## \_source字段

\_source字段是一个自动生成的字段，用来存储实际提交的JSON数据（即存储的索引的原始文档），他是不索引的（不可搜索的），只用来存储。自动0.90开始，所以存储的字段（包括\_source）总是被压缩的。

将一个field的store属性设置为true,这个会在lucene层面处理（即存储也是在lucene层面）

es并不需要单独存储需要返回的每一个field的值，因为默认情况下每一个文档的完整信息都已经存储了（存储在\_source字段中）

（从stored field中获取值都需要一次磁盘io，如果想获取多个field的值，就需要多次磁盘io，但是从\_source字段中获取多个field的值，只需要一次磁盘io，因为\_source只是一个字段而已）

目前来看，可以通过修改源码来实现插件修改（好像不对）

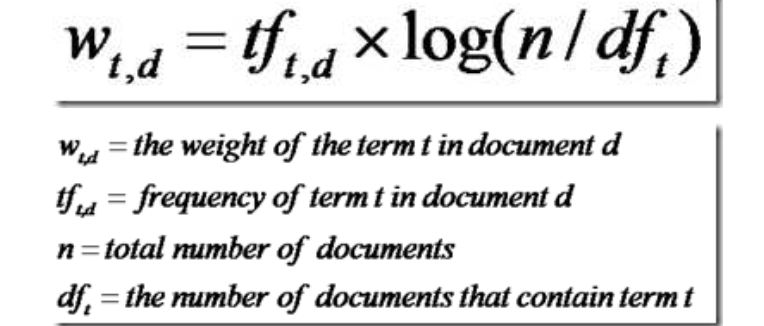
## Lucene层

倒排表而言，文档号的排序是看评分来的？然后按名称？

### 词的权重

影响一个词在一篇文档的重要性主要有两个因素：

1. Term frequency（tf）：即此term在此文档出现了多少次。Tf越大说明越重要。
2. Document frequency（df）：即有多少文档包含此term。Df越大说明越不重要。



从判断term之间的关系从而得到文档相关性的过程（即向量空间模型的算法vsm）

## 测试

准备几个索引，indextest1是自动映射的索引、indextest2是存储字段内容的索引、indextest3是禁用source字段的索引、indextest4是禁用all字段的索引、indextest5是禁用\_source禁用\_all的索引、indextest6是禁用\_source禁用\_all禁用doc\_value

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Fdt（此段包含的文档，文档包含的域以及域的信息） | Dvd（保存被docvalue标记的字段建立的文档到值的映射关系） | Tim（词典） | Doc（词频率信息） | Pos（词位置信息） |
| Indextest1  (自动映射) | 70791 | 27125 | 26680 | 38135 | 8156 |
| Indextest2  （字段存储） | 103676 | 25547 | 25177 | 36901 | 8394 |
| Indextest3  禁用source | 4899 | 24158 | 23480 | 33723 | 6703 |
| Indextest4  禁用all | 64125 | 26698 | 15803 | 25521 | 450 |
| Indextest5禁用source禁用all | 5617 | 27700 | 16194 | 25902 | 311 |
| Indextest6  Source、all、docvalue全禁用 | 5616 | 125 | 16203 | 25869 | 308 |

## Lucene文件结构

参考：<http://www.cnblogs.com/seaspring/articles/5459186.html>

### Segment\_N文件

这里描述的结构是由header（版本数字和用以标识文件的字符串）、version（记录了索引被更改的次数）、namecounter（用于生成新的段文件的名字）、segcount（包含的段的数量）、segname（段的名字，同时用于同一个段的所有索引文件的文件名前缀）、delgen（段执行删除文件操作的次数）、DeletionCount（段中被删除即标记为删除的文件的数量）、checksum（用于验证打开的索引文件的完整性）、segcodec（段的编解码器的名字）、commituserdata（存储一个用户从字符串到字符串的映射map？）、fieldinfosgen（记录段生成fieldinfos文件的次数？）、updatesfiles（存储在段中更新过的文件的列表）

