Ceph Monitor

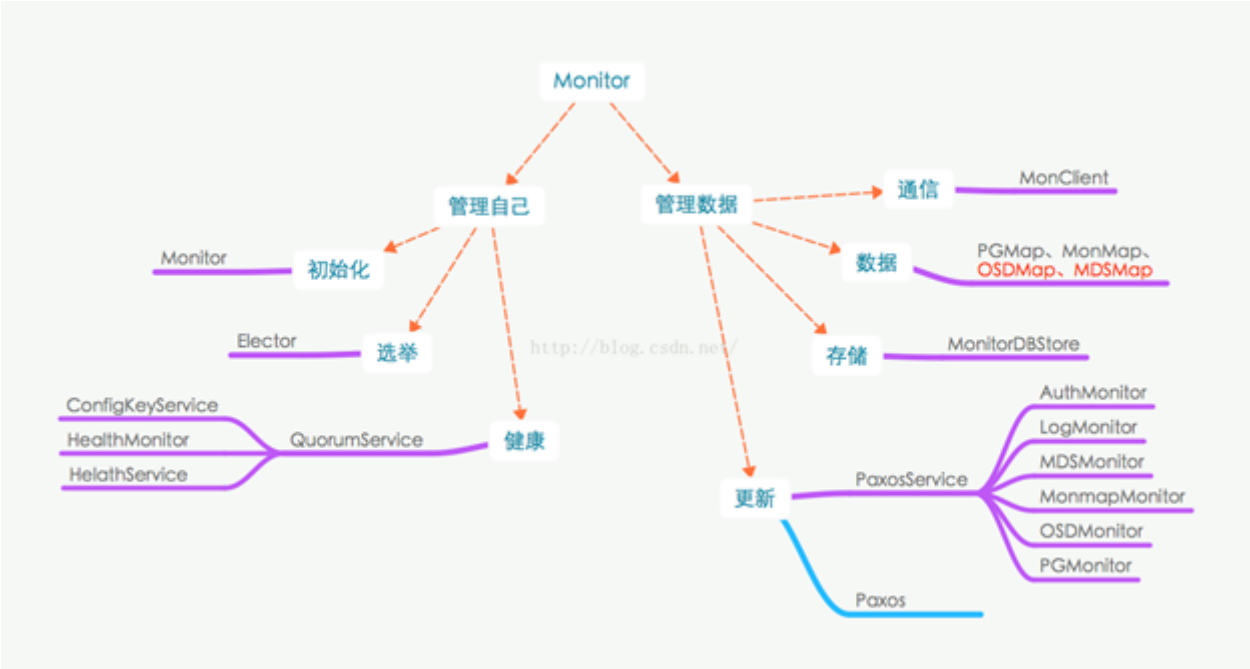
ceph版本：https://github.com/ceph/ceph/tree/v18.2.1

