

Cosmos-Tendermint

2022年3月30日 星期三

下午2:39

1.Cosmos

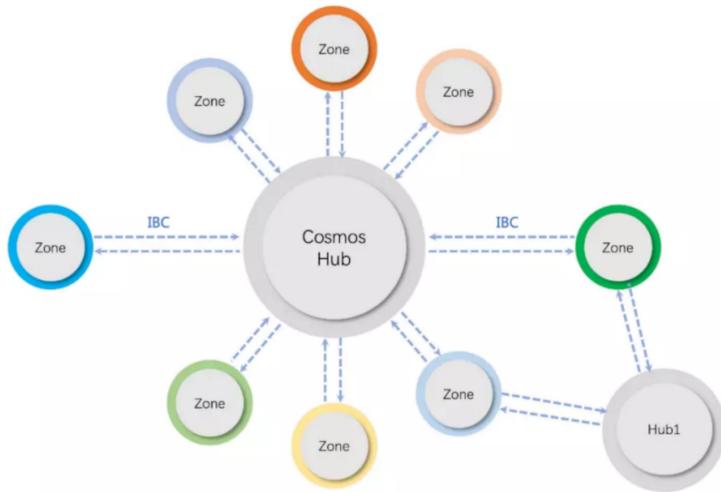
Cosmos并不是一条独立的区块链，而是一个可扩展的模块化区块链网络。

Cosmos的目标是建立一个可互通的网络生态系统。

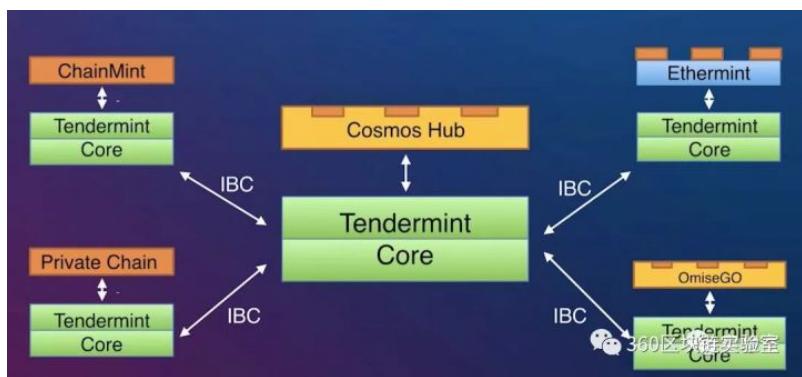
为此将区块链的共识层和应用层分离，实现如下核心模块

- 。Tendermint共识引擎，主要实现区块链的底层基础模块，共识，P2P，mempool，RPC，Store等模块
- 。Cosmos SDK 的模块化开发框架，主要实现各种dapp应用，比如Account，Bank，Staking，Slashing和Gov等
- 。ABCI 适应大多数编程语言的接口，主要用于在Tendermint和应用层通信
- 。IBC通信协议，实现区块链之间的信息和资产转移，打通不同区块链的孤岛，形成互联网。

1.1 Cosmos架构



1.2 Tendermint在Cosmos生态中的位置



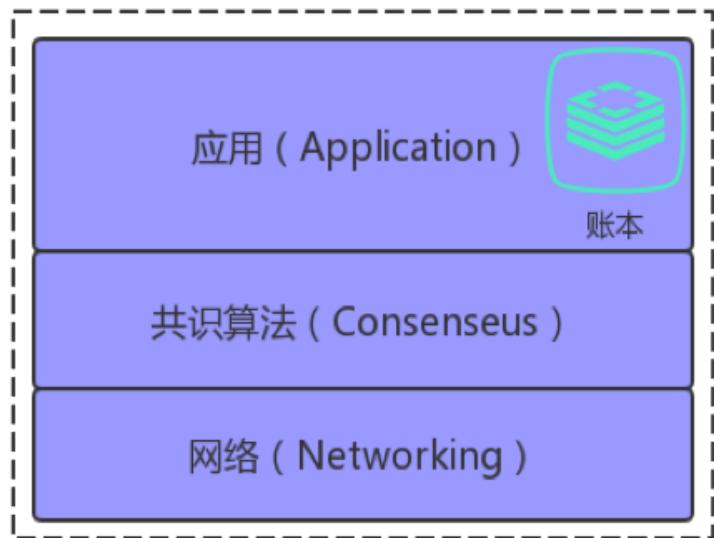
2.Tendermint

2.1 区块链架构

从架构的层面上，区块链可以简单分为三个概念层：

- 网络层（Networking）：负责交易和数据传输和同步。
- 共识算法（Consensus）：负责不同的验证节点处理完交易后，保证状态的一致，也就是将交易打包到区块中。
- 应用程序（Application）：交易的真正执行者。