通过其文件可以查看的部分，得到的信息包括该文件的后缀，节点包含的段的名称，段的格式是lucene54，translog\_uuid、translog\_generation、sync\_id等

### Write.lock文件

这个文件存在的目的确保同一个时间只有一个writer修改index

### SI文件

SegVersion（段所基于的lucene的版本）、SegSize（段所包含的文档数）、InCompundFile（记录了这个段是不是合并过的，-1不是，1是）、CheckSum（用于确认文件的完整性）、The FiaGnostics Map（包含了lucene的版本，os信息，java版本，段创建的信息等元信息）、Attributes（解编码器自己的属性，是个map）、files（段所包含的文件列表）

实际文件中可以看出来包含的内容有The FiaGnostics Map、时间信息、包含的文件，其它内容暂无法解读。

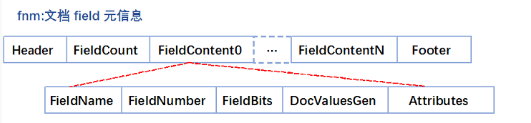
### Cfs、cfe文件

复合索引文件

Cfe文件较小，主要内容就是包含的文件的信息，包括一个header、文件数、文件信息（文件名，数据偏移量，数据长度）

Cfs存储文件内容，包括一个header，然后就是filecount个filedata

### Fnm文件



段的域信息，包括域个数，名称，类型等信息

包括header、fieldscount（域数量）、fieldscount个fieldinfo【包括FieldName（域名称）、FieldNumber（域的数量）、FieldBits（一个字节，包含了域的各种选项，从低位开始依次是：

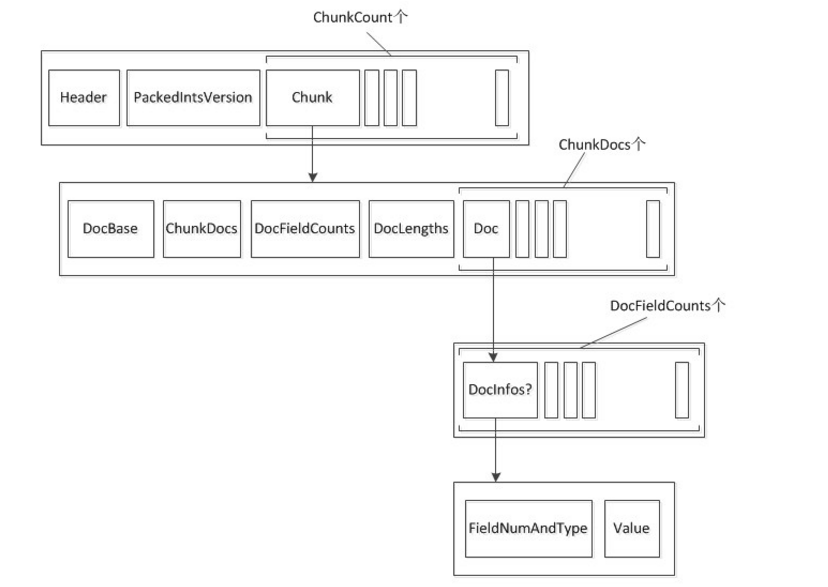
1. 标志这个域是否是索引的（1表示索引，0表示没有）
2. 标志这个域是否有term向量。1表示有，0表示没有
3. 标志偏移量是否设置。如果是1则在位置后添加偏移量
4. 没有使用
5. 如果设置，这个域忽略标准化（？）
6. 如果设置，则用户的有效荷载（payload）会被存储在这个域。
7. 如果设置，term的频率和位置会被忽略
8. 如果设置，term的位置会被忽略

）、DocValuesBits（一个字节，记录了每个文档的值类型。低位的4个bit表示DocValues的可选特征，高危的四个bit则表示了标准化的可选特征。Docvalues的选择包括，0：域没有docvalues，1：numericDocValues，2：BinaryDocValues，3：SortedDocValues）、DocValuesGen（表示DocValues的代数，如果是-1，表示这个域没有更新DocValues）、Attributes（编解码器自己的属性，是一个map）】

