

两阶段生成索引文件之第一阶段

在前面的文章中，介绍了大部分的[索引文件](#)的数据结构，而从这篇文章开始，用两篇文章的篇幅来介绍例如索引文件是如何生成的，索引文件之间生成的先后顺序等内容。索引文件两个阶段生成：

- 第一阶段：添加文档阶段，也就是IndexWriter调用addDocument(..)或updateDocument(...)，在此阶段会生成.fdx、.fdt、.tvd、.tvx索引文件
- 第二阶段：flush或commit阶段，在此阶段会生成.liv、.dim、.dij、.tim、.tip、.doc、.pos、.pay、.nvd、.nvm、.dvm、.dvd索引文件。

本篇文章介绍第一阶段，建议大家可以先看下索引文件的索引结构，因为在下文的介绍中，会根据随着流程进度介绍对应索引文件的数据结构。

预备知识

term

term指的是对域值分词后的token，例如下图中域名为"content"的域(Field)，它的域值为"it is good"：

图1：

```
doc.add(new Field( name: "content", value: "it is good", type));
```

在使用空格分词器后，我们就可以得到3个term："it"、"is"、"good"。

IndexOptions

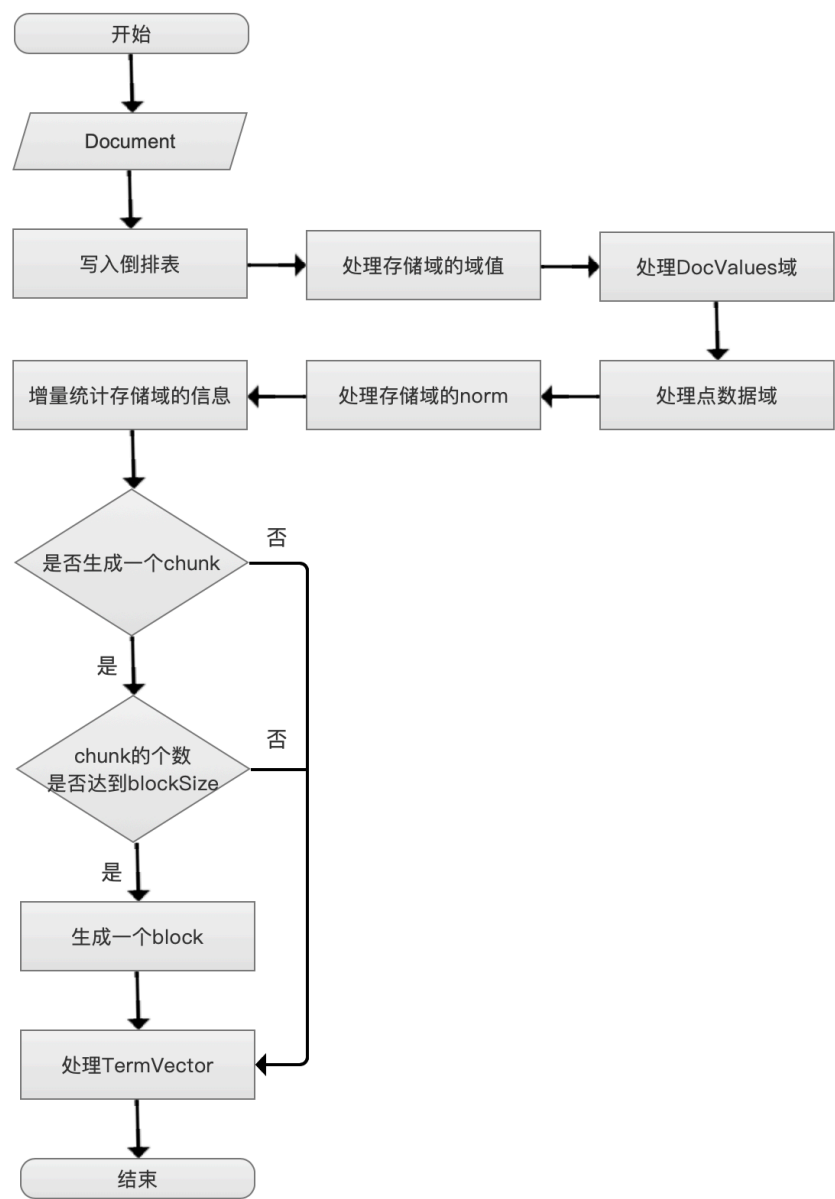
索引选项(IndexOptions)，如果你希望定义的域在搜索阶段能根据该域进行搜索，那么必须至少设置一个非NONE的索引选项，这样才能将这个域的写入到倒排表中，并最后写入到索引文件，索引选项有以下几种：

