Tendermint文档及代码初步分析

V2019.3.25

一、区块链的一些组成部分：

- P2P:peer discovery、connect and data transferral
- mempool:broadcasting tx
- consensus: agreement on block
- storage:account states
- VM: run smart contract
- user-level permission

Tendermint的思路是对其中某些模块进行解耦，有利于代码复用，降低升级维护难度等等。

二、概述

1. 两个主要部件

- Tendermint Core

  共识层

- ABCI(Application Blockchain Interface)

  接口层，包括有TSP(Tendermint Socket Protocol)

2. ABCI概述

对基于ABCI开发的应用来说，职责划分如下。

共识引擎的责任：

- 传播交易
- 有效交易顺序的确认

应用程序的责任：

- 生成交易
- 检查交易是否有效
- 处理交易（包括状态更改）

实现一个Application，应该有实现下面这些方法。

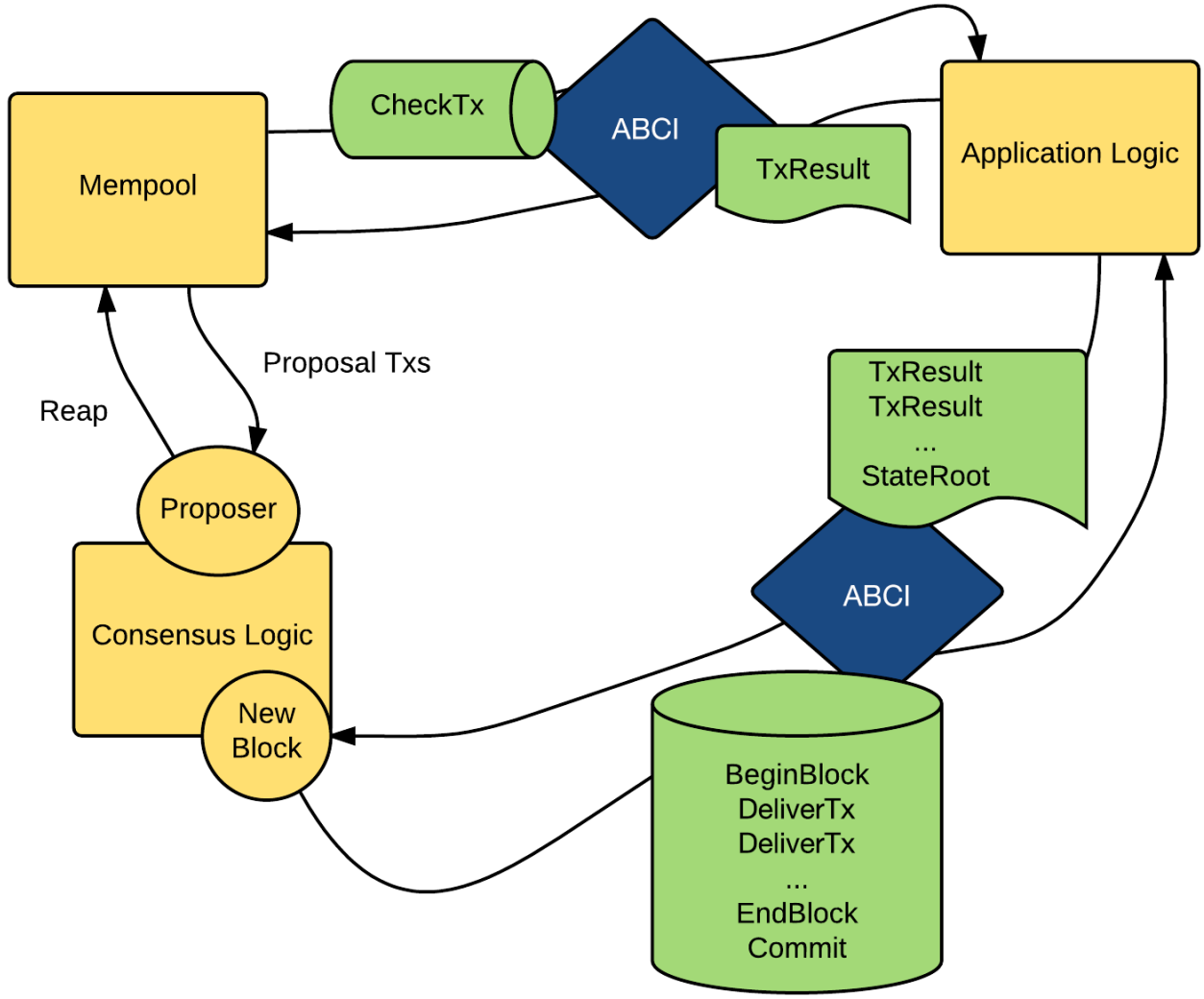
```
$GOPATH/src/github.com/tendermint/tendermint/abci/types/types.proto
service ABCIApplication {
  rpc Echo(RequestEcho) returns (ResponseEcho) ; //Echo 用于调试
  rpc Flush(RequestFlush) returns (ResponseFlush); //Tendermint端提醒应用
  rpc Info(RequestInfo) returns (ResponseInfo);//Tendermint询问应用一些状态信息，比如最后一次块高等等
  rpc SetOption(RequestSetOption) returns (ResponseSetOption);//选项相关
  rpc DeliverTx(RequestDeliverTx) returns (ResponseDeliverTx); //见下面
  rpc CheckTx(RequestCheckTx) returns (ResponseCheckTx); //见下面
  rpc Query(RequestQuery) returns (ResponseQuery);//通过RPC可以Query值
  rpc Commit(RequestCommit) returns (ResponseCommit);//见下面
  rpc InitChain(RequestInitChain) returns (ResponseInitChain); //处在Genesis时，Tendermint给应用一
个InitChain消息
  rpc BeginBlock(RequestBeginBlock) returns (ResponseBeginBlock);//块开始
  rpc EndBlock(RequestEndBlock) returns (ResponseEndBlock);//快结束
}
```