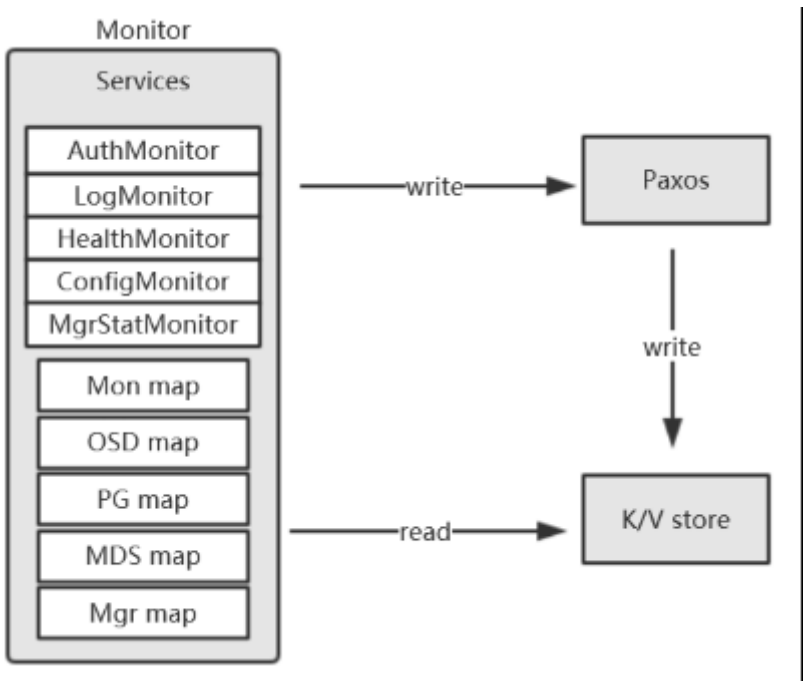
关于paxos的基本原理，本文不再介绍，网上有各路大神的分享

官方介绍

可靠分布式系统-paxos的直观解释

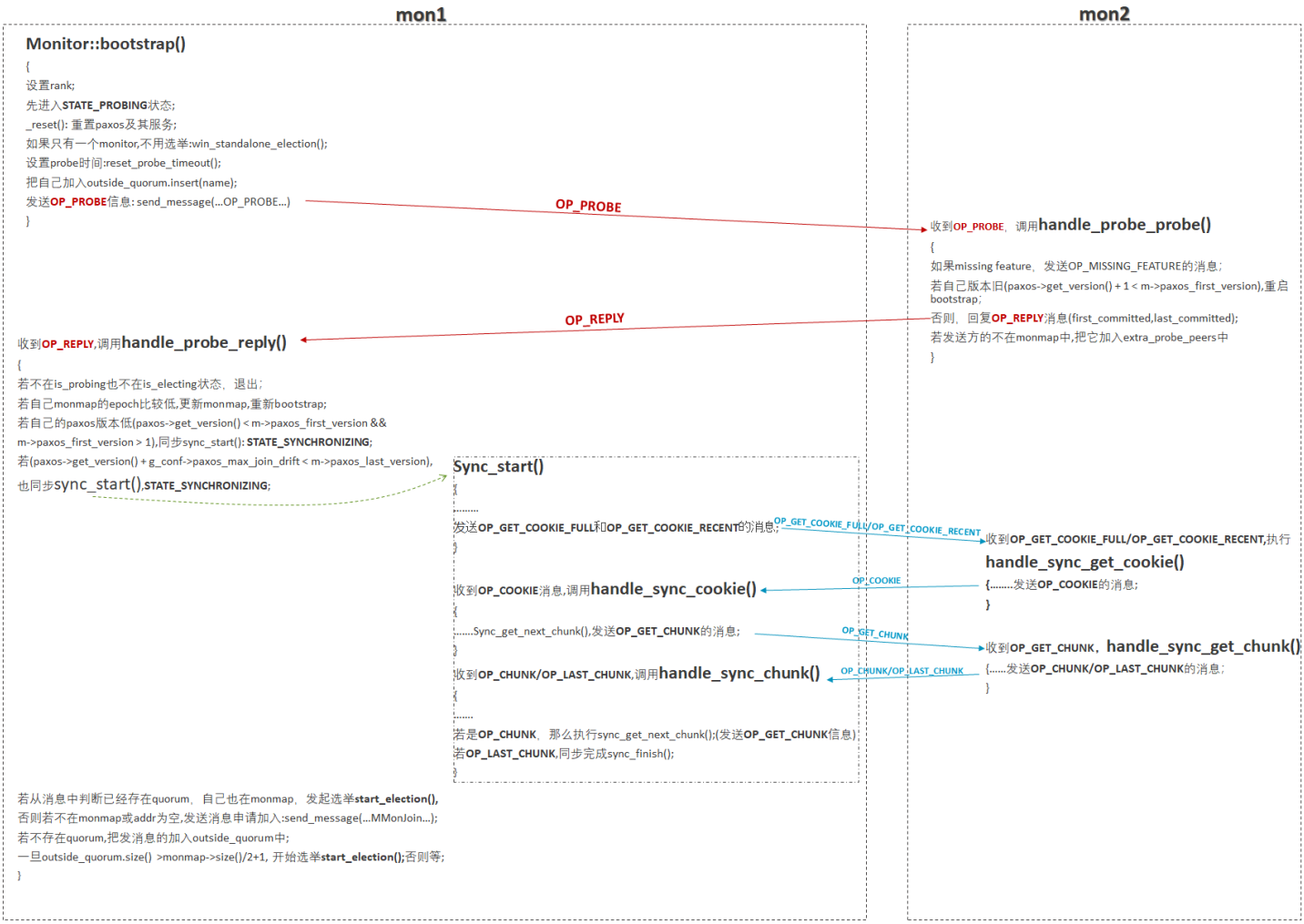
Mon





Mon一共三层，PaxosServices，Paxos，MonitorDBStore

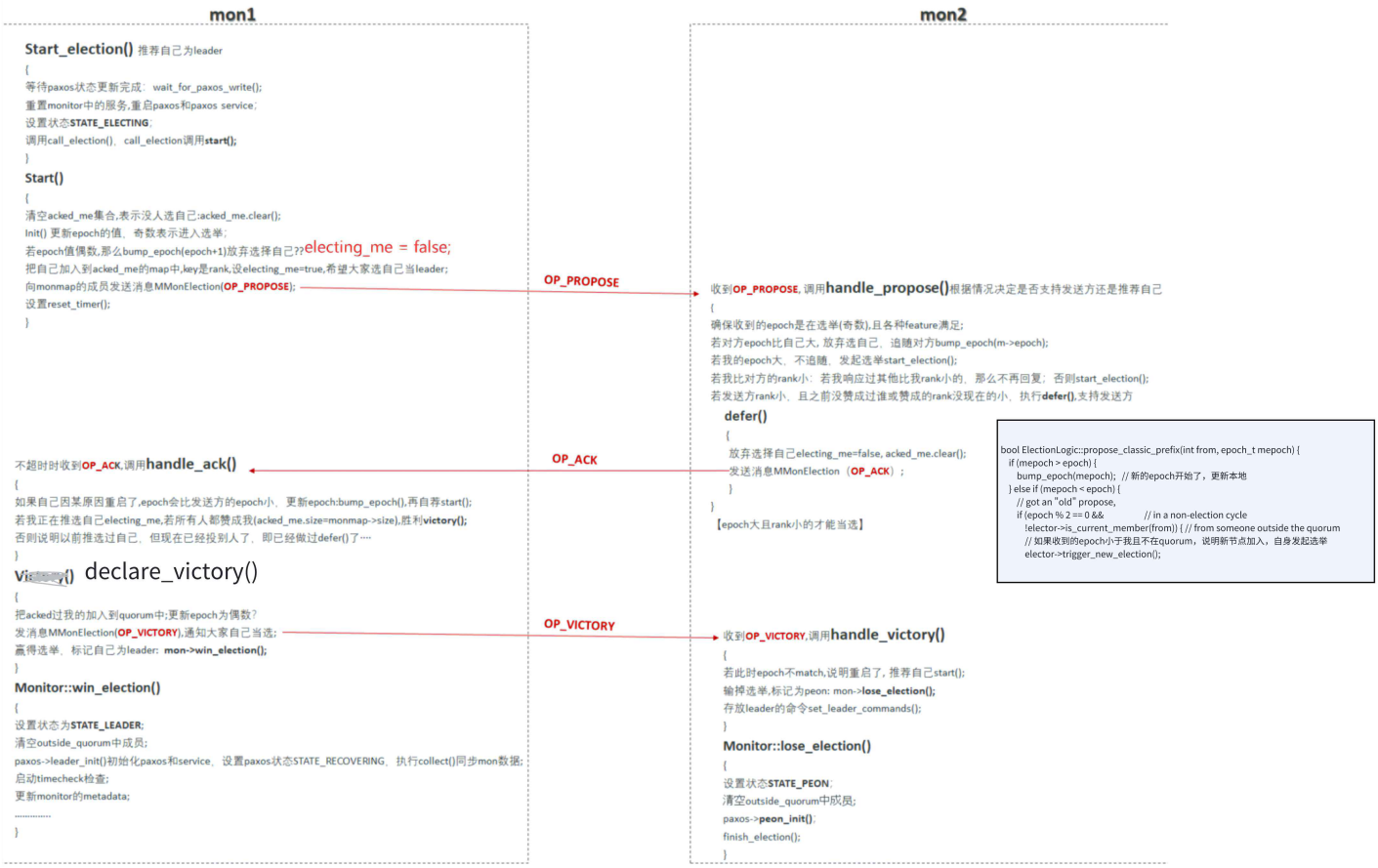