大致框架如下：



早期大部分的区块链实现都是采用上面的框架，实现成单一的程序，有如下不足：

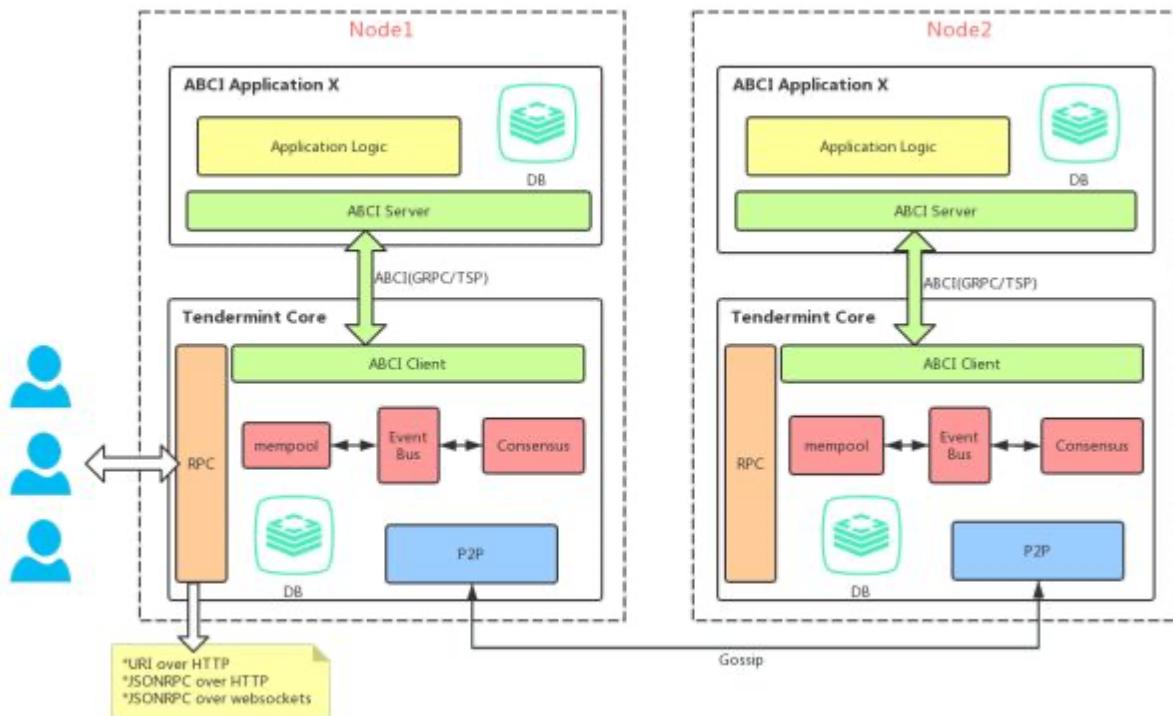
1. 代码复用困难，代码库的分支管理变得复杂。
2. 限制了应用开发的语言。

2.2 Tendermint 架构

Tendermint设计了自己的一套框架，其设计原则是易使用，易理解，高性能，适用于各种分布式应用。

它的创新之处在于，将区块链应用（状态）与底层共识进行了分离，将共识引擎和P2P网络层封装组成Tendermint Core。同时提供ABCI接口与应用层进行交互，应用逻辑可以用任何语言编写，应用做的事情实际上就是状态机控制。

基于这种架构，应用的开发者可以方便地实现自己的区块链。



2.3 共识模块

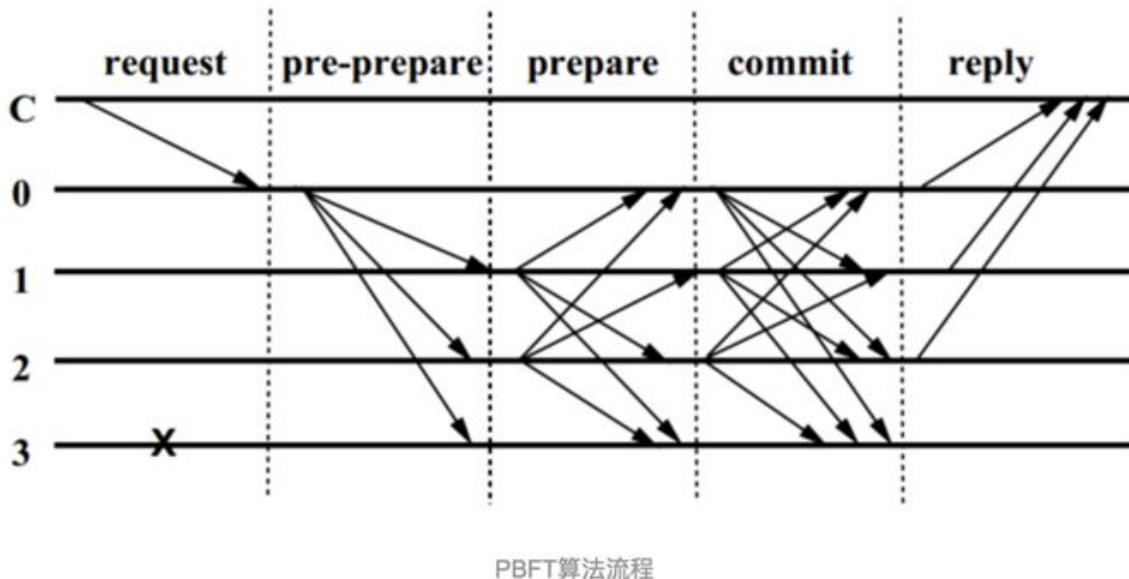
2.3.1 PBFT介绍

算法的核心三个阶段分别是 **pre-prepare** 阶段（预准备阶段），**prepare** 阶段（准备阶段），**commit** 阶段（提交阶段）。

- 。图中的C代表客户端
- 。0, 1, 2, 3 代表节点的编号，
- 。打叉的3代表可能是故障节点或者是问题节点，这里表现的行为就是对其它节点的请求无响应。

。0 是主节点。

下面介绍 pbft 算法的核心三阶段两次投票流程，如下图所示：



2.3.1.1 ViewChange (视图更改) 事件

当主节点挂了（超时无响应）或者从节点集体认为主节点是问题节点时，就会触发 ViewChange 事件， ViewChange 完成后，视图编号将会加 1。

下图展示 ViewChange 的三个阶段流程：

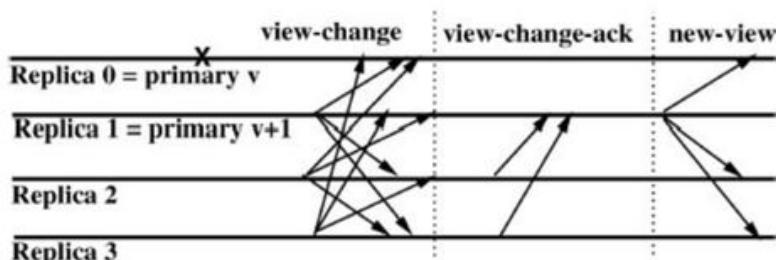


Fig. 2. View-change protocol: the primary for view v (replica 0) fails causing a view change to view $v + 1$.

