索引文件的生成（十二）（Lucene 8.4.0）

本文承接索引文件的生成（十一），继续介绍剩余的内容，为了便于下文的介绍，先给出生成索引文件.dim&&.dii的流程图以及流程点`构建BKD树的节点值（node value）`的流程图：

图1：

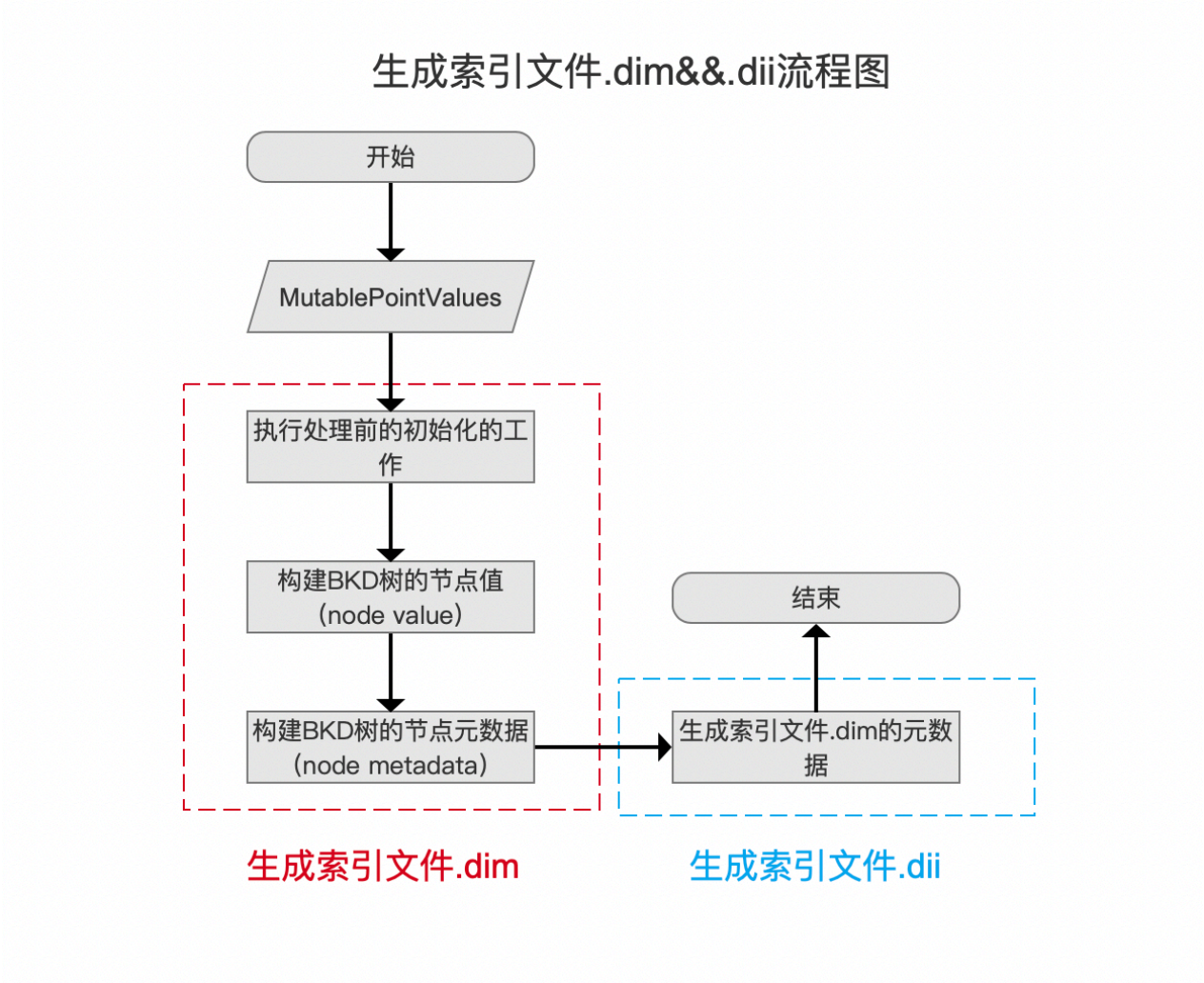


图2：

在前面的文章中，我们介绍了图2中处理内部节点的所有流程点，在介绍处理叶子节点前，我们先填个坑，即当某个内部节点划分后生成的左右子树为叶子节点时，内部节点为左右子树提供哪些准备数据，即流程点`设置左子树的准备数据`、`设置右子树的准备数据`。

