

# 文档的增删改（下）（part 1）

本文承接[文档的增删改（上）](#)、[文档的增删改（中）](#)、继续介绍文档的增删改，为了能深入理解，还是得先介绍下几个预备知识。

## 预备知识

### DocumentsWriterStallControl

在[文档的增删改（中）](#)我们知道，多线程（持有相同的IndexWriter对象的引用）执行添加/更新操作时，每个线程都会获取一个ThreadState，每次执行完一次添加/更新的后，如果持有的DWPT对象收集的索引信息没有达到flush的要求，该索引信息的大小会被累加到activeBytes，否则会被累加到flushBytes中，并且执行flush操作，这种方式即添加/更新和flush为并行操作。

并行操作即某些ThreadState执行添加/更新，而其他ThreadState执行flush，故可能会存在添加/更新的速度一直快于flush的情况，导致内存中堆积索引信息，那么很容易出现OOM的错误。所以DocumentsWriterStallControl类就是用来通过阻塞添加/更新的操作来保证写入(indexing)的健康度(This class used to block incoming indexing threads if flushing significantly slower than indexing to ensure the healthiness)

满足下面的条件时需要阻塞添加/更新的操作：

```
(activeBytes + flushBytes) > limit && activeBytes < limit
```

其中limit的值为  $2 * ramBufferSizeMB$ ，ramBufferSizeMB描述了索引信息被写入到磁盘前暂时缓存在内存中允许的最大使用内存值。

