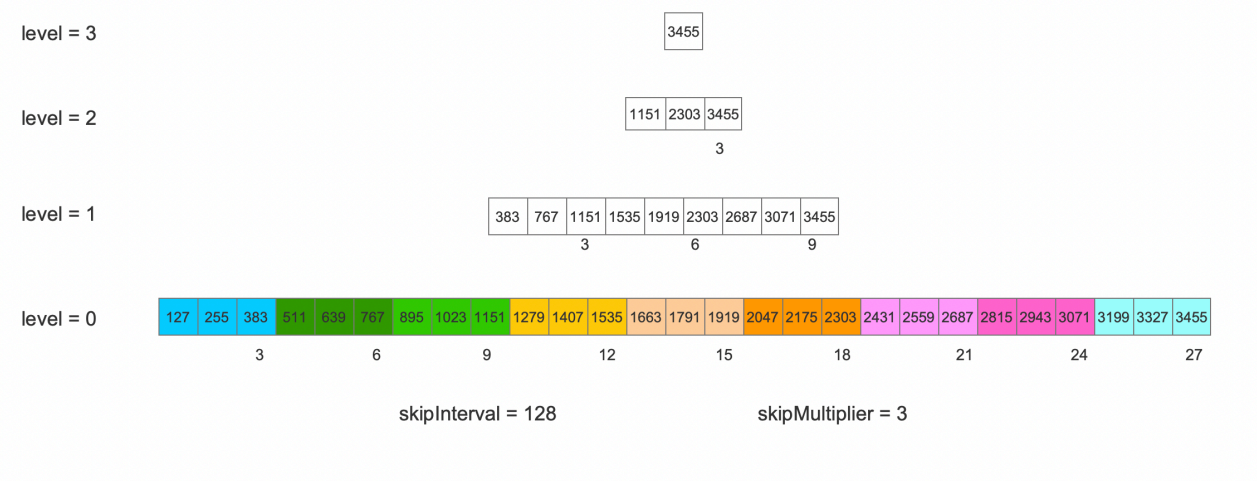
索引文件的生成（四）

在文章索引文件的生成（三）中我们介绍了在Lucene中生成跳表SkipList的流程，通过流程图的方法介绍了源码中的实现方式，而对于读取SkipList的内容，决定直接以例子的方式来介绍其读取过程，下文中出现的名词如果没有作出介绍，请先阅读文章索引文件的生成（三）。

例子

直接给出一个生成后的跳表：

图1：



在图1中，为了便于介绍，我们处理的是文档号0~3455的3456篇文档，我们另skipInterval为128，即每处理128篇文档生成一个PackedBlock，对应一个datum；另skipMultiplier为3（源码中默认值为8），即每生成3个datum就在上一层生成一个新的索引，新的索引也是一个datum，它是3个datum中的最后一个，并且增加了一个索引值SkipChildLevelPointer来实现映射关系（见索引文件的生成（三）），每一层的数值为PackedBlock中的最后一篇文档的文档号，例如level=2的三个数值1151、2303、3455。

哨兵数组skipDoc

哨兵数组skipDoc的定义如下所示：

```
1     int[] skipDoc;
```

该数组用来描述每一层中正在处理的datum，datum对应的PackedBlock中的最后一篇文档的文档号作为哨兵值添加到哨兵数组中，在初始化阶段，skipDoc数组中的数组元素如下所示（见图2红框标注的数值）：

```
1     int[] skipDoc = {127, 383, 1151, 3455}
```

初始化阶段将每一层的第一个datum对应的PackedBlock中的最后一篇文档的文档号作为哨兵值。

docDeltaBuffer

docDeltaBuffer是一个int类型数组，总是根据docDeltaBuffer中的文档集合来判断SkipList中是否存在待处理的文档号。

在初始化阶段，docDeltaBuffer数组中的数组元素是level=0的第一个datum对应的PackedBlock中文档集合。

SkipList在Lucene中的应用

了解SkipList在Lucene中的应用对理解读取跳表SkipList的过程很重要，在Lucene中，使用SkipList实现文档号的**递增遍历**，每次判断的文档号是否在SkipList中时，待处理的文档号必须大于上一个处理的文档号，例如我们在文章文档号合并（SHOULD）中，找出满足查询要求的文档就是通过SkipList来实现。