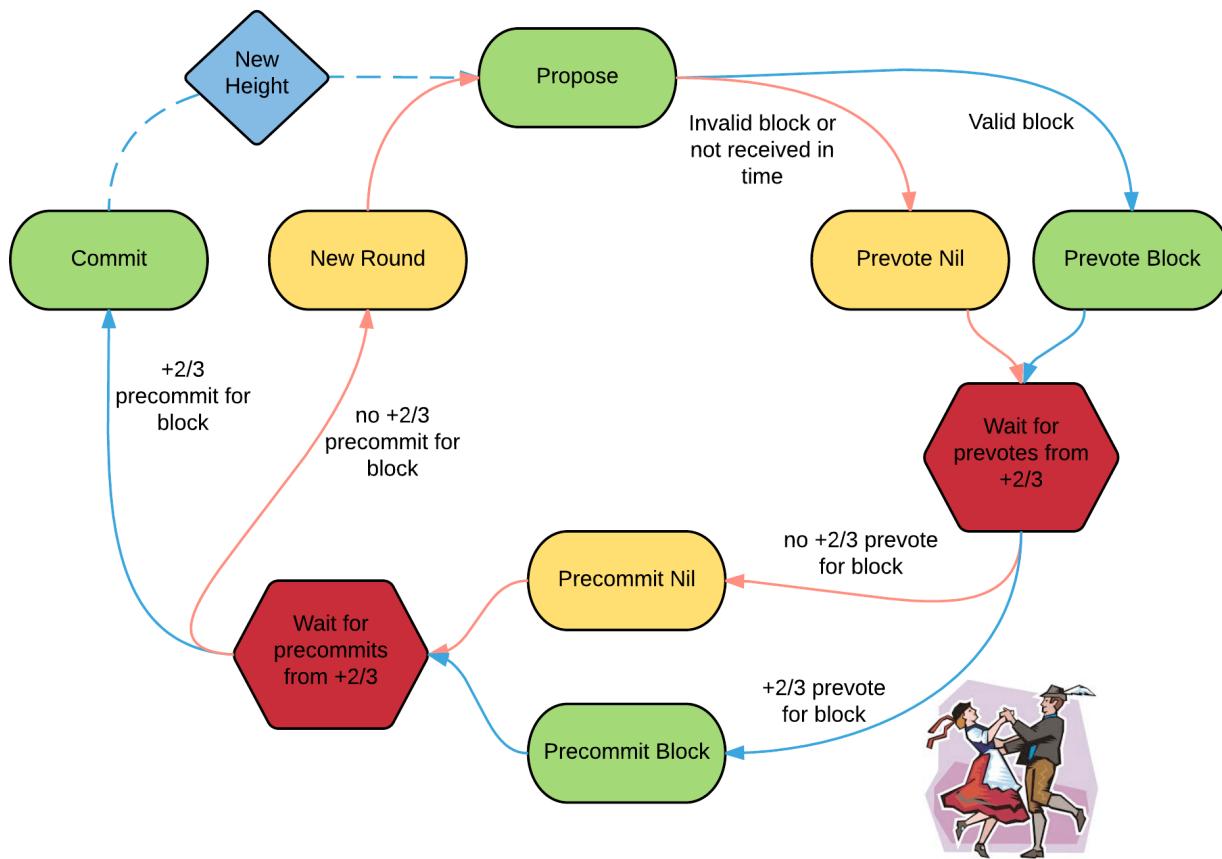
如图所示， viewchange 会有三个阶段，分别是 view-change ， view-change-ack 和 new-view 阶段。

。从节点认为主节点有问题时，会向其它节点发送 view-change 消息，当前存活的节点编号最小的节点将成为新的主节点。

。当新的主节点收到 $2f$ 个其它节点的 view-change 消息，则证明有足够多人的节点认为主节点有问题，于是就会向其它节点广播 New-view 消息。

2.3.2 Tendermint 共识

Tendermint 是一种易于理解的、异步的 BFT 共识协议。该协议遵循如下所示的简单状态机：



由于Tendermint BFT没有PBFT的那种view-change机制，转而是以提交空块复用主流程实现。

2.3.2.1 Tendermint BFT锁定机制：

锁定机制其实是Tendermint提出的一种容错的机制，用来确保没有两个验证器在同一高度提交不同的块。

由于Tendermint BFT没有PBFT的那种view-change机制，转而是以提交空块复用主流程实现，从而使得空块的提交对流程产生影响，考虑到如下情况：

1. 在propose阶段，proposer节点广播出了新块blockX
2. A的超时时间内没有收到这个新块，向外广播pre-vote nil，B,C,D都收到了，向外广播pre-vote投给blockX
3. 现在四个节点进入了pre-commit阶段，A处于红色内圈，B,C,D处于蓝色外圈
4. 假设A由于自身网络不好，又没有在规定时间内收到超过2/3个对blockX的投票，于是只能发出pre-commit nil投票消息投给空块
5. D收到了B和C的pre-vote消息，加上自己的，就超过了2/3了，于是D在本机区块链里commit了blockX
6. B和C网络出现问题，收不到D在pre-commit消息，这时B和C只能看到2票投给了blockX，一票投给了空块，全部不足2/3，于是B和C都只能commit空块，高度不变，进入R+1轮，
A也只能看到2票投给了blockX，一票投给了空块，也只能commit空块，高度不变，进入R+1轮
7. 在R+1轮，由于新换了一个proposer，提议了新的区块blockY，A,B,C三个可能会达成共识，提交blockY，于是在同样的高度，就有blockX和blockY两个块，产生了分叉。

Tendermint加上了锁的机制，具体就是，在第7步，即使proposer出了新块blockY，A,B,C只能被锁定在第6步他们的pre-commit块上，即A在第6步投给了空块，那么在第R+1轮，只能继续投给空块，B在第6步投给了blockX，那么在新一轮，永远只能投给blockX，C也是类似。

这样在R+1轮，就会有1票投给空块，两票投给blockX，最终达成共识blockX，A,B,C三人都会commit blockX，与D一致，没有产生冲突。

2.3.3 Tendermint共识和PBFT共识比较

Tendermint共识算法和PBFT非常相似的，可以说是PBFT的变种，那我们来比较一下：

相同点：

1. 同属BFT体系。
2. 抗1/3拜占庭节点攻击。
3. 三阶段提交，第一阶段广播交易（区块），后两阶段广播签名（确认）。
4. 两者都需要达到法定人数才能提交块。

不同点：

1. Tendermint与PBFT的区别主要是在超过1/3节点为拜占庭节点的情况下。
 - a. 当拜占庭节点数量在验证者数量的1/3和2/3之间时，PBFT算法无法提供保证，使得攻击者可以将任意结果返回给客户端。
 - b. 而Tendermint共识模型认为必须超过2/3数量的precommit确认才能提交块。
- 举个例子，如果1/2的验证者是拜占庭节点，Tendermint中这些拜占庭节点能够阻止区块的提交，但他们自己也无法提交恶意块。而在PBFT中拜占庭节点却是可以提交块给客户端。
2. 另一个不同点在于拜占庭节点概念不同，PBFT指的是节点数，而Tendermint代表的是节点的权益数，也就是投票权力。
3. 最后一点，PBFT需要预设一组固定的验证人，而Tendermint是通过要求超过2/3法定人数的验证人员批准会员变更，从而支持验证人的动态变化。

