NumericDocValues

本篇文章只是介绍NumericDocValues在.dvd、.dvm文件中的数据结构，NumericDocValues的应用跟概念介绍不会在本篇文章中赘述，大家可以参考官方文档给出的介绍。.dvd、.dvm文件存放了所有DocValues的信息，所以如果在索引阶段，还添加了其他DocValues的Document，那么他们在.dvd、.dvm文件中的布局如下图： 图1：

.dvd

DocValues之间的前后关系则是根据IndexWriter的添加对应的域的先后关系来确定。同理在.dvm文件中，不赘述。

预备知识

在详细介绍数据结构前，先介绍在处理NumericDocValues的过程中会碰到的一些知识点。

固定bit位

源码中采用固定bit位个数来存放每一个域值，利用公式 (max - min) / gcd 来计算出存储每一个域值需要的固定bit位个数。其中max是最大的域值，min是最小的域值，gcd为所有域值的最大公约数。

gcd (greatest common divisor，最大公约数)

使用gcd的目的在于存储域值时，尽可能的降低空间占用。

例子

> 3个域值分别是 150、140、135，不使用gcd的情况下，按照公式(max - min)的计算出存储每一个域值需要的固定bit位个数，那么 150 - 135 = 15，15的二进制表示为 0b00001111 即每个域值需要固定的4个bit位来存储。
>
> 如果我们先求出150、140、135的最大公约数，即最大公约数为5，然后按照(max - min) / gdc来计算每个域值需要的固定bit位个数，即 (150 -135) / 5 = 3，3的二进制表示为0b00000011，那么每个域值需要的固定bit位个数只要2个即可，所以实际存储的3个域值的按照公式 (v - min) / gcd 的结果为 3 ((150 -135) / 5 = 3)、1 ((140 -135) / 5 = 1)、0，((135 -135) / 5 = 0)。gcd、min的值会保存到.dvm文件中。在读取阶段，通过gcd、min 就可以解码出域值。

按块处理域值

在处理域值时，会分为 **单个block** （SingleBlock）或 **多个block**（MultipleBlocks）来存储域值，这么做的目的也是为了尽可能降低空间的使用。 当需要处理的域值个数超过 16384 (NUMERIC_BLOCK_SIZE)个时，Lucene会判断采用 多个block存储是否会减少空间的使用。

例子

使用单个block存储域值

> 考虑这么一种情况，如果待处理的域值个数为16644个(16384 + 260)，如果前16384个域值的取值要么是3，要么是4，并且剩余的260个域值的值都为大于2000的且各不相同的值，并且最大值为3000。很明显这16644个域值的gcd为1，所以根据(max - min) / gdc，存储每一个域值需要的bit位个数为(3000 - 3) / 1 = 2997，2997的二进制位为0b00001011_10110101,即每一个域值需要占用12个bit位。这个例子将所有的域值当成一个block进行处理。

使用多个block存储域值

> 当待处理的域值个数达到16384个时，先将这些值作为一个block处理，由于这16384个域值不是3就是4，所以很明显gcd的值为1，所以根据(max - min) / gdc，存储每一个域值需要的bit位个数为(4 - 3) / 1 = 1， 1的二进制位为0b1，即每一个域值只要占用1个bit位。最后剩余的260个域值按照一个block处理，并且同样地按照(max - min) / gdc计算每一个域值需要的bit位个数。

域值种类个数小于256的优化

这种优化的目的还是处于尽可能减少空间的使用。 满足的优化的条件需要两点：

第一点：待处理的域值种类个数(不是域值的个数)不超过256。

至于为什么是256这个值，我没有参透~不好意思。

第二点：预先计算判断优化后的空间使用量是否能小于优化前

优化后的域值存储方式与非优化的方法截然不同，下面通过特定的例子来介绍

例子

待处理的域值有 5、6、5、6、3000，域值种类个数为 3，即5、6、3000。

不优化存储

根据公式 (max - min) / gcd，计算出存储每一个域值需要的固定bit位个数， (3000 - 5) / 1 = 2995，2995的二进制为0b00001011_10110011，即存储每一个域值需要12个bit位。最后将域值存放到dvd文件中。

