
```

部分 又高效。倒排索引是根据 来排序的，所以我 首先在 列表中 到 brown ，然后描所有列， 到包含 brown 的文 。我 可以快速看到 Doc_1 和 Doc_2 包含 brown 个 token。

然后， 于聚合部分，我 需要 到 Doc_1 和 Doc_2 里所有唯一的 。用倒排索引做 件事情代 很高：我 会迭代索引里的 个 并收集 Doc_1 和 Doc_2 列里面 token。 很慢而且 以 展：随着 和文 的数量 加， 行 也会 加。

Doc values 通 置 者 的 系来解决 个 。倒排索引将 映射到包含它 的文 ，doc values 将文 映射到它 包含的 ：

```
Doc      Terms
-----------------------------------------------------------------
Doc_1 | brown, dog, fox, jumped, lazy, over, quick, the
Doc_2 | brown, dogs, foxes, in, lazy, leap, over, quick, summer
Doc_3 | dog, dogs, fox, jumped, over, quick, the
-----------------------------------------------------------------
```

当数据被 置之后，想要收集到 Doc_1 和 Doc_2 的唯一 token 会非常容易。 得 个文 行，取所有的 ，然后求 个集合的并集。

因此，搜索和聚合是相互 密 的。搜索使用倒排索引 文 ，聚合操作收集和聚合 doc values 里的数据。

NOTE	Doc values 不 可以用于聚合。 任何需要 某个文 包含的 的操作都必 使用它。除了聚合， 包括排序， 字段 的脚本，父子 系 理（参 [parent-child] ）。

深入文

在上一 一 我 就 文 （doc values） 是 "更快、更高效并且内存友好" 。 听起来好像是不的 ，不 回来文 到底是如何工作的 ？

文 是在索引 与倒排索引同 生的。也就是 文 是按段来 生的并且是不可 的，正如用于搜索的倒排索引一 。同 ，和倒排索引一 ，文 也序列化到磁 。 些 于性能和伸 性很重要。

通 序列化一个持久化的数据 到磁 ，我 可以依 于操作系 的 存来管理内存，而不是在 JVM 堆 里 留数据。 当 "工作集（working set）" 数据要小于系 可用内存的情况下，操作系会自然的将文 留在内存， 将会 来和直接使用 JVM 堆 数据 相同的性能。

不 ，如果 的工作集 大于可用内存，操作系 会 始根据需要 文 行分 / 。 会 著慢于内存 留的数据 ，当然，它也 有使用 大于服 器内存容量的伸 性的好 。 如果 些数据 是粹的存 于 JVM 堆内存，那 唯一的 只能是随着内存溢出（OutOfMemory）而崩 （或是一个分 模式，正如操作系 的那 ）。

因 文 不是由 JVM 来管理，所以 Elasticsearch 服 器可以配置一个很小的 JVM 堆 。 会 操作系 来更多的内存来做 存。同 也 来一个好 就是 JVM 的 回收器工作在一个很小的堆 ， 果就是更快更高效的回收周期。

NOTE

上，我 会建 分配机器内存的 50% 来 JVM 堆 。随着文 的引入， 个建 始不再 用。 在 64gb 内存的机器上，也 可以考 堆 分配 4-16gb 的内存，而不是之前建 的 32gb。

有 更 的 ， 看 [heap-sizing].

列式存 的

从广 来 ，文 本 上是一个序列化的 列式存 。 正如我 上一 所 的，列式存 擅 某些操作，因 些数据的存 天然 合 些 。

而且，他 也同 擅 数据 ，特 是数字。 于 省磁 空 和快速 很重要。 代 CPU 的 理速度要比磁 快几个数量 （尽管即将到来的 NVMe 器正在迅速 小差距）。 意味着 少必 从磁 取的数据量 是有益的，尽管需要 外的 CPU 算来 行解 。

要了解它如何 助 数据，来看一 数字 型的文 ：

```
Doc      Terms
-----------------------------------------------------------------
Doc_1 | 100
Doc_2 | 1000
Doc_3 | 1500
Doc_4 | 1200
Doc_5 | 300
Doc_6 | 1900
Doc_7 | 4200
-----------------------------------------------------------------
```

按列布局意味着我 有一个 的数据 ： `[100,1000,1500,1200,300,1900,4200]` 。因 我 已 知道他 都是数字（而不是像文 或行中看到的 集合），所以我 可以使用 一的偏移来将他 排列。

而且， 的数字有很多 技巧。 会注意到 里 个数字都是 100 的倍数，文 会 一个段里面的所有数 ，并使用一个 最大公 数，方便做 一 的数据 。

如果我 保存 `100` 作 此段的除数，我 可以 个数字都除以 100，然后得到： `[1,10,15,12,3,19,42]` 。 在 些数字 小了，只需要很少的位就可以存 下，也 少了磁 存放的大小。

文 正是使用了像 的一些技巧。它会按依次 以下 模式:

1. 如果所有的数 各不相同（或 失）， 置一个 并 些
2. 如果 些 小于 256，将使用一个 的 表
3. 如果 些 大于 256， 是否存在一个最大公 数
4. 如果没有存在最大公 数，从最小的数 始， 一 算偏移量 行

会 些 模式不是 的通用的 方式，比如 DEFLATE 或是 LZ4。 因 列式存 的 是格且良好定 的，我 可以通 使用 的模式来 到比通用 算法（如 LZ4 ）更高的 效果。

NOTE

也 会想 "好 ，貌似 数字很好，不知道字符串 ？" 通 借助 序表（ordinal table），字符 型也是 似 行 的。字符 型是去重之后存放到 序表的，通 分配一个 ID，然后 些 ID 和数 型的文 一 使用。 也就是 ，字符 型和数型一 有相同的 特性。

序表本身也有很多 技巧，比如固定 度、 或是前 字符 等等。

禁用文

文 所有字段 用，除了分析字符 型字段。也就是 所有的数字、地理坐 、日期、IP 和不分析（ `not_analyzed` ）字符 型。

分析字符 型 不使用文 。分析流程会 生很多新的 token， 会 文 不能高效的工作。我 将在 聚合与分析 如何使用分析字符 型来做聚合。

因 文 用， 可以 数据集里面的大多数字段 行聚合和排序操作。但是如果 知道 永 也不会 某些字段 行聚合、排序或是使用脚本操作？

尽管 ，但当 些情况出 ， 是希望有 法来 特定的字段禁用文 。 回 省磁 空 （因 文 再也没有序列化到磁 ），也 能提升些 索引速度（因 不需要生成文 ）。

要禁用文 ，在字段的映射（mapping） 置 `doc_values: false` 即可。例如， 里我 建了一个新的索引，字段 `"session_id"` 禁用了文 ：