设置左子树的准备数据、设置右子树的准备数据

左右子树为叶子节点

在文章索引文件的生成（十一）之dim&&dii中我们提到，处理节点（叶子节点或者内部节点）需要的准备数据有好几个，这些准备数据在源码中其实就是 https://github.com/LuXugang/Lucene-7.5.0/blob/master/solr-8.4.0/lucene/core/src/java/org/apache/lucene/util/bkd/BKDWriter.java 类中的build(...)方法的参数：

图3：

```
private void build(int nodeID, int leafNodeOffset,
                   MutablePointValues reader, int from, int to,
                   IndexOutput out,
                   byte[] minPackedValue, byte[] maxPackedValue,
                   int[] parentSplits,
                   byte[] splitPackedValues,
                   long[] leafBlockFPs,
                   int[] spareDocIds) throws IOException {
```

对于左右子树为叶子节点的情况，我们只需要关心图3中的leafBlockFPs数组。

leafBlockFPs数组

每当处理完一个叶子节点，我们需要将叶子节点对应的信息写入到索引文件.dim中，又因为所有叶子节点对应的信息以字节流形式存储在索引文件.dim中，leafBlockFPs数组正是用来记录每一个叶子节点对应的信息在索引文件.dim中的起始位置，同样具体的内容将在下文中展开。

选出叶子节点的排序维度

图4：

选出叶子节点的排序维度前需要先计算两个数组：commonPrefixLengths数组以及usedBytes数组。

commonPrefixLengths数组

commonPrefixLengths为int类型的数组，该数组的下标用来描述维度编号（见文章索引文件的生成（十）之dim&&dii），数组元素描述的是维度编号对应的所有维度值的最长公共前缀的长度。

usedBytes数组

usedBytes为FixedBitSet类型数组，如果你不熟悉FixedBitSet对象也没关系，我们只需要知道usedBytes数组的下标同样用来表示维度编号，数组元素描述的是维度编号的对应的所有维度值的公共前缀后的**下一个字节**的种类数量（按照字节的不同值作为种类的划分），只是种类数量用FixedBitSet来统计而已。

图5：

假设叶子节点中有3个点数据，我们以维度编号2为例，图4中，点数据蓝色部分的字节是相同的，即点数据在这个维度的维度值的最长公共前缀的长度为2（2个字节），即在commonPrefixLengths数组中，下标值为2的数组元素的值为3，其他维度同理，故commonPrefixLengths数组如下所示：

图6：

同样以维度编号2为例，除去相同的2个前缀字节的下一个字节的种类数量如图4中绿框与红框标注共有2个种类，即有两种，其他维度同理，故usedBytes数组如下所示：

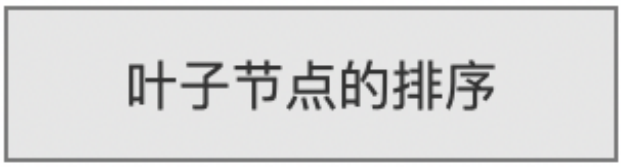
图7：

在源码实现上，需要先计算出commonPrefixLengths数组，利用该数组再计算出usedBytes数组。

在计算出usedBytes数组之后，遍历该数组，找到**第一个**数组元素最小的值，对应的数组下标，即维度编号，作为叶子节点的排序维度，在图6中，编号维度1将作为排序维度。

叶子节点的排序

图8：



使用**内省排序（IntroSorter）**对叶子节点中的点数据进行排序，而不是在文章索引文件的生成（十）之dim&&dii中提到的内部节点排序使用的**最大有效位的基数排序(MSB radix sort)**，源码中也给出了注释：

```
No need for a fancy radix sort here, this is called on the leaves only so there are not many values to sort
```

简单来说就是叶子节点中的点数据的数量相对于内部节点较少而选择使用**内省排序（IntroSorter）**。

叶子节点中的点数据数量的取值范围是什么

先给出源码中的注释：

```
The tree is fully balanced, which means the leaf nodes will have between 50% and 100% of the requested maxPointsInLeafNode
```

上述注释中的maxPointsInLeafNode指的是叶子节点中最多包含的点数据数量，默认是1024，也就说当内部节点中的点数据数量大于1024个时就需要继续切分，那么叶子节点中包含的点数据的数量区间为 [512, 1024]。

同内部节点的排序一样，叶子节点中点数据之间的排序关系同样用ord数组（见文章索引文件的生成（十）之dim&&dii）来描述，不赘述。

更新leafBlockFPs数组

图9：

到此流程点，我们即将开始把叶子节点的信息写入到索引文件.dim中，故需要将索引文件.dim当前可写入的位置记录到leafBlockFPs数组中，在读取阶段，就可以通过leafBlockFPs数组找到当前叶子节点的信息在索引文件.dim中的起始读取位置，如下所示：

