**BlueStore源码分析-状态机**

文章参考自:https://blog.csdn.net/qq\_23929673/article/details/103935453?utm\_medium=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase

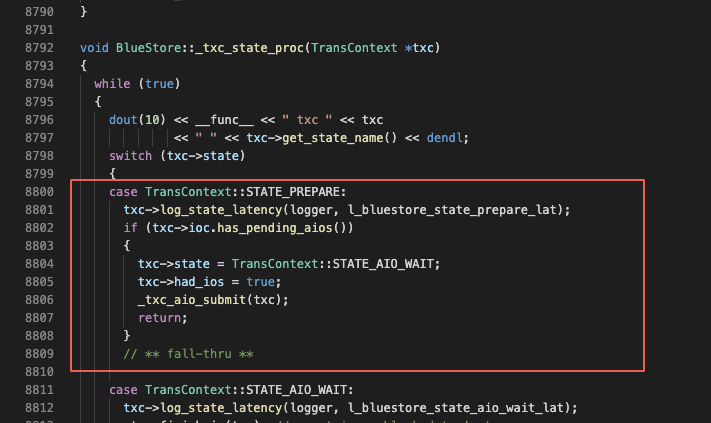
如有侵权请告知.

本篇我们分析不同状态下的动作.

**STATE\_PREPARE**

主要是调用\_txc\_add\_transaction进行事务的生成和提交.上一篇的写流程就是这个状态操作的.最后会调用状态机在该状态下对需要提交的异步IO进行提交,然后把状态流转到STATE\_AIO\_WAIT.

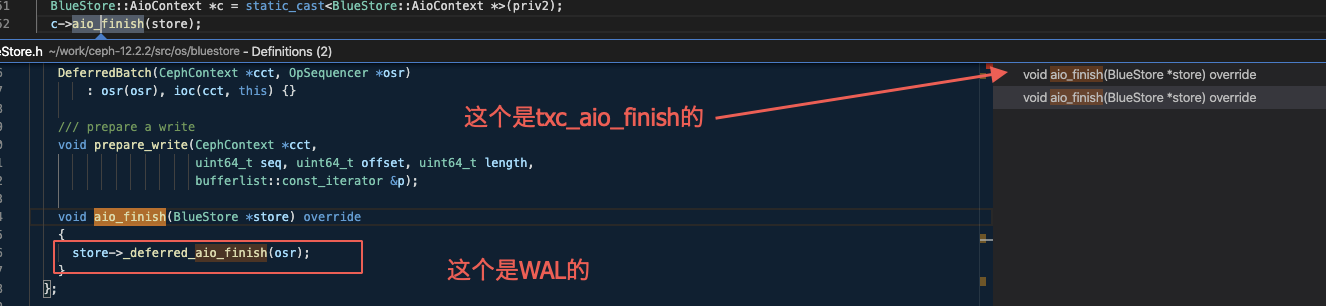
代码如下:

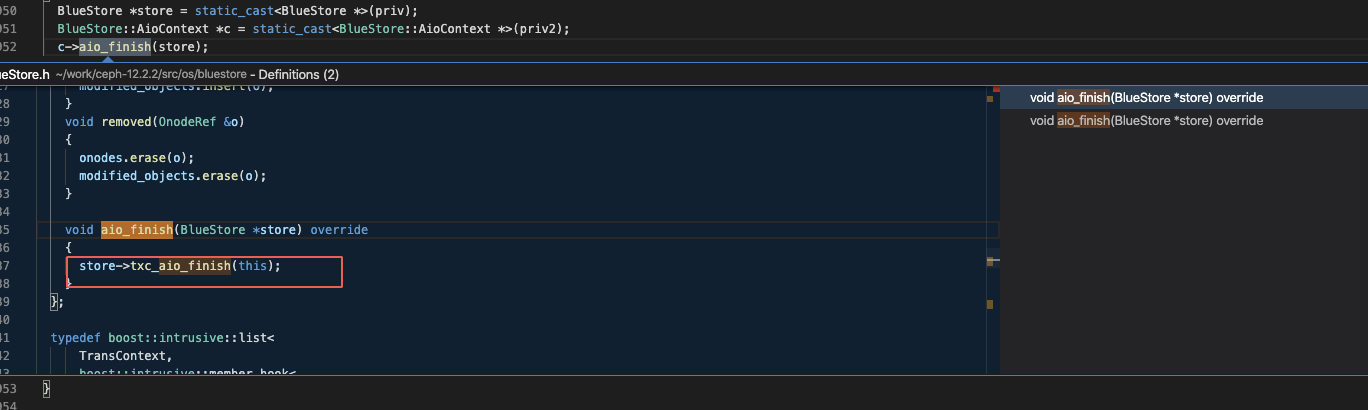


图中的has\_pending\_aios()就是判断是否有pending状态的io事件,这个在\_do\_write\_small或者\_do\_write\_big的时候有个aio\_write就做了这个记录,是一个原子类型的计数器.