```
PUT my_index
{
  "mappings": {
    "my_type": {
      "properties": {
        "session_id": {
          "type":       "string",
          "index":      "not_analyzed",
          "doc_values": false ①
        }
      }
    }
  }
}
```

① 通 置 `doc_values: false` ， 个字段将不能被用于聚合、排序以及脚本操作

反 来也是可以 行配置的： 一个字段可以被聚合，通 禁用倒排索引，使它不能被正常搜索，例如：

```
PUT my_index
{
  "mappings": {
    "my_type": {
      "properties": {
        "customer_token": {
          "type":       "string",
          "index":      "not_analyzed",
          "doc_values": true, ①
          "index": "no" ②
        }
      }
    }
  }
}
```

① 文　　被　用来允　聚合

② 索引被禁用了，　　字段不能被　　/搜索

通　　置 `doc_values: true` 和 `index: no` ，我　得到一个只能被用于聚合/排序/脚本的字段。无可否　，　是一个非常　　的需求，但有　很有用。

聚合与分析

有些聚合，比如 `terms` 桶，　操作字符串字段。字符串字段可能是 `analyzed` 或者 `not_analyzed` ，　那　　来了，分析是　　影　聚合的　？

答案是影　“很多”，有　个原因：分析影　聚合中使用的　tokens　，并且　doc　values　不能使用于分析字符串。

　我　解决第一个　　：分析 tokens 的　生如何影　聚合。首先索引一些代表美国各个州的文　：

```
POST /agg_analysis/data/_bulk
{ "index": {}}
{ "state" : "New York" }
{ "index": {}}
{ "state" : "New Jersey" }
{ "index": {}}
{ "state" : "New Mexico" }
{ "index": {}}
{ "state" : "New York" }
{ "index": {}}
{ "state" : "New York" }
```

我　希望　建一个数据集里各个州的唯一列表，并且　数。　　，　我　使用 `terms` 桶：

```
GET /agg_analysis/data/_search
{
    "size" : 0,
    "aggs" : {
        "states" : {
            "terms" : {
                "field" : "state"
            }
        }
    }
}
```

得到 果：

```
{
...
   "aggregations": {
      "states": {
         "buckets": [
            {
               "key": "new",
               "doc_count": 5
            },
            {
               "key": "york",
               "doc_count": 3
            },
            {
               "key": "jersey",
               "doc_count": 1
            },
            {
               "key": "mexico",
               "doc_count": 1
            }
         ]
      }
   }
}
```

宝 儿， 完全不是我 想要的！没有 州名 数，聚合 算了 个 的数目。背后的原因很 ：聚合是基于倒排索引 建的，倒排索引是 后置分析（*post-analysis*）的。

当我 把 些文 加入到 Elasticsearch 中 ，字符串 `"New York"` 被分析/分析成 `["new", "york"]` 。些 独的 tokens，都被用来填充聚合 数，所以我 最 看到 `new` 的数量而不是 `New York` 。

然不是我 想要的行 ，但幸 的是很容易修正它。

我 需要 state 定 multifield 并且 置成 `not_analyzed` 。 可以防止 New York

被分析，也意味着在聚合 程中它会以 个 token 的形式存在。 我 完整的 程，但 次指定一个 *raw* multifield：

```
DELETE /agg_analysis/
PUT /agg_analysis
{
  "mappings": {
    "data": {
      "properties": {
        "state" : {
          "type": "string",
          "fields": {
            "raw" : {
              "type": "string",
              "index": "not_analyzed"①
            }
          }
        }
      }
    }
  }
}

POST /agg_analysis/data/_bulk
{ "index": {}}
{ "state" : "New York" }
{ "index": {}}
{ "state" : "New Jersey" }
{ "index": {}}
{ "state" : "New Mexico" }
{ "index": {}}
{ "state" : "New York" }
{ "index": {}}
{ "state" : "New York" }

GET /agg_analysis/data/_search
{
  "size" : 0,
  "aggs" : {
    "states" : {
        "terms" : {
            "field" : "state.raw" ②
        }
    }
  }
}
```

① 次我 式映射 state 字段并包括一个 `not_analyzed` 字段。

② 聚合 state.raw 字段而不是 state 。

在 行聚合，我 得到了合理的 果：

