在ElasticSearch 2.4版本中，文档存储的介质分为内存和硬盘：内存速度快，但是容量有限；硬盘速度较慢，但是容量很大。同时，ElasticSearch进程自身的运行也需要内存空间，必须保证ElasticSearch进程有充足的运行时内存。为了使ElasticSearch引擎达到最佳性能，必须合理分配有限的内存和硬盘资源。

一，倒排索引（Inverted Index）

ElasticSearch引擎把文档数据写入到倒排索引（Inverted Index）的数据结构中，倒排索引建立的是分词（Term）和文档（Document）之间的映射关系，在倒排索引中，数据是面向词（Term）而不是面向文档的。
举个例子，文档和词条之间的关系如下图：

DocID	Field
1	Study after school
2	Study English

字段值被分析之后，存储在倒排索引中，倒排索引存储的是分词（Term）和文档（Doc）之间的关系，简化版的倒排索引如下图：

Term	Counter	DocID
After	1	1
School	1	1
Study	2	1,2
English	1	2

从图中可以看出，倒排索引有一个词条的列表，每个分词在列表中是唯一的，记录着词条出现的次数，以及包含词条的文档。实际上，ElasticSearch引擎创建的倒排索引比这个复杂得多。

1，段是倒排索引的组成部分

倒排索引是由段（Segment）组成的，段存储在硬盘（Disk）文件中。索引段不是实时更新的，这意味着，段在写入硬盘之后，就不再被更新。在删除文档时，ElasticSearch引擎把已删除的文档的信息存储在一个单独的文件中，在搜索数据时，ElasticSearch引擎首先从段中执行查询，再从查询结果中过滤被删除的文档，这意味着，段中存储着被删除的文档，这使得段中含有”正常文档“的密度降低。多个段可以通过段合并（Segment Merge）操作把“已删除”的文档将从段中物理删除，把未删除的文档合并到一个新段中，新段中没有”已删除文档“，因此，段合并操作能够提高索引的查找速度，但是，段合并是IO密集型操作，需要消耗大量的硬盘IO。

在ElasticSearch中，大多数查询都需要从硬盘文件（索引的段数据存储在硬盘文件中）中获取数据，因此，在全局配置文件elasticsearch.yml 中，把**结点的路径（Path）**配置为性能较高的硬盘，能够提高查询性能。默认情况下，ElasticSearch使用基于安装目录的相对路径来配置结点的路径，安装目录由属性path.home显示，在home path下，ElasticSearch自动创建config，data，logs和plugins目录，一般情况下不需要对结点路径单独配置。结点的文件路径配置项：

- **path.data：设置ElasticSearch结点的索引数据保存的目录，多个数据文件使用逗号隔开，例如，path.data: /path/to/data1,/path/to/data2；**
- **path.work：设置ElasticSearch的临时文件保存的目录；**

2，分词和原始文本的存储

映射参数index决定ElasticSearch引擎是否对文本字段执行分析操作，也就是说分析操作把文本分割成一个一个的分词，也就是标记流（Token Stream），把分词编入索引，使分词能够被搜索到：

- 当index为analyzed时，该字段是分析字段，ElasticSearch引擎对该字段执行分析操作，把文本分割成分词流，存储在倒排索引中，使其支持全文搜索；
- 当index为not_analyzed时，该字段不会被分析，ElasticSearch引擎把原始文本作为单个分词存储在倒排索引中，不支持全文搜索，但是支持词条级别的搜索；也

就是说，字段的原始文本不经过分析而存储在倒排索引中，把原始文本编入索引，在搜索的过程中，查询条件必须全部匹配整个原始文本；

- 当index为no时，该字段不会被存储到倒排索引中，不会被搜索到；

字段的原始值是否被存储到倒排索引，是由映射参数store决定的，默认值是false，也就是，原始值不存储到倒排索引中。

映射参数index和store的区别在于：

- **store用于获取（Retrieve）字段的原始值，不支持查询，可以使用投影参数fields，对stroe属性为true的字段进行过滤，只获取（Retrieve）特定的字段，减少网络负载；**
- **index用于查询（Search）字段，当index为analyzed时，对字段的分词执行全文查询；当index为not_analyzed时，字段的原始值作为一个分词，只能对字段的原始文本执行词条查询；**

3，单个分词的最大长度

如果设置字段的index属性为not_analyzed，原始文本将作为单个分词，其最大长度跟UTF8 编码有关，默认的最大长度是32766Bytes，如果字段的文本超过该限制，那么ElasticSearch将跳过（Skip）该文档，并在Response中抛出异常消息：

> operation[607]: index returned 400 _index: ebrite _type: events _id: 76860 _version: 0 error: Type: illegal_argument_exception Reason: "Document contains at least one immense term in field="event_raw" (whose UTF8 encoding is longer than the max length 32766), all of which were skipped. Please correct the analyzer to not produce such terms. The prefix of the first immense term is: '[112, 114,... 115]...', original message: bytes can be at most 32766 in length; got 35100" CausedBy:Type: max_bytes_length_exceeded_exception Reason: "bytes can be at most 32766 in length; got 35100"

可以在字段中设置ignore_above属性，该属性值指的是字符数量，而不是字节数量；由于一个UTF8字符最多占用3个字节，因此，可以设置

"ignore_above"：10000

这样，超过30000字节之后的字符将会被分析器忽略，单个分词（Term）的最大长度是30000Bytes。

> The value for `ignore_above` is the *character count*, but Lucene counts bytes. If you use UTF-8 text with many non-ASCII characters, you may want to set the limit to `32766 / 3 = 10922` since UTF-8 characters may occupy at most 3 bytes.