2.3.4 出块选举

Tendermint本身是BFT+POS共识，在BFT之前需要先通过POS方法选出一个proposer来进行提案，

Tendermint提供了一个votingPower更新算法，算法的规则如下：

- Validator的初始votingPower与其stake是相等的，stake是什么？之前提到过Tendermint的共识算法是POS+BFT，这里的stake就是POS算法的权重，类似于POW算法的算力，用来衡量一个节点的权重的。
如果Validator A在创世块中的stake是1，那么它的votingPower也会被初始化为1
- 每一轮结束后都会对Validator的votingPower做一次更新
 - 如果一个Validator在当前轮中没有被选中为proposer，那么它的votingPower将增加，增加的值为它初始的stake，
例如Validator A的初始化stake为1，如果A没有被选中为proposer，那么它的 $votingPower = pre_votingPower + stake$ 。
 - 如果一个Validator在当前轮中被选中为proposer，那么它的votingPower将减少，减少的值为数组中其他Validator的stake之和，
例如：Validator集合={A:1,B:2,C:3}，如果C被选中为proposer，那么C的 $votingPower = pre_votingPower - (stake_a + stake_b)$

下面这张图演示了如何选择proposer的过程：



例：Validator集合={A:1,B:2,C:3}

首先假设我们在创世块中一个配置了三个Validator分别是A, B, C他们的stake分别是1, 2, 3，因此这三个Validator的votingPower也分别为：1, 2, 3。

在第一轮中，根据上文介绍的规则，Tendermint会选择votingPower最大的Validator作为proposer，

所以在第一轮中proposer为C，就是上图中第一列中标红的方块。

根据规则，每一轮结束后会对所有的Validator的votingPower做一次更新，所以从上图可以看出在第二轮中A, B, C的votingPower分别变成2, 4, 0。

从上图中可以看出，Validator C 在四轮中有两次被选中为proposer，

这是由于每个Validator称为proposer的比率是与它的stake占所有Validator的stake比率有关，

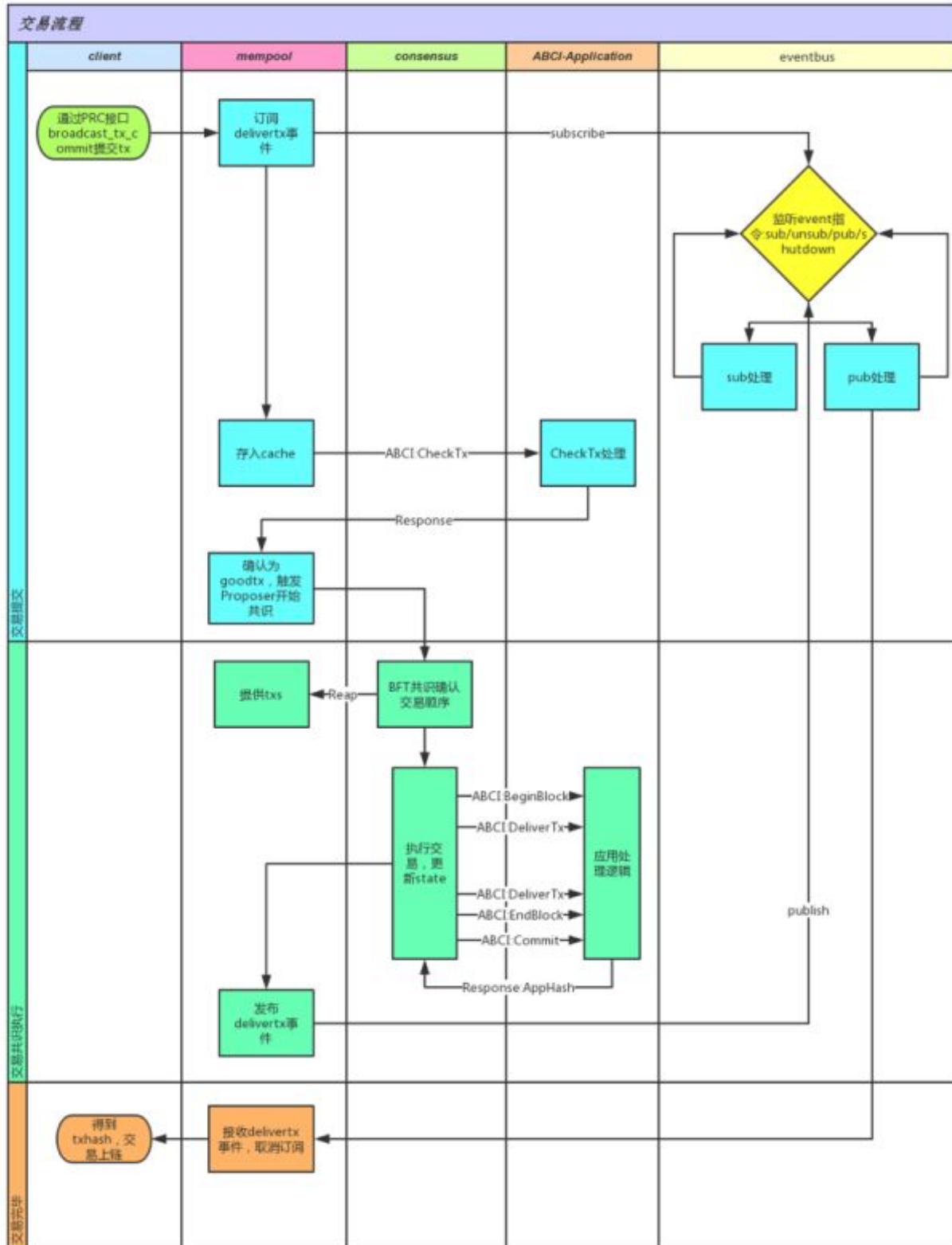
Validator C 的stake是3，其中总的stake为6，所以C称为proposer的比率是50%。

验证节点的结构体信息：

每个height对应的验证集合是按照votePower大小排序的，ProposerPriority是提案优先级。
是否被选中为提案者由这个优先级决定。

```
type Validator struct {
    Address      Address
    PubKey       crypto.PubKey
    VotingPower  int64
    ProposerPriority int64
}
```