```
{
...
   "aggregations": {
      "states": {
         "buckets": [
            {
               "key": "New York",
               "doc_count": 3
            },
            {
               "key": "New Jersey",
               "doc_count": 1
            },
            {
               "key": "New Mexico",
               "doc_count": 1
            }
         ]
      }
   }
}
```

在 中， 的 很容易被察 ，我 的聚合会返回一些奇怪的桶，我 会 住分析的 。
之，很少有在聚合中使用分析字段的 例。当我 疑惑 ，只要 加一个 multifield 就能有 。

分析字符串和 Fielddata（Analyzed strings and Fielddata）

当第一个 及如何聚合数据并 示 用 ，第二个 主要是技 和幕后。

Doc values 不支持 `analyzed` 字符串字段，因 它 不能很有效的表示多 字符串。 Doc values 最有效的是，当 个文 都有一个或几个 tokens ， 但不是无数的，分析字符串（想象一个 PDF ，可能有几兆字 并有数以千 的独特 tokens）。

出于 个原因，doc values 不生成分析的字符串，然而， 些字段 然可以使用聚合，那 可能 ？

答案是一 被称 *fielddata* 的数据 。与 doc values 不同，fielddata 建和管理 100% 在内存中，常 于 JVM 内存堆。 意味着它本 上是不可 展的，有很多 情况下要提防。本章的其余部分是解决在分析字符串上下文中 fielddata 的挑 。

NOTE

从 史上看，fielddata 是 所有 字段的 置。但是 Elasticsearch 已 移到 doc values 以 少 OOM 的几率。分析的字符串是 然使用 fielddata 的最后一 地。 最 目 是建立一个序列化的数据 似于 doc values ，可以 理高 度的分析字符串，逐 淘汰 fielddata。

高基数内存的影 （High-Cardinality Memory Implications）

避免分析字段的 外一个原因就是：高基数字段在加 到 fielddata 会消耗大量内存。 分析的 程会

常（尽管不 是 ）生成大量的 token， 些 token 大多都是唯一的。 会 加字段的整体基数并且来更大的内存 力。

有些 型的分析 于内存来 度 不友好，想想 n-gram 的分析 程， New York 会被 n-gram 分析成以下 token：

- ne
- ew
- w{nbsp}
- {nbsp}y
- yo
- or
- rk

可以想象 n-gram 的 程是如何生成大量唯一 token 的，特 是在分析成段文本的 候。当 些数据加 到内存中，会 而易 的将我 堆空 消耗殆尽。

因此，在聚合字符串字段之前， 估情况：

- 是一个 not_analyzed 字段 ？如果是，可以通 doc values 省内存 。
- 否 ， 是一个 analyzed 字段，它将使用 fielddata 并加 到内存中。 个字段因 ngrams 有一个非常大的基数？如果是， 于内存来 度不友好。

限制内存使用

一旦分析字符串被加 到 fielddata ，他 会一直在那里，直到被 逐（或者 点崩 ）。由于 个原因，留意内存的使用情况，了解它是如何以及何 加 的， 限制 集群的影 是很重要的。

Fielddata 是 延 加 。如果 从来没有聚合一个分析字符串，就不会加 fielddata 到内存中。此外，fielddata 是基于字段加 的， 意味着只有很活 地使用字段才会 加 fielddata 的 担。

然而， 里有一个令人 的地方。假 的 是高度 性和只返回命中的 100 个 果。大多数人 fielddata 只加 100 个文 。

情况是，fielddata 会加 索引中（ 特定字段的） 所有的 文 ，而不管 的特 性。是 ：如果 会 文 X、Y 和 Z，那很有可能会在下一个 中 其他文 。

与 doc values 不同，fielddata 不会在索引 建。相反，它是在 行 ， 填充。 可能是一个比 的操作，可能需要一些 。 将所有的信息一次加 ，再将其 持在内存中的方式要比反 只加 一个 fielddata 的部分代 要低。

JVM 堆 是有限 源的， 被合理利用。 限制 fielddata 堆使用的影 有多套机制， 些限制方式非常重要，因 堆 的乱用会 致 点不 定（感 慢的 回收机制），甚至 致 点宕机（通常伴随 OutOfMemory 常）。

堆大小（Choosing a Heap Size）

在 置 Elasticsearch 堆大小 需要通 `$ES_HEAP_SIZE` 境 量 用 个 ：

不要超 可用 *RAM* 的 *50%*

Lucene 能很好利用文件系 的 存，它是通 系 内核管理的。如果没有足 的文件系存空 ，性能会受到影 。 此外， 用于堆的内存越多意味着其他所有使用 doc values 的字段内存越少。

不要超 *32 GB*

如果堆大小小于 32 GB，JVM 可以利用指 ， 可以大大降低内存的使用： 个指 4 字 而不是 8 字 。

更 和更完整的堆大小 ， 参 [heap-sizing]

Fielddata的大小

`indices.fielddata.cache.size` 控制 fielddata 分配的堆空 大小。 当 起一个 ，分析字符串的聚合将会被加 到 fielddata，如果 些字符串之前没有被加 。如果 果中 fielddata 大小超 了指定 大小，其他的 将会被回收从而 得空 。

情况下， 置都是 *unbounded*，Elasticsearch 永 都不会从 fielddata 中回收数据。

个 置是刻意 的：fielddata 不是 存。它是 留内存里的数据 ，必 可以快速 行 ，而且 建它的代 十分高昂。如果 个 求都重 数据，性能会十分糟 。

一个有界的大小会 制数据 回收数据。我 会看何 置 个 ，但 首先 以下警告：

WARNING

个 置是一个安全 士，而非内存不足的解决方案。

如果没有足 空 可以将 fielddata 保留在内存中，Elasticsearch 就会 刻从磁重 数据，并回收其他数据以 得更多空 。内存的回收机制会 致重度磁 I/O，并且在内存中生成很多 ， 些 必 在 些 候被回收掉。

想我 正在 日志 行索引， 天使用一个新的索引。通常我 只 去一 天的数据感 趣，尽管我会保留老的索引，但我 很少需要 它 。不 如果采用 置，旧索引的 fielddata 永 不会从 存中回收！fieldata 会保持 直到 fielddata 生断熔（ 参 断路器）， 我 就无法 入更多的 fielddata。

个 候，我 被困在了死胡同。但我 然可以 旧索引中的 fielddata，也无法加 任何新的 。相反，我 回收旧的数据，并 新 得更多空 。

了防止 生 的事情，可以通 在 `config/elasticsearch.yml` 文件中 加配置 fielddata 置一个上限：