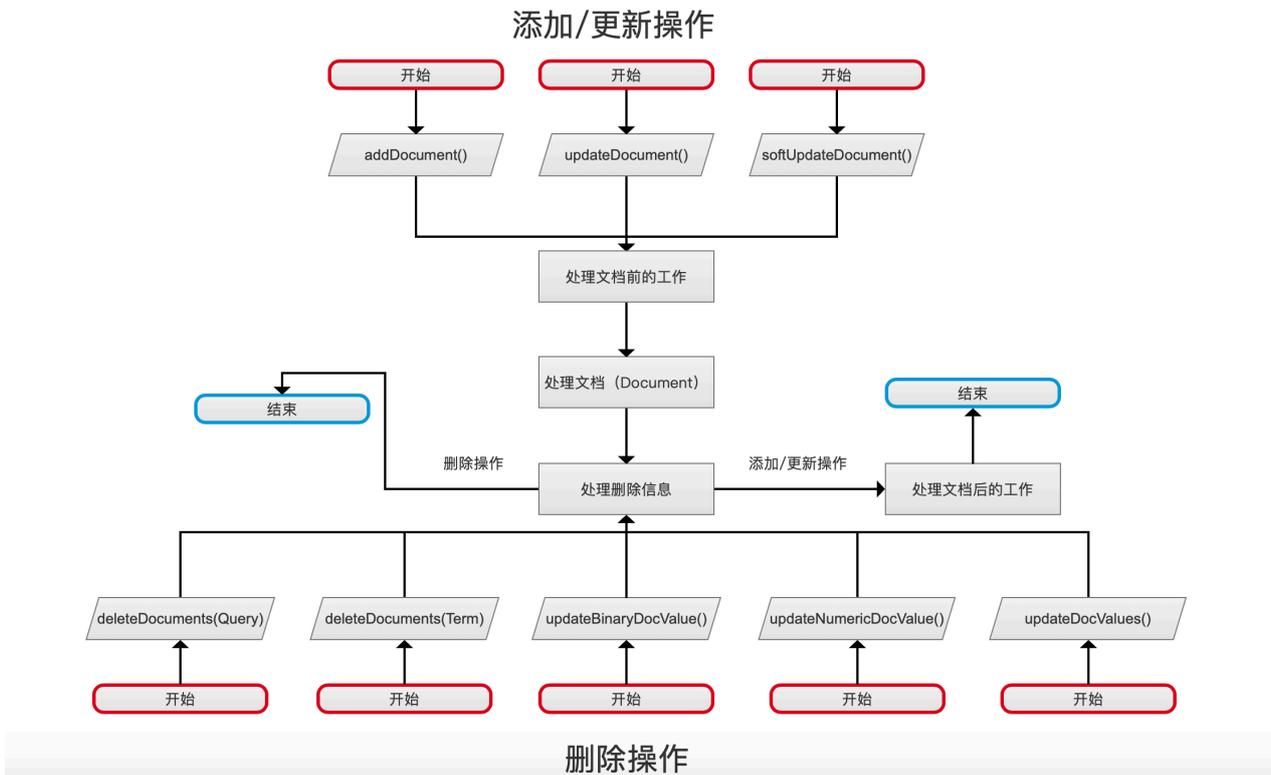
如何阻塞执行添加/更新操作的线程：

- wait(1000)：等待1秒的方法来实现阻塞，源码中的注释给出了使用这种方式的原因：Defensive, in case we have a concurrency bug that fails to .notify/All our thread, just wait for up to 1 second here, and let caller re-stall if it's still needed
- stall：该值是一个对所有线程可见的变量，当stall的值为true，那么需要执行阻塞

## 文档的增删改流程图

在[文档的增删改（上）](#)的文章中，我们给出了文档的增删改流程图，这篇文章对该流程图的每一个流程点进行介绍，由于当文档跟多文档的增删改流程是雷同的，故下文中只介绍单文档的增删改流程图：

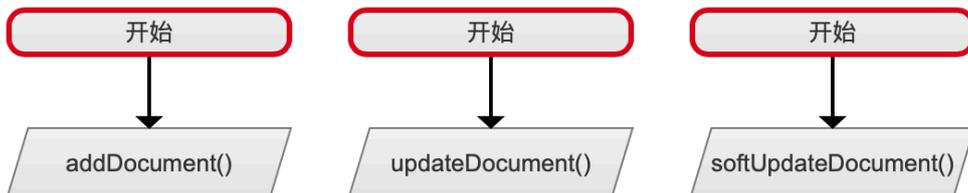
图1：



[点击查看大图](#)

## 添加/更新文档

图2:



在[文档的增删改（上）](#)中我们知道，updateDocument( )、softUpdateDocument( )的操作分为删除跟添加，其中添加的逻辑跟addDocument( )是一致，故这三种方法使用相同的流程先处理文档的添加操作，而不同点在于处理删除的操作，下文会详细介绍。