图10：

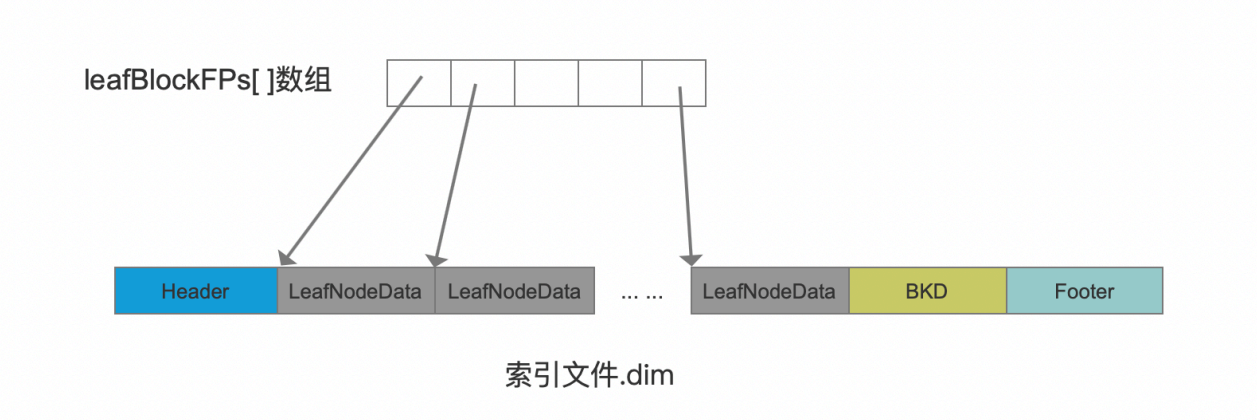


图10中，leafNodeData为叶子节点的信息，leafBlockFPs数组的下标值为叶子节点的**差值节点编号**。

差值节点编号是什么

在文章索引文件的生成（十）之dim&&dii我们介绍了节点编号的概念以及最左叶子节点的节点编号的获得方式，差值节点编号值的就是叶子节点编号跟最左叶子节点的节点编号的差值：

```
leafBlockFPs[nodeID - leafNodeOffset]
```

上述代码中，nodeID即节点编号，leafNodeOffset为最左叶子节点的节点编号，如果当前处理的是最左叶子节点，那么差值节点编号为0 ，即下标值为0，那么对应的数组元素描述的就是最左叶子节点的信息在索引文件.dim中的起始读取位置。

写入文档号信息到索引文件.dim中

图11：

该流程点描述是将叶子节点中的点数据对应的文档号写入到索引文件.dim中。

如果获得每个点数据对应的文档号

在文章索引文件的生成（八）之dim&&dii中我们说到，在点数据的收集阶段，使用了 docIDs数组存储了文档号信息，该数组的数组元素为文档号，下标值为numPoints，那么我们只需要知道点数据对应的numPoints就可以获取点数据对应的文档号，而在ord数组（见文章索引文件的生成（十）之dim&&dii）中，数组元素正是numPoints，所以我们只要遍历ord数组就能获得当前叶子节点中的文档号信息，并且这些文档号是有序的，注意的是：**不是文档号的值有序，而是文档号对应的点数据是有序，排序规则即上文中的流程点`选出叶子节点的排序维度`、`叶子节点的排序`中的内容**。

图12：

叶子节点中包含的文档号信息在索引文件.dim中的位置如下所示：

图13：

上图中，红框标注的两个字段为当前流程点写入的信息，其中Count描述的是文档号的数量，DocIds为文档号的集合。DocIds的详细数据结构见文章索引文件之dim&&dii 。

写入相同前缀信息到索引文件.dim中

图14：

在当前流程点，我们根据在上文中计算出的commonPrefixLengths数组，将每个维度的最长相同前缀的值写入到索引文件.dim中，对应在索引文件.dim中的位置如下所示：

图15：

上图中，Length描述的每一个维度的最长公共前缀的长度，Value为最长公共前缀的值，在读取阶段就可以根据Length的值确定Values在索引文件.dim中的位置区间。

以图5的维度编号2为例，Length跟Value的值如下所示：

```
Length: 2
Value: {10000000, 00000000}
```

结语

基于篇幅，剩余的内容将在下一篇文章中展开。

点击下载附件