```
indices.fielddata.cache.size:  20% ①
```

① 可以 置堆大小的百分比，也可以是某个 ，例如： `5gb` 。

有了 个 置，最久未使用（LRU）的 fielddata 会被回收 新数据 出空 。

WARNING

可能 在 文 有 外一个 置： `indices.fielddata.cache.expire` 。

个 置 永 都不会 被使用！它很有可能在不久的将来被 用。

个 置要求 Elasticsearch 回收那些 期 的 fielddata，不管 些 有没有被用到。

性能是件 很糟 的事情。回收会有消耗性能，它刻意的安排回收方式，而没能得任何回 。

没有理由使用 个 置：我 不能从理 上假 一个有用的情形。目前，它的存在只是 了向前兼容。我 只在很有以前提到 个 置，但不幸的是 上各 文章都将其作 一 性能 的小 来推 。

它不是。永 不要使用！

控 fielddata（Monitoring fielddata）

无 是仔 控 fielddata 的内存使用情况， 是看有无数据被回收都十分重要。高的回收数可以 示 重的 源 以及性能不佳的原因。

Fielddata 的使用可以被 控：

- 按索引使用 `indices-stats` API ：

```
GET /_stats/fielddata?fields=*
```

- 按 点使用 {ref}/cluster-nodes-stats.html[`nodes-stats` API] ：

```
GET /_nodes/stats/indices/fielddata?fields=*
```

- 按索引 点：

```
GET /_nodes/stats/indices/fielddata?level=indices&fields=*
```

使用 置 `?fields=*` ，可以将内存使用分配到 个字段。

断路器

机敏的 者可能已 fielddata 大小 置的一个 。fielddata 大小是在数据加 之后 的。如果一个 加 比可用内存更多的信息到 fielddata 中会 生什 ？答案很丑 ：我 会 到 OutOfMemoryException 。

Elasticsearch 包括一个 *fielddata* 断熔器 ， 个 就是 了 理上述情况。 断熔器通 内部 （字段的 型、基数、大小等等）来估算一个 需要的内存。它然后 要求加 的 fielddata 是否会 致 fielddata 的 量超 堆的配置比例。

如果估算　的大小超出限制，就会　触　断路器，　会被中止并返回　常。　都　生在数据加　之前，也就意味着不会引起 OutOfMemoryException 。

可用的断路器（Available Circuit Breakers）

Elasticsearch 有一系列的断路器，它　都能保　内存不会超出限制：

`indices.breaker.fielddata.limit`
　`fielddata` 断路器　置堆的 60% 作　fielddata 大小的上限。

`indices.breaker.request.limit`
　`request`　断路器估算需要完成其他　求部分的　大小，例如　建一个聚合桶，限制是堆内存的 40%。

`indices.breaker.total.limit`
　`total` 揉合 `request` 和 `fielddata` 断路器保　者　合起来不会使用超　堆内存的 70%。

断路器的限制可以在文件 `config/elasticsearch.yml` 中指定，可以　更新一个正在　行的集群：

```
PUT /_cluster/settings
{
  "persistent" : {
    "indices.breaker.fielddata.limit" : "40%" ①
  }
}
```

① 个限制是按　内存的百分比　置的。

最好　断路器　置一个相　保守点的　。　住　fielddata　需要与　`request`　断路器共享堆内存、索引　冲内存和　器　存。Lucene　的数据被用来　造索引，以及各　其他　的数据　。正因如此，它　非常保守，只有　60%　。　于　的　置可能会引起潜在的堆　溢出（OOM）　常，　会使整个　点宕掉。

一方面，　度保守的　只会返回　常，　用程序可以　常做相　理。　常比服　器崩　要好。些　常　也能促　我　行重新　估：　什　个　需要超　堆内存的 60% 之多？

TIP

在 Fielddata的大小 中，我　提　于　fielddata 的大小加一个限制，从而　保旧的无用 fielddata　被回收的方法。　`indices.fielddata.cache.size`　和 `indices.breaker.fielddata.limit`　之　的　系非常重要。　如果断路器的限制低于　存大小，没有数据会被回收。　了能正常工作，断路器的限制 必　要比　存大小要高。

得注意的是：断路器是根据　堆内存大小估算　大小的，而　非　根据　堆内存的使用情况。　是由于各　技　原因造成的（例如，堆可能看上去是　的但　上可能只是在等待　回收，　使我以　行合理的估算）。但作　端用　，　意味着　置需要保守，因　它是根据　堆内存必要的，而　不是可用堆内存。

Fielddata 的

想我 正在 行一个 站允 用 收听他 喜 的歌曲。 了 他 可以更容易的管理自己的音，用 可以 歌曲 置任何他 喜 的 ， 我 就会有很多歌曲被附上 rock（ ） 、 hiphop（哈） 和 electronica（ 音） ，但也会有些歌曲被附上 my_16th_birthday_favorite_anthem 的 。

在 想我 想要 用 展示 首歌曲最受 迎的三个 ，很有可能 rock 的 会排在三个中的最前面，而 my_16th_birthday_favorite_anthem 不太可能得到 。 尽管如此，了 算最受 迎的 ，我 必 制将 些一次性使用的 加 到内存中。

感 fielddata ，我 可以控制 状况。我 知道 自己只 最流行的 感 趣，所以我 可以 地避免加 那些不太有意思的 尾 ：