- NONE：该域的信息不会写入到倒排表中，那么在索引阶段无法通过该域名进行搜索
- DOCS：只有域所属的文档(Document)号被写入到倒排表中，在搜索阶段，PhraseQuery，或者需要位置信息的Query会抛出异常，由于不记录term(分词后的域值)的位置词频信息，所以对文档打分时，即使该term在一篇文档中出现多次，也会被当成只出现一次进行处理。
- DOCS_AND_FREQS：只有域所属的文档号和term的词频(frequency)被写入到倒排表中，但是由于没有记录term在文档中的位置信息，所以同样的在搜索阶段，PhraseQuery，或者需要位置信息的Query会抛出异常
- DOCS_AND_FREQS_AND_POSITIONS：只有域所属的文档号和term的词频(frequency)及位置(position)被写入到倒排表中，该选项是需要进行全文搜索的域的默认的索引选项
- DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS：域所属的文档号和term的词频(frequency)及位置(position)及偏移(offset)被写入到倒排表中

文档号、词频、位置、偏移的概念在[倒排表\(上\)](#)中有详细介绍。

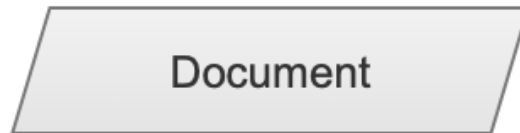
流程图

图2：



Document

图3：



Document即我们定义的Document对象，一个Document对象中可以有多种域，下图中一个Document中包含了域名为"content"的存储域、域名为"space"的点数据域、域名为"author"的DocValues域。

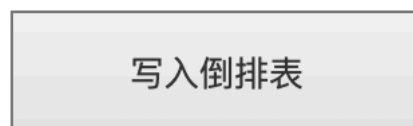
图4:

```
FieldType type = new FieldType();
type.setStored(true);
type.setStoreTermVectors(true);
type.setStoreTermVectorPositions(true);
type.setStoreTermVectorPayloads(true);
type.setStoreTermVectorOffsets(true);
type.setTokenized(true);
type.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS);

Document doc = new Document();
doc.add(new Field( name: "content", getRandomValue(), type));
doc.add(new IntPoint( name: "space", ...point: 3, 4, 6));
doc.add(new SortedDocValuesField( name: "author", new BytesRef(getRandomValue())));
indexWriter.addDocument(doc);
```

写入倒排表

图5:



只有IndexOptions不为NONE的域才会被写入到倒排表中，在搜索阶段，这样的域才能作为Query的条件进行搜索，下图是一个最基本的根据域进行查询的Query:

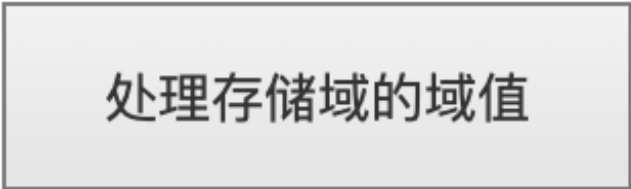
图6:

```
Query query = new TermQuery(new Term( fld: "content", text: "good"));
```

上图中，我们的查询条件是 满足条件的文档必须包含一个叫"good"的term，并且该term是域名为"content"的域值中的完整值或部分值。 写入[倒排表](#)的过程在前面的文章中已经介绍，不赘述。

处理存储域的域值

图7:



在当前流程中，需要将域的域值跟域值类型写入到一个bufferedDocs[]数组（在源码中bufferedDocs其实是一个对象，它用来将数据写入到buffer[]中，为了方便介绍，所以我们直接认为bufferedDocs是一个数组）中，目前Lucene7.5.0支持的存储域的域值类型fieldType为下面几种：