启动/同步： bootStrap



选举：election

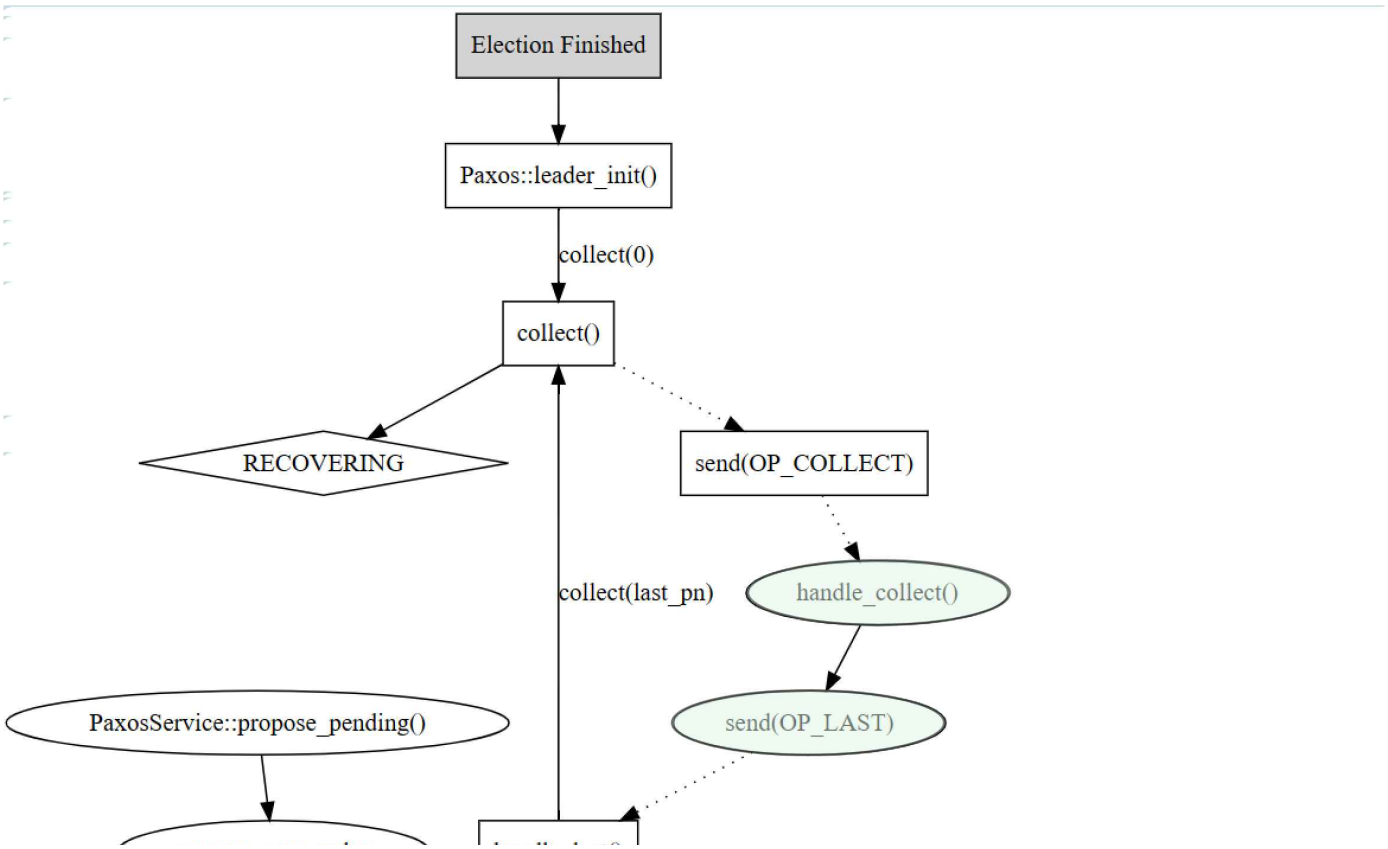
下图中的{若epoch是偶数，则...}，实际上在init中，epoch会设置为奇数，也就是说electing_me=false这段分支根本走不到(当时看的时候很矛盾，先设置为了false，后面又设置为true，最终发现根本没有false的前提)