在aio线程完成io操作之后会有个回调函数:txc\_aio\_finish():







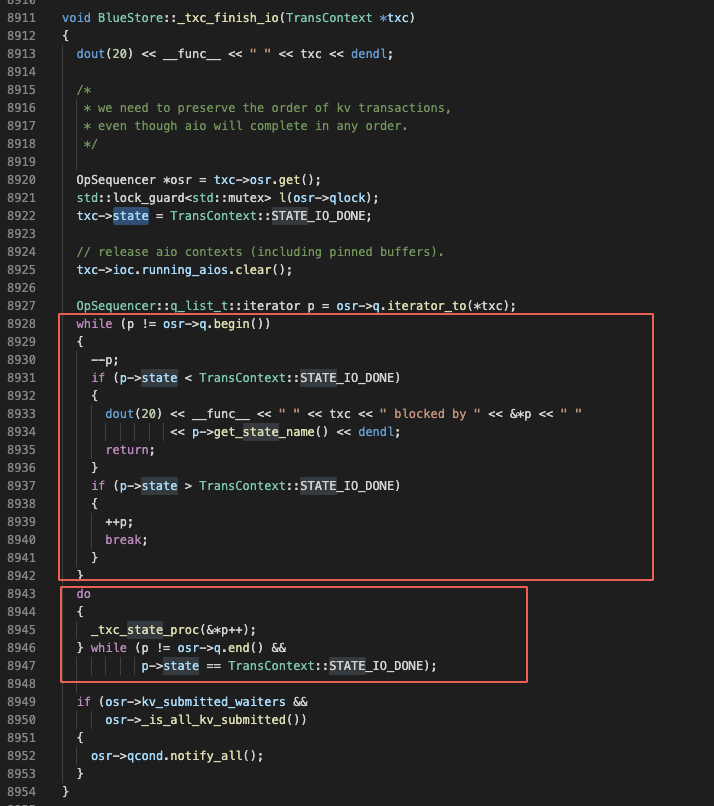
调用完成后会进入STATE\_IO\_DONE状态.

**作用:** 准备工作，生成大小写、初始化TransContext、deferred\_txn、分配磁盘空间等

**延迟指标:** l\_bluestore\_state\_prepare\_lat,从进入状态机到prepare阶段完成

**STATE\_AIO\_WAIT**

调用\_txc\_finish\_io进行SimpleWrite的IO保序等处理，然后将状态设置为STATE\_IO\_DONE再调用\_txc\_state\_proc进入下一个状态的处理。



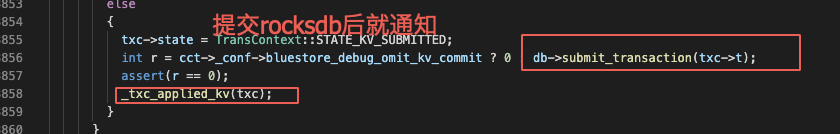
**作用:** 对IO保序，等待AIO的完成。

**延迟指标:** l\_bluestore\_state\_aio\_wait\_lat，从prepare阶段完成开始到AIO完成，平均延迟受限于设备

**STATE\_IO\_DONE**

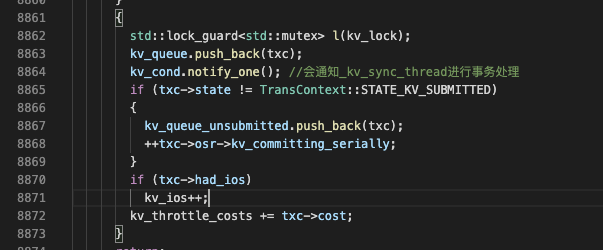
完成AIO，并进入STATE\_KV\_QUEUED阶段。会根据bluestore\_sync\_submit\_transaction做不同处理。该值为布尔值，默认为false。

如果为true，设置状态为STATE\_KV\_SUBMITTED并且同步提交kv到RocksDB但是没有sync落盘(submit\_transaction)，然后applied\_kv:



如果为false，则不用做上面的操作，但是以下操作都会做:

最后将事物放在kv\_queue里，通过kv\_cond通知kv\_sync\_thread去同步IO和元数据:



**作用:** 将事物放入kv\_queue，然后通知kv\_sync\_thread，osr的IO保序可能会block。

**延迟指标：** l\_bluestore\_state\_io\_done\_lat，通常很小主要耗在对SimpleWrite的IO保序处理上。

**STATE\_KV\_QUEUED**

在kv\_sync\_thread中进行数据提交,主要是meta数据和数据,然后状态设置为STATE\_KV\_SUBMITTED

**作用:** 从kv\_queue中取出事务然后执行它.

**延迟指标:** l\_bluestore\_state\_kv\_queued\_lat,从事物进入队列到取出事物因为是单线程顺序处理的，所以依赖于kv\_sync\_thread处理事物的速度。

**STATE\_KV\_SUBMITTED**

等待kv\_sync\_thread中kv元数据和IO数据的Sync完成，然后将状态设置为STATE\_KV\_DONE并且回调finisher线程。

**作用:** 等待kv元数据和IO数据的Sync完成

**延迟指标:** l\_bluestore\_state\_kv\_committing\_lat, 从队列取出事物到完成kv同步.

**STATE\_KV\_DONE**



**延迟指标:** l\_bluestore\_state\_kv\_done\_lat,deferred入队耗时,忽略不计.

**STATE\_FINISHING**

**作用：** 设置状态为STATE\_DONE，如果还有DeferredIO也会提交。

**延迟指标：** l\_bluestore\_state\_finishing\_lat，平均延迟0.001ms。

**STATE\_DONE**

**作用：** 标识整个IO完成。

**延迟指标：** l\_bluestore\_state\_done\_lat。

**STATE\_DEFERRED\_CLEANUP**

**作用：** 清理defferred IO在RocksDB上的WAL。

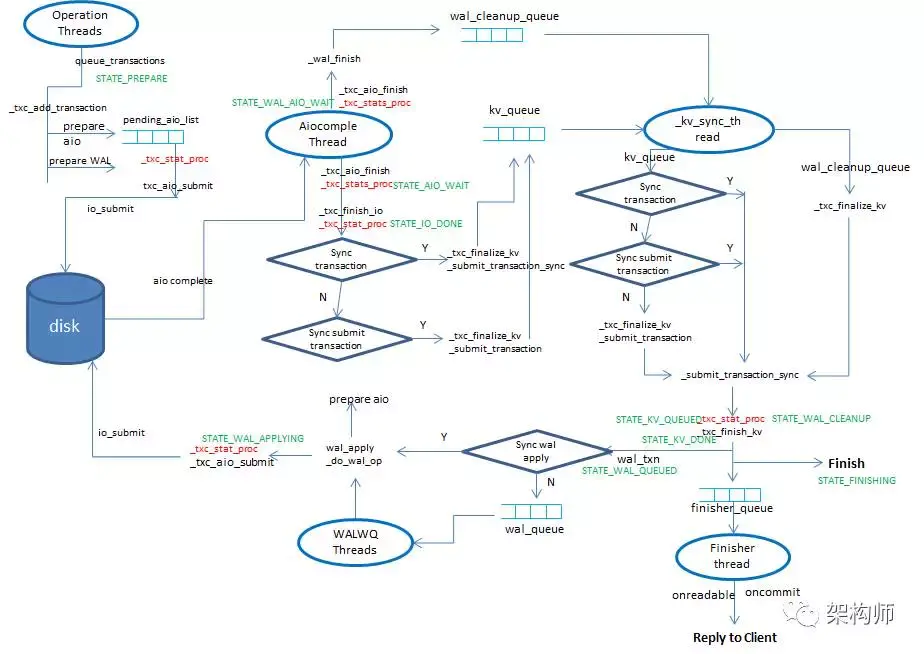
**延迟指标：** l\_bluestore\_state\_deferred\_cleanup\_lat

cleanup的时候是device回调了aio\_finish:



然后调用defferred的finish函数,然后将事务的状态设置为STATE\_DEFERRED\_CLEANUP

整体状态机在事务中的流转:(图片来自https://www.jianshu.com/p/880033db42a7 侵权请联系删除)



**核心线程队列**

**kv\_sync\_thread**

主要针对deferred\_done\_queue, deferred\_stable\_queue, kv\_queue进行操作.sync数据和meta数据.

