Elasticsearch 是默认延迟加载fielddata到内存里的。当elasticsearch第一次遇到一个查询需要一个指定field的fielddata的时候，就会把索引的每个段中整个field加载到内存。对于小段，这是个可以忽略不计的时间，但是如果你有一些5G大小的段并且需要加载10GB的fielddata到内存里，这个过程需要数十秒，习惯于秒内响应时间的用户会被网突如其来的迟钝所打击。

有三种方法应对这种延迟尖峰：  
Eagerly load fielddata（饿汉式（预）加载fielddata）  
Eagerly load global ordinals（饿汉式（预）加载全局序数）  
Prepopulate caches with warmers （使用warmer提前加载缓存）。

所有这些都是一个意思：预先加载fielddata到内存里，这样当用户需要执行一个搜索的时候就感受不到延迟了。

饿汉式（预）加载fielddata

首先是预先加载（而不是默认的延迟加载），当一个新的段形成时（无乱是刷新，写入或者是合并），可以预先加载的field会提前把段的fielddata加载到内存里，在这段可以用于搜索之前。

这意味着当你第一次查询的时候，如果碰到在这个段上，你不需要再触发加载fielddata的操作，它们已经在内存中了，这会防止你的用户遇到一些冷到缓存而发生延迟尖峰。

预先加载是基于每个field的，所以你可以控制哪些field进行预加载。

PUT /music/\_mapping/\_song

{

"price\_usd": {

"type": "integer",

"fielddata": {

"loading" : "eager"

}

}

}

备注: 通过设置fielddata.loading: eager，告诉elasticsearch预先加载这个field的内容到内存里。

fielddata的加载可以设置成饿汉模式（预先加载）还是懒惰模式（延迟加载），使用update-mapping的api。

预先加载是简单对fielddata加载的开销的转移，从查询时间转义到刷新时刻。

大段的刷新时间会比小段的时间长，通常大段的产生都是由哪些已经可搜素的小段合并而来的，所以慢一点的刷新时间不是那么重要（译者注：意思是大段的刷新时间长不影响你的搜索，在大段合并成前的小段可以用于搜索）。

****全局序数****

其中一项用于减少string类型的fielddata占用内存的技术叫做序数。

假设我们有十亿条文档，每个文档都有一个status的field，只有三个值：status\_pending, status\_published, status\_deleted，如果我们把所有的status加载到内存里，每个文档需要14-16byte，也就是说15GB

相反，我们可以确认这三个特殊的字符串，对他们排序，依次编号0,1,2

Ordinal | Term

-------------------

0 | status\_deleted

1 | status\_pending

2 | status\_published

序号对应的字符串值要在序号列表中存储一次，每个文档只要使用他们的编号来表示他们所包含的值就可以了。

Doc | Ordinal

-------------------------

0 | 1 # pending

1 | 1 # pending

2 | 2 # published

3 | 0 # deleted

这个可以把15GB的内存占用减少到小于1GB

但是这里有个问题，记住fielddata缓存是针对每个段的，如果一个段包含两个状态-status\_deleted 和 status\_published，这个序号是(0 和1），如果其它段中有三个不同的状态，那相同的序号的含义在两个段中是不同的。

如果我们试图执行一个status 的field的汇总操作（aggregation ），我们需要在实际的字符串上进行汇总。以为着我们要在所有的段上识别出有相同值的文档，一个单纯的方式就是在每个段上进行汇总操作，从每个段上返回字符串的值，然后合并他们得到最终的结果，这样是可行的，这会很慢很耗CPU。

想反，我们实验了一个数据结构叫做全局序数，全局序号只是在fielddata之上，一小部分内存数据结构，把所有的段的唯一值存储在一个序号列表里，就像前面我们描述的那样。

现在，我们的汇总操作只要在全局序号上进行汇总，把序号转换成实际字符串，只要在最后一个汇总中进行就可以了，这大大提高了在只要三四个值的field上进行汇总（还有排序）操作的性能。

****构建全局序数****

当然，天下没有免费的午餐，全局序数是针对一个索引的所有段的，所以如果增加了一个段或者删除了一个段，这个全局序数都要进行重建，长袖需要读取每个段的每个唯一term。存储基数越大，唯一tems越多，这个过程会更长。