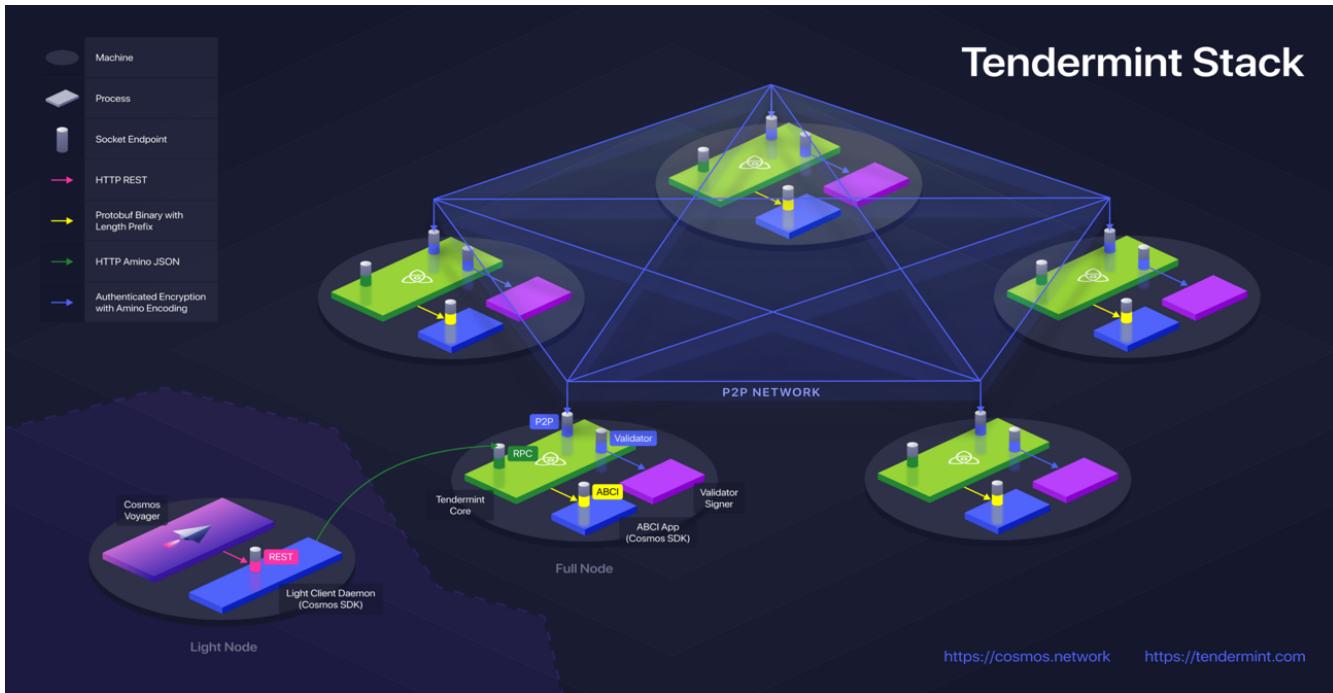
2.3.5 Tendermint 工作流程：



上图简单描述了Tendermint的工作流。大致为：

1. client通过RPC接口broadcast_tx_commit提交交易；
2. mempool调用ABCI接口CheckTx用于校验交易的有效性，比如交易序号、发送者余额等，同时订阅交易执行后的事件并等待监听。
3. 共识从mempool中获取交易开始共识排序，打包区块，确定之后依次调用ABCI相关接口更新当前的世界状态，并触发事件。
4. 最终将交易信息返回client。

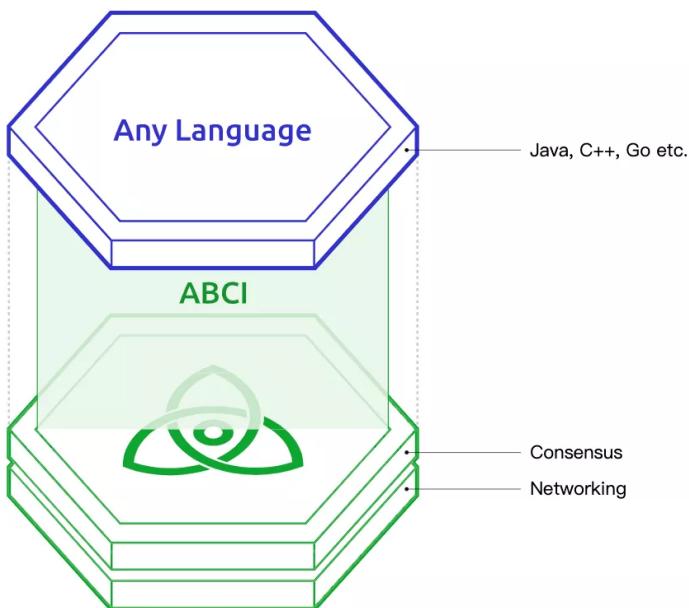
2.3.6 基于Tendermint 底层实现的区块链网络部署结构



- 。最大的绿色方块是Tendermint节点
- 。蓝色的方块是开发的链应用层
- 。 粉红色的方块是签名节点，用来对区块进行签名，单独部署签名节点目的是从安全方面考虑的。

3.ABCI

3.1 定义一个通用的Tendermint和app应用通信的方法集



ABCI 应用程序和Tendermint 共识引擎可以在同一个进程或者分开运行。

- 。一起运行时Tendermint直接调用ABCI 的方法。Cosmos-SDK就是基于go语言实现的app应用库。
- 。分开在不同的进程运行时，Tendermint 为 ABCI 方法打开四个连接。每个连接都处理 ABCI 方法调用的一个子集。

Consensus connection

- 由共识协议驱动，负责区块执行。
- 处理InitChain、BeginBlock、DeliverTx、EndBlock和Commit方法调用。

Mempool connection

- 用于在共享或包含在块中之前验证新交易。
- 处理CheckTx呼叫。

Info connection

- 用于初始化和来自用户的查询。
- 处理Info和Query调用。

Snapshot connection

- 用于提供和恢复状态同步快照
- 处理ListSnapshots、LoadSnapshotChunk、OfferSnapshot和ApplySnapshotChunk调用。

