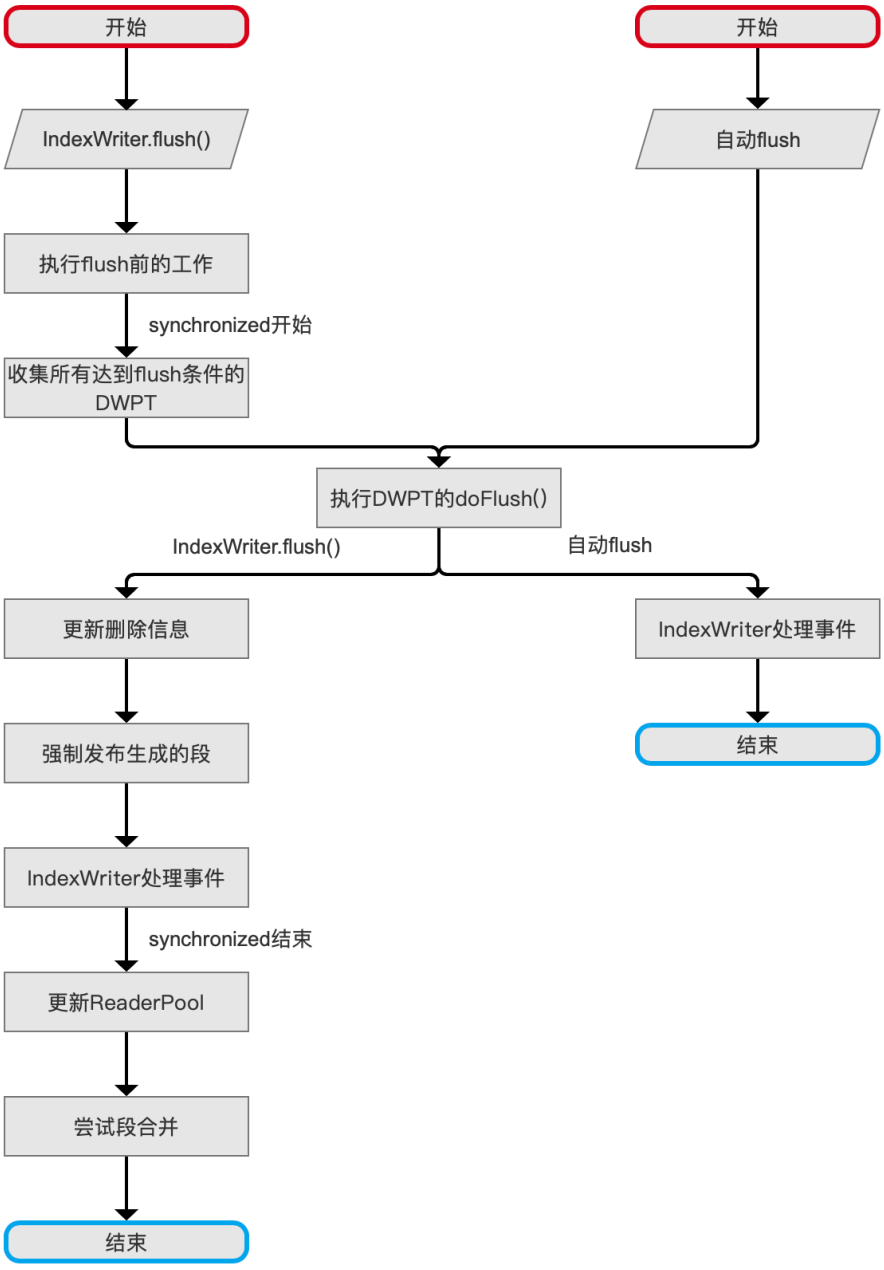
文档提交之flush（二）

本文承接文档提交之flush（一），继续依次介绍每一个流程点。

文档提交之flush的流程图

图1：



点击查看大图

执行DWPT的doFlush()