全局序数是创建在内存里的fielddata和doc values之上的，实际上，他们也是doc values性能表现的主要原因。

和fielddata的加载一样，全局序数的构建默认也是延迟进行的，当第一个请求需要fielddata时候，会触发全局序数的构建，和你的field的基数有关，这可能导致用户的一个延迟响应。一旦全局序数构建完成，他们将被使用，直到索引中的段发生变化：刷新，写入和合并。

****饿汉式（预先加载）全局序数****

PUT /music/\_mapping/\_song

{

"song\_title": {

"type": "string",

"fielddata": {

"loading" : "eager\_global\_ordinals"

}

}

}

注：设置为eager\_global\_ordinals 也代表实现了预先加载fielddata。

就像预先加载fielddata一样，全局序数会在一个新的段可进行搜索之前进行构建。

注意：序数只用于字符串类型的field，数值类型(整型，地理位置，时间等）都不需要一个序数映射，因为他们自身就是一个内存的序数映射。

所以，你只能开启字符串类型的全局序数的预加载。

文档的值也可以有他们的全局序数：

PUT /music/\_mapping/\_song

{

"song\_title": {

"type": "string",

"doc\_values": true,

"fielddata": {

"loading" : "eager\_global\_ordinals"

}

}

}

注：这种情况，fielddata就不加载到内存里了，而是文档的values会被加载到系统级缓存里。

和fielddata的预加载不同，对全局序数的预加载会对数据的实时性有影响，队友一个很大基数的field，构建全局序数会导致数秒的延迟刷新，是把时间花在每次刷新上呢，还是每次刷新之后的第一次查询上面这是一个选择。如果你插入数据频繁，查询很少，最好是把代价花在查询时刻，而不是每次刷新上。

注意：让你的全局序号自己处理，如果你有一个很大基数的field，导致很长时候构建，你可以增加refresh\_interval 配置，这样回增加全局序数的有效时间，减少CPU的利用率，同样构建全局序数的次数也会减少。

****Index warmers（索引热身）****

最后，我们到了index warmers，wamers可以提早加载fielddata和全局序数，但是他还有一个目的。一个索引的warmer允许你指定一个查询或者汇聚操作在你的新段可被查询之前执行，这个idea就叫做预热、或者叫暖身，缓存等，那你的用户就永远不会遇到延迟尖峰时刻了。

期初，这个warmers的最重要的用处是为了确保fielddata的预先加载，这通常是开销最大的部分，这是我们之前讨论的利用这个技术控制的最好的地方，然而，warmers可以用于构建filter 缓存，也可以用于预加载fielddata，这个目的随你选择。

先让我们注册一个warner再去讨论发生了什么：

PUT /music/\_warmer/warmer\_1

{

"query" : {

"filtered" : {

"filter" : {

"bool": {

"should": [

{ "term": { "tag": "rock" }},

{ "term": { "tag": "hiphop" }},

{ "term": { "tag": "electronics" }}

]

}

}

}

},

"aggs" : {

"price" : {

"histogram" : {

"field" : "price",

"interval" : 10

}

}

}

}

注：这表示在music索引上注册一个id为warmer\_1的warmer，三大流行音乐流派的filter 缓存都会被预先创建，对price字段的fielddata和全局序数都会被预先加载

warmers是注册在一个特定的索引上面的，每一个warmer都有一个唯一的id，因为你可以为一个索引指定多个warmer。

然后你可以之低昂一个查询，或者多个查询，他包含查询语句，filters，aggregations，sort，脚本等有效的查询语句。重点是要注册那些你的用户将要进行，有代表性的查询，从而适当的缓存可以被预先加载。

当一个新的段产生的时候，elasticsearch就会执行你这些warmers里的查询语句。执行这些查询会导致缓存进行加载，只有索引warmers都被执行了，新的段才会变成可搜素状态。

注意：类似于预先加载，warmers 把对冷缓存的开销转移到了刷新时刻。当你注册了warmers，你必须是经过明智判断的。你可以增加成千上万个wamer确保预热所有的缓存，但是这将极大的增加一个新的段从创建变为可搜索的时间。

实践中，选择几条代表大部分用户的查询语句进行注册。

一些管理方面的细节（例如获取存在的warmer，删除warmer）就不在这里过多解释了，阅读warmer相关文档，获取其它细节。

原文地址：https://www.elastic.co/guide/en/elasticsearch/guide/current/preload-fielddata.html