PaxosLease

使用paxos本身实现选主功能，即将value部分定义为主节点leader id + 租期Time，根据paxos的特性，最后会唯一的确定下来一个value，即唯一的leader和对应租期，而这也是lamport在paxos make simple论文中所提到的，论文中还提出成员变更也可采取此方法实现。

Question：

依据此想法，如果一切正常没有过大的网络延迟现象出现，那么acceptor回复给proposer的消息中应该是空value，这样最早参与选主的proposer可以放上自身的提案（即选择自己为leader，并指定租期），然后顺利的当选。

但是，如果acceptor回复给proposer的不是空value，如：5个节点，其中proposer 1发出prepare，被acceptor 2，3 回复后，正常进入第二阶段，但是在proposer 1 发出accept消息到达acceptor 3之前，有另外的proposer 5发出的prepare给acceptor 3，4使之亦进入第二阶段，并且accept消息先行到达了acceptor 3，但是proposer 5出现故障，迟迟未能进入learn节点；于是proposer 1 发起新的提案，此时网络顺畅，最终达成了一致，此时的value实际上是proposer 5提出的value，但却由proposer 1实现。

那么proposer 1会错误的认为自己是leader（论文中的`leasrOwner = true`），但是value中的内容却表明leader是节点5，而且节点5自身都不知道自己已经成为了leader。

所以在PaxosLease中需要避免出现acceptor已经接受过提案的情况发生，即acceptor需要清空自己之前accept过的提案值。（多久清空？？？或者说什么时间点清空？？）

不过清空自己accept过的提案值会不会与正常的选举之间冲突呢？？或者说正常的选举时accepted的提案值何时清空

> 源码中acceptor会清空上一次已过期的提案值，而针对上述例子，采取在发起2a时检测value中的id和本机是否一致

KeySpace PaxosLease源码分析：

PLeaseState.h

```
class PLeaseProposerState
{
public:
    bool      preparing;
    bool      proposing;
    uint64_t  proposalID;    //提案号，单调递增，各节点间互不相同
    uint64_t  higestReceivedProposalID;//曾接受过的最大的提案号，收到多个有value的accept回复后，选择此值最大的对应的value
    unsigned  leaseOwner;//leader id 即本机id
    unsigned  duration;//leader租期（以上两行共同构成value）
    uint64_t  expireTime;//过期时间点
    //相比于一般paxos，state中删去了accepted的最大的提案号，此值用来选择下一个提案号；似乎此值被设置为proposer的本地变量
};
```

```
class PLeaseAcceptorState
{
public:
    uint64_t promisedProposalID;//承诺过的最大的提案号
    bool     accepted;
    uint64_t acceptedProposalID;//accept过的提案号
    unsigned acceptedLeaseOwner;//accept过的leader
    unsigned acceptedDuration;//accept过的租期（以上两行共同构成value）
    uint64_t acceptedExpireTime;//accept过的过期时间点
};
```

```
class PLeaseLearnerState
{
  public:
    bool     learned;
    unsigned leaseOwner;//是否为leader
    uint64_t expireTime;//过期时间点
    uint64_t leaseEpoch;//租约任期
};
```

每个节点的租期是单独计算的，不是全局时间，各自从确认accept value那一刻开始计算过期时间

paxos流程：

1）相比于一般的paxos流程，acceptor阶段新增了状态的清空（在第一和第二阶段中均检测）：针对那些已经accept过value，并且租期已经过期的情况；也就是acceptor处于上一已过期租期，此时清空状态以防止影响下一次的选举

为防止上述例子中的情况产生，proposer会在进入第二阶段初检查，如果value中的leaderOwner不是本机（即不是此value的提出方），直接终止，但这样的话，第二阶段为什么需要再次检测

（能不能提前，直接在1b阶段检查终止掉）

update：第一阶段时的检测应该是为了以上理由，第二阶段的检测难道是为了应对原leader续租

2）acceptor在启动后会Sleep M秒，论文中解释为防止破坏租约不等式（任何时间点都不会有多于一个proposer持有租约）

计时器：

Proposer：

`CdownTimer acquireLeaseTimeout;`

获取租约可用的时间间隔；在StartPrepare时开始计时，进入learn时移除；如果此时间范围内没能获取到租约，则执行超时回调`onAcquireLeaseTimeout`重新开始Prepare

`Timer extendLeaseTimeout;`

扩展的租约时间间隔，时长为五分之一的租期`Now() + (state.expireTime - Now()) / 5`，在开始learn阶段开始计时，一旦proposer又进入prepare阶段此计时器马上移除，超出则执行回调`onExtendLeaseTimeout`重新开始Prepare，但需要确保此时proposer不是prepare和proposing阶段，即超时前的proposer已经明确进入了learn阶段

Acceptor：

`Timer leaseTimeout;`

因为PaxosLease没有全局的时间概念，各自维护自身的租期，所以此值为acceptor各自的租约计时器，在accept提案值时便确定下来，loop中开始计时，但如果状态清空时会从loop中移除此计时器；超时处理为`onLeaseTimeout`清空acceptor的状态

Learner：

`Timer leaseTimeout;`

~~在当前时间基础上加上本次租期在学习阶段的剩余时长~~（learner的租期也是单独计算）；超时处理为`OnLeaseTimeout`清空learner的状态

PaxosLease.h PaxosLease.cpp

待补充

> keyspace的回调形式很有特点，不是分为static的两段式回调，使用了模板。需要深入看下