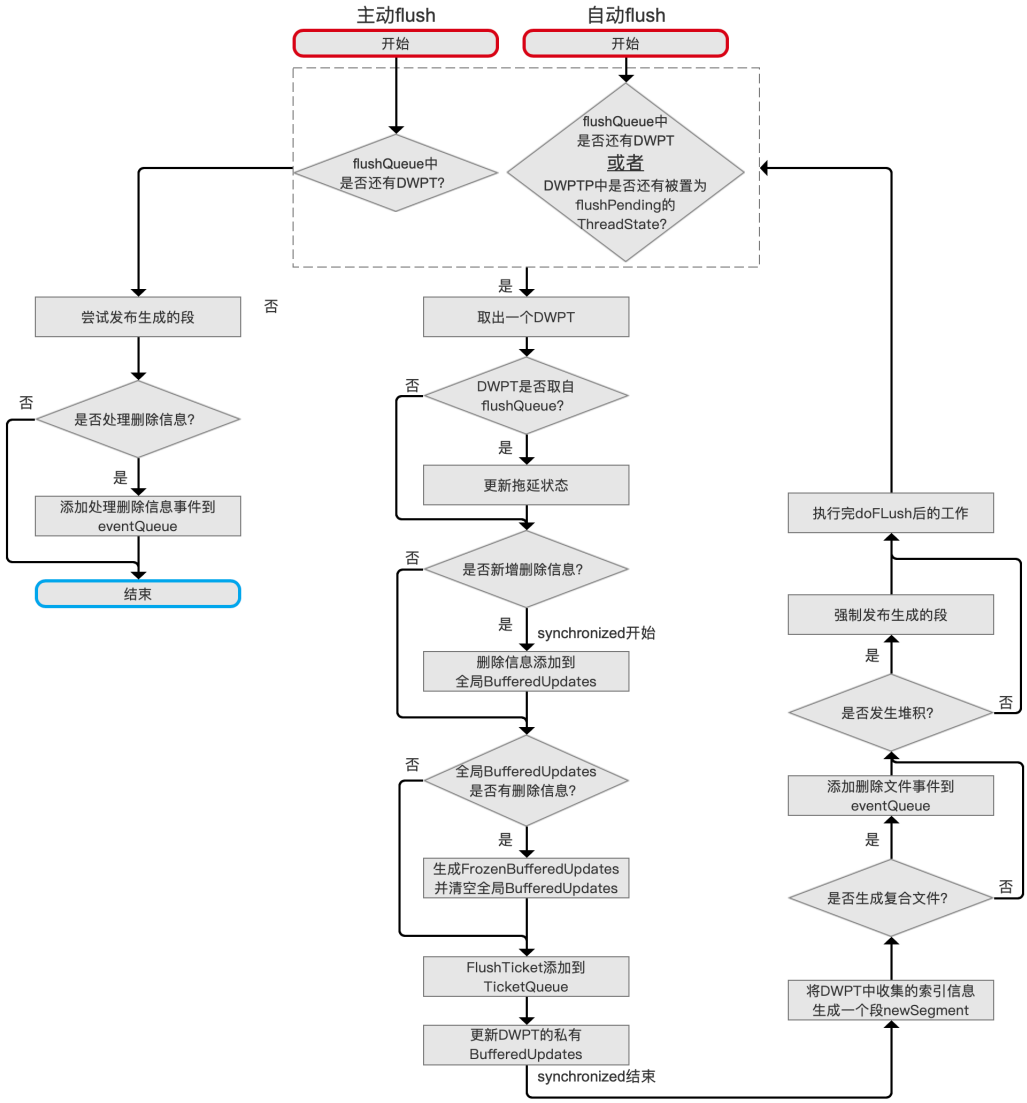
图2：

在文档提交之flush（一）的文章中，我们知道，执行主动flush并完成流程点`收集所有达到flush条件的DWPT`之后，具有相同删除队列deleteQueue的DWPT都被添加到了flushQueue中，而在当前流程点中，则是从flushQueue中依次去取出每一个DWPT，执行doFlush( )操作。