```
PUT /music/_mapping/song
{
  "properties": {
    "tag": {
      "type": "string",
      "fielddata": { ①
        "filter": {
          "frequency": { ②
            "min":              0.01, ③
            "min_segment_size": 500  ④
          }
        }
      }
    }
  }
}
```

① fielddata 字允 我 配置 fielddata 理 字段的方式。

② frequency 器允 我 基于 率 加 fielddata。

③ 只加 那些至少在本段文 中出 1% 的 。

④ 忽略任何文 个数小于 500 的段。

有了 个映射，只有那些至少在 本段 文 中出 超 1% 的 才会被加 到内存中。我 也可以指定一个最大 ，它可以被用来排除 常用 ，比如 停用 。

情况下， 是按照段来 算的。 是 的一个限制：fielddata 是按段来加 的，所以可 的只是 段内的 率。但是， 个限制也有些有趣的特性：它可以 受 迎的新 迅速提升到 部。

比如一个新 格的歌曲在一夜之 受大 迎，我 可能想要将 新 格的歌曲 包括在最受 迎列表中，但如果我 倚 索引做完整的 算 取 ，我 就必 等到新 得像 rock 和 electronica）一 流行。 由于 度 的 方式，新加的 会很快作 高 出在新段内，也当然会迅速上升到 部。

min_segment_size 参数要求 Elasticsearch 忽略某个大小以下的段。 如果一个段内只有少量文 ，它的

会非常粗略没有任何意 。 小的分段会很快被合并到更大的分段中，某一刻超 个限制，将会被入 算。

TIP	通 次来 并不是唯一的 ，我 也可以使用正 式来决定只加 那些匹配的 。例如，我 可以用 `regex` 器 理 twitte 上的消息只将以 `#` 号 始的 加 到内存中。 假 我 使用的分析器会保留 点符号，像 `whitespace` 分析器。

Fielddata 内存使用有 巨大的 影 ， 衡也是 而易 的：我 上是在忽略数据。但 于很多 用， 衡是合理的，因 些数据根本就没有被使用到。内存的 省通常要比包括一个大量而无用的 尾 更 重要。

加 fielddata

Elasticsearch 加 内存 fielddata 的 行 是 延 加 。 当 Elasticsearch 第一次 某个字段 ，它将会完整加 个字段所有 Segment 中的倒排索引到内存中，以便于以后的 能 取更好的性能。

于小索引段来 ， 个 程的需要的 可以忽略。但如果我 有一些 5 GB 的索引段，并希望加 10 GB 的 fielddata 到内存中， 个 程可能会要数十秒。 已 秒 的用 很 会接受停 数秒 着没反 的 站。

有三 方式可以解决 个延 高峰：

- 加 fielddata
- 加 全局序号
- 存

所有的 化都基于同一概念： 加 fielddata， 在用 行搜索 就不会 到延 高峰。

加 fielddata（Eagerly Loading Fielddata）

第一个工具称 加 （与 的 延 加 相 ）。随着新分段的 建（通 刷新、写入或合并等方式）， 字段 加 可以使那些 搜索不可 的分段里的 fielddata 提前 加 。

就意味着首次命中分段的 不需要促 fielddata 的加 ，因 fielddata 已 被 入到内存。避免了用 遇到搜索 的情形。

加 是按字段 用的，所以我 可以控制具体 个字段可以 先加 ：

```
PUT /music/_mapping/_song
{
  "tags": {
    "type": "string",
    "fielddata": {
      "loading" : "eager" ①
    }
  }
}
```

① 置 fielddata.loading: eager 可以告 Elasticsearch 先将此字段的内容 入内存中。

Fielddata 的 入可以使用 update-mapping API 已有字段 置 lazy 或 eager 模式。

WARNING	加 只是 的将 入 fielddata 的代 移到索引刷新的 候，而不是 ，从而大大提高了搜索体 。 体 大的索引段会比体 小的索引段需要更 的刷新 。通常，体 大的索引段是由那些已 可 的小分段合并而成的，所以 慢的刷新 也不是很重要。

全局序号（Global Ordinals）

有 可以用来降低字符串 fielddata 内存使用的技 叫做 序号 。

想我 有十 文 ， 个文 都有自己的 status 状 字段，状 共有三 ： status_pending 、 status_published 、 status_deleted 。如果我 个文 都保留其状 的完整字符串形式，那 个文 就需要使用 14 到 16 字 ，或 共 15 GB。

取而代之的是我 可以指定三个不同的字符串， 其排序、 号：0，1，2。

```
Ordinal | Term
------------------
0       | status_deleted
1       | status_pending
2       | status_published
```

序号字符串在序号列表中只存 一次， 个文 只要使用数 号的序号来替代它原始的 。

```
Doc     | Ordinal
------------------------
0       | 1  # pending
1       | 1  # pending
2       | 2  # published
3       | 0  # deleted
```

可以将内存使用从 15 GB 降到 1 GB 以下！

但 里有个 ， 得 fielddata 是按分 段 来 存的。如果一个分段只包含 个状 （ status_deleted 和 status_published ）。那 果中的序号（0 和 1）就会与包含所有三个状 的分段不一 。

如果我 status 字段 行 terms 聚合，我 需要 字符串的 行聚合，也就是 我需要 所有分段中相同的 。一个 粗暴的方式就是 个分段 行聚合操作，返回 个分段的字符串 ，再将它 得出完整的 果。 尽管 做可行，但会很慢而且大量消耗 CPU。

取而代之的是使用一个被称 全局序号 的 。 全局序号是一个 建在 fielddata 之上的数据 ，它只占用少量内存。唯一 是 跨所有分段 的，然后将它 存入一个序号列表中，正如我描述 的那 。

在， terms 聚合可以 全局序号 行聚合操作，将序号 成真 字符串 的 程只会在聚合 束 生一次。 会将聚合（和排序）的性能提高三到四倍。

建全局序号（Building global ordinals）

当然，天下没有免 的 餐。 全局序号分布在索引的所有段中，所以如果新 或 除一个分段 ，需要 全局序号 行重建。 重建需要 取 个分段的 个唯一 ，基数越高（即存在更多的唯一 ） 个 程会越 。

全局序号是 建在内存 fielddata 和 doc values 之上的。 上，它 正是 doc values 性能表 不 的一个主要原因。

和 fielddata 加 一 ，全局序号 也是延 建的。首个需要 索引内 fielddata 的 求会促 全局序号的 建。由于字段的基数不同， 会 致 用 来 著延 一糟 果。一旦全局序号 生重建， 会使用旧的全局序号，直到索引中的分段 生 化：在刷新、写入或合并之后。

建全局序号（Eager global ordinals）

个字符串字段 可以通 配置 先 建全局序号：