二，列式存储（doc_values）

默认情况下，大多数字段被索引之后，能够被搜索到。倒排索引是由一个有序的词条列表构成的，每一个词条在列表中都是唯一存在的，通过这种数据存储模式，你可以很快查找到包含某一个词条的文档列表。但是，排序和聚合操作采用相反的数据访问模式，这两种操作不是查找词条以发现文档，而是查找文档，以发现字段中包含的词条。ElasticSearch使用列式存储实现排序和聚合查询。

文档值**（doc_values）**是存储在硬盘上的数据结构，在索引时（index time）根据文档的原始值创建，文档值是一个列式存储风格的数据结构，非常适合执行存储和聚合操作，除了字符类型的分析字段之外，其他字段类型都支持文档值存储。默认情况下，字段的文档值存储是启用的，除了字符类型的分析字段之外。如果不需要对字段执行排序或聚合操作，可以禁用字段的文档值，以节省硬盘空间。

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

三，顺排索引（fielddata）

字符类型的分析字段，不支持文档值（doc_values），但是，支持fielddata数据结构，fielddata数据结构存储在JVM的堆内存中。相比文档值（数据存储在硬盘上），fielddata字段（数据存储在内存中）的查询性能更高。默认情况下，ElasticSearch引擎在第一次对字段执行聚合或排序查询时（（query-time）），创建fielddata数据结构；在后续的查询请求中，ElasticSearch引擎使用fielddata数据结构以提高聚合和排序的查询性能。

在ElasticSearch中，倒排索引的各个段（segment）的数据存储在硬盘文件上，从整个倒排索引的段中读取字段数据之后，ElasticSearch引擎首先反转**词条和文档**之间的关系，创建**文档和词条**之间的关系，即创建顺排索引，然后把顺排索引存储在JVM的堆内存中。把倒排索引加载到fielddata结构是一个非常消耗硬盘IO资源的过程，因此，数据一旦被加载到内存，最好保持在内存中，直到索引段(segment)的生命周期结束。默认情况下，倒排索引的每个段(segment)，都会创建相应的fielddata结构，以存储字符类型的分析字段值，但是，需要注意的是，分配的JVM堆内存是有限的，Fileddata把数据存储在内存中，会占用过多的JVM堆内存，甚至耗尽JVM赖以正常运行的内存空间，反而会降低ElasticSearch引擎的查询性能。

1，format属性

fielddata会消耗大量的JVM内存，因此，尽量为JVM设置大的内存，不要为不必要的字段启用fielddata存储。通过format参数控制是否启用字段的fielddata特性，字符类型的分析字段，fielddata的默认值是paged_bytes，这就意味着，默认情况下，字符类型的分析字段启用fielddata存储。一旦禁用fielddata存储，那么字符类型的分析字段将不再支持排序和聚合查询。

```
"mappings": {
   "my_type": {
     "properties": {
       "text": {
         "type": "string",
         "fielddata": {
           "format": "disabled"
         }
       }
     }
   }
 }
```

2，加载属性（loading）

loading属性控制fielddata加载到内存的时机，可能的值是lazy，eager和eager_global_ordinals，默认值是lazy。

- lazy：fielddata只在需要时加载到内存，默认情况下，在第一次搜索时，fielddata被加载到内存中；但是，如果查询一个非常大的索引段（Segment），lazy加载方式会产生较大的时间延迟。
- eager：在倒排索引的段可用之前，其数据就被加载到内存，eager加载方式能够减少查询的时间延迟，但是，有些数据可能非常冷，以至于没有请求来查询这些数据，但是冷数据依然被加载到内存中，占用紧缺的内存资源。
- eager_global_ordinals：按照global ordinals积极把fielddata加载到内存。

四，JVM进程使用的内存和堆内存

1，配置ElasticSearch使用的内存

ElasticSearch使用JAVA_OPTS环境变量（Environment Variable）启动JVM进程，在JAVA_OPTS中，最重要的配置是：-Xmx参数控制分配给JVM进程的最大内存，-Xms参数控制分配给JVM进程的最小内存。通常情况下，使用默认的配置就能满足工程需要。
ES_HEAP_SIZE 环境变量控制分配给JVM进程的堆内存（Heap Memory）大小，顺排索引（fielddata）的数据存储在堆内存（Heap Memory）中。

2，内存锁定

大多数应用程序尝试使用尽可能多的内存，并尽可能把未使用的内存换出，但是，内存换出会影响ElasticSearch引擎的查询性能，推荐启用内存锁定，禁用ElasticSearch内存的换进换出。
在全局配置文档 elasticsearch.yml中，设置 bootstrap.memory_lock为ture，这将锁定ElasticSearch进程的内存地址空间，阻止ElasticSearch内存被OS换出（Swap out）。