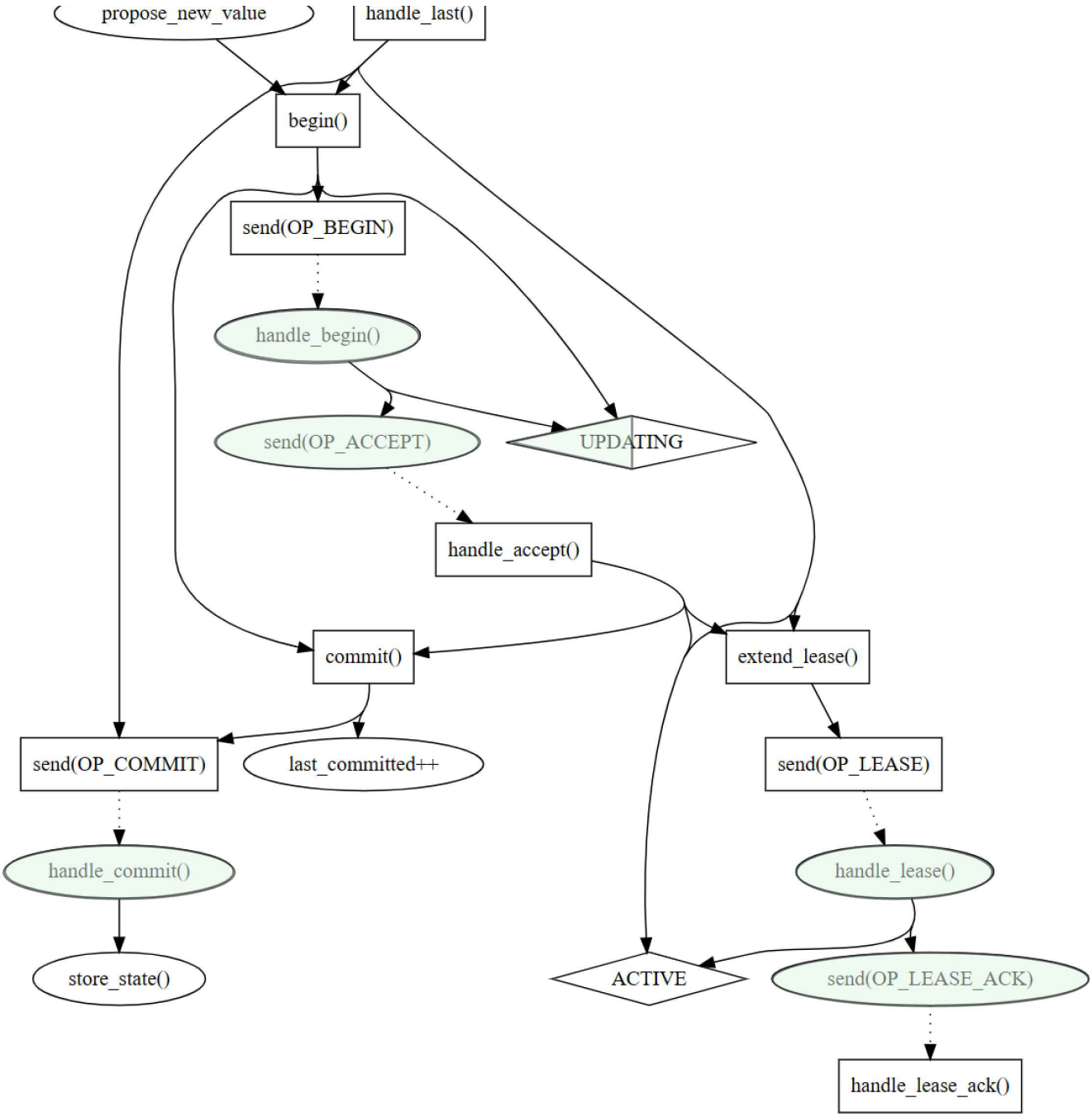
关于epoch的奇偶性对paxos的影响并不只是版本号，还表示一种状态，如果是奇数，表示当前正在进行选举；否则当前表示当前是除了选举之外的状态。而这也是针对ceph实现，更宽泛的说法，epoch为奇数表示当前正在商议提案，为偶数则表示已经商议完毕。对当前情况下的选举来说，就是指商议阶段。