```
PUT /music/_mapping/_song
{
  "song_title": {
    "type": "string",
    "fielddata": {
      "loading" : "eager_global_ordinals" ①
    }
  }
}
```

① 置 eager_global_ordinals 也暗示着 fielddata 是 加 的。

正如 fielddata 的 加 一 ， 建全局序号 生在新分段 于搜索可 之前。

NOTE

序号的 建只被 用于字符串。数 信息（integers（整数）、geopoints（地理 度）、dates（日期）等等）不需要使用序号映射，因 些 自己本 上就是序号映射。

因此，我 只能 字符串字段 建其全局序号。

也可以 Doc values 行全局序号 建：

```
PUT /music/_mapping/_song
{
  "song_title": {
    "type":       "string",
    "doc_values": true,
    "fielddata": {
      "loading" : "eager_global_ordinals" ①
    }
  }
}
```

① 情况下，fielddata 没有 入到内存中，而是 doc values 被 入到文件系 存中。

与 fielddata 加 不一 ， 建全局序号会 数据的 性 生影 ， 建一个高基数的全局序号会使一个刷新延 数秒。 在于是 次刷新 付出代 ， 是在刷新后的第一次 。如果 常索引而 少，那 在 付出代 要比 次刷新 要好。如果写大于 ，那 在 在 重建全局序号将会是一个更好的 。

TIP 景 化全局序号的重建 次。如果我 有高基数字段需要花数秒 重建， 加 `refresh_interval` 的刷新的 从而可以使我 的全局序号保留更 的有效期， 也会 省 CPU 源，因 我 重建的 次下降了。

索引 器（Index Warmers）

最后我 索引 器 。 器早于 fielddata 加 和全局序号 加 之前出 ，它 然尤其存在的理由。一个索引 器允 我 指定一个 和聚合 要在新分片 于搜索可 之前 行 。 个想法是通 先填充或 存 用 永 无法遇到延 的波峰。

原来， 器最重要的用法是 保 fielddata 被 先加 ，因 通常是最耗 的一 。 在可以通前面 的那些技 来更好的控制它，但是 器 是可以用来 建 器 存，当然我 也 是能 用它来 加 fielddata。

我 注 一个 器然后解 生了什 ：

```
PUT /music/_warmer/warmer_1 ①
{
  "query" : {
    "bool" : {
      "filter" : {
        "bool": {
          "should": [ ②
            { "term": { "tag": "rock"        }},
            { "term": { "tag": "hiphop"      }},
            { "term": { "tag": "electronics" }}
          ]
        }
      }
    }
  },
  "aggs" : {
    "price" : {
      "histogram" : {
        "field" : "price", ③
        "interval" : 10
      }
    }
  }
}
```

① 器被 到索引（music）上，使用接入口 _warmer 以及 ID （warmer_1）。

② 三 最受 迎的曲 建 器 存。

③ 字段 price 的 fielddata 和全局序号会被 加 。

器是根据具体索引注 的， 个 器都有唯一的 ID，因 个索引可能有多个 器。

然后我 可以指定 ，任何 。它可以包括 、 器、聚合、排序 、脚本，任何有效的 表式都 不夸 。 里的目的是想注 那些可以代表用 生流量 力的 ，从而将合 的内容 入 存。

当新建一个分段 ，Elasticsearch 将会 行注 在 器中的 。 行 些 会 制加 存，只有在所有 器 行完， 个分段才会 搜索可 。

WARNING

与 加 似， 器只是将冷 存的代 移到刷新的 候。当注 器 ，做出明智的决定十分重要。 了 保 个 存都被 入，我 可以 加入上千的 器，但 也会使新分段 于搜索可 的 急 上升。

中，我 会 少量代表大多数用 的 ，然后注 它 。

有些管理的 （比如 得已有 器和 除 器）没有在本小 提到，剩下的 内容可以参考 {ref}/indices-warmers.html[器文 （warmers documentation）]。

化聚合

“elasticsearch 里面桶的叫法和 SQL 里面分 的概念是 似的，一个桶就 似 SQL 里面的一个 group，多 嵌套的 aggregation， 似 SQL 里面的多字段分 （group by field1,field2, ...），注意 里 是概念 似，底 的 原理是不一 的。 — 者注”

terms 桶基于我 的数据 建桶；它并不知道到底生成了多少桶。 大多数 候 个字段的聚合 是非常快的， 但是当需要同 聚合多个字段 ，就可能会 生大量的分 ，最 果就是占用 es 大量内存，从而 致 OOM 的情况 生。

假 我 在有一些 于 影的数据集， 条数据里面会有一个数 型的字段存 表演 影的所有演 的名字。

```
{
  "actors" : [
    "Fred Jones",
    "Mary Jane",
    "Elizabeth Worthing"
  ]
}
```

如果我 想要 出演影片最多的十个演 以及与他 合作最多的演 ，使用聚合是非常 的：