最主要的三种如下：

- `DeliverTx`：是应用的主要部分。链中的每笔交易都通过这个消息进行传送。应用需要基于当前状态，应用协议，和交易的加密证书上，去验证接收到 `DeliverTx` 消息的每笔交易。一个经过验证的交易递送后，需要去更新应用状态（State—） - 比如通过将绑定一个值到键值存储，或者通过更新 `UTXO` 数据库。
- `CheckTx`：类似于 `DeliverTx`，但是它仅用于验证交易。`Tendermint Core` 的内存池首先通过 `CheckTx` 检验一笔交易的有效性，并且只将有效交易中继到其他节点。比如，一个应用可能会检查在交易中不断增长的序列号，如果序列号过时，`CheckTx` 就会返回一个错误。又或者，他们可能使用一个基于容量的系统，该系统需要对每笔交易重新更新容量。
- `Commit`：用于计算当前应用状态的一个加密保证（`cryptographic commitment`），这个加密保证会被放到下一个区块头。这有一些比较方便的属性。现在，更新状态时的不一致性显得像是区块链的分叉，它会捕获这一整类的编程错误。这同样也简化了保障轻节点客户端安全的开发，因为 `Merkel-hash` 证明可以通过在区块哈希上的检查得到验证，区块链哈希由一个 `Quorum`（法定人数） 签署。

  一个应用可能有多个 `ABCI socket` 连接。`Tendermint Core` 给应用创建了三个 `ABCI` 连接：一个用于内存池广播时的交易验证，一个用于运行提交区块时的共识引擎，还有一个用于查询应用状态。

3. ABCI大致运行流程：



1. 一个tx要加入mempool，首先Tendermint core调用checkTx，让Application验证是否合法，合法保留，不合法丢弃。
2. 在某些条件下，选出的Proposer（相当于主节点）会从mempool获得tx，并按照规则打包成块，经过共识，即表示出一新块。

