RoaringDocIdSet

RoaringDocIdSet的设计灵感来源于RoaringBitmap，Lucene根据自身需求有着自己的的实现方法，来实现对文档号的处理（存储，读取）。

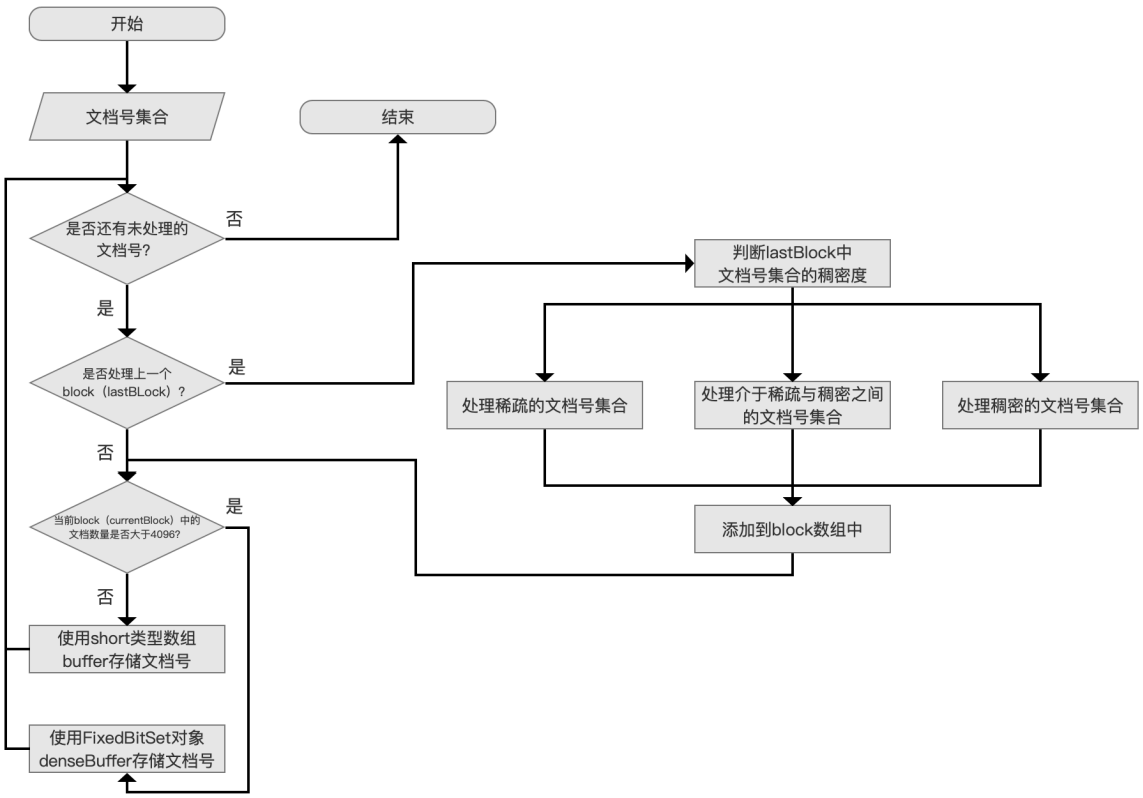
RoaringBitmap

位集合（bitsets）通常也被成为位图（bitMaps），它是一种常用的快速数据结构（fast data structures），但由于它对内存的开销较大，所以我们通常会使用经过压缩处理的bitMaps，而这便是RoaringBitmap。

本篇文章不会对RoaringBitmap展开介绍，因为Lucene有着自己的实现方法，感兴趣的同学可以看这两个链接：http://roaringbitmap.org、https://github.com/RoaringBitmap/RoaringBitmap。