- STRING：固定值：0x00，域值为String类型
- BYTE_ARR：固定值：0x01，域值为字节类型
- NUMERIC_INT：固定值：0x02，域值为int类型
- NUMERIC_FLOAT：固定值：0x03，域值为float类型
- NUMERIC_LONG：固定值：0x04，域值为longt类型
- NUMERIC_DOUBLE：固定值：0x05，域值为double类型

bufferedDocs[]数组数据结构如下：

图8:

FieldNumAndType	Value
-----------------	-------

FieldNumAndType

FieldNumAndType是域的编号fieldNum跟域值类型FieldType的组合值，域的编号是一个从开0开始的递增值，根据域名来获得一个域的编号，FieldNumAndType = fieldNum << 3 | FieldType。比如有以下的一个Document：

Value

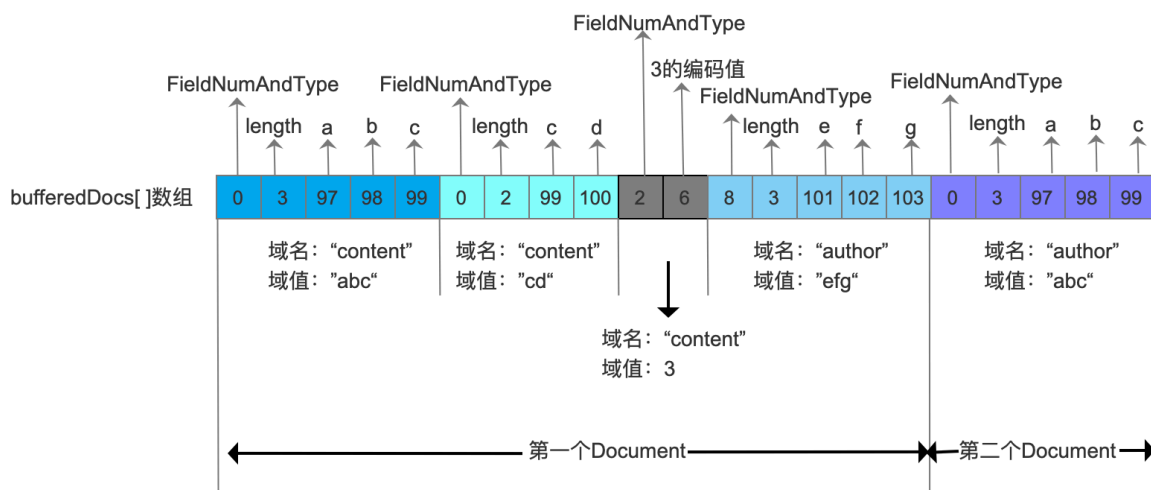
Value即域值，如果是数值类型，那么通过zigZag编码后存储，如果不是，那么按字节存储存储域值，并存储一个length来表示字节的个数。

图9:

```
// 0
doc = new Document();
doc.add(new Field( name: "content", value: "abc", type));
doc.add(new Field( name: "content", value: "cd", type));
doc.add(new StoredField( name: "content", value: 3));
doc.add(new Field( name: "author", value: "efg", type));
indexWriter.addDocument(doc);
// 1
doc = new Document();
doc.add(new Field( name: "content", value: "abc", type));
indexWriter.addDocument(doc);
```