3.2 ABCI方法集：

```

type Application interface {
    // Info/Query Connection
    Info(RequestInfo) ResponseInfo    // Return application info
    Query(RequestQuery) ResponseQuery // Query for state

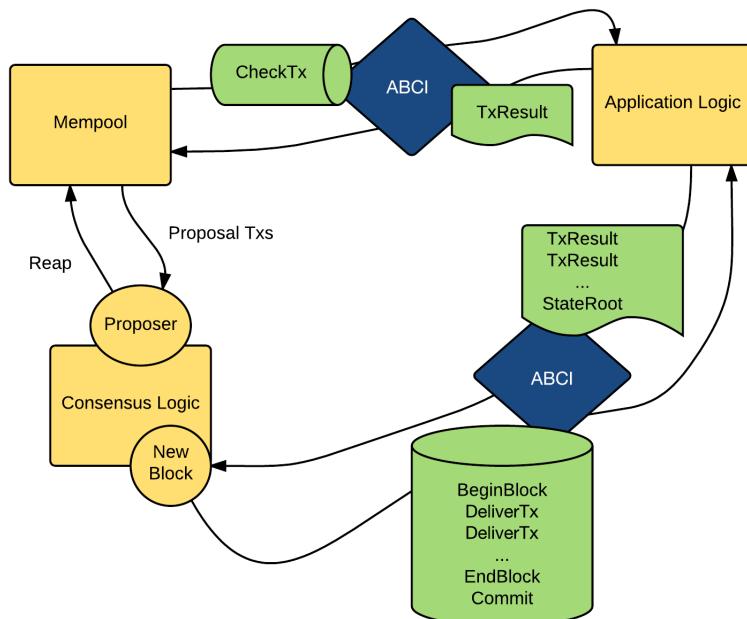
    // Mempool Connection
    CheckTx(RequestCheckTx) ResponseCheckTx // Validate a tx for the mempool

    // Consensus Connection
    InitChain(RequestInitChain) ResponseInitChain // Initialize blockchain w validators/other info from TendermintCore
    PrepareProposal(RequestPrepareProposal) ResponsePrepareProposal
    ProcessProposal(RequestProcessProposal) ResponseProcessProposal
    // Commit the state and return the application Merkle root hash
    Commit() ResponseCommit
    // Create application specific vote extension
    ExtendVote(RequestExtendVote) ResponseExtendVote
    // Verify application's vote extension data
    VerifyVoteExtension(RequestVerifyVoteExtension) ResponseVerifyVoteExtension
    // Deliver the decided block with its txs to the Application
    FinalizeBlock(RequestFinalizeBlock) ResponseFinalizeBlock

    // State Sync Connection
    ListSnapshots(RequestListSnapshots) ResponseListSnapshots           // List available snapshots
    OfferSnapshot(RequestOfferSnapshot) ResponseOfferSnapshot          // Offer a snapshot to the application
    LoadSnapshotChunk(RequestLoadSnapshotChunk) ResponseLoadSnapshotChunk // Load a snapshot chunk
    ApplySnapshotChunk(RequestApplySnapshotChunk) ResponseApplySnapshotChunk // Apply a snapshot chunk
}