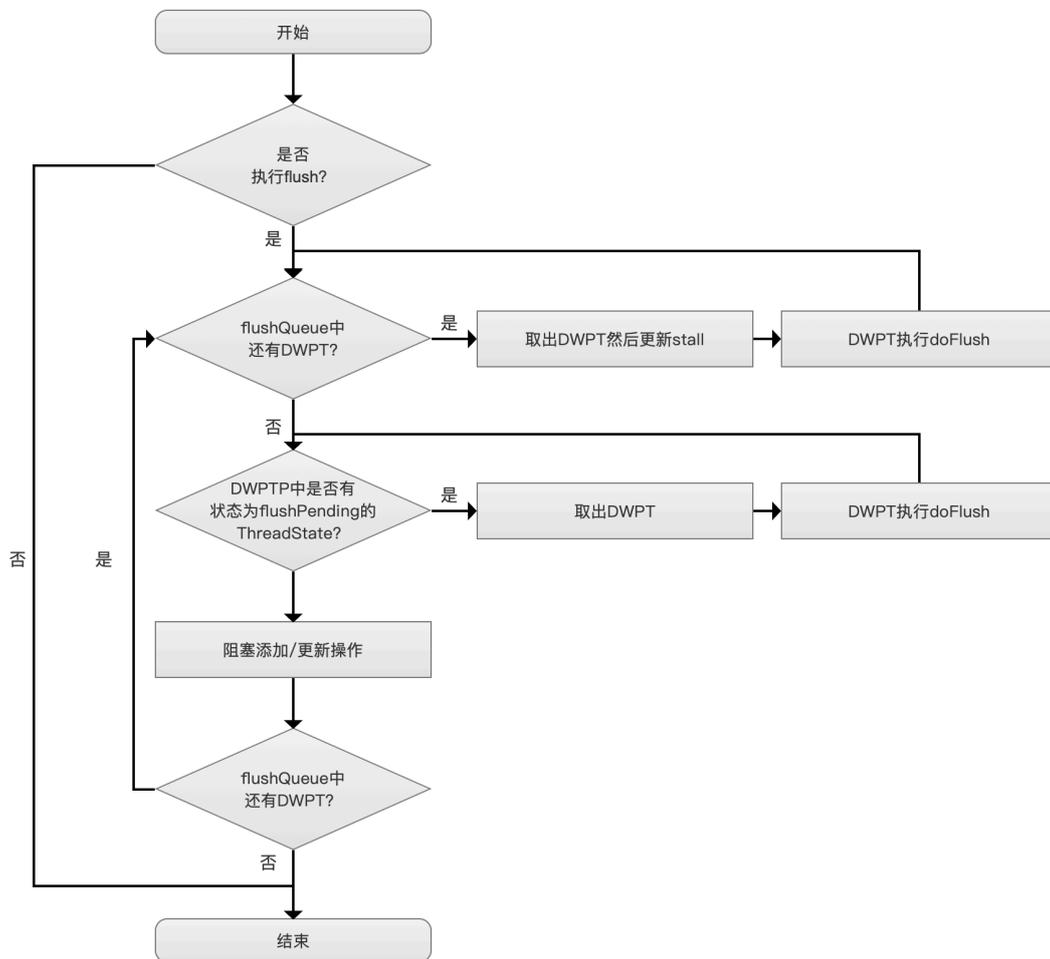
## 处理文档前的工作

图3:

## 处理文档前的工作

### 处理文档前的工作的流程图

图4:



[点击查看大图](#)

这个在源码中即调用DocumentWriter的preUpdate()方法。在处理文当前，如果满足下面的两种条件之一，需要执行上图的流程：

- stall为true：该值即上文中提到的DocumentsWriterStallControl中的变量stall，如果该值为true，说明当前flush的速度赶不上添加/更新
- checkPendingFlushOnUpdate为true && flushQueue.size > 0: checkPendingFlushOnUpdate

默认值为true，可以通过LiveIndexWriterConfig自定义设置，它描述了是否需要在每一次的添加/更新操作时去检查flushQueue中是否有待flush的DWPT。flushQueue的概念以及在后面的介绍flush的文章中会展开介绍，这里只要知道flushQueue中存放了待执行doFlush操作的DWPT集合

## 是否执行flush?

图5:



是否执行flush的条件即上文提到的两种条件。

## 处理flushQueue中的DWPT

图6:



该流程点从flushQueue中依次取出每一个DWPT，依次执行以下两个操作：

- 更新stall：重新计算 $(activeBytes + flushBytes) > limit \ \&\& \ activeBytes < limit$ ，更新stall的值
- 执行doFlush()：将DWPT中收集的索引信息生成索引文件，doFlush()的流程会在介绍flush时候详细展开

## 处理DWPTP中状态为flushPending的ThreadState

图7:



该流程点从DWPTP (DocumentsWriterPerThreadPool) 中取出每一个被标记为flushPending的ThreadState, 将其持有的DWPT对象引用执行doFlush的操作。

为什么要处理DWPTP中状态为flushPending的ThreadState:

- 这种情况主要考虑的是多线程的情况下, 因为ThreadState被置为flushPending和被添加到flushQueue这两个操作不是一个事务操作, 目的是尽量让所有达到flush条件的ThreadState中的DWPT执行doFlush()操作

## 阻塞添加/更新操作

图8:



由于上面执行了更新stall的操作, 所以在该流程点, stall的值可能被置为了false, 就不用阻塞等待, 反之需要等待1秒。

## 再次判断判断flushQueue中是否有了新的DWPT

图9:



同样地, 由于没有事务操作, 当执行到该流程点可能又存在新的待flush的DWPT, 这个流程点说明了当需要进行flush(满足flush的两个条件之一)时, Lucene会尽可能先将待flush的DWPT执行doFlush()操作。

## 处理文档 (Document)

---

图10:

## 处理文档 (Document)

在该流程点，我们先获得一个ThreadState，然后开始处理文档的操作，获得ThreadState的流程在[文档的增删改 \(中\)](#)已经介绍，不赘述。而处理文档的逻辑则在文章[两阶段生成索引文件之第一阶段](#)中已经介绍。在这篇文档中，详细的介绍了收集并生成.fdx、fdt、.tvd、tvm的过程，其他索引文件信息的收集流程会在随后介绍两阶段生成索引文件之第二阶段中详细展开，总之当处理文档 (Document) 的流程结束后，DWPT就收集到了所有的索引信息，基于这些信息生成索引文件。

## 结语

---

由于处理删除信息跟处理文档后的工作这两个流程点涉及的知识点较多，会造成本篇文章篇幅过长，故在挪到下篇文章中介绍

[点击下载](#)附件