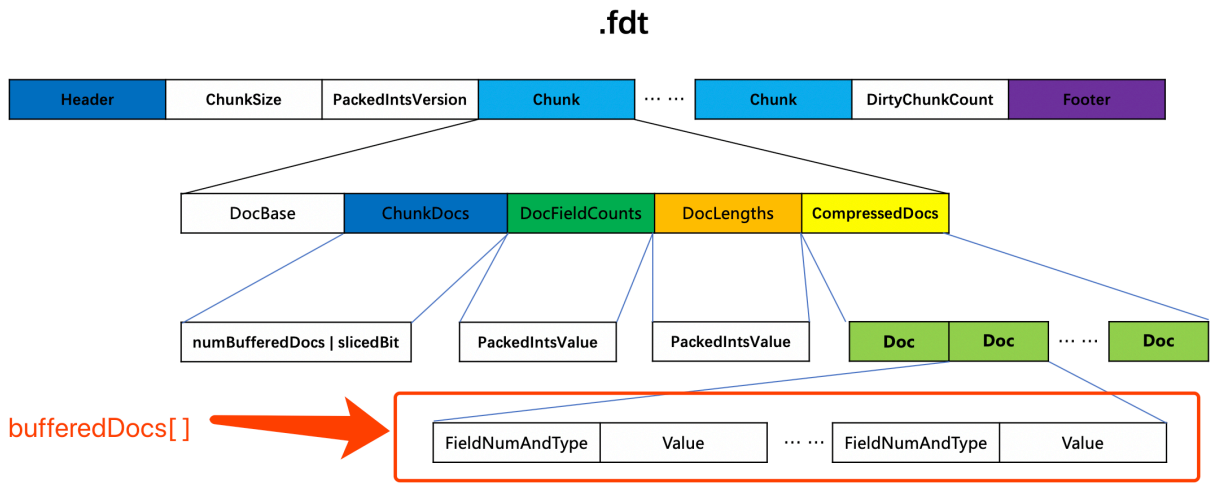
上图中两个Document，共5个存储域，假设当前域"content"的编号是0，域"author"的编号是1，并且根据域值类型，他们的FieldType分别是 (0x00)、(0x00)、(0x02)、(0x00)、(0x00)，所以他们FieldNumAndType分别是：(0 << 3 | 0x00 = 0)、(0 << 3 | 0x00 = 0)、(0 << 3 | 0x02 = 2)、(1 << 3 | 0x00 = 8)、(0 << 3 | 0x00 = 0)。故bufferedDocs[]数组如下：

图10:



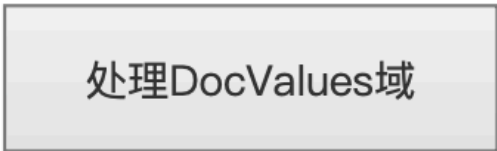
存储域的域值的信息对应应在.fdx、.fdt索引文件中的位置如下图，强调的是下图只是其中一种情况的.fxt的索引文件，详情请看.fdx、.fdt的介绍：

图11:



处理DocValues域

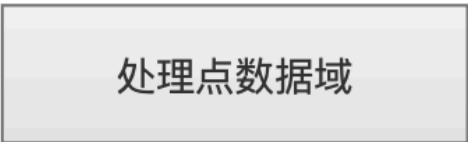
图12：



在生成索引文件的第一阶段，负责收集一些数据，根据这些数据在生成索引文件的第二阶段生成 [.dvm_](#)、[.dvd](#) 索引文件，在下一篇文章中，会详细介绍在第一阶段收集了哪些数据。

处理点数据域


图13：



在生成索引文件的第一阶段，负责收集一些数据，根据这些数据在生成索引文件的第二阶段生成 [.dim_](#)、[.dii](#) 索引文件，在两阶段生成索引文件之第二阶段的文章中，会详细介绍在第一阶段收集了哪些数据。

处理存储域的norm

图14:




处理存储域的norm

在生成索引文件的第一阶段，负责收集一些数据，根据这些数据在生成索引文件的第二阶段生成 [nvd_、.nvm](#) 索引文件，在两阶段生成索引文件之第二阶段的文章中，会详细介绍在第一阶段收集了哪些数据。

增量统计存储域的信息

图15:

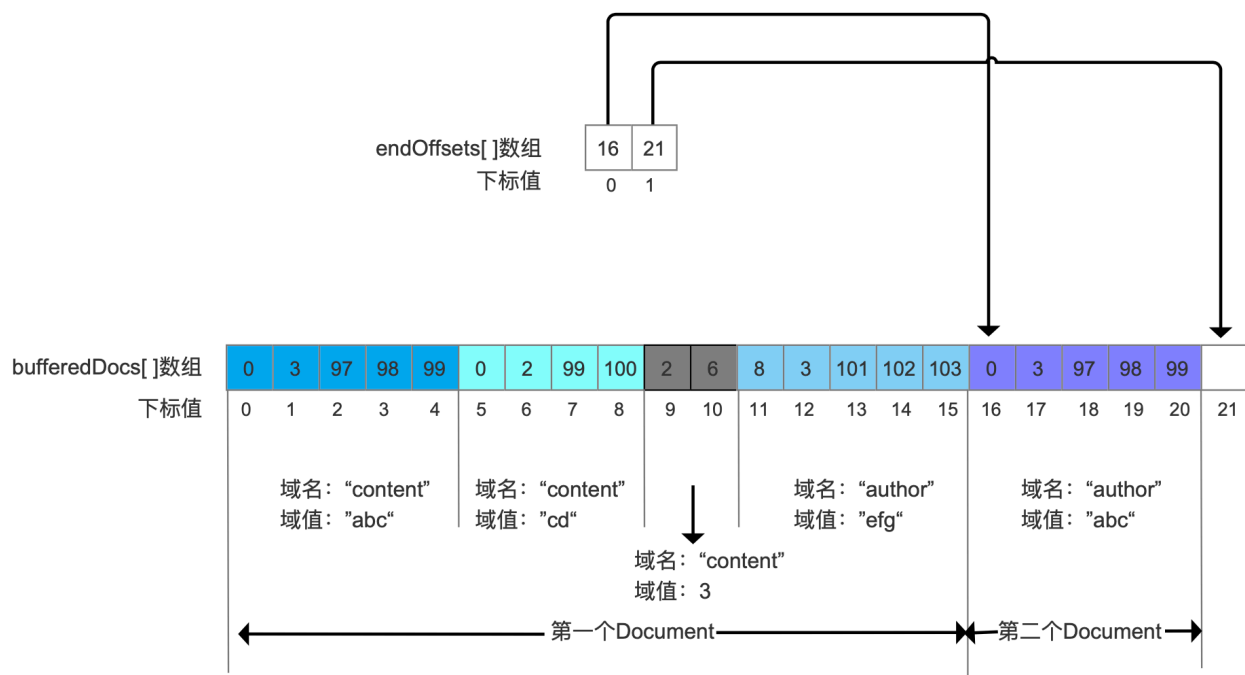


增量统计存储域的信息

在这个流程点，需要增量的记录下面的数据：

- **numBufferedDocs**：该值是一个从0开始递增的值，每次处理一个Document，该值+1，它描述了一个Document的内部文档号，同时该值也描述了当前生成一个chunk(如果你没看过[.fdx_、.fdt](#)，下文会介绍)前已经处理的文档数量
- **numStoredFields[]数组**：该数组的下标值是numBufferedDocs，数组元素是numBufferedDocs对应的Document中包含的存储域的个数
- **endOffsets[]数组**：该数组的下标值是numBufferedDocs，数组元素是一个索引值，该值作为bufferedDocs[]数组的下标值，用来得到numBufferedDocs对应的Document中所有存储域的域值在bufferedDocs[]数组中的偏移位置

图16:



是否生成一个chunk

图17:



IndexWriter每次添加完一篇文档，就会判断两个条件，如果满足其中任意一个，那么需要生成一个chunk。

- 条件1：是否已经添加了maxDocsPerChunk篇文档
- 条件2：是否存储域的域值的大小是否超过chunkSize(字节)

通过numBufferedDocs来判断是否满足条件1，通过bufferedDocs[]数组来判断是否满足条件2。chunkSize的值根据存储域的配置选项(Configuration option for stored fields)有两种值：