在types.proto文件下，还可以看到一些其他定义，比如块头：

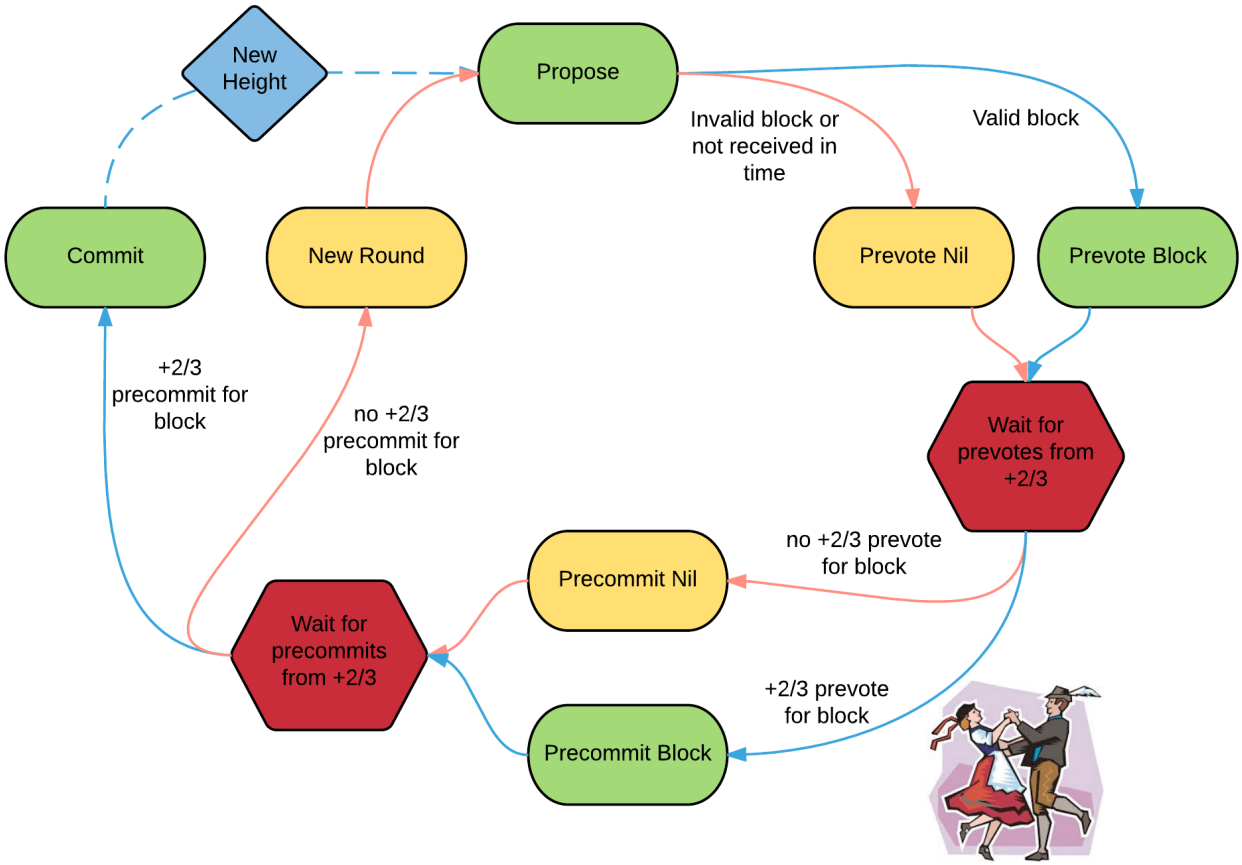
```
$GOPATH/src/github.com/tendermint/tendermint/abci/types/types.proto
// Blockchain Types

message Header {
  // basic block info
  Version version = 1 [(gogoproto.nullable)=false];
  string chain_id = 2 [(gogoproto.customname)="ChainID"];
  int64 height = 3;
  google.protobuf.Timestamp time = 4 [(gogoproto.nullable)=false, (gogoproto.stdtime)=true];
  int64 num_txs = 5;
  int64 total_txs = 6;

  // prev block info
  BlockID last_block_id = 7 [(gogoproto.nullable)=false];
```

```
  // hashes of block data
  bytes last_commit_hash = 8; // commit from validators from the last block
  bytes data_hash = 9;        // transactions

  // hashes from the app output from the prev block
  bytes validators_hash = 10;   // validators for the current block
  bytes next_validators_hash = 11;   // validators for the next block
  bytes consensus_hash = 12;   // consensus params for current block
  bytes app_hash = 13;         // state after txs from the previous block
  bytes last_results_hash = 14;// root hash of all results from the txs from the previous block

  // consensus info
  bytes evidence_hash = 15;    // evidence included in the block
  bytes proposer_address = 16; // original proposer of the block
}

message Version {
  uint64 Block = 1;
  uint64 App = 2;
}


message BlockID {
  bytes hash = 1;
  PartSetHeader parts_header = 2 [(gogoproto.nullable)=false];
}
```