一致性决策阶段

白色部分是leader的行为，绿色部分是peon的行为




propose_new_value
handle_last()
begin()
send(OP_BEGIN)
handle_begin()
send(OP_ACCEPT)
UPDATING
handle_accept()
commit()
extend_lease()
send(OP_COMMIT)
last_committed++
send(OP_LEASE)
handle_commit()
handle_lease()
store_state()
ACTIVE
send(OP_LEASE_ACK)
handle_lease_ack()

需要持久化的数据

名称	含义	其他
PN(Proposal Number)	提案编号，用于标识提案，维护提案顺序	Leader当选之后，会执行一 确定PN，在其为Leader的这 Phase 2共用一个PN
last_pn	上次当选leader后生成的PN	get_new_proposal_numb 选后，接着生成
accepted_pn	我接受过的PN，可能是别的leader提议的PN	peon根据这个值拒绝较小的
first_committed	本节点记录的第一个被commit的版本	更早的版本(日志)，本节点没
last_committed	本节点记录的最后一次被commit的版本	往后的版本，未被commit，
uncommitted_v	本节点记录的未commit的版本，如果有，只能等于last_commit+1	ceph只允许有一个未comm
uncommitted_pn	未commit的版本对应的PN	与uncommitted_v，uncor 一个事务中记录
uncommitted_value	未commit的版本的内容	与uncommitted_v，uncor 一个事务中记录

leader_init

```
1  void Paxos::leader_init()
2  {
3    cancel_events();
4  // 清除之前提出的提议值
5    new_value.clear();
6
7  // 重置正在等待决议的提议
8    pending_proposal.reset();
9  // 重置commit的回调函数
10   reset_pending_committing_finishers();
11
12 // 根据法定人数设置状态为ACTIVE或RECOVERING
13   if (mon.get_quorum().size() == 1) {
14     state = STATE_ACTIVE;
15     return;
16   }
17
18 // 进入recovery状态
19   state = STATE_RECOVERING;
20   lease_expire = {};
21 // 收集其它节点信息，进行提议值选择和一致性决议
```

```
  collect(0);
}
```

collect

```
void Paxos::collect(version_t oldpn) {

    state = STATE_RECOVERING;
    ceph_assert(mon.is_leader());

// 重置未提交提议的记录
    uncommitted_v = 0;
    uncommitted_pn = 0;
    uncommitted_value.clear();
    peer_first_committed.clear();
    peer_last_committed.clear();

// 检查本地存储,获取未提交的提议值
    if (get_store()->exists(get_name(), last_committed + 1)) {
        version_t v = get_store()->get(get_name(), "pending_v");
        version_t pn = get_store()->get(get_name(), "pending_pn");
        if (v && pn && v == last_committed + 1) {
            uncommitted_pn = pn;
        } else {
            uncommitted_pn = accepted_pn;
        }
        uncommitted_v = last_committed + 1;

        get_store()->get(get_name(), last_committed + 1, uncommitted_value);
        ceph_assert(uncommitted_value.length());
    }

// pick new pn
    accepted_pn = get_new_proposal_number(std::max(accepted_pn, oldpn));
    accepted_pn_from = last_committed;
    num_last = 1;

// send collect
    for (auto p = mon.get_quorum().begin();
         p != mon.get_quorum().end();
         ++p) {
        if (*p == mon.rank)
            continue;

```

```
40          MMonPaxos *collect = new MMonPaxos(mon.get_epoch(),
    MMonPaxos::OP_COLLECT,
41                                             ceph_clock_now());
42          collect->last_committed = last_committed;
43          collect->first_committed = first_committed;
44          collect->pn = accepted_pn;
45          mon.send_mon_message(collect, *p);
46      }
47
48  // set timeout event
49      collect_timeout_event = mon.timer.add_event_after(
50          g_conf()->mon_accept_timeout_factor *
51              g_conf()->mon_lease,
52          new C_MonContext{&mon, [this](int r) {
53                                  if (r == -ECANCELED)
54                                      return;
55                                  collect_timeout();
56                              }});
57  }
58
```

每次clloect，都会生成新的accept_pn，get_new_proposal_number:

```
1  version_t Paxos::get_new_proposal_number(version_t gt) {
2      if (last_pn < gt)
3          last_pn = gt;
4
5      // update. make it unique among all monitors.
6      last_pn /= 100; // 将最后两位数字清零
7      last_pn++;      // 让前几位数字增1
8      last_pn *= 100; // 重新恢复最后两位为00
9      last_pn += (version_t)mon.rank; // 将rank作为最后两位
10     // 利用低两位来区分不同的mon，这样可以保证mon生成的pn唯一
11 ...
12     return last_pn;
13 }
```

Peon: hanle_collect

```
1  主要功能包括:
2  设置状态为RECOVERING表示正在进行恢复
3  更新副本节点lease过期时间
4  如果Leader提交的版本号太高则需要进行引导恢复
```