```

3.3 ABCI消息流程：

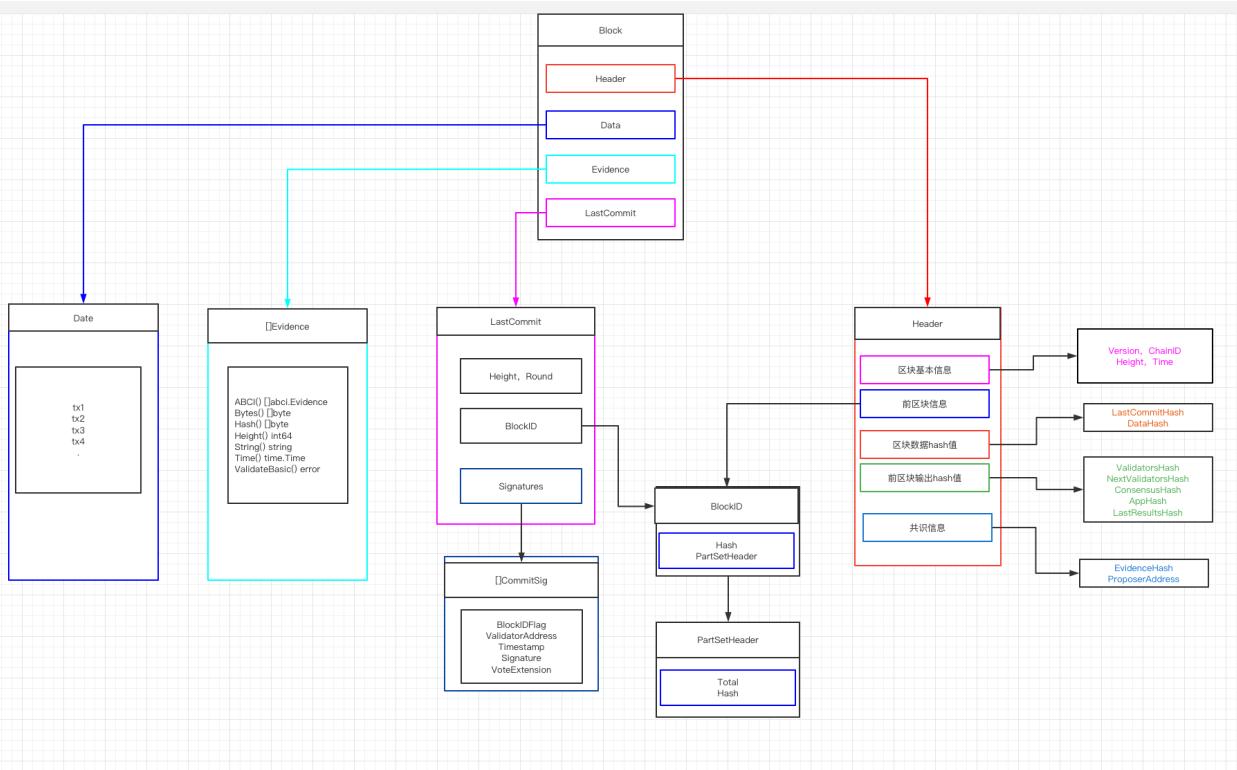


3.4 ABCI和Tendermint Core之间的apphash值。Tendermint是先共识后执行

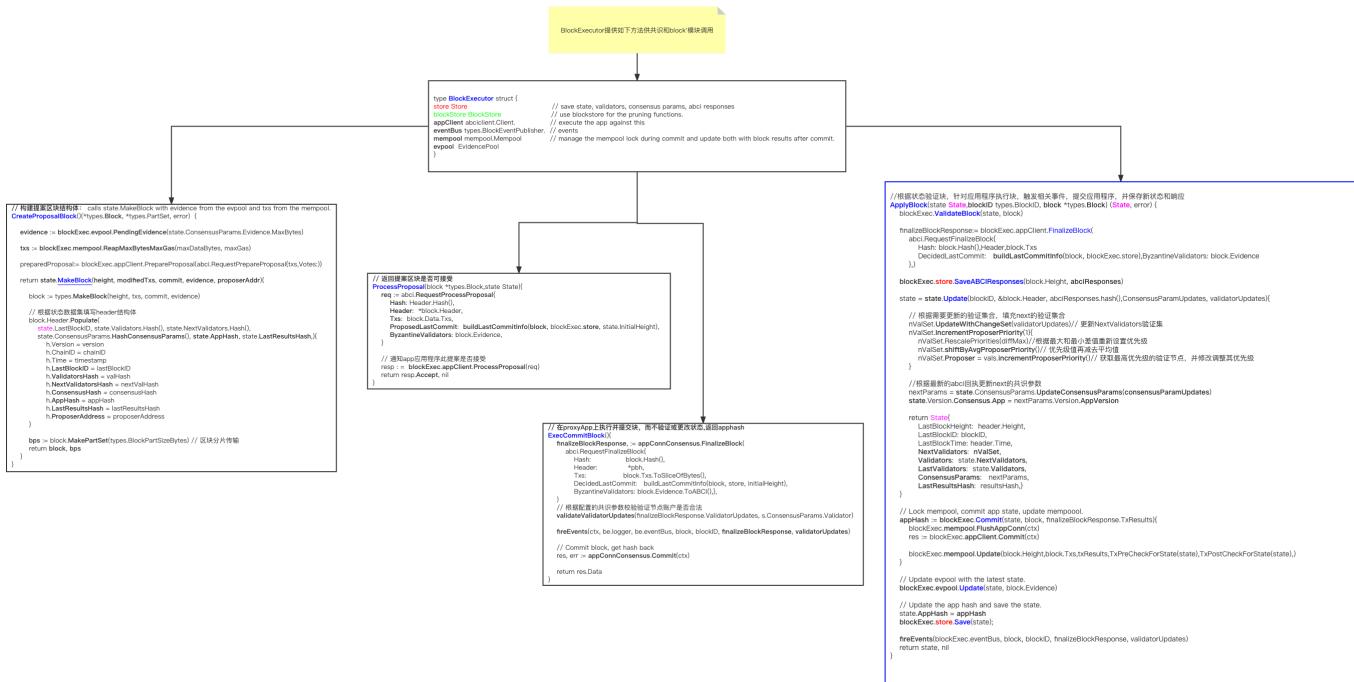


4.核心模块代码机构

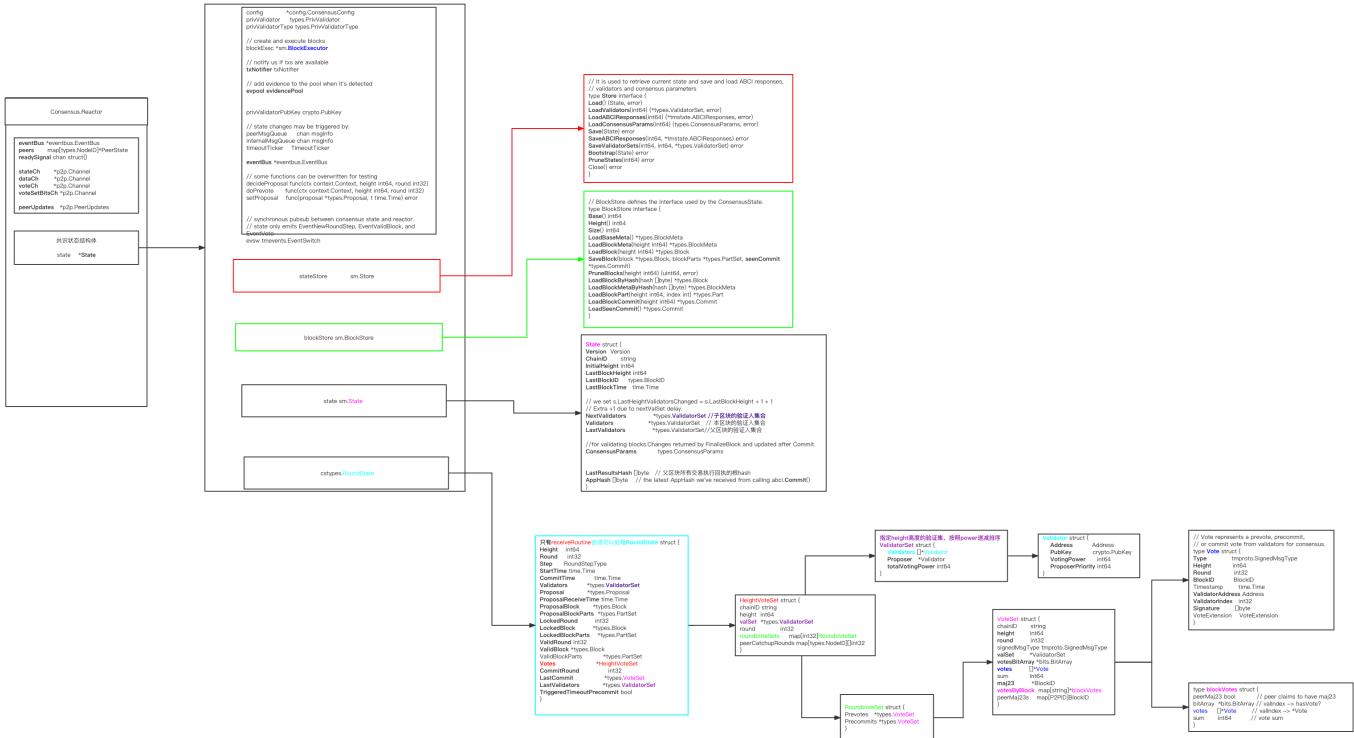
4.1 Block结构体:



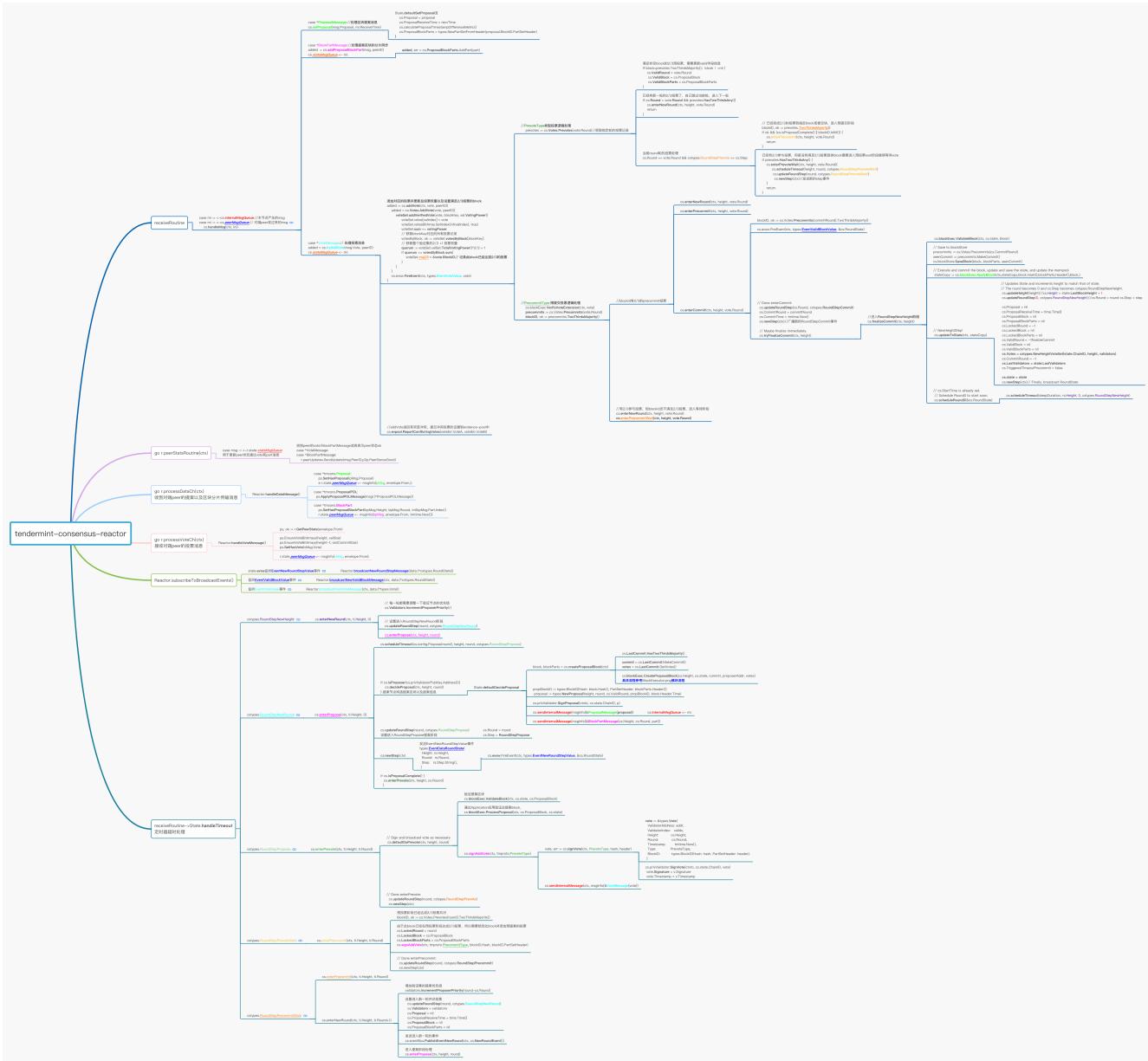
4.2 BlockExecutor执行模块



4.3 Consensus-State结构体：



4.4 TM共识引擎相关业务处理流程



4.4.1 cs.receiveRoutine()主

要是更新共识模块的状态数据

主要处理VoteMsg消息：

- 添加对应的投票信息以及计算投票的权重是否满足2/3

PrevoteType预投票阶段vote消息处理逻辑：

- 已经产生了满足2/3投票权重的block，直接进入enterPrecommit阶段
- 已经有2/3的参与投票了，但是没有达成统一的block，需要进入RoundStepPrevoteWait超时阶段

PrecCommitType预提交阶段vote消息的处理

- 已经产生满足2/3 precommit投票的blockid

进入enterCommit处理阶段：

更新状态进入RoundStepCommit

尝试完成提交tryFinalizeCommit:

验证提交的区块blockExec.ValidateBlock

存储区块信息到数据库blockStore.SaveBlock

调用应用层执行具体交易blockExec.ApplyBlock

更新状态并开启RoundStepNewHeight阶段

开启RoundStepNewHeight新高度的超时器

- 有2/3参与投票，但blockid还不满足2/3投票，进入enterPrecommitWait等待阶段：

4.4.2 handleTimeout () 各阶段超时处理

case: stepType:

RoundStepNewHeight: `enterNewRound()`:

- 调整验证节点的优先级, `Validators.IncrementProposerPriority(r)`
- 设置进入RoundStepNewRound阶段, `updateRoundStep(RoundStepNewRound)`
- 进入提案阶段处理: `enterPropose()`

RoundStepNewRound: `enterPropose()`

- 开启超时器`scheduleTimeout(RoundStepPropose)`
- 提案节点构造提案区块以及提案消息: `defaultDecideProposal`
 - 根据LastCommit和evidence和txs和proposerAddr构造提案区块
 - 构造提案消息NewProposal并签名SignProposal并更新状态
 - 更新RoundStep进入RoundStepPropose阶段
 - 广播EventNewRoundStepValue通知其他peer节点
 - 如果提案阶段已经完成, 进入预投票阶段`enterPrevote`

RoundStepPropose: `enterPrevote()`

- 签名投票并广播: `defaultDoPrevote()`
 - 验证提案区块: `blockExec.ValidateBlock`
 - 通知应用层验证此区块: `blockExec.ProcessProposal()`
 - 对投票签名并通知状态模块继续处理: `signAddVote(PrevoteType)`
- 更新RoundStepPrevote阶段并广播

RoundStepPrevoteWait: `enterPrecommit()`

- 如果预投票阶段没有达成2/3的共识, 那就直接在预提交阶段头空区块
- 如果预投票阶段已经达成2/3的共识, 需要将节点锁定到此block和对应的round上
- 签名添加预提交的投票: `signAddVote(PrecommitType)`
- 更新到RoundStepPrecommit阶段并广播

RoundStepPrecommitWait:

- `enterPrecommit()`:
- 进入新一轮投票: `cs.enterNewRound(ctx, ti.Height, ti.Round+1)`
 - 调整验证阶段的提案优先级根据round: `validators.IncrementProposerPriority(round-cs.Round)`
 - 设置进入新一轮的共识投票: `cs.updateRoundStep(round, cstypes.RoundStepNewRound)`
 - 进入提案阶段的处理逻辑: `cs.enterPropose(ctx, height, round)`

5.总结

Cosmos的目标是区块链互联网, 也就是万链互通的生态。

为此将区块链分成共识层和应用层;

为支持应用层多语言开发抽象出ABCI模块;

为支持资产跨链实现了IBC跨链协议。

由于Tendermint底层是基于POS的BFT共识,

投票权和权益质押绑定

支持1/3拜占庭容错

秒级出块的高TPS

快速实现最终确认, 也就是出块即确认

特点:

每条链只针对一个特定的应用场景: 比如, HUB链, EVM链, DEX链, Defi链, 隐私链, 和NFT+GameFi链等。

各个应用链之间通过标准的IBC可以实现资产的跨链, 从而可以参与到不通的应用生态中。

不追求一个能满足所有应用场景的超级链。

可优化点:

此共识继承了PBFT的三阶段二次投票, 投票阶段是所有验证节点之间互相通信, 导致通信量很大, 影响共识效率。默认上限100个

目前已由一些基于PBFT共识的项目在投票阶段为了减少不必要的通信, 使用聚合签名算法将各个验证节点的投票收集起来组合成一个聚合签名然后再广播出去, 从而减少两两之间的通信, 提高共识效率。

缺点：

为了保证共识效率验证节点数量有限制，从而表现的不够去中性化。