4. 代码的确定性

时钟不确定、浮点数不确定、线程竞态，Golang的map迭代等等会造成对相同初始状态，结果不同，难以达成共识，在区块链代码里应避免不确定性。

5. Tendermint共识



上图就是Tendermint达成共识的大致流程：

协议中参与者叫做验证者（Validators），验证者提议块以及给块投票；链上提交的块（Block），每个块都有它的块高（Height，可以理解为块的编号）。如果块在当下轮次（Round）未提交成功，根据协议，可在下一轮次让下一个Proposer接着提交，块高是不会重复的。期间会经过讲过阶段：预投票（Pre-Vote）和预提交（Pre-Commit），只有在同一轮次有2/3的验证者预提交了该块，此块才算出块，块一出即确定，不会产生分叉。

上图右下角有一对夫妇在跳波卡舞，因为验证人做的事情就像是在跳波卡舞。当超过 2/3 的验证人对同一个块进行了预投票，我们就把它叫做一个“波卡”（Polka）（可以理解为一个证书-Certificate）。每一个预提交都必须先被同一轮中的一个波卡所验证。

• **提议（Proposals）**：在每一轮（round）中，新区块的提议者必须是有效的，并且告诉（gossiped）其他验证者。如果在一定时间内没有收到当轮提议（proposal），当前提议者将被后面的提议者接替。

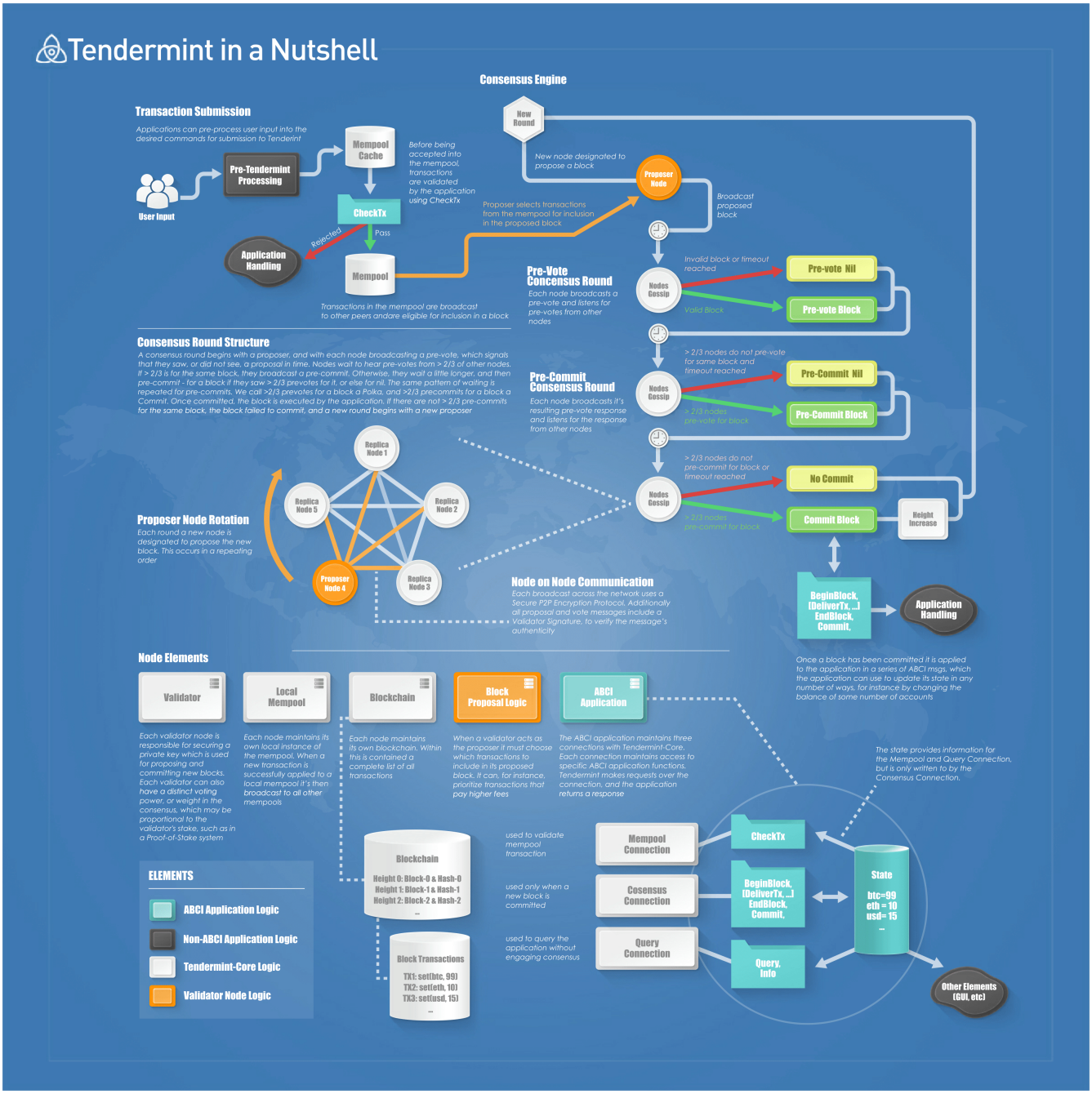
• **投票（Votes）**：两阶段的投票基于优化的拜占庭容错。它们分别被称作预投票（pre-vote）和预提交（pre-commit）。对于同一个区块同一轮如果存在超过2/3的预提交（pre-commit）则对应产生一个提交(commit)。 • **锁(Locks)**：在拜占庭节点数少于节点总数的1/3的情况下，Tendermint中的锁机制可以确保没有两个验证者在同一高度提交（commit）了两个不同的区块。锁机制确保了在当前高度验证者的下一轮预投票或者预提交依赖于这一轮的预投票或者预提交。有了那个块的一个波卡，它才能够解锁，并为一个新块进行预提交。