从图1中可以知道除了主动flush，自动flush的DWPT也会执行doFlush( )，由于两种情况使用同一个流程点，但实际流程点中的具体逻辑还是有所区别。

执行DWPT的doFlush()的流程图

图3：



点击查看大图

取出一个DWPT

图4：

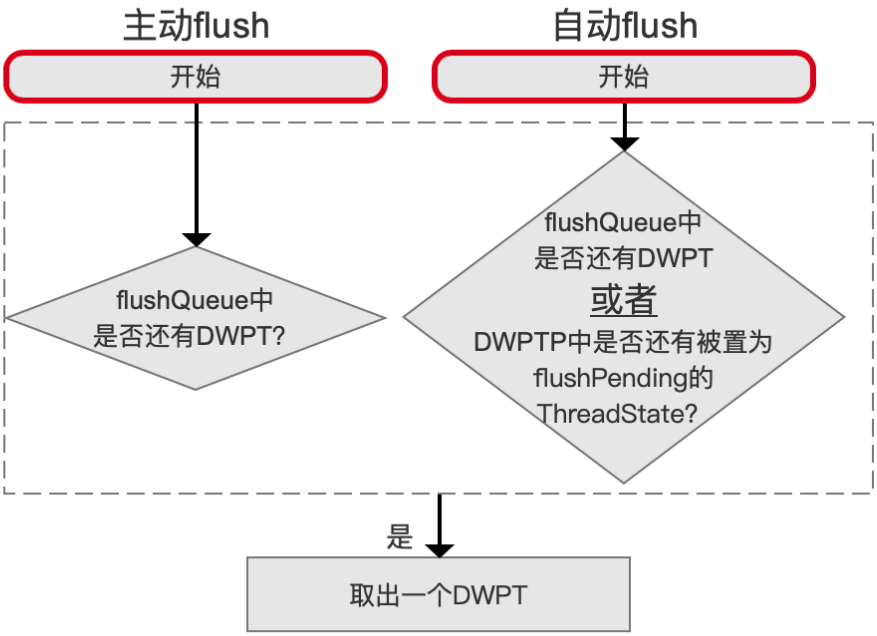


图4中，尽管主动flush跟自动flush都是执行DWPT的doFlush()，但是DWPT的来源是不一样的，在介绍为何来源不同之前，我们先说一个重要的事实：

- 主动flush跟自动flush不会同时进入`执行DWPT的doFlush()的流程图`这个流程点（该流程点的入口对应源码中DocumentsWriter.doFlush(DocumentsWriterPerThread)）

首先介绍下通过什么方式防止主动flush跟自动flush同时进入`执行DWPT的doFlush()的流程图`这个流程点：

- 在线程A触发了全局flush后，并且在flush期间（全局变量fullFlush为true）如果线程B（持有的DWPT跟当前正在flush的DWPT持有不一样的删除队列deleteQueue）中的DWPT执行了添加/更新的操作后，如果因为达到了maxBufferedDocs而触发自动flush，那么该DWPT会被添加到blockedFlushes中；如果因为达到ramBufferSizeMB而触发自动flush，并且该DWPT收集的索引量是DWPTP中所有跟它具有相同删除队列的DWPT集合中最大的，那么它也会被添加到blockedFlushes中，否则什么也不做。**另外当主动flush在执行完后，会马上将blockedFlushes中的DWPT添加到flushQueue中**
- 上面的描述同时也填了我们在文档的增删改（下）（part 3）中介绍blockedFlushes时挖的二号坑，即blockedFlushes的另一个作用是防止主动flush跟自动flush同时进入`执行DWPT的doFlush()的流程图`这个流程点