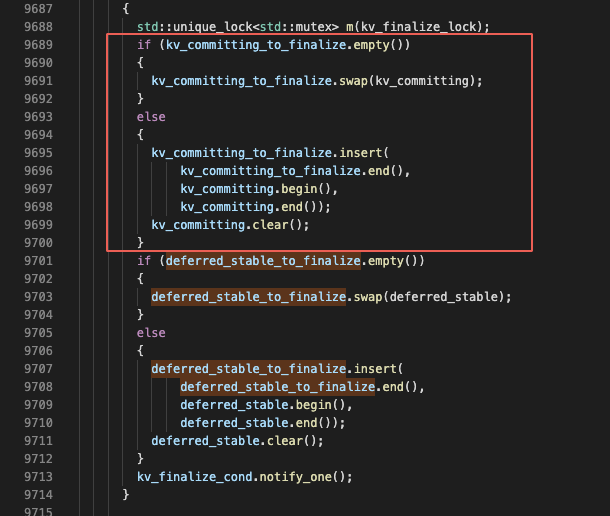
事件生成后,把txc放入kv\_queue,由kv\_sync\_thread处理.

simpleWrite的写,都是些新的block,如果存在cow,也是写新block,然后对旧block回收,写完device之后更新meta信息,这样突然掉电是不会破坏之前已经写入的数据的,而且也没有对用户返回成功,不会造成不一致的问题.

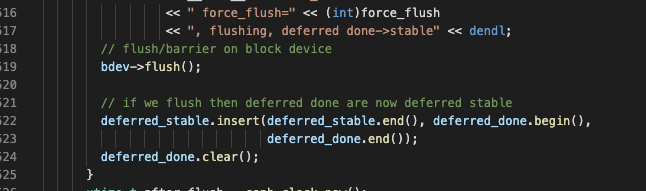
deferredWrite会在prepare阶段写WAL数据到rocksDBde db\_transaction中,kv\_sync\_thread进行commit,将wal持久化到kv系统.

具体的涉及到的队列的操作如下:

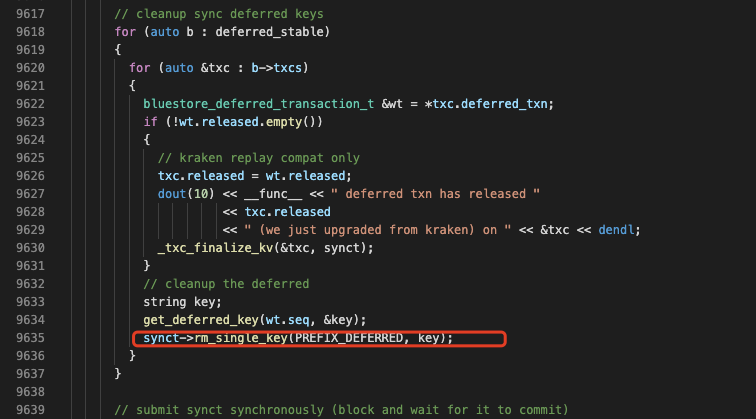
kv\_queue:执行txc中需要commit的事务,把kv\_queue的txc放到kv\_commiting,然后执行把txc里面的kvDB的transaction提交到rocksDB,设置状态为STATE\_KV\_SUBMITTED,然后把kv\_committing放入kv\_committing\_to\_finalize.等到kv\_finalize\_thread执行.

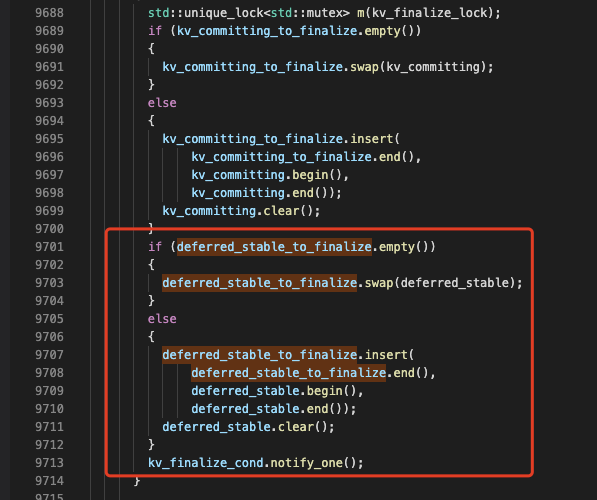


deferred\_done\_queue:已经完成deferredIO的事务还没有flush到stable存储中,如果已经flush了就会放到deferred\_stable\_queue队列中.



deferred\_stable\_queue: deferredIO已经flush到stable存储了,准备清理rocksdb的WAL,操作txc删除,.然后txc写入到deferred\_stable\_to\_finalize等待kv\_finalize\_thread执行.