为了应对单个拜占庭故障节点，Tendermint网络至少需要包括4个验证者。每个验证者拥有一对非对称密钥，其中私钥用来进行数字签名，公钥用来标识自己的身份ID。验证者们从公共的初始状态开始，初始状态包含了一份验证者列表。所有的提议和投票都需要各自的私钥签名，便于其他验证者进行公钥验证。

由于一些原因，验证人可能在提交一个块时失败：当前提议者可能离线，或者网络非常慢。每轮的开始对同步有弱的依赖性。每一轮开始期间，存在一个用来计时的本地同步时钟，如果验证者在`TimeoutPropose`时间内没有收到提议，验证者将参与投票来决定是否跳过当前提交者。`TimeoutPropose`会随着轮数的增加而增加。每轮收到提议以后，进入完全异步模式。之后验证者的每一个网络决定需要得到`2/3`验证者以上的同意。这样降低了对同步时钟的依赖或者网络的延迟。但是这也意味着如果得不到`1/3`以上验证者的响应，整个网络将瘫痪。简言之，每轮，开始提议弱同步，之后投票完全异步。

不同的`Validator`的权重（`weight`）是不一样的，对于加密货币，可以运行权益证明（`POS`），行为不当可销毁其抵押，对于一个利己节点，在作恶前会估计成本，即在`PBFT`上又加入了一些经济上的考量。

下图是`Tendermint`整个运行流程的概述。



三、Tendermint安装

首先运行

```
mkdir -p $GOPATH/src/github.com/tendermint
cd $GOPATH/src/github.com/tendermint
git clone https://github.com/tendermint/tendermint.git
cd tendermint


make get_tools
make get_vendor_deps
```

然后安装grpc相关

```
mkdir -p $GOPATH/src/google.golang.org
cd $GOPATH/src/google.golang.org
git clone https://github.com/grpc/grpc-go.git grpc
git clone https://github.com/google/go-genproto.git genproto
```

然后进入Tendermint的main文件所在处，运行go install

```
cd $GOPATH/src/github.com/tendermint/tendermint/cmd/tendermint
go install
```

此时你可能还会看到某些包不存在，可手动安装，如下：

```
go get -u github.com/go-kit/kit
go get -u github.com/go-logfmt/logfmt
go get -u github.com/gorilla/websocket
go get -u github.com/rcrowley/go-metrics
go get -u github.com/syndtr/goleveldb
go get -u github.com/golang/snappy
go get -u github.com/rs/cors
go get -u github.com/spf13/cobra
go get -u github.com/spf13/viper
go get -u github.com/tendermint/go-amino
```