Lucene中使用读取跳表SkipList的过程

读取过程分为下面三步：

- 步骤一：获得需要更新哨兵值的层数N
  - 从skipDoc数组的第一个哨兵值开始，依次与待处理的文档号比较，找出所有比待处理的文档号小的层
- 步骤二：从N层开始依次更新每一层在skipDoc数组中的哨兵值
  - 如果待处理的文档号大于当前层的哨兵值，那么另当前层的下一个datum对应的PackedBlock中的最后一篇文档的文档号作为新的哨兵值，直到待处理的文档号小于当前层的哨兵值
  - 在处理level=0时，更新后的datum对应的PackedBlock中的文档集合更新到docDeltaBuffer中
- 步骤三：遍历docDeltaBuffer数组
  - 取出PackedBlock中的所有文档号到docDeltaBuffer数组中，依次与待处理的文档号作比较，判断SkipList中是否存在该文档号

读取跳表SKipList

我们依次处理下面的文档号，判断是否在跳表SKipList中来了解读取过程：

```
1    文档号：{23, 700, 701, 3000}
```

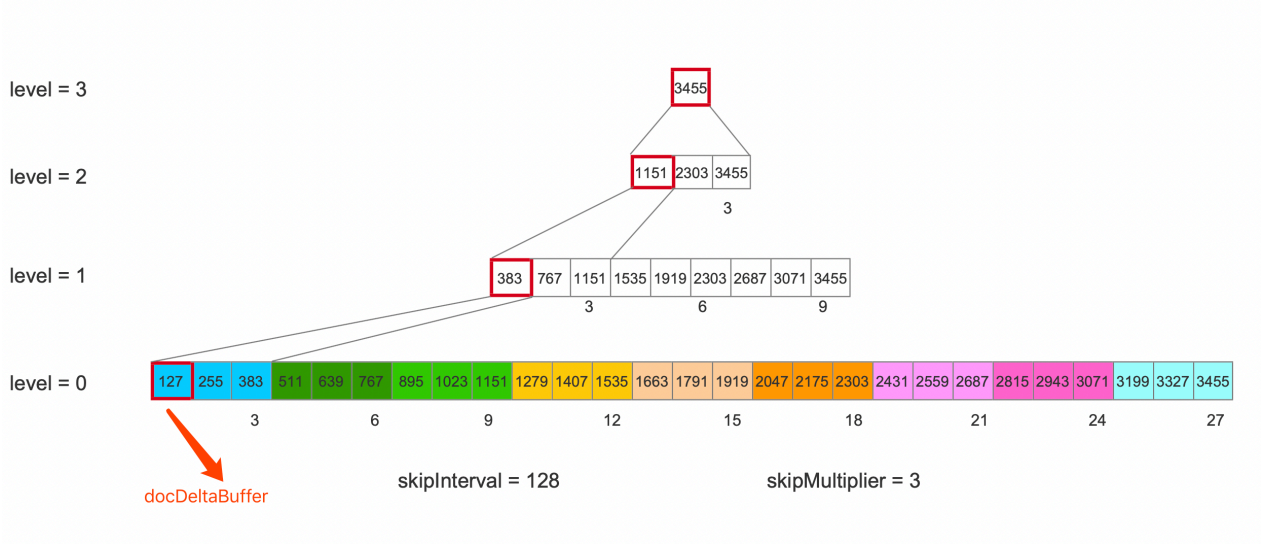
文档号：23

更新前的skipDoc数组如下所示：

```
1    int[] skipDoc = {127, 383, 1151, 3455}
```

当前skipDoc数组对应的SkipList如下所示：

图2：



由于文档号23小于skipDoc数组中的所有哨兵值，故不需要更新skipDoc数组中的哨兵值，那么直接遍历docDeltaBuffer，判断文档号23是否在该数组中即可。

文档号：700

更新前的skipDoc数组如下所示：

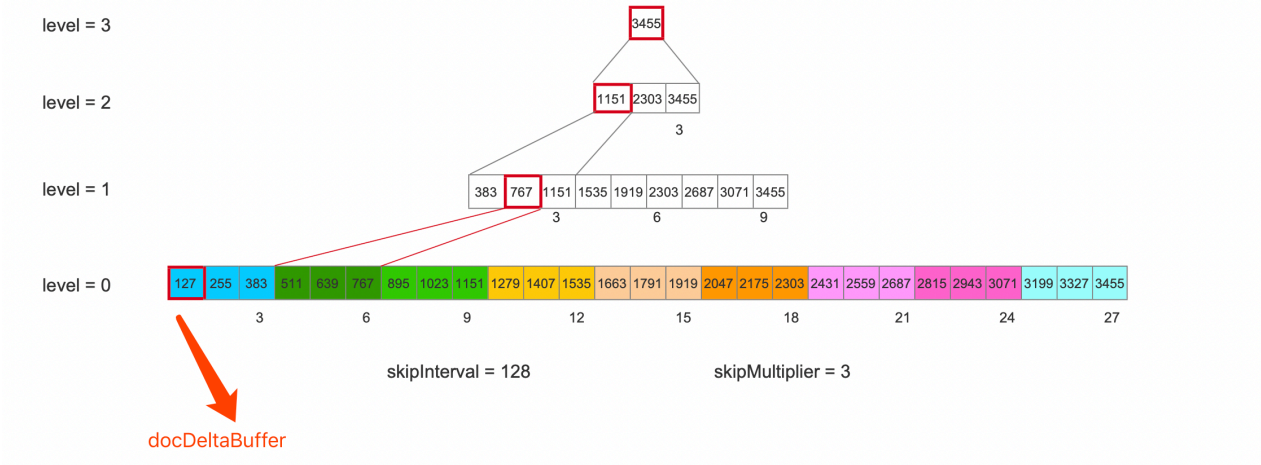
```
1    int[] skipDoc = {127, 383, 1151, 3455}
```