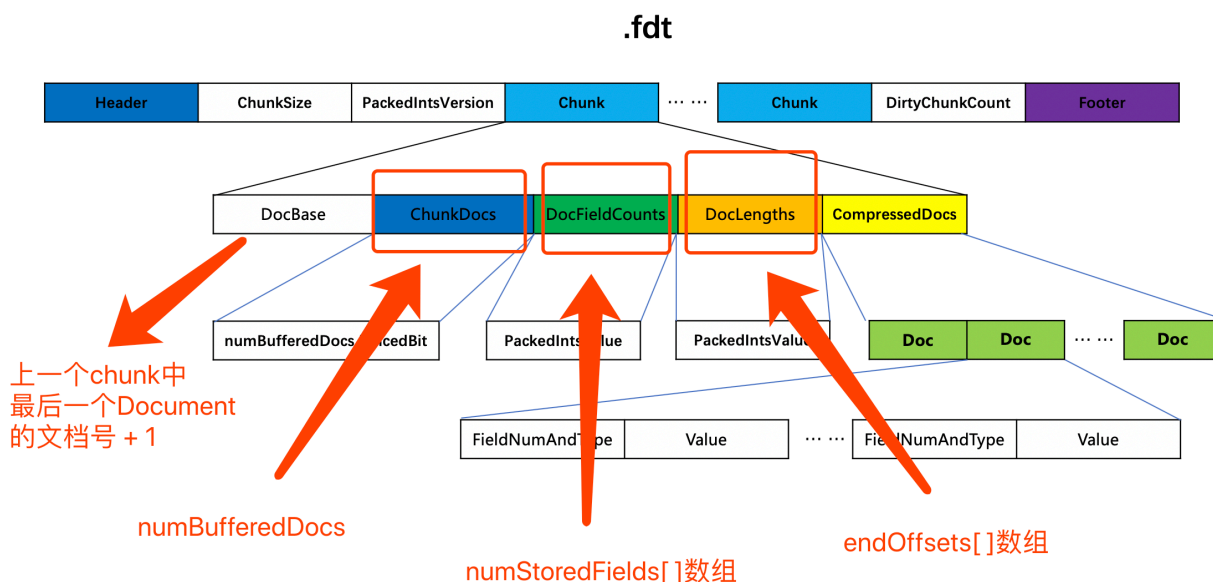
- BEST_SPEED：该模式下maxDocsPerChunk的值为128，chunkSize的值为1 << 14，即16384，这种模式有较快的检索索引速度，较低的压缩率，即生成的索引文件较大，是一种 压缩率换检索速度 的方式(Trade compression ratio for retrieval speed)

- **BEST_COMPRESSION**: 该模式下maxDocsPerChunk的值为512, chunkSize的值为 61440, 这种模式有较高的压缩率, 即生成的索引文件较小, 但较慢的索引速度, 是一种 检索速度换压缩率的方式(Trade retrieval speed for compression ratio)

在创建IndexWriter对象时, 用户可以通过IndexWriterConfig来自行选择一种模式, 默认是**BEST_SPEED**模式。

存储域的一个chunk信息对应应在.fdx、.fdt索引文件中的位置如下图:

图18:



这里需要强调的是, endOffsets[]中记录的是每个Document的域值在bufferedDocs[]数组中的偏移位置, 也就是告诉我们每个Document的存储域的域值长度, 另外DocBase的值是上一个chunk中最后一篇文档的文档号加1。

另外每生成一个chunk, 我们需要记录跟chunk相关的信息, 在后面的流程需要用到这些信息来生成索引文件, 即.fdx文件。

- **startPointerDeltas[] 数组**: 该数组的下标值是一个从0开始的递增值, 它用来代表一个chunk, 数组元素是该chunk在.fdt文件中的偏移位置, 偏移位置使用差值存储
- **docBaseDeltas[] 数组**: 该数组的下标值是一个从0开始的递增值, 同样地代表一个chunk, 数组元素是该chunk中包含的文档个数, 文档个数使用差值存储

chunk的个数是否达到blockSize

图19:



当chunk的个数达到blockSize，那么需要在索引文件.fdx中生成一个block，用该block来描述这blockSize个chunk在.fdt中的位置。