优化存储

优化步骤如下：给每一种域值一个编号，在源码中，对3种域值进行排序，然后赋予每一种域值一个从0开始的值，如下表：

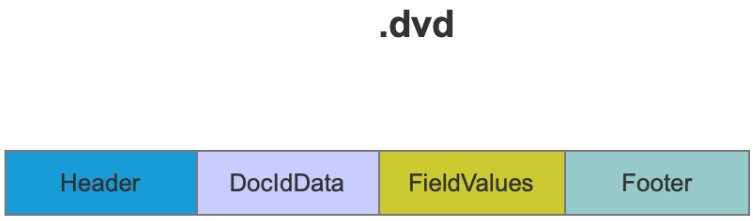
域值	5	6	3000
编号	0	1	2

接着原本应该将所有域值，即5、6、5、6、3000存放到.dvd文件中，换成对应的编号存放到.dvd文件中，即实际存储到.dvd的值为0、1、0、1、2。并且每个值需要的固定bit位的个数为编号中的最大值，在当前例子中，需要的固定bit位的个数为2，即存储每一个值只需要2个bit位。 这种优化的方式需要将域值与编号的对应关系信息存放到.dvm文件中，在读取阶段，先从.dvd文件读取到一个编号，然后根据.dvm中存放的 域值与编号的映射关系，获得最终的域值。

数据结构

dvd

在下图中给出了只有NumericDocValues的.dvd、.dvm文件的数据结构。 图2：



DocIdData描述了包含NumericDocValues数据的文档号的信息。 FieldValues描述了域值信息。

DocIdData

如果IndexWriter添加的document中不都包含当前域，那么需要将包含当前域的文档号记录到DocIdData中，并且使用IndexedDISI类来存储文档号，IndexedDISI存储文档号后生成的数据结构单独的作为一篇文章介绍，在这里不赘述，看这里。

FieldValues

单个block存储域值

单个block存储方式又根据是否采用了 域值种类个数小于256的优化 生成两种数据结构。

优化

正如上文中的说明，优化后的域值存储方式，实际存储的是域值对应的编号，然后采用PackedInt对编号进行编码存储。PackedInt编码后的FieldValues格式在这里不赘述，在BulkOperationPacked中介绍了其中一种压缩方法。

未优化

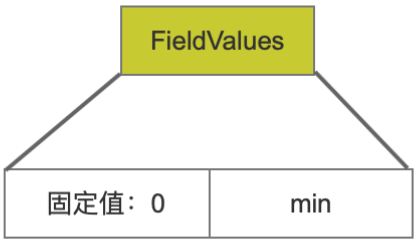
未优化的域值存储方式，只能根据 (v - min) / gcd 公式将域值存放到FieldValues中, 其中v是待存储的域值，min为所有域值的最小值，上文中的预备知识介绍了为何使用 (v - min) / gcd 公式。同样的采用PackedInt对域值进行编码存储。

多个block存储域值

多个block存储方式根据域值是否都是一样生成两种数据结构。

域值都相同

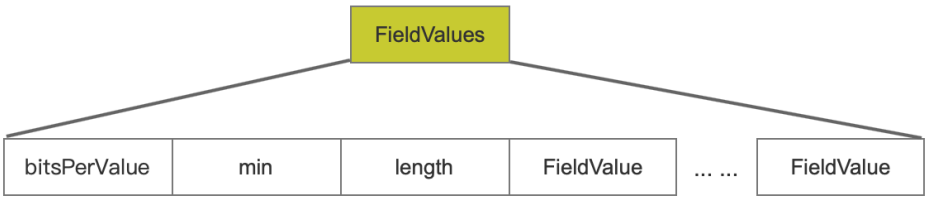
图3：



由于域值都相同，只要往.dvd文件中写入一个固定的标记值0跟其中一个域值即可。

域值不都相同

图4：



bitsPerValue

存储每一个域值需要的bit位的个数。

min

实际存储到.dvd文件的域值是经过 (v - min) / gcd 公式计算后的值，所以这里要记录当前block中min值，在读取阶段用来解码域值。对于每一个block，min的值可能会不同，但是gcd的值通过所有域值计算出来的，所以不用在每一个block中存储gcd。gcd的值记录在.dvm文件中。

length

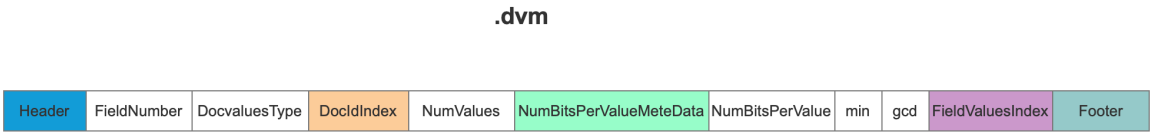
length用来描述在读取阶段需要往后读取的字节区间，这个字节区间内包含了当前block的所有域值信息。