文档号700小于1151、3455，执行了步骤一之后，判断出需要更新哨兵值的层数N为2（两层），即只要从level=1开始更新level=1以及level=0对应的skipDoc数组中的哨兵值。

对于level=1层，在执行了步骤二后，更新后的skipDoc数组如下所示：

```
1    int[] skipDoc = {127, 767, 1151, 3455}
```

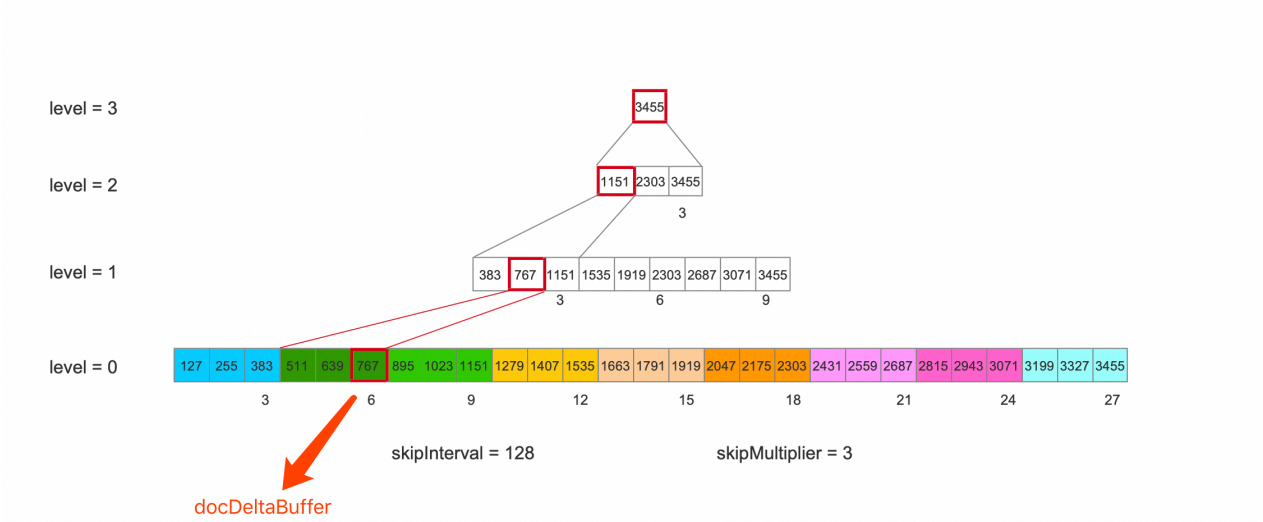
图3：



随后更新level=0层，在执行了步骤二后，更新后的skipDoc数组如下所示：

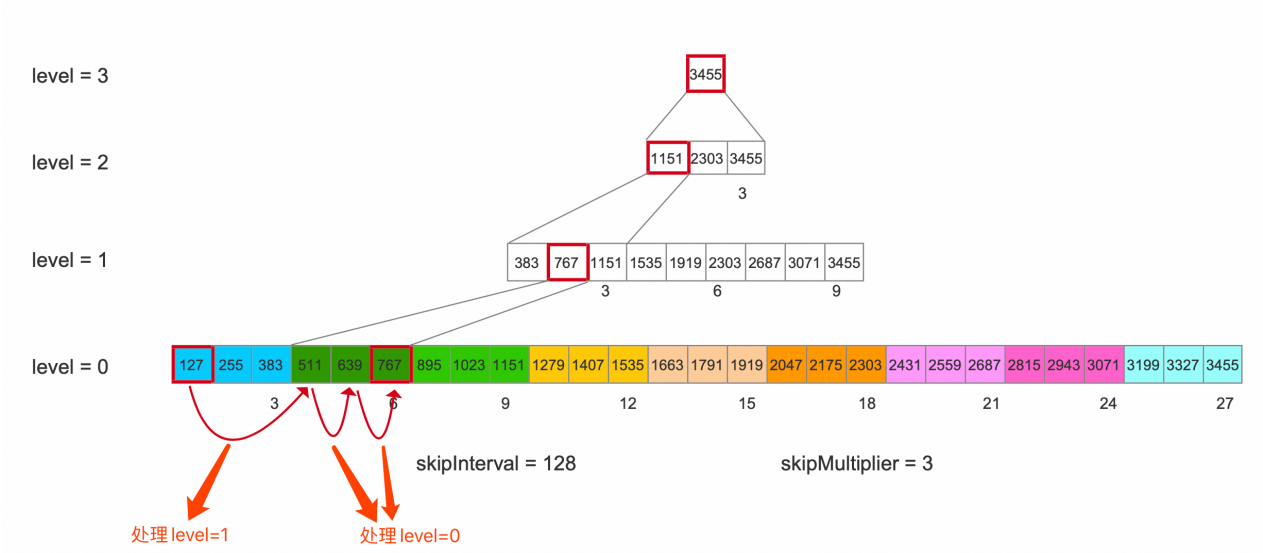
```
int[] skipDoc = {767, 767, 1151, 3455}
```

图4：



这里要注意的是，level=0层的datum更新过程如下所示：

图5：