RoaringDocIdSet存储文档号流程图

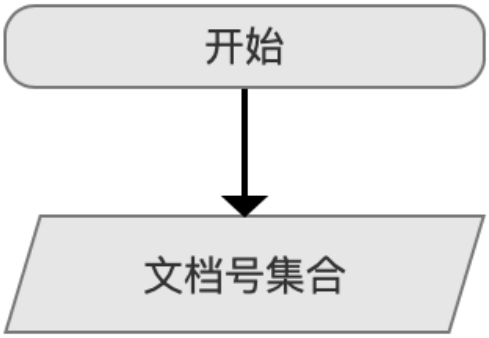
图1：



点击查看大图

文档号集合

图2：

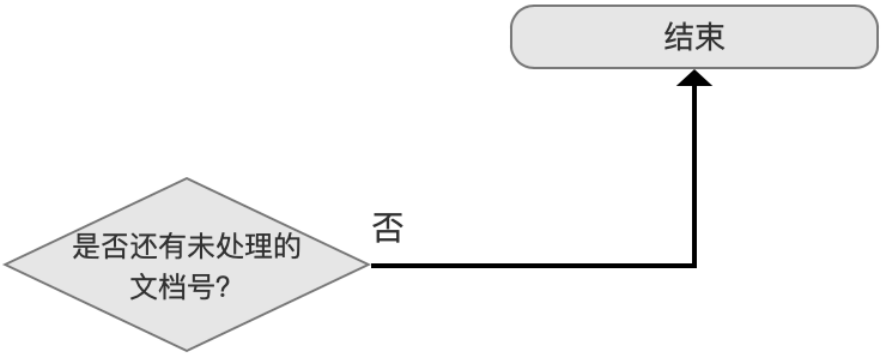


RoaringDocIdSet只允许处理按照文档号从小到大递增的文档号集合，每当处理一个新的文档号，称之为docId，会判断与上一个处理的文档号lastDocId的大小，如果小于等于lastDocId，说明文档号集合不满足要求，抛出如下的异常：

```
if (docId <= lastDocId) {
    throw new IllegalArgumentException("Doc ids must be added in-order, got " +
docId + " which is <= lastDocID=" + lastDocId);
}
```

是否还有未处理的文档号?

图3：



我们从最小的文档号开始，逐个处理文档号集合中的文档号，当处理完所有的文档号，那么就完成了图1所示的流程。

是否处理上一个block（lastBLock）？

图4：

block是什么：

- 根据规则将文档号划分到一个或多个block中处理，在同一个block中的所有文档号会以同一种方式存储（存储的方式在下文中会介绍），不同block之间的存储方式没有相互联系。

规则是什么：

- 规则如下:

```
docId >>> 16
```

- 由上面的规则可以看出文档号从0开始，每65536个文档号被划分为一个block，如果我们有下面的文档号集合：

图5：

{0, 3, 5, 6, 65532, 65536, 65537, 131072, 131073}

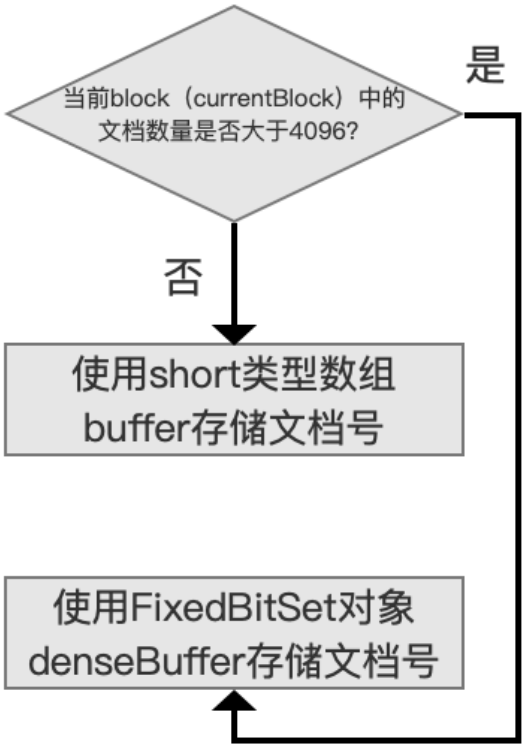
- 划分后的block如下所示，根据规则，可以知道一个block中最多可以包含65536个文档号：

图6：

我们以图5中的文档号集合为例，当处理到文档号65536时，该文档号会被划分到第二个block中，那么此时我们就需要处理第一个block，此时第一个block成为图4中的lastBlock，同时第二个block成为currentBlock。

根据block中的文档号数量选择存储方式

图7：



上文中我们说到使用block来存储文档号，并且一个block中最多存储65536个文档号，但是并没有给出block的数据结构，原因是block的数据结构取决于block中包含的文档号数量。

block的数据结构

block一共有两种数据结构：

- short类型数组：当block中包含的文档号数量小于4096个时，使用该数据结构
- FixedBitSet：当block中包含的文档号大于等于4096个时，使用该数据结构，FixedBitSet在前面的文章中已经介绍，这里不赘述，感兴趣的可以点击链接查看：https://www.amazingkoala.com.cn/Lucene/gongjulei/2019/0404/45.html