FieldValue

当前block中的所有域值，并且使用了 PackedInt进行编码存储。

dvm

图5：



FieldNumber

域的编号。

DocValuesType

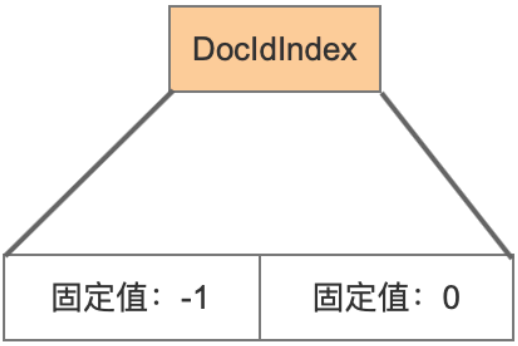
Docvalues的类型，本文中，这个值就是 NUMERIC。

DocIdIndex

DocIdIndex是对.dvd文件的一个索引，用来描述 .dvd文件中DocIdData在.dvd文件中的开始跟结束位置。

情况1：

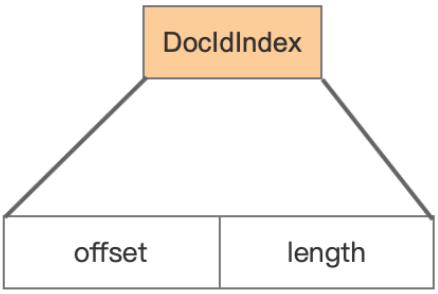
图6：



如果IndexWriter添加的document中都包含当前域，那么只需要在DocIdIndex中添加标志信息即可。

情况2：

图7：



如果IndexWriter添加的document中不都包含当前域，那么.dvd文件中需要将包含当前的域的文档号信息都记录下来。

offset

```
.dvd文件中存放文档号的DocIdData在文件中的开始位置。
```

length

```
length为DocIdData在.dvd文件中的数据长度。
```

在读取阶段，通过offset跟length就可以获得所有的DocIdData数据。

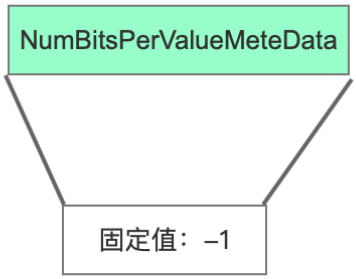
NumValues

```
当前域的域值个数。
```

NumBitsPerValueMeteData

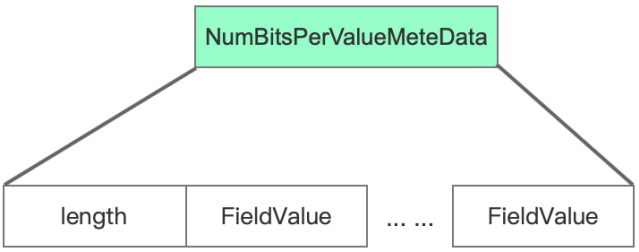
情况1

如果处理的域值都是相同的，那么只要写入标志信息即可。 图8：



情况2

如果处理的域值满足上文介绍的 域值种类个数小于256的优化。 图9：



length

域值的种类的个数。在上文中，我们给出的例子是 5、6、5、6、3000，那么length的值就是3。

FieldValue

上文中我们提到，在.dvm文件中需要保存 域值跟编号 的映射关系，在上面的例子中，FieldValue的分别会存放，5、6、3000的域值（原始域值存储）。在读取阶段，根据读取先后顺序，给每一个域值一个从0开始计数的一个编码值，就可以获得 域值跟编号 的映射关系。

NumBitsPerValue

存储每一个域值需要的固定bit位个数。

min

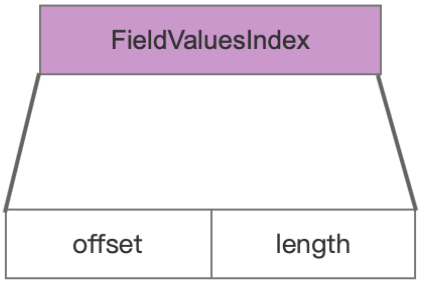
用于对编码的域值进行解码。

gcd

用于对编码的域值进行解码。

FieldValuesIndex

FieldValuesIndex是对.dvd文件的一个索引，用来描述 .dvd文件中FieldValues在.dvd文件中的开始跟结束位置。 图10：



offset

```
.dvd文件中存放文档号的FieldValues在文件中的开始位置。
```

length

```
length为FieldValues在.dvd文件中的数据长度。
```

在读取阶段，通过offset跟length就可以获得所有的FieldValues数据。

结语

NumericDocValues的索引文件数据结构相对SortedDocValues比较简单。之前介绍的SortedDocValues（上）的文章会在以后进行重写，内容保持跟本篇文章一致。 大家可以看我的源码注释来快速理解源码，地址在这里：https://github.com/luxugang/Lucene-7.5.0/blob/master/solr-7.5.0/lucene/core/src/java/org/apache/lucene/codecs/lucene70/Lucene70DocValuesConsumer.java

点击下载Markdown文件