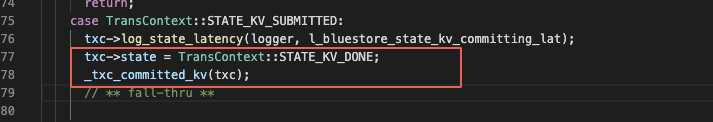


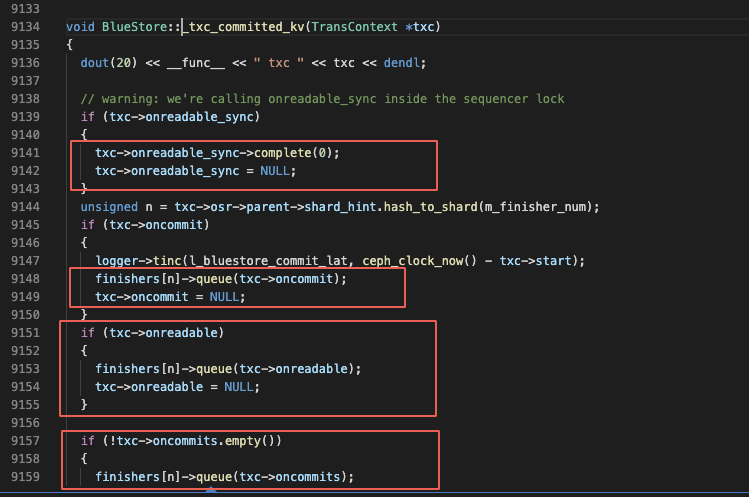


**kv\_finalize\_thread**

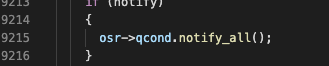
队列为kv\_committing\_to\_finalize、deferred\_stable\_to\_finalize，执行清理功能.

kv\_committing\_to\_finalize: 设置状态机状态为STATE\_KV\_DONE,执行回调通知用户io操作完成.

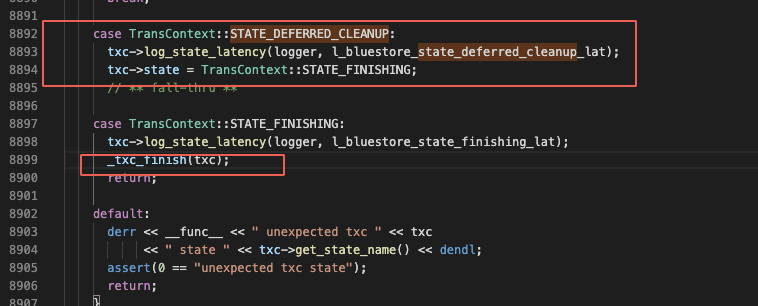




在FINISHING状态下调用\_txc\_finish会通知用户:

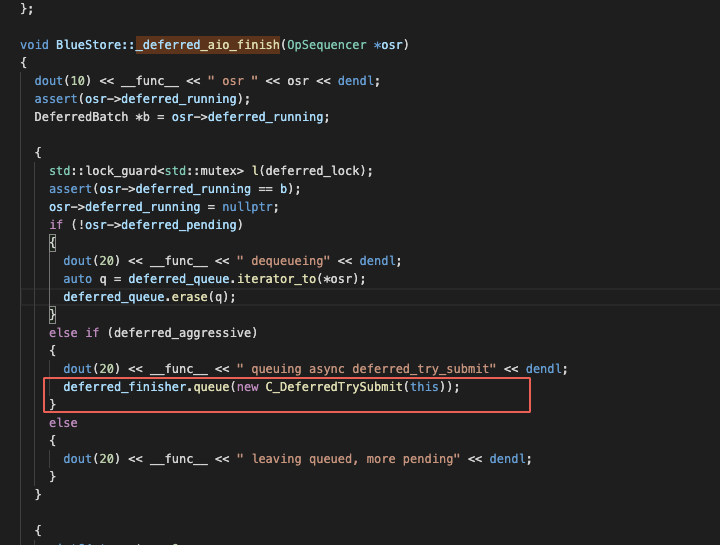


deferred\_stable\_to\_finalize: 遍历deferred\_stable，调用\_txc\_state\_proc进入状态机，设置状态为STATE\_FINISHING，继续调用\_txc\_finish，设置状态为STATE\_DONE，状态机结束，事物完成.



**deferred\_finisher**

在aio\_cb中的aio\_finish回调中触发的,会调用回调函数提交DeferredIO的请求



其它的线程队列暂时没有仔细了解.待进一步学习补充.