










SimpleDB lab4 实验报告

一、Git Commit History

May 24, 2023 - 1 commit	
 lab4第三次提交, 完成了lab4 Wentao Tang authored 1 day ago	f3b6ccba  
May 19, 2023 - 1 commit	
 lab4第二次提交, 完成了exercise3、4 Wentao Tang authored 6 days ago	a746a1be  
May 16, 2023 - 1 commit	
 lab4第一次提交, 完成了exercise1、2 Wentao Tang authored 1 week ago	a5cccec28  

二、设计思路

Exercise1

加锁思路为：用 HashMap 记录每个 Page 上的锁，锁的类型用整数表示，0 为 exclusive 锁，大于 0 则为 shared 锁，其值表示 shared 锁的重数。HashMap 的 key 为 PageId，若其中没有这个 key，说明 Page 上没锁。另外还有一个 HashMap 记录每个事务所拥有的锁，key 为 TransacId，value 为 HashSet<PageId>。当每个事务获取锁时，根据加锁逻辑判断阻塞或获取锁。

如下为加锁逻辑

- 1、事务读 page 前需要加 shared 锁
- 2、事务写 page 前需要加 exclusive 锁
- 3、shared 锁可共享
- 4、只有一个事务能拥有对一个 page 的 exclusive 锁
- 5、当事务为某个 shared 锁的唯一拥有者时，可以将锁升级为 exclusive 锁

画出如下的加锁流程图：

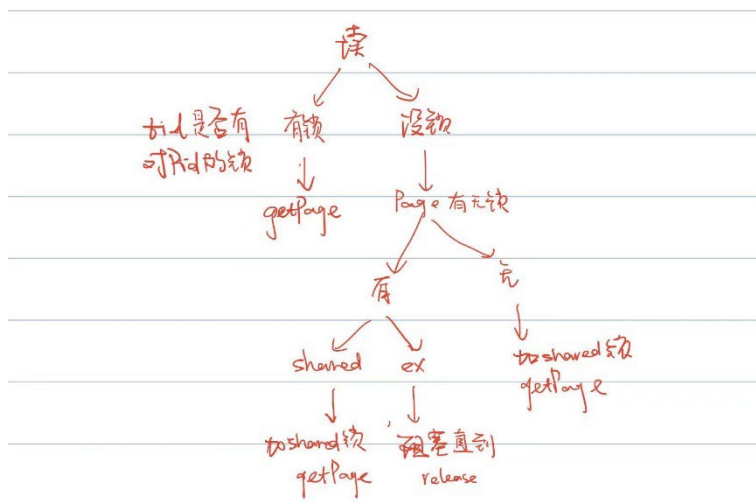


图 1 READ_ONLY 加锁流程图

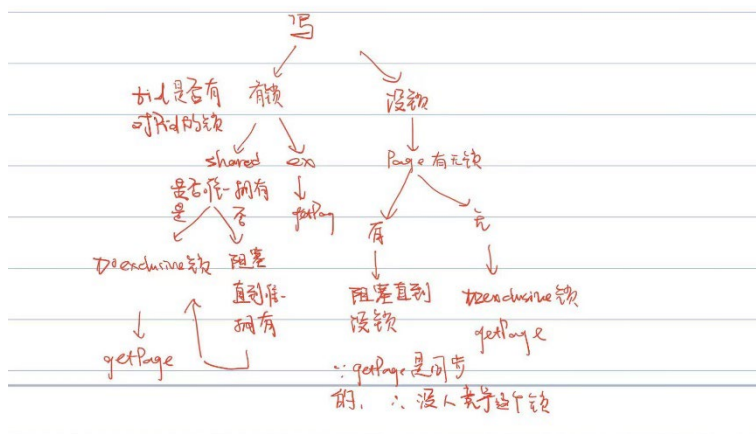


图 1 READ_WRITE 加锁流程图

Exercise2

为实现 2PL，只需确认适当地使用了 BufferPool.getPage()，即可保证在一个事务中进行正确的加锁。此外需要考虑一些特殊情况。

- 1、向 HeapFile 后写入新 Page 时，Page 层面的锁无法避免竞争。为解决该问题，我将 inert 方法进行同步，避免多线程同时插入。
- 2、当在寻找有空 slot 的 page 进行插入时，若确定当前 page 无空 slot，可直接释放该 page 上的锁。

Exercise3

为实现 NO STEAL，需要保证一个事务内不 flush 脏页。所以需要在 evictPage 时选择非脏页，若 BufferPool 中全都是脏页，抛出 DbException。

Exercise4

事务结束时，需要释放该事务拥有的所有锁，并去除事务对锁的所有权。若为 commit，需要将 BufferPool 中所有的更改写入磁盘；若为 abort，需要将磁盘中的数据恢复到 BufferPool。

Exercise5

采用基于时间限制的死锁检测策略，当线程阻塞的 while 循环次数超过一定限制时，就抛出 TransactionAbortException。

三、思考题

实现基于依赖的死锁检测。由于精确的找出事务之间的等待依赖开销较大，影响性能。我结合基于时间限制与依赖图两种方法，采用部分条件进行死锁的检测：当线程阻塞的 while 循环超过一定次数后，每个循环步进行死锁检测；检测时仅判断当前事务所请求的 page 的锁拥有者是否也在阻塞，即等待其他线程，若是，则认为有死锁。

另外，在拥有 shared 锁的事务请求 exclusive 锁，即锁的升级时，可以简单地实现死锁的精确检测。若此时该事务处于阻塞，且循环超过一定次数后，每个循环步进行死锁检测：在等候列表中寻找一个 TransactionId id，若 id 应不等于当前 tid，id 拥有对 pid 的锁，并且希望得到对 pid 的锁，即 id 也希望升级，则出现了多个拥有同一个 Page 的 shared 锁的事务都希望得到 exclusive 锁的死锁情况。

在 TransactionTest、DeadLockTest 等测试中验证性能，明显感受到应用混合死锁检测算法后，运行速度得到了明显提升。

四、重难点

- 1、在判断线程是否能获取锁时，需要用 synchronized 进行同步，确保只有一个线程获得锁。加锁时也需要同步，防止多线程同时改写共享变量导致出现无法预测的情况。
- 2、在遍历事务拥有锁的列表释放锁时，需要将遍历对象进行 clone() 深拷贝，防止对同一对象一边遍历一边修改，出现 ConcurrentModificationException
- 3、采用部分条件进行死锁的检测：当线程阻塞的 while 循环超过一定次数后，每个循环步进行死锁检测；检测时仅判断当前事务所请求的 page 的锁拥有者是否也在阻塞，即等待其他线程，若是，则认为有死锁。