2014/12/23重新按照边看别完整写理解的方式，有了不少新的理解。

**协议介绍部分：**

      1. 异步模型:  1) 各个进程的初始值可能不同； 2) 消息可能丢失，改变顺序、重复；3) 只需要经历三次消息传输导致的延迟，就可以让所有的参与者知道最终结果；TODO: 是指learning阶段？还是整个阶段？还是在正常情况下？

                          4) Crash的进程可以被替代，不影响运行，除非是大量进程都crash了。

      2. Atomic Broadcast: 保证所有参与者按照相同顺序收到消息。注: 应该是收到结果的顺序，而不是各个消息的顺序，因为消息本身就可能被reorder。

                     通过执行多个consensus instances提高性能(我理解就是多实例)。

                     注意： Abroadcast是指client发起的操作。

      3. 关于几个特性的说明中，Validity一项，这个A diliver是一定发生的吗？ 后面的uniform Integrity，说了at most one，那就是可能压根没有deliver，在故障的情况下。

      2.2.1  各个learner必须按照iid依次做deliver，比如iid 1-10已经被改learner deliver，这时发现iid 13已经达成一致，11-12还没有达到一致(消息可能没按照顺序到达或者其他原因)，那么必须等待11, 12，或者主动询问，然后按照顺序去deliver。

               一旦一个value(对某个iid)被选择，那么就认为被永久接受和持久化了，不会丢失。在Crash-stop模型中，即使某个机器crash，也认为其起来后能够找到以前的持久化信息。如果找不到持久化信息，则该节点不应该继续工作。

      2.2.2 注意这些内容都是针对某个iid的。Acceptor保存了一个四元组信息(iid, B, V, VB)组合：

               B: 被accepted或promised的最高的ballot number；  V: 已经被accept的value;   VB: 已经被accept，对应V的ballot number。TODO: 这个是不是应该也可以是被promised的，而不一定是一件被Accept的？

**Proposer                                                            Acceptor**

                                                                          <i, Nil, Nil, Nil>

                         ==Prepare, <i,b>==>

                                   <==Promise, <i, b, Nil, Nil>==         <i, b, Nil, Nil>

                                   ==><i, b, v>, Accept==>

                                                                                         <i, b, v, b>

                                   <==<i, b, v>  Learn==

             注意，一个非常重要的点，在prepare阶段，返回的v和b，如果非空，v就一定需要Phase2使用(If instead some value was found in the promises, the instance is reserved.

The coordinator is forced to select the value V with the highest associated ballot VB and execute phase 2 with it. 即从promise中，选择最大的VB对应的V，但是b应该还是phase1的b，而不是别的)

             另外，在answer前，acceptor必须将内容同步写盘，这样在crash后，能找到自己的状态。

             Acceptor的个数是预定义和固定的，只要大多数活着，就能继续运转。

             TODO: Acceptor如何做reject?  估计需要看代码才能知道。

        2.2.4 Leader根据client提交的value，选择一个未用的iid，通过broadcast发送出去，分为两个阶段：

            Phase 1, Prepare    Proposer等待直到majority 个收到promise或者timeout。如果acceptor认为不能接受，为什么不直接reject? 或者有没有直接reject? 还是说promise中返回的v, b如果非空，然后Proposer在Phase 2使用收到acceptor发送的v，就是一种reject?

                         TODO: 如果收到不同的acceptor发送的v都不同，那么怎么处理？选择vb最大的？

                         如果timeout，则增加ballot number，然后重试。

           Phase 2,

                        Reserved是啥含义？ 用返回的更高ballot number对应的v替代？

                        比如，promose返回的是<i, b, v1, b1>，那么应该是b1>b，所以accept请求应为<i, b, v1>，注意我理解**是b，而不是b1**

                        如果发生超时，则增加ballot number，**从phase1**重试。

       2.4  倒数第二段，P2 is forced to execute phase 2 using vi rather than any other value。

                   这个对保证一致性应该很重要，是个重要原则。TODO: 但是如果不这么干，会出现什么问题？估计还要从其他地方找答案。

TODO:Leader的election，是个什么过程？跟实例执行过程中超时等机制什么关系？

==>得到的一些信息：

       1) 考虑多个server需要按照同一顺序进行更新的情况，会有多个Proposer，但是实际上任意一个时刻，还是有一个Leader(coordinator)，负责接收所有client的请求，并发起propose，其他proposer一般处于idle状态。

                 问题： Leader怎么选举的？跟instance什么关系？ Proposers relies on an external leader election service, which should nominate a coordinator (or leader) among them

       2) 既然有Leader，为什么还需要Paxos? 因为Leader也可能死掉，而且重新选举不是瞬间的，这个过程需要Paxos处理。

       3) Multi-Instance也是在Leader上发起的，这个是为了提高多个Instance执行的并行性。Multi-Instance并不是说让多个Proposer都一起去发起Proposal。只有认为Leader故障时，其他Proposer才发起proposal。

       4) 下面这段没怎么看懂： In case the application using PAXOS has strict ordering guarantees, e.g., it requires FIFO order with respect to each proposer, it is quite easy to enforce

            those requirements on top of the the Atomic Broadcast layer, for example by embedding a vector clock[Lam78] or a hclient ID,sequence numberi pair directly into the value.