回复Leader本地最后提交和首提交信息
判断是否可以接受Leader的提议号,如果可以更新提议号状态
将本节点已提交但Leader未知的信息同步给Leader
如果本地有已提交但未确认的值,也发送给Leader
这段代码结合Paxos算法流程很清晰地实现了副本节点收到Leader COLLECT请求后的处理逻辑:
检查自己是否需要进行引导恢复
回传自己的状态信息
判断是否需要采纳Leader新的提议号
与Leader同步状态,将自己更高版本的信息传递给Leader

```
void Paxos::handle_collect(MonOpRequestRef op) {
    op->mark_paxos_event("handle_collect");
    auto collect = op->get_req<MMonPaxos>();
    ceph_assert(mon.is_peon());
    state = STATE_RECOVERING;

// 更新副本节点lease过期时间
    reset_lease_timeout();
// 版本过于落后，需要引导修复
    if (collect->first_committed > last_committed + 1) {
        op->mark_paxos_event("need to bootstrap");
        mon.bootstrap();
        return;
    }

    // reply
    MMonPaxos *last = new MMonPaxos(mon.get_epoch(),
             MMonPaxos::OP_LAST, ceph_clock_now());
    last->last_committed = last_committed;
    last->first_committed = first_committed;

    version_t previous_pn = accepted_pn;

    // can we accept this pn?
    if (collect->pn > accepted_pn) {
        // ok, accept it
        accepted_pn = collect->pn;
        accepted_pn_from = collect->pn_from;
        auto t(std::make_shared<MonitorDBStore::Transaction>());
        t->put(get_name(), "accepted_pn", accepted_pn);

        JSONFormatter f(true);
        t->dump(&f);
        f.flush(*_dout);
        *_dout << dendl;
```

```
// 将接受的提案值通过事务写到存储
        auto start = ceph::coarse_mono_clock::now();
        get_store()->apply_transaction(t);
        auto end = ceph::coarse_mono_clock::now();

    } else {
    }
    last->pn = accepted_pn;
    last->pn_from = accepted_pn_from;

// 本地已提交，但leader未知的信息，将leader缺少的记录给添加到last中
    if (collect->last_committed < last_committed)
        share_state(last, collect->first_committed, collect->last_committed);

    bufferlist bl;
    if (collect->last_committed <= last_committed &&
        get_store()->exists(get_name(), last_committed + 1)) {
        get_store()->get(get_name(), last_committed + 1, bl);
        ceph_assert(bl.length() > 0);

        last->values[last_committed + 1] = bl;

        version_t v = get_store()->get(get_name(), "pending_v");
        version_t pn = get_store()->get(get_name(), "pending_pn");
        if (v && pn && v == last_committed + 1) {
            last->uncommitted_pn = pn;
        } else {
            last->uncommitted_pn = previous_pn;
        }

        logger->inc(l_paxos_collect_uncommitted);
    }

// send reply
    collect->get_connection()->send_message(last);
}

```

handle_last

处理逻辑

```
更新收到回复节点的first_committed和last_committed状态
```

```
判断状态是否一致,不一致触发引导恢复
保存LAST消息中的已确认值
判断其他副本是否都接收当前提议号,是则进入ACTIVE状态
处理可能学习到的未确认值
如果还没有全体副本响应,继续等待
否则完成本轮提议,切换状态
```

```
void Paxos::handle_last(MonOpRequestRef op) {
    op->mark_paxos_event("handle_last");
    auto last = op->get_req<MMonPaxos>();
    bool need_refresh = false;
    int from = last->get_source().num();

    if (!mon.is_leader()) {
        return;
    }


    peer_first_committed[from] = last->first_committed;
    peer_last_committed[from] = last->last_committed;

// 引导修复
    if (last->first_committed > last_committed + 1) {
        op->mark_paxos_event("need to bootstrap");
        mon.bootstrap();
        return;
    }

    ceph_assert(g_conf()->paxos_kill_at != 1);

    // store any committed values if any are specified in the message
    need_refresh = store_state(last);

    ceph_assert(g_conf()->paxos_kill_at != 2);

// 更新每一个peer的信息
    // is everyone contiguous and up to date?
    for (auto p = peer_last_committed.begin();
         p != peer_last_committed.end();
         ++p) {
        if (p->second + 1 < first_committed && first_committed > 1) {
            op->mark_paxos_event("need to bootstrap");
            mon.bootstrap();
            return;
        }
```

```
39          if (p->second < last_committed) {
40              // share committed values
41              MMonPaxos *commit = new MMonPaxos(mon.get_epoch(),
42                                                MMonPaxos::OP_COMMIT,
43                                                ceph_clock_now());
44              share_state(commit, peer_first_committed[p->first], p->second);
45              mon.send_mon_message(commit, p->first);
46          }
47      }
48
49  // 说明leader的pn落后，开始新的提案
50      if (last->pn > accepted_pn) {
51          // cancel timeout event
52          mon.timer.cancel_event(collect_timeout_event);
53          collect_timeout_event = 0;
54
55          collect(last->pn);
56      } else if (last->pn == accepted_pn) {
57  // 增加已接受提案的计数
58          num_last++;
59          // did this person send back an accepted but uncommitted value?
60          if (last->uncommitted_pn) {
61              if (last->uncommitted_pn >= uncommitted_pn &&
62                  last->last_committed >= last_committed &&
63                  last->last_committed + 1 >= uncommitted_v) {
64                  uncommitted_v = last->last_committed + 1;
65                  uncommitted_pn = last->uncommitted_pn;
66                  uncommitted_value = last->values[uncommitted_v];
67              }
68          }
69
70  // 接收提案的节点达到了指定人数
71          if (num_last == mon.get_quorum().size()) {
72  // 取消超时事件，并清理peer的状态信息
73              // cancel timeout event
74              mon.timer.cancel_event(collect_timeout_event);
75              collect_timeout_event = 0;
76              peer_first_committed.clear();
77              peer_last_committed.clear();
78
79              // almost...
80
81  // 如果当前节点收到了一个有效的未提交值，设置状态为UPDATING_PREVIOUS
82              if (uncommitted_v == last_committed + 1 &&
83                  uncommitted_value.length()) {
84                  state = STATE_UPDATING_PREVIOUS;
85                  begin(uncommitted_value);
```

```
            } else {
// 否则如果所有的节点都接受了提案编号且没有新的未提交值，延长租约
                // active!
                extend_lease();

                need_refresh = false;
                if (do_refresh()) {
                    finish_round();
                }
            }
        }
    } else {
        // no, this is an old message, discard
    }

    if (need_refresh)
        (void)do_refresh();
}