```
{
  "aggs" : {
    "actors" : {
      "terms" : {
         "field" : "actors",
         "size" :  10
      },
      "aggs" : {
        "costars" : {
          "terms" : {
            "field" : "actors",
            "size" :  5
          }
        }
      }
    }
  }
}
```

会返回前十位出演最多的演 ，以及与他 合作最多的五位演 。 看起来是一个 的聚合 ，最 只返回 50 条数据！

但是， 个看上去 的 可以 而易 地消耗大量内存，我 可以通 在内存中 建一个 来 看 个 terms 聚合。 actors 聚合会 建 的第一 ， 个演 都有一个桶。然后，内套在第一 的 个 点之下， costar 聚合会 建第二 ， 个 合出演一个桶， 参 Build full depth tree 所示。 意味着

部影片会生成 n^2 个桶！

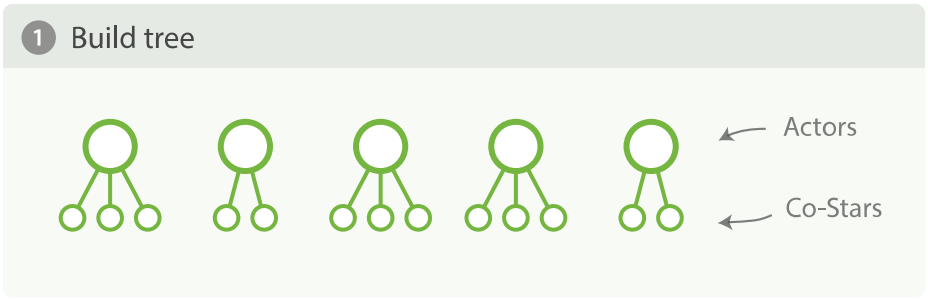

Figure 1. Build full depth tree

用真 点的数据， 想平均 部影片有 10 名演 ， 部影片就会生成 10^2 == 100 个桶。如果 共有 20 ，000 部影片，粗率 算就会生成 2，000，000 个桶。

在， 住，聚合只是 的希望得到前十位演 和与他 合出演者， 共 50 条数据。 了得到最的 果，我 建了一个有 2，000，000 桶的 ，然后 其排序，取 top10。 Sort tree 和 Prune tree 个 程 行了 述。

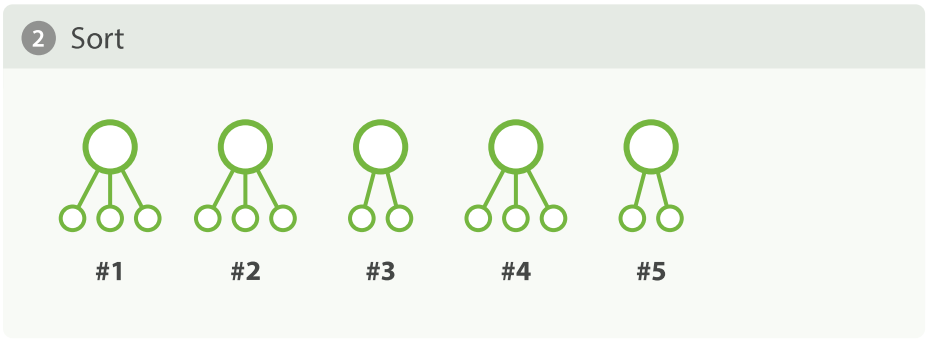

Figure 2. Sort tree

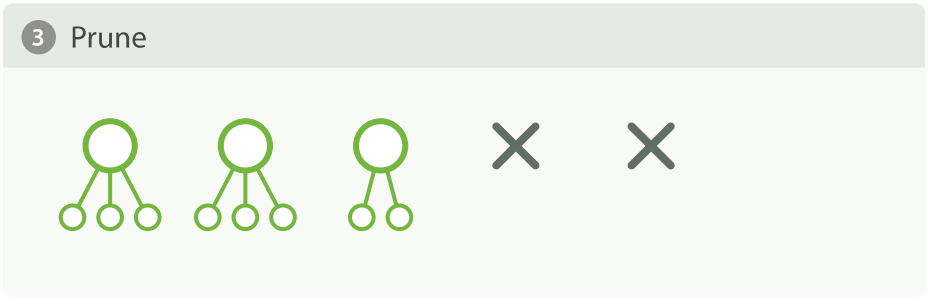

Figure 3. Prune tree

我 一定非常 狂，在 2 万条数据下 行任何聚合 都是 无 力的。如果我 有 2 文 ，想要得到前 100 位演 以及与他 合作最多的 20 位演 ，作 的最 果会出 什 情况 ？

可以推 聚合出来的分 数非常大，会使 策略 以 持。世界上并不存在足 的内存来支持 不受控制的聚合 。

深度　先与广度　先（Depth-First Versus Breadth-First）

Elasticsearch 允　我　改　聚合的 集合模式 ，就是　了　　状况。 我　之前展示的策略叫做 深度　先　　，它是　　置，　　先　建完整的　，然后修剪无用　点。　　深度　先　　的方式　于大多数聚合都能正常工作，但　于如我　演　和　合演　　例子的情形就不太　用。

　了　　些特殊的　用　景，我　　使用　一　集合策略叫做　　广度　先　　。　策略的工作方式有些不同，它先　行第一　聚合，　再　　下一　聚合之前会先做修剪。 Build first level 和　Prune first level　　个　程　行了　述。

在我　的示例中，　`actors`　聚合会首先　行，在　个　候，我　的　只有一　，但我　已　知道了前　10 位的演　！　就没有必要保留其他的演　信息，因　它　无　如何都不会出　在前十位中。

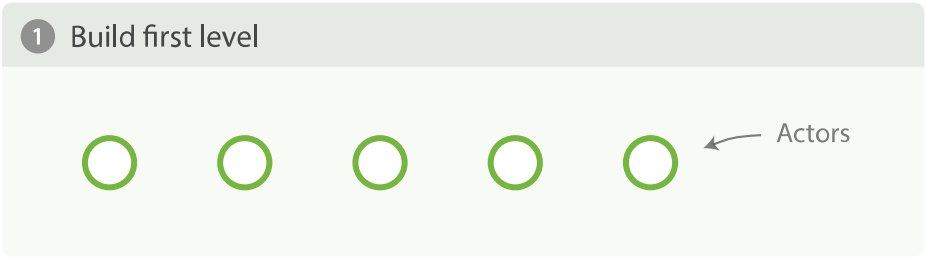

Figure 4. Build first level

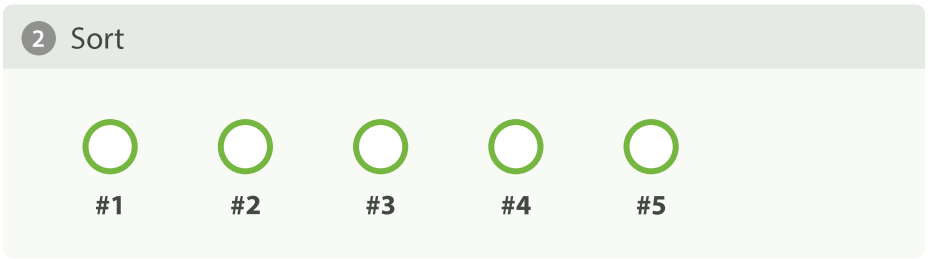

Figure 5. Sort first level

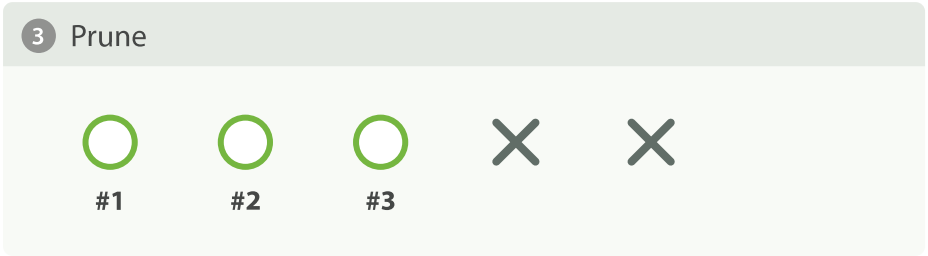

Figure 6. Prune first level

因　我　已　知道了前十名演　，我　可以安全的修剪其他　点。修剪后，下一　是基于　它的　　行模式

入的，重　　行　个　程直到聚合完成，如　　Populate full depth for remaining nodes 所示。
　　景下，广度　先可以大幅度　省内存。

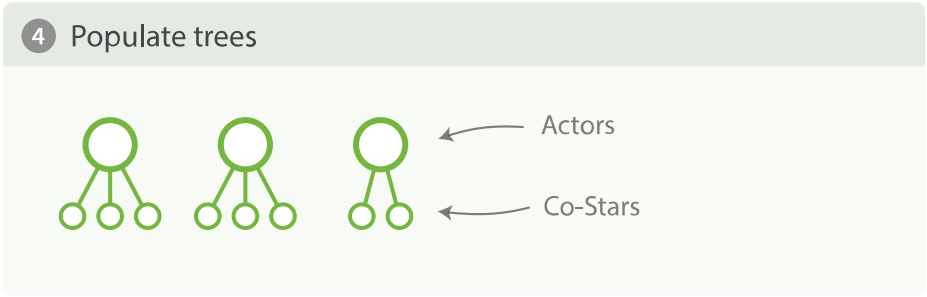

Figure 7. Populate full depth for remaining nodes

要使用广度　先，只需　　的通　参数 `collect`　　：