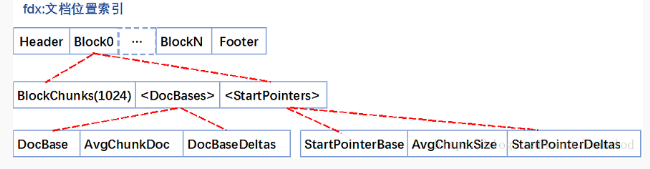
实际文件中能直观看出来的信息主要是域的信息内容

### Fdt、fdx文件

Fdt文件保存此段包含的文档，每篇文档中包含的域以及每个域的信息。Fdx保存元信息。



Chunkcount（表示段中存储所有文档所需要的chunk数）、chunk【包括DocBase（这个chunk中的第一篇文档的ID）、ChunkDocs（这个chunk所包含的文档数）以及DocFieldCount（这个chunk中所包含的的每一个文档所包含的域的数量）、Doclengths（这个chunk中所有文档的长度）、以及chunkdocs个doc】，doc的内容如下：包含DocFieldCount个域信息（包括FieldNumAndType，一个VLong类型，最后三位表示Type（0为string、1为BinaryValue、2为Int、3为Float、4为Long、5为Double、6,7值未使用），其它的位则是FieldNum；Value也就是具体的值，值的类型依据type，其中BinaryValue的格式为BinaryValue->ValueLength ValueLength）



Fdx主要内容就是chunkindex的内容，包括PackedIntsVersion、BlocksEndMarker（0标志着块的结束）、Block【blockChunks（块中编码的块数）、DocBase（文档的基础信息，包括DocBase（块的第一个文档id）、AvgChunkDocs（单个块中的平均文档数）、BitsPerDocBaseDelta（使用ZigZag编码从平均值表示增量所需的位数）、DocBaseDelta（？））、StartPointers（包括StartPointerBase（块的第一个起始指针）、AvgChunkSize（一组压缩文档的平均大小）、BitsPerstartPointerDelta（使用ZigZag编码从平均值表示增量所需的位数）、StartPointerDeltas（上面那个的元素数组，表示使用ZigZag编码的平均起始指针的增量））】

**fdt为文档值，里面一个chunk就是一个块，Lucene索引文档时，先缓存文档，缓存大于16KB时，就会把文档压缩存储。一个chunk包含了该chunk起始文档、多少个文档、压缩后的文档内容。**

**fdx为文档号索引，倒排表存放的时文档号，通过fdx才能快速定位到文档位置即chunk位置，它的索引结构比较简单，就是跳跃表结构，首先它会把1024个chunk归为一个block,每个block记载了起始文档值，block就相当于一级跳表。**

**所以查找文档，就分为三步：**

**第一步二分查找block，定位属于哪个block。**

**第二步就是根据从block里根据每个chunk的起始文档号，找到属于哪个chunk和chunk位置。**

**第三步就是去加载fdt的chunk，找到文档。这里还有一个细节就是存放chunk起始文档值和chunk位置不是简单的数组，而是采用了平均值压缩法。所以第N个chunk的起始文档值由 DocBase + AvgChunkDocs \* n + DocBaseDeltas[n]恢复而来，而第N个chunk再fdt中的位置由 StartPointerBase + AvgChunkSize \* n + StartPointerDeltas[n]恢复而来。**

**从上面分析可以看出，lucene对原始文件的存放是行是存储，并且为了提高空间利用率，是多文档一起压缩，因此取文档时需要读入和解压额外文档，因此取文档过程非常依赖随机IO，以及lucene虽然提供了取特定列，但从存储结构可以看出，并不会减少取文档时间。**

### Tim文件

记录从域到term的信息，包括每个域中的term的统计信息（如docfreq）所形成的列表，还包括了指向频率，位置，payload，跳跃表信息的指针，这些信息在.doc,.pos,.pay文件中

Tim在块中排列，blocks包括不同数量（默认25到48个）入口，每个入口对应一个term或者指向一个次级block。

## DocValues

参考：<http://blog.csdn.net/zteny/article/details/60633374>