```

propose_pending

```
void Paxos::propose_pending() {
    ceph_assert(is_active());
    ceph_assert(pending_proposal);

    cancel_events();

    bufferlist bl;
    pending_proposal->encode(bl);

    pending_proposal.reset();

    committing_finishers.swap(pending_finishers);
    state = STATE_UPDATING;
    begin(bl);
}
```

begin

```
void Paxos::begin(bufferlist &v) {
    ceph_assert(mon.is_leader());
```

```
3     ceph_assert(is_updating() || is_updating_previous());
4
5     // we must already have a majority for this to work.
6     ceph_assert(mon.get_quorum().size() == 1 ||
7                 num_last > (unsigned)mon.monmap->size() / 2);
8
9     // and no value, yet.
10    ceph_assert(new_value.length() == 0);
11
12 // accept it ourselves，并设置新的提案
13    accepted.clear();
14    accepted.insert(mon.rank);
15    new_value = v;
16
17 // 第一个commit，只有第一次提出提案的时候才会遇到
18    if (last_committed == 0) {
19        auto t(std::make_shared<MonitorDBStore::Transaction>());
20        // initial base case; set first_committed too
21        t->put(get_name(), "first_committed", 1);
22        decode_append_transaction(t, new_value);
23
24        bufferlist tx_bl;
25        t->encode(tx_bl);
26
27        new_value = tx_bl;
28    }
29
30 // 将编码后的值添加到存储事务中
31    auto t(std::make_shared<MonitorDBStore::Transaction>());
32    t->put(get_name(), last_committed + 1, new_value);
33
34    // note which pn this pending value is for.
35    t->put(get_name(), "pending_v", last_committed + 1);
36    t->put(get_name(), "pending_pn", accepted_pn);
37
38 // 将要发起的提案通过事务保存在本地
39    auto start = ceph::coarse_mono_clock::now();
40    get_store()->apply_transaction(t);
41    auto end = ceph::coarse_mono_clock::now();
42
43    logger->tinc(l_paxos_begin_latency, to_timespan(end - start));
44
45    ceph_assert(g_conf()->paxos_kill_at != 3);
46 // 集群只有一个mon，就直接commit
47    if (mon.get_quorum().size() == 1) {
48        // we''re alone, take it easy
49        commit_start();
```

```
50          return;
51      }
52
53  // 通知其它节点，处理这个提案
54      // ask others to accept it too!
55      for (auto p = mon.get_quorum().begin();
56           p != mon.get_quorum().end();
57           ++p) {
58          if (*p == mon.rank)
59              continue;
60
61          MMonPaxos *begin = new MMonPaxos(mon.get_epoch(), MMonPaxos::OP_BEGIN,
62                                           ceph_clock_now());
63          begin->values[last_committed + 1] = new_value;
64          begin->last_committed = last_committed;
65          begin->pn = accepted_pn;
66
67          mon.send_mon_message(begin, *p);
68      }
69
70  // set timeout event
71      accept_timeout_event = mon.timer.add_event_after(
72          g_conf()->mon_accept_timeout_factor * g_conf()->mon_lease,
73          new C_MonContext{&mon, [this](int r) {
74                                  if (r == -ECANCELED)
75                                      return;
76                                  accept_timeout();
77                              }});
78  }
79
```

Peon: handle_begin

```
1  void Paxos::handle_begin(MonOpRequestRef op) {
2      op->mark_paxos_event("handle_begin");
3      auto begin = op->get_req<MMonPaxos>();
4
5  // can we accept this?
6      if (begin->pn < accepted_pn) {
7          op->mark_paxos_event("have higher pn, ignore");
8          return;
9      }
10     ceph_assert(begin->pn == accepted_pn);
11     ceph_assert(begin->last_committed == last_committed);
12
```

```
    ceph_assert(g_conf()->paxos_kill_at != 4);

    logger->inc(l_paxos_begin);

// set state.
    state = STATE_UPDATING;
    lease_expire = {}; // cancel lease

// yes.
    version_t v = last_committed + 1;

// 这里和leader一样，先将数据放在事务中，然后提交事务
    auto t(std::make_shared<MonitorDBStore::Transaction>());
    t->put(get_name(), v, begin->values[v]);

    // note which pn this pending value is for.
    t->put(get_name(), "pending_v", v);
    t->put(get_name(), "pending_pn", accepted_pn);

    auto start = ceph::coarse_mono_clock::now();
    get_store()->apply_transaction(t);
    auto end = ceph::coarse_mono_clock::now();

    logger->tinc(l_paxos_begin_latency, to_timespan(end - start));

    ceph_assert(g_conf()->paxos_kill_at != 5);

// 回复接收提案的消息
    MMonPaxos *accept = new MMonPaxos(mon.get_epoch(), MMonPaxos::OP_ACCEPT,
                                      ceph_clock_now());
    accept->pn = accepted_pn;
    accept->last_committed = last_committed;
    begin->get_connection()->send_message(accept);
}
```