```
{
  "aggs" : {
    "actors" : {
      "terms" : {
         "field" :        "actors",
         "size" :         10,
         "collect_mode" : "breadth_first" ①
      },
      "aggs" : {
        "costars" : {
          "terms" : {
            "field" : "actors",
            "size" :  5
          }
        }
      }
    }
  }
}
```

① 按聚合来　　`breadth_first`。

广度　先　　用于　个　的聚合数量　　小于当前　　数的情况下，因　广度　先会在内存中　存裁剪后的　　需要　存的　个　的所有数据，以便于它的子聚合分　　可以　用上　聚合的数据。

广度　先的内存使用情况与裁剪后的　存分　数据量是成　性的。　于很多聚合来　，　个桶内的文　数量是相当大的。　　想象一　按月分　的直方　，　　数肯定是固定的，因　　年只有12个月，　个　候　个月下的数据量可能非常大。　使广度　先不是一个好的　　，　也是　什　深度　先作　　策略的原因。

　　上面演　的例子，如果数据量越大，那　　的使用深度　先的聚合模式生成的　分　数就会非常多，但是　估二　的聚合字段分　后的数据量相比　的分　数会小很多所以　　情况下使用广度　先的模式能大大　省内存，从而通　　化聚合模式来大大提高了在某些特定　景下聚合　　的成功率。

本小 涵 了 多基本理 以及很多深入的技 。聚合 Elasticsearch 来了 以言 的大能力和 活性。桶与度量的嵌套能力，基数与百分位数的快速估算能力，定位信息中 常的能力，所有的 些都在近乎 的情况下操作的，而且全文搜索是并行的，它 改 了很多企 的游 。

聚合是 一 功能特性：一旦我 始使用它，我 就能 到很多其他的可用 景。 表与分析 于很多 来 都是核心功能（无 是 用于商 智能 是服 器日志）。

Elasticsearch 大多数 字段 用 doc values，所以在一些搜索 景大大的 省了内存使用量，但是需要注意的是只有不分 的 string 型的字段才能使用 特性。

内存的管理形式可以有多 形式， 取决于我 特定的 用 景：

- 在 ， 好数据，使聚合 行在 `not_analyzed` 字符串而不是 `analyzed` 字符串，可以有效的利用 doc values 。
- 在 ， 分析 不会在之后的聚合 算中 建高基数字段。
- 在搜索 ，合理利用近似聚合和数据 。
- 在 点 ， 置硬内存大小以及 的断熔限制。
- 在 用 ，通 控集群内存的使用情况和 Full GC 的 生 率，来 整是否需要 集群源添加更多的机器 点

大多数 施会 用到以上一 或几 方法。 切的 合方式与我 特定的系 境高度相 。

无 采取何 方式， 于 有的 行 估，并同 建短期和 期 ，都十分重要。先决定当前内存的使用情况和需要做的事情（如果有），通 估数据 速度，来决定未来半年或者一年的集群的 ，使用何 方式来 展。

最好在建立集群之前就 好 些内容，而不是在我 集群堆内存使用 90% 的 候再 抱佛脚。