再次运行go install，应该就OK了。

顺便再安装abci-cli：

```
cd $GOPATH/src/github.com/tendermint/tendermint/abci/cmd/abci-cli
go install
```

PS：若提示某些golang.com的包找不到，可在github上下载镜像

```
mkdir -p $GOPATH/src/golang.org/x/
cd $GOPATH/src/golang.org/x/
git clone https://github.com/golang/crypto.git
```

非crypto包，改名字一样下载

四、ABCI测试

1. Init

```shell
tendermint init
$I[2019-03-24|23:00:31.877] Found private validator module=main
$keyFile=/home/xcshuan/.tendermint/config/priv_validator_key.json
stateFile=/home/xcshuan/.tendermint/data/priv_validator_state.json
$I[2019-03-24|23:00:31.877] Found node key                                 module=main
path=/home/xcshuan/.tendermint/config/node_key.json
$I[2019-03-24|23:00:31.877] Found genesis file                             module=main
path=/home/xcshuan/.tendermint/config/genesis.json

```

2. 启动测试应用

```
tendermint node --proxy_app=kvstore --consensus.create_empty_blocks=false
```

`--proxy_app`运行一个指定的内置应用，包括kvstore和counter，`--consensus.create_empty_blocks=false`表示不出空块，否则会按时出块，即使没有任何Tx。 另一种运行方法是：

```
abci-cli kvstore
//再在新开终端运行
tendermint node --consensus.create_empty_blocks=false
```

3. RPC测试

可用的RPC接口如下：

```
//$GOPATH/src/github.com/tendermint/tendermint/rpc/core/routes.go
var Routes = map[string]*rpc.RPCFunc{
    // subscribe/unsubscribe are reserved for websocket events.
    "subscribe":       rpc.NewWSRPCFunc(Subscribe, "query"),
    "unsubscribe":     rpc.NewWSRPCFunc(Unsubscribe, "query"),
    "unsubscribe_all": rpc.NewWSRPCFunc(UnsubscribeAll, ""),

    // info API
    "health":               rpc.NewRPCFunc(Health, ""),
    "status":               rpc.NewRPCFunc(Status, ""),
    "net_info":             rpc.NewRPCFunc(NetInfo, ""),
    "blockchain":           rpc.NewRPCFunc(BlockchainInfo, "minHeight,maxHeight"),
    "genesis":              rpc.NewRPCFunc(Genesis, ""),
    "block":                rpc.NewRPCFunc(Block, "height"),
    "block_results":        rpc.NewRPCFunc(BlockResults, "height"),
    "commit":               rpc.NewRPCFunc(Commit, "height"),
    "tx":                   rpc.NewRPCFunc(Tx, "hash,prove"),
    "tx_search":            rpc.NewRPCFunc(TxSearch, "query,prove,page,per_page"),
    "validators":           rpc.NewRPCFunc(Validators, "height"),
    "dump_consensus_state": rpc.NewRPCFunc(DumpConsensusState, ""),
    "consensus_state":      rpc.NewRPCFunc(ConsensusState, ""),
```

```
    "consensus_params":     rpc.NewRPCFunc(ConsensusParams, "height"),
    "unconfirmed_txs":      rpc.NewRPCFunc(UnconfirmedTxs, "limit"),
    "num_unconfirmed_txs":  rpc.NewRPCFunc(NumUnconfirmedTxs, ""),

    // broadcast API
    "broadcast_tx_commit": rpc.NewRPCFunc(BroadcastTxCommit, "tx"),
    "broadcast_tx_sync":   rpc.NewRPCFunc(BroadcastTxSync, "tx"),
    "broadcast_tx_async":  rpc.NewRPCFunc(BroadcastTxAsync, "tx"),

    // abci API
    "abci_query": rpc.NewRPCFunc(ABCIQuery, "path,data,height,prove"),
    "abci_info":  rpc.NewRPCFunc(ABCIInfo, ""),
}
```