short类型数组向FixedBitSet的转化（重要）：

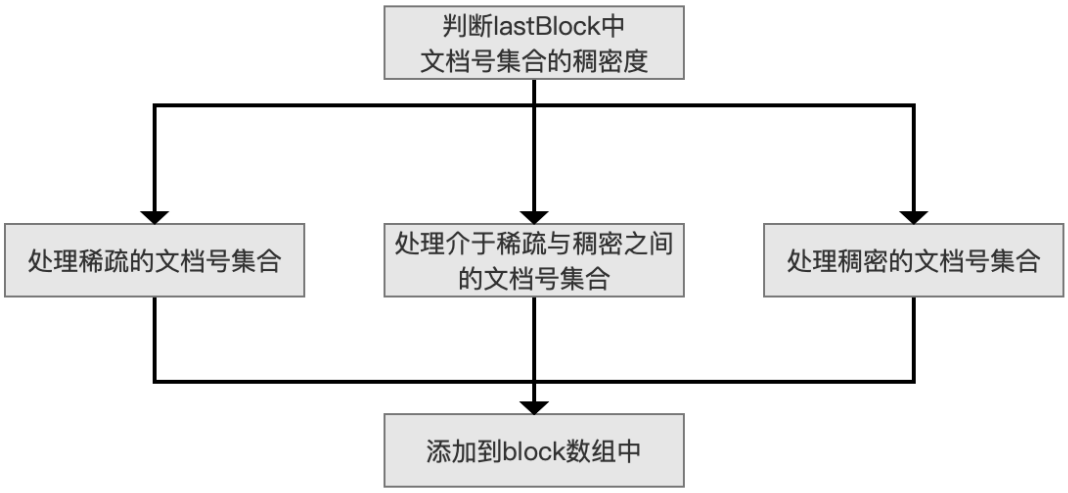
- 上文中我们说到，从图2中的文档号集合中依次处理每一个文档号，该文档号总是先用short类型数组存储，该数组的变量名为buffer，当该数组中的元素超过4096个时，那么如果下一个文档号仍

然属于同一个block，**那么需要先将short类型数组中所有元素存放到FixedBitSet对象中**，该对象的变量名为denseBuffer，并且后续属于同一个block的文档号都使用denseBuffer存放，这就是转化的过程

- **（重要）**图7的流程没有描述转化的过程，会让读者误以为使用了两种数据结构存储了同一个block中的文档号集合

处理上一个block（lastBlock）

图8：



根据block中包含的文档号的数量来判断稠密度：

- 稀疏：block中**存储**的文档号数量**小于等于**4096个认为是稀疏的
- 稠密：block中**未存储**的文档号数量**小于**4096个认为是稠密度，一个block最多可以存储65536个文档号，即稠密的block中**存储**的文档号数量大于等于61440个文档号
- 既不稀疏也不稠密：block中**存储**的文档号数量大于4095并且小于61440个认为是既不稀疏也不稠密

不同稠密度的文档号集合如何存储：

- 稀疏：使用short类型数组存储
- 稠密：计算出那些未存储的文档号，然后使用short类型数组存储
- 既不稀疏也不稠密：使用FixedBitSet存储

例如我们以下的文档号集合：

图9：

$$\{0, 1, 2, ..., 65533, 65536, 65537, 131072, 131073\}$$

图10描述了将图9中的文档号划分到不同的block，**其中第一个block中包含了0~65533共65534个文档号**：

图10：

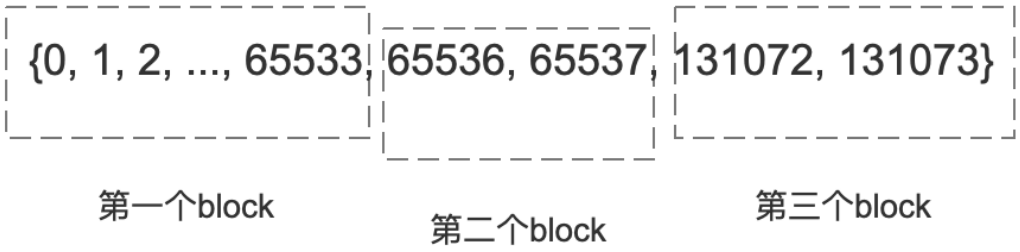


图11描述了进行了稠密度计算后，将处理后的block添加到block数组中，第一个block包含了65534个文档号，故认为是稠密的，由于一个block最多只能存储65536个，那么计算出那些未存储的文档号，即65534跟65535两个文档号，然后存储它们两个，而第二个block跟第三个block都只包含了2个文档号，所以认为是稀疏的，直接存储即可：

图11：

RoaringDocIdSet读取文档号

RoaringDocIdSet提供了两种方法来读取文档号:

- 获取所有文档号：这种方法只需要逐个block数组的每一个元素，如果block使用short类型数组存储，那么顺序遍历该数组中的元素，如果使用FixedBitSet存储，其遍历方法见https://www.amazingkoala.com.cn/Lucene/gongjulei/2019/0404/45.html
- 判断某个文档号是否存在：根据下面的规则，找到该文档号属于block数组中的哪一个block，如果block使用short类型数组存储，那么使用**二分法**尝试在该数组中找，如果使用FixedBitSet存储，其查找方法见https://www.amazingkoala.com.cn/Lucene/gongjulei/2019/0404/45.html

```
    docId >>> 16
```

结语

个人觉得直接看源码应该比看我写的文章能更快的了解RoaringDocIdSet😄，所以点击这个链接看下吧：https://github.com/LuXugang/Lucene-7.5.0/blob/master/solr-7.5.0/lucene/core/src/java/org/apache/lucene/util/RoaringDocIdSet.java。

点击下载附件