hanle_accept

```
void Paxos::handle_accept(MonOpRequestRef op) {
    op->mark_paxos_event("handle_accept");
    auto accept = op->get_req<MMonPaxos>();
    int from = accept->get_source().num();

    if (accept->pn != accepted_pn) {
```

```
        op->mark_paxos_event("have higher pn, ignore");
        return;
    }
    if (last_committed > 0 &&
        accept->last_committed < last_committed - 1) {
        op->mark_paxos_event("old round, ignore");
        return;
    }
    ceph_assert(accept->last_committed == last_committed ||    // not committed
                accept->last_committed == last_committed - 1); // committed

    ceph_assert(is_updating() || is_updating_previous());
    ceph_assert(accepted.count(from) == 0);
    accepted.insert(from);

    ceph_assert(g_conf()->paxos_kill_at != 6);

// 只有通过数达到了quorum，才能提交
    if (accepted == mon.get_quorum()) {
        op->mark_paxos_event("commit_start");
        commit_start();
    }
}
```

第二阶段：commit_start

```
void Paxos::commit_start() {

    ceph_assert(g_conf()->paxos_kill_at != 7);

    auto t(std::make_shared<MonitorDBStore::Transaction>());

// commit locally
    t->put(get_name(), "last_committed", last_committed + 1);

    // decode the value and apply its transaction to the store.
    // this value can now be read from last_committed.
    decode_append_transaction(t, new_value);

    commit_start_stamp = ceph_clock_now();

// 提交事务，与之前不同的是，这里是异步调用
// 设置的回调函数是commit_finish
    get_store()->queue_transaction(t, new C_Committed(this));

```

```
    if (is_updating_previous())
        state = STATE_WRITING_PREVIOUS;
    else if (is_updating())
        state = STATE_WRITING;
    else
        ceph_abort();
    ++commits_started;

    if (mon.get_quorum().size() > 1) {
        // cancel timeout event
        mon.timer.cancel_event(accept_timeout_event);
        accept_timeout_event = 0;
    }
}
```

commit_finish

```
void Paxos::commit_finish() {
    utime_t end = ceph_clock_now();
    logger->tinc(l_paxos_commit_latency, end - commit_start_stamp);

    ceph_assert(g_conf()->paxos_kill_at != 8);

// 取消当前租约
    lease_expire = {}; // cancel lease

    last_committed++;
    last_commit_time = ceph_clock_now();

// refresh first_committed; this txn may have trimmed.
    first_committed = get_store()->get(get_name(), "first_committed");

    _sanity_check_store();

// 通知所有人commit
    for (auto p = mon.get_quorum().begin();
         p != mon.get_quorum().end();
         ++p) {
        if (*p == mon.rank)
            continue;


        MMonPaxos *commit = new MMonPaxos(mon.get_epoch(),
MMonPaxos::OP_COMMIT,
                                          ceph_clock_now());
```

```
28          commit->values[last_committed] = new_value;
29          commit->pn = accepted_pn;
30          commit->last_committed = last_committed;
31
32          mon.send_mon_message(commit, *p);
33      }
34
35      ceph_assert(g_conf()->paxos_kill_at != 9);
36
37  // 清空value，准备下一次循环
38      // get ready for a new round.
39      new_value.clear();
40
41      // WRITING -> REFRESH
42      // among other things, this lets do_refresh() -> mon.bootstrap() ->
43      // wait_for_paxos_write() know that it doesn''t need to flush the store
44      // queue. and it should not, as we are in the async completion thread now!
45      ceph_assert(is_writing() || is_writing_previous());
46      state = STATE_REFRESH;
47      ceph_assert(commits_started > 0);
48      --commits_started;
49
50  // 如果有新一轮的提案，则延长租约
51  // 完成当前循环
52      if (do_refresh()) {
53          commit_proposal();
54          if (mon.get_quorum().size() > 1) {
55              extend_lease();
56          }
57
58          ceph_assert(g_conf()->paxos_kill_at != 10);
59
60          finish_round();
61      }
62  }
```

Peon: hanedle_commit

```
1  void Paxos::handle_commit(MonOpRequestRef op) {
2      op->mark_paxos_event("handle_commit");
3      auto commit = op->get_req<MMonPaxos>();
4
5      logger->inc(l_paxos_commit);
6
7      if (!mon.is_peon()) {
```

```
        ceph_abort();
        return;
    }

    op->mark_paxos_event("store_state");
    store_state(commit);

    (void)do_refresh();
}

```

参考

http://bean-li.github.io/ceph-paxos/

http://bean-li.github.io/ceph-paxos-2/

ceph monitor----初始化和选举 - yimuxi - 博客园

https://www.cnblogs.com/yi-mu-xi/p/10364797.html