为什么不允许主动flush跟自动flush同时进入`执行DWPT的doFlush()的流程图`这个流程点：

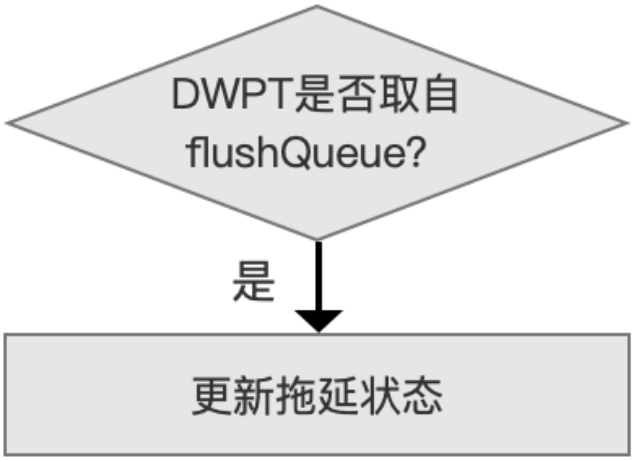
- 这里先暂时给出结论：为了能正确处理删除信息。在下面的流程点中会展开

为什么主动flush跟自动flush都是执行DWPT的doFlush()，但是DWPT的来源是不一样的：

- 主动flush：在文档提交之flush（一）的文章中我们介绍了，由于主动flush跟添加/更新文档是并行操作，此时的DWPTP中可能有最多两种包含不同的删除队列deleteQueue的DWPT类型，所以主动flush需要从DWPTP中挑选出这次flush的DWPT，并存放到flushQueue中，随后依次执行doFLush( )
- 自动flush：全局flush没有被触发，那么DWPTP中的DWPT肯定都包含相同的deleteQueue（原因见文档提交之flush（一）），所以可以从DWPTP中把所有满足flush条件的DWPT都依次取出来执行doFLush( )，为什么还要`先去`flushQueue中找DWPT呢：
    - 上文中我们提到，由于线程A正在执行主动flush，线程B中的满足flush条件的DWPT被存到了blockedFlushes中，并且在主动flush结束后，这些DWPT被添加到了flushQueue中，所以先到要flushQueue中找DWPT，并且这些DWPT能被优先执行doFLush( )，毕竟是它们的存在说明内存中堆积了不少的索引内存量

更新拖延状态

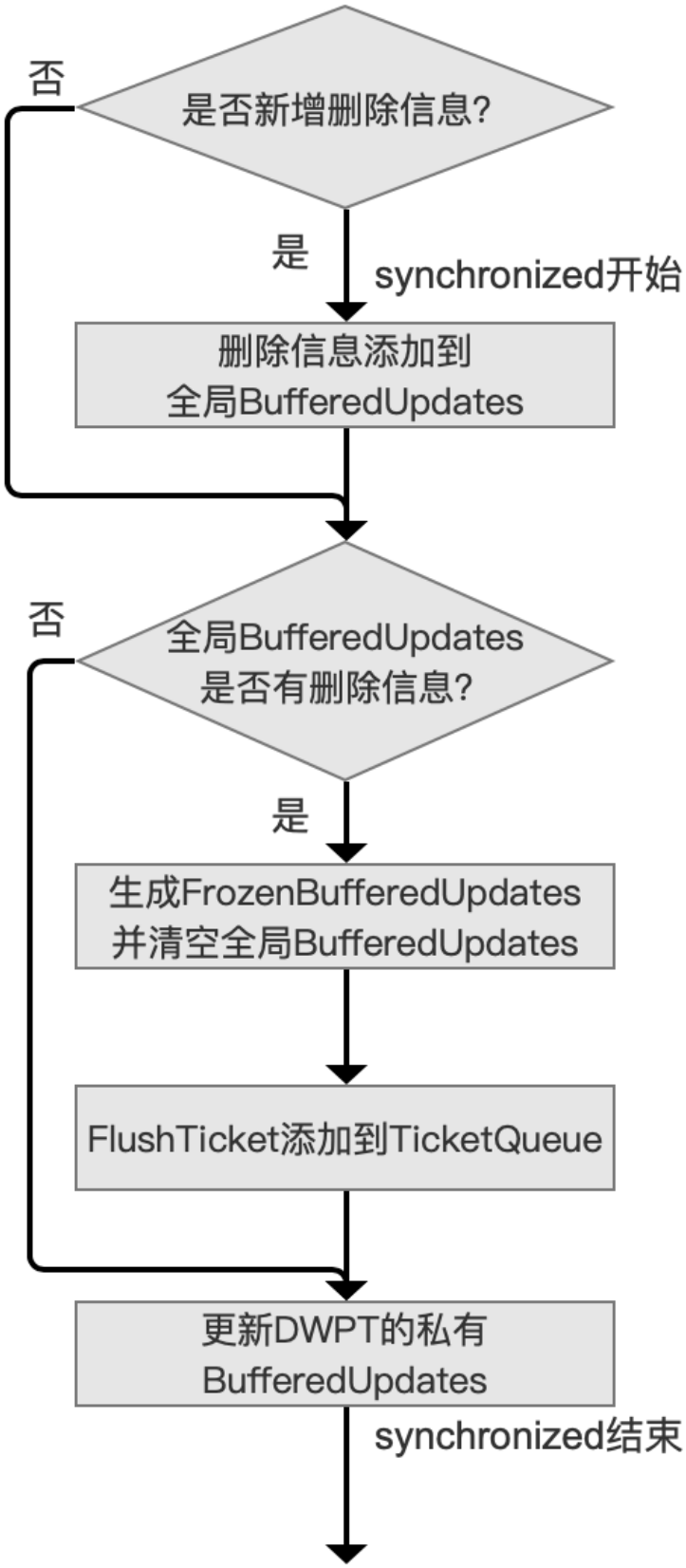
图5：



如果是从flushQueue中取出的DWPT，说明正在执行主动flush或者刚刚结束，那么需要更新拖延状态，其更新的原因在文档提交之flush（一）文章关于updateStallState的概念已经作了介绍，不赘述。