blockSize在Lucene7.5.0版本中是固定值1024，跟BEST_SPEED还是BEST_COMPRESSION模式无关。

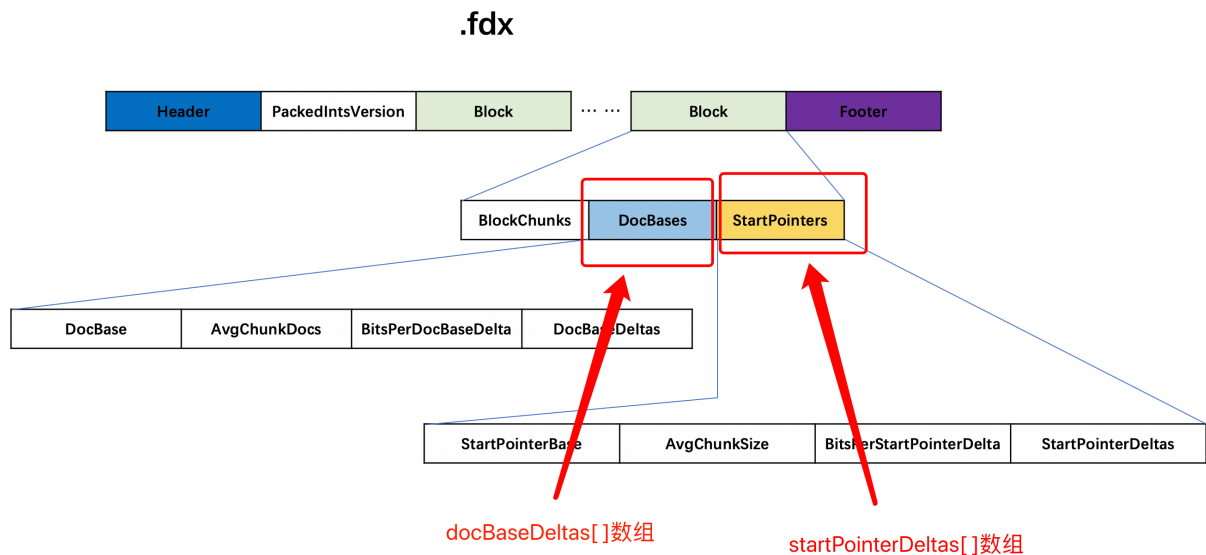
生成一个block

图20:



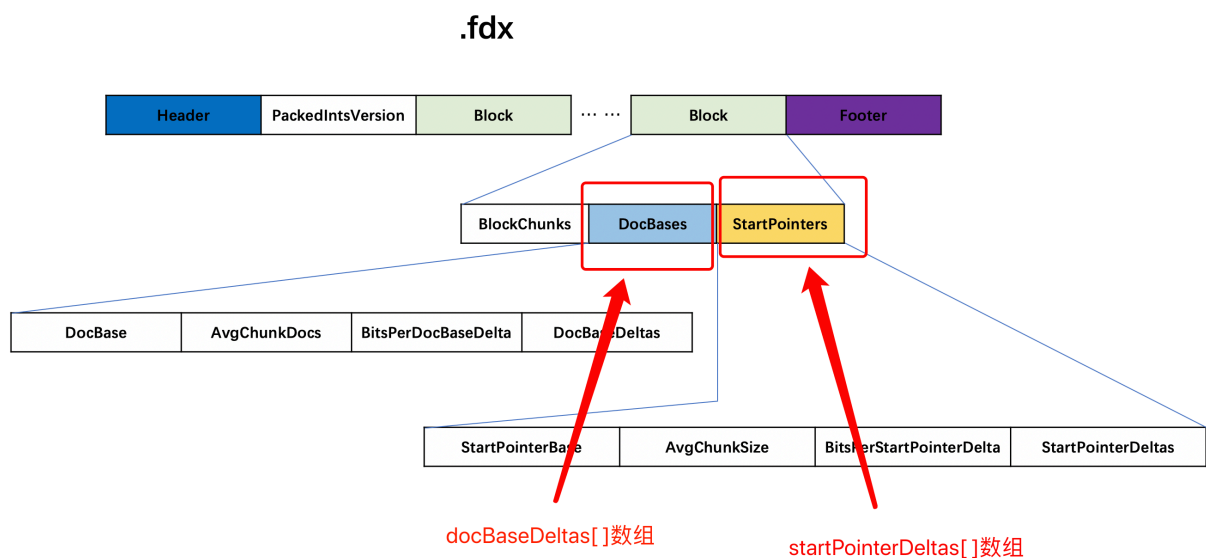
当chunk的个数达到1024个，那么把这1024个chunk在.fdx中的偏移位置及包含的文档数生成一个block，写到.fdx文件中。

图21:



处理TermVector

图21:



大家通过看TermVector的[.tvd](#)、[.tvx](#)索引文件会发现其索引结构跟存储域的索引文件及其相似，故其逻辑是类似的，所以就不赘述了。

结语

没啥好说的。

[点击下载](#)Markdown文件