从图5中可以看出，在更新的过程中，**跳过了两个datum**，其原因是在图3中，当更新完level=1的datum之后，该datum通过它包含的**SkipChildLevelPointer**字段（见索引文件的生成（三））重新设置在level=0层的哨兵值，随后在处理level=0时，根据待处理的文档号继续更新哨兵值。

文档号：701

更新前的skipDoc数组如下所示：

```
int[] skipDoc = {767, 767, 1151, 3455}
```

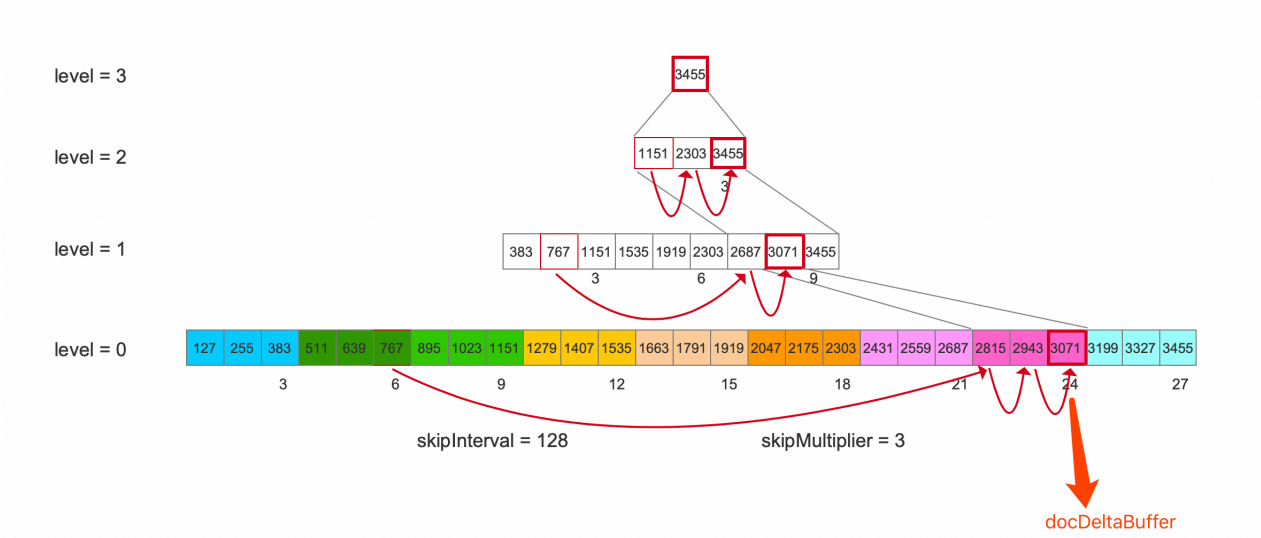
文档号701小于所有的哨兵值，所以直接遍历docDeltaBuffer数组即可。

文档号：3000

不重复用文字描述其更新过程了，直接以下图给出每一层最终的哨兵值以及更新过程，自己品...，更新后的skipDoc数组如下所示：

```
1    int[] skipDoc = {3071, 3071, 3455, 3455}
```

图6：



结语

可以看出，SkipList在Lucene中的实现适用于存储有序递增的文档号，性能上取决于待处理的文档号在datum的分布，分布在越少的datum，性能越高。

点击下载附件