处理全局BufferedUpdates、更新DWPT的私有BufferedUpdates

图6：



从图5中的流程点`是否新增删除信息`可以看出，这个流程点只针对于自动flush，因为在主动flush阶段，新增的删除信息会被添加到新的全局删除队列newQueue中（见文档提交之flush（一）），所以主动flush使用的删除队列flushingQueue不会新增删除信息。

首先回顾下之前讲的内容，在文档的增删改（下）（part 2）的文章中我们提到，每个DWPT有一个私有的deleteSlice，该deleteSlice描述了作用于DWPT中的文档的删除信息，该删除信息最后会保存在DWPT的私有BufferedUpdates中，另外我们还介绍了一个全局的删除队列deleteQueue，它描述了所有的删除信息，该删除信息会被保存在全局BufferedUpdates（源码中的变量名是globalBufferedUpdates）中。

生成FrozenBufferedUpdates并清空全局BufferedUpdates

全局BufferedUpdates会被冻结（Frozen）为一个全局FrozenBufferedUpdates，它包含了删除信息，并且该删除信息在后面的流程中会作用于之前的所有段中的文档，之后清空全局BufferedUpdates。

为什么要清空全局BufferedUpdates：

- 主动flush：如果不清空，那么每一个DWPT都会生成一个全局FrozenBufferedUpdates，并且都携带了相同的删除信息（flushingQueue不再改变），这是没有必要的
- 自动flush：如果不清空，那么后生成的FrozenBufferedUpdates会携带之前生成的FrozenBufferedUpdates中的冗余删除信息

FlushTicket添加到TicketQueue

FlushTicket类中主要的两个变量如下所示：

```
static final class FlushTicket {
    private final FrozenBufferedUpdates frozenUpdates;
    private FlushedSegment segment;
    ... ....
}
```

其中frozenUpdates就是上文中包含删除信息且作用于其他段中的文档的全局FrozenBufferedUpdate，而segment则是DWPT对应生成的段，只是segment在随后的流程中才会被添加进来，最后FlushTicket对象被添加到TicketQueue中，TicketQueue的内容在后面的流程中会详细介绍。

更新DWPT的私有BufferedUpdates

不管是主动flush还是自动flush，DWPT在最后一次收集文档后，其他线程可能新增了deleteQueue中的删除信息，这些删除信息需要作用于该DWPT，故需要更新DWPT的私有deleteSlice，即让deleteSlice的sliceTail持有跟tail一样的对象引用（见文档的增删改（下）（part 2）），最后将deleteSlice中的删除信息更新到私有BufferedUpdates中。

在后面的`将DWPT中收集的索引信息生成一个段newSegment`流程中，我们会了解到，**DWPT的私有BufferedUpdates也会被冻结为其私有FrozenBufferedUpdates，其包含的删除信息只作用于DWPT自身收集的文档，或者说只作用于DWPT对应生成的段，而全局的FrozenBufferedUpdates则是作用于其他的段。**

为什么这里需要使用synchronized同步：

- 源码中给出了解释，为了避免可能翻译造成的不准确描述，所以这里直接原文：Since with DWPT the flush process is concurrent and several DWPT could flush at the same time we must maintain the order of the flushes before we can apply the flushed segment and the frozen

global deletes it is buffering. The reason for this is that the global deletes mark a certain point in time where we took a DWPT out of rotation and freeze the global deletes

- Example: A flush 'A' starts and freezes the global deletes, then flush 'B' starts and freezes all deletes occurred since 'A' has started. if 'B' finishes before 'A' we need to wait until 'A' is done otherwise the deletes frozen by 'B' are not applied to 'A' and we might miss to deletes documents in 'A'