新开一个终端，运行下面的代码即为提交一个transaction，可在Tendermint窗口看到区块高度变化。

```
curl 'http://localhost:26657/broadcast_tx_commit?tx="12345"'
$
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "check_tx": {
      "gasWanted": "1"
    },
    "deliver_tx": {
      "tags": [
        {
          "key": "YXBwLmNyZWF0b3I=",
          "value": "Q29zbW9zaGkgTmV0b3dva28="
        },
        {
          "key": "YXBwLmtleQ==",
          "value": "MTIzNDU="
        }
      ]
    },
    "hash": "5994471ABB01112AFCC18159F6CC74B4F511B99806DA59B3CAF5A9C173CACFC5",
    "height": "5"
  }
}
```

然后可以查询该transaction的存在与否：

```
curl -s 'localhost:26657/abci_query?data="12345"'
$
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "response": {
      "log": "exists",
      "key": "MTIzNDU=",
      "value": "MTIzNDU="
    }
  }
}
```

查看当前链的状态:

```
curl -s 'localhost:26657/status'
$
{
  "jsonrpc": "2.0",
  "id": "",
  "result": {
    "node_info": {
      "protocol_version": {
        "p2p": "7",
        "block": "10",
        "app": "1"
      },
      "id": "0808516b4351ad896b5a3c605337ad20a9be3bdc",
      "listen_addr": "tcp://0.0.0.0:26656",
      "network": "test-chain-CnStT5",
      "version": "0.31.0",
      "channels": "4020212223303800",
      "moniker": "dasheng",
      "other": {
        "tx_index": "on",
        "rpc_address": "tcp://0.0.0.0:26657"
      }
    },
    "sync_info": {
      "latest_block_hash": "888F815CA01970F7F9449DFF2D99A6C0A8331B478C4A6DDF57E6E30311BE199B",
      "latest_app_hash": "0400000000000000",
      "latest_block_height": "6",
      "latest_block_time": "2019-03-24T15:07:30.378415336Z",
      "catching_up": false
    },
    "validator_info": {
      "address": "5477FEAF8A105C4DDEAB417D48573204C1C0A0CE",
      "pub_key": {
        "type": "tendermint/PubKeyEd25519",
        "value": "0aByLoLssM6noy9Uzcnmyn3+r01VbZ6NxAyoZdV6sOU="
      },
```

```
      "voting_power": "10"
    }
  }
}
```

如果想重新从头测试，可运行`tendermint unsafe_reset_all`，清除数据，否则运行其他应用会提示Hash不匹配。

4. 测试代码分析

可以找到`kv-store`这个应用实现的代码：

```
$GOPATH/src/github.com/tendermint/tendermint/abci/example/kvstore/kvstore.go
type KVStoreApplication struct {
    types.BaseApplication

    state State
}

func NewKVStoreApplication() *KVStoreApplication {
    state := loadState(dbm.NewMemDB())
    return &KVStoreApplication{state: state}
}

func (app *KVStoreApplication) Info(req types.RequestInfo) (resInfo types.ResponseInfo) {
    return types.ResponseInfo{
        Data:       fmt.Sprintf("{\"size\":%v}", app.state.Size),
        Version:    version.ABCIVersion,
        AppVersion: ProtocolVersion.Uint64(),
    }
}

// tx is either "key=value" or just arbitrary bytes
func (app *KVStoreApplication) DeliverTx(tx []byte) types.ResponseDeliverTx {
    var key, value []byte
    parts := bytes.Split(tx, []byte("="))
    if len(parts) == 2 {
        key, value = parts[0], parts[1]
    } else {
        key, value = tx, tx
    }
    app.state.db.Set(prefixKey(key), value)
    app.state.Size += 1

    tags := []cmn.KVPair{
        {Key: []byte("app.creator"), Value: []byte("Cosmoshi Netowoko")},
        {Key: []byte("app.key"), Value: key},
    }
    return types.ResponseDeliverTx{Code: code.CodeTypeOK, Tags: tags}
}

func (app *KVStoreApplication) CheckTx(tx []byte) types.ResponseCheckTx {
    return types.ResponseCheckTx{Code: code.CodeTypeOK, GasWanted: 1}
}
```

```
func (app *KVStoreApplication) Commit() types.ResponseCommit {
    // Using a memdb - just return the big endian size of the db
    appHash := make([]byte, 8)
    binary.PutVarint(appHash, app.state.Size)
    app.state.AppHash = appHash
    app.state.Height += 1
    saveState(app.state)
    return types.ResponseCommit{Data: appHash}
}

func (app *KVStoreApplication) Query(reqQuery types.RequestQuery) (resQuery types.ResponseQuery)
{
    if reqQuery.Prove {
        value := app.state.db.Get(prefixKey(reqQuery.Data))
        resQuery.Index = -1 // TODO make Proof return index
        resQuery.Key = reqQuery.Data
        resQuery.Value = value
        if value != nil {
            resQuery.Log = "exists"
        } else {
            resQuery.Log = "does not exist"
        }
        return
    } else {
        resQuery.Key = reqQuery.Data
        value := app.state.db.Get(prefixKey(reqQuery.Data))
        resQuery.Value = value
        if value != nil {
            resQuery.Log = "exists"
        } else {
            resQuery.Log = "does not exist"
        }
        return
    }
}
```

