提交线程的工作

提交线程处理了事务非常核心的工作，例如更新系统的最新数据版本，写日志，同步日志等。由于这些工作都是需要在多个事务间同步，所以在设计上它是一个单线程，严格按照事务的提交顺序进行处理。在这篇文章中，我们主要详述提交线程的每一步工作。

# 提交线程的基础设计

提交线程继承了SeqQueueThread。该类的主要字段如下：

|  |  |
| --- | --- |
| 字段名 | 解释 |
| static const int64\_t QUEUE\_WAIT\_TIME = 10 \* 1000; | 从任务队列中取包的等待时间 |
| bool inited\_; | 是否已经初始化 |
| pthread\_t pd\_; | 处理线程的句柄 |
| volatile bool run\_flag\_; | 线程运行的标志 |
| int64\_t idle\_interval\_; | 空闲等待 |
| int64\_t last\_idle\_time\_; | 上次空闲时间 |

QUEUE\_WAIT\_TIME, idle\_interval\_, last\_idle\_time\_ 这几个字段是为了构造线程的“空闲机制”。线程会尝试从任务队列中取包，如果超过 QUEUE\_WAIT\_TIME 后还是获得，那么就会转入空闲判断的分支。如果发现自上次空闲（last\_idle\_time\_）至今已经经过了idle\_interval\_长度的时间（5ms），那么认为系统已经再次空闲，一旦空闲就会触发空闲任务（将缓冲区中的日志写入磁盘）。

## 线程调度逻辑

提交线程会不断的从任务队列中取出任务，根据子类（）定义的逻辑处理每一个任务。每次取任务都会尝试性的等待一定的时间，超时后就会进行空闲处理，主要是将缓冲区中的日志写入磁盘，同步备机；发布已经写完日志的任务，并将其加入下一个线程池中（这些逻辑都定义在TransCommitThread中）。以下是提交线程的基础调度逻辑。



(ps: ObSeqQueue::get 的行为非常有意思，整个类是专门为了提交线程设计的，多写者，单读者，并且严格按照编号顺序读)

## 任务出入队列的策略

提交任务插入到提交队列（ObSeqQueue）时，首先会获得一个全局的版本号，这是一个递增的结构，提交任务在获得这个编号后插入队列中。所有的任务都严格按照这个编号出队列。（所以并不是先进先出），这种出队列的方式是为了满足提交的语义（严格按照版本号对事务进行提交）。以下是任务入队列的逻辑：



# 提交线程的详细设计

TransExecutor继承了提交线程类，继承结构为：TransExecutor : TransCommitThread : SeqQueueThread。以下是继承后与提交线程相关的重要函数为：

handle\_commit(void \*ptask, void \*pdata); //处理提交任务

get\_seq(void \*task); //任务入队列时编号的获得方案

这两个函数包含了最主要的提交流程。我们将重点介绍在提交过程中需要执行的任务。以下是 get\_seq 的执行逻辑，这里最核心的是以下及各项任务：

1. 获得版本号；
2. 设置上下文版本号；
3. 预提交；



以下是 handle\_commit 在执行过程中需要完成以下任务。



这里主要的工作是

1. 处理已经完成写完日志的任务
2. 执行当前任务的提交工作。
3. 对特殊提交任务，需要写将已有的日志写完，然后执行特殊任务

以下是handle\_write\_commit\_的处理流程，负责处理事务日志的提交工作。



主要工作有以下几点：

1. 触发 ES\_COMMIT任务；
2. 增加对版本号的检验
3. fill\_log 中的一些任务
4. 日志缓冲区满的时候进行写日志

5、提交队列中后继任务尚未准备好时进行写日志

fill\_log负责将日志加入日志缓冲区，并且将提交任务加入到等待队列中，其执行逻辑如下：



主要的工作为：

1. 增加Memtable的检验
2. 将版本号，检验值等信息填入日志
3. 将日志加入日志缓冲区
4. 将任务加入flush\_queue 等待队列中，当任务的日志确定写入磁盘，同步备机后将被唤醒调度
5. 冻结事务上下文，防止被杀死

事务日志加入缓冲区后不会立刻写入磁盘，同步备机（commit\_log\_）；而是要等待与其他事务成组提交。成组提交的触发提交有以下几个：

1. 日志缓冲区满 （handle\_write\_commit中）
2. 没有后继的提交任务（hanlde\_write\_commit中）
3. 提交线程空闲时（on\_commit\_idle, SeqQueueThread::on\_idle）
4. 遇到特殊任务，需要确保已有日志都完成（handle\_commit 中）

commit\_log\_ 的逻辑在本文中不做介绍，其核心工作是处理日志缓冲区，写磁盘，同步备机；与事务管理的关联性比较小。

当事务的日志完成写入后，会有handle\_flushed\_log\_处理这些任务。其核心逻辑如下：



主要工作为：

1. 判断对尾任务的日志是否已经小于当前已提交的最大日志号，是则说明任务日志全部完成；否则需要进一步等待，函数退出
2. 触发任务的 publish 任务。
3. 修改任务类型，加入 CommitEndHandlePool线程池中，这将会释放上下文，回应客户端。

以上是提交线程在处理 事务日志提交 时的完整流程。目前还缺少对特殊任务的处理逻辑。

# 总结

提交线程中处理的核心任务是需要串行执行的，修改的数据对象是需要并发控制的。我们总结提交线程处理的任务如下：

1. 获得版本号，对事务进行预提交。这部分工作实际上不是在提交线程中完成的，是插入线程任务队列中触发的。
2. 触发 ES\_COMMIT任务；
3. 触发 ES\_PUBLISH任务。
4. 检验任务：版本号的检验，Memtable的检验
5. 冻结事务上下文，防止被杀死
6. 维护日志内容，写入版本号，检验码，将日志加入日志缓冲区
7. 成组提交日志，同步备机
8. 将完成日志操作的任务加入CommitEndHandlePool队列。

提交工作由于是单线程完成，容易成为系统瓶颈（可以添加实验数据佐证）。对它的优化是极为重要的，我觉得有以下工作可以尝试

1. 对比使用多线程（修改日志缓冲区，全局发布版本号，MemTable校验码等采用并发控制）与单线程串行提交的优点与缺点；
2. 并行一些开销巨大的子任务，例如将日志写入缓冲区的工作并行化。