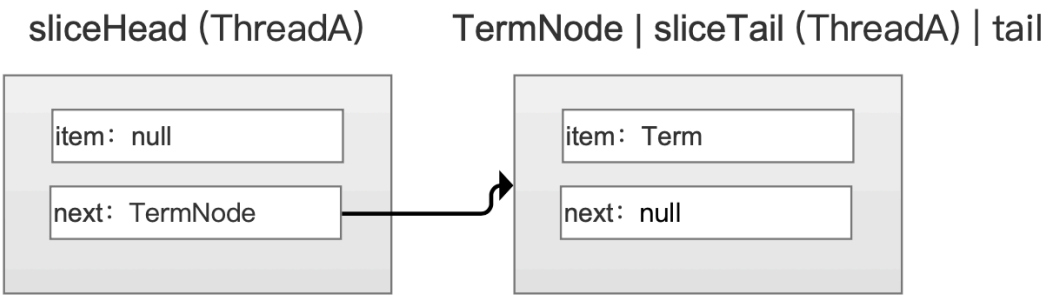
下面用一个例子来说明上文中描述的内容。

例子

有两个线程ThreadA、ThreadB，他们都是执行添加文档的操作，并且假设他们在执行完添加操作后两个线程中的DWPT都会生成一个段，并且全局删除队列deleteQueue为空。

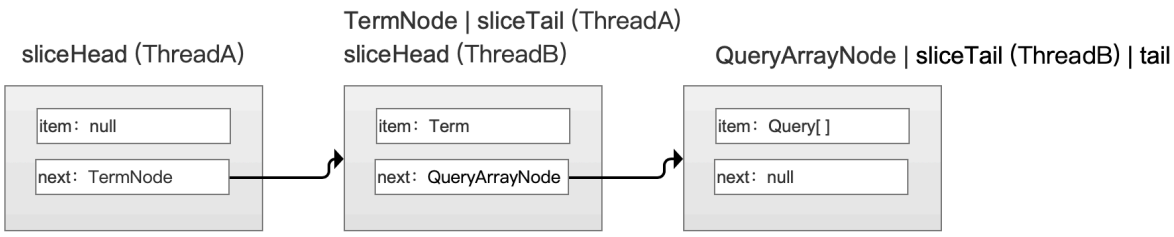
ThreadA在执行完添加操作之后，其他线程新增了一个删除信息（对应图6中的TermNode）到全局删除队列deleteQueue中，当ThreadA中的DWPT执行doFlush时，更新DWPT私有的deleteSlice（即更新deleteSlice中的sliceTail，deleteSlice的概念见文档的增删改（下）（part 2）），并且全局的删除信息会被冻结到全局FrozenBufferedUpdates（包含TermNode对应的删除信息），我们称之为F1，即删除信息TermNode将要作用于**已有的段中的文档**。

图7：



ThreadB在执行完添加操作之后，其他线程新增了一个删除信息（对应图7中的QueryArrayNode）到全局删除队列deleteQueue中，当ThreadB中的DWPT执行doFlush时，更新DWPT私有的deleteSlice，并且全局的删除信息会被冻结到全局FrozenBufferedUpdates（包含QueryArrayNode、TermNode对应的删除信息），我们称之为F2，即删除信息QueryArrayNode、TermNode将要作用于**已有的段中的文档**。

图8：



如果此时ThreadB优先ThreadA先生成一个段，那么F2中的删除信息，即QueryNode无法作用于ThreadA中的DWPT对应的段中的文档，并且在上面的例子中，F1中的删除信息还会错误的作用到ThreadB中的DWPT对应的段中的文档。

如果能同步两个线程的操作，那么在上面的例子中ThreadA中的DWPT会先生成一个段，并且F1中的删除信息TermNode作用于已有的段中的文档，然后情况全局BufferedUpdates，接着ThreadB中的DWPT会先生成一个段，并且F2中的删除信息QueryArrayNode（注意F2包含的删除信息只有一个QueryArrayNode）能作用于ThreadA中的DWPT对应生成的段以及其他已有的段中的文档，最后清空全局BufferedUpdates。

上面的描述同时解释了为什么不允许主动flush跟自动flush同时进入`执行DWPT的doFlush()的流程图`这个流程点，自动flush中的删除信息可能会无法作用于主动flush中所有的DWPT对应的段中的文档。

结语

下一个流程点`将DWPT中收集的索引信息生成一个段newSegment`会利用一篇文章的篇幅来介绍，同时也是填了之前的一个坑，即迟迟未写的两阶段生成索引文件之第一阶段系列的第二阶段，在这个阶段，会生成除了.fdx、.fdt和.tvx、.tvd之外其他所有的索引文件。

点击下载附件