可以看到，kvstore的CheckTx不做任何检查，直接通过；DeliverTx只是把值加入到DB里，而Query只是从DB里找值。

5. 基于ABCI写Application

由上面的测试可知，我们只要写一个Application，实现了那几个方法即可，以下函数仅做示例，不做具体实现。

```
package common

import (
    "github.com/tendermint/tendermint/abci/types"
)

type state struct {
    balance int  //余额
}
```

```
type ChainApp struct {
    types.BaseApplication
    Accounts map[string]state //账户包括一个地址映射到状态
}

func NewChainApp() *ChainApp {
    accounts := make(map[string]state)
    return &ChainApp{Accounts: accounts}
}


func (app *ChainApp) Query(req types.RequestQuery) (rsp types.ResponseQuery) {
    //
    return
}

func (app *ChainApp) DeliverTx(raw []byte) (rsp types.ResponseDeliverTx) {
    //
    return
}

func (app *ChainApp) CheckTx(raw []byte) (rsp types.ResponseCheckTx) {
    //
    return
}
```

五、TendermintCore

1.Init解析

运行下面命令将初始化，Tendermint 根目录，若没有指定TMHOME，则创建在`$HOME/.tendermint`

```
tendermint init
```

这个命令将会在`TMHOME/config`下创建一个私钥文件(priv_validator.json)和一个创世状态文件(genesis.json):

`priv_validator.json`:

```
{
  "address": "5477FEAF8A105C4DDEAB417D48573204C1C0A0CE",
  "pub_key": {
    "type": "tendermint/PubKeyEd25519",
    "value": "OaByLoLssM6noy9Uzcnmyn3+r01VbZ6NxAyoZdV6sOU="},
    "priv_key": {
        "type": "tendermint/PrivKeyEd25519",
        "value":
 "aTIjjMmtA40a0F________IL0ce1Nuk03Ro3gNrFY5oHIuguywzqejL1TNyebKff6vTVVtno3EDKhl1Xqw5Q=="
    }
}
```

`genesis.json`:

```
{
    "genesis_time": "2019-03-24T12:23:07.464397179Z",
    "chain_id": "test-chain-CnStT5",                                    "consensus_params": {
        "block": {                                                          "max_bytes":
"22020096",                                                 "max_gas": "-1",
                                    "time_iota_ms": "1000"
          },
        "evidence": {                                                       "max_age":
"100000"                                                  },
        "validator": {
"pub_key_types": [                                                  "ed25519"
                                            ]
                    }                                                                   },
    "validators": [
        {
        "address": "5477FEAF8A105C4DDEAB417D48573204C1C0A0CE",              "pub_key": {
                                                    "type": "tendermint/PubKeyEd25519",
                                "value": "0aByLoLssM6noy9Uzcnmyn3+r01VbZ6NxAyoZdV6sOU="
      },
        "power": "10",                                                      "name": ""
    }
    ],
      "app_hash": ""
}
```

对于一个本地测试网的Validator而言，这两个文件已经足够了。

- `genesis_time`: 此区块链启动的时间
- `chain_id`: 区块链的ID，每条链都应该是独一无二的。
- `validators`: 区块链初始状态的验证者
    - `address`: 生成的地址
    - `pub_key`: 默认 `pub_key` 类型为 Ed25519. 第二项为编码后的公钥.
    - `power`: 验证者的权重.
    - `name`: 验证者的名字（可选）.
- `app_hash`: 想运行的Application的Hash，不匹配会Panic，所以测试的时候，如果想换Application，应先运行`tendermint unsafe_reset_all`。
- `app_state`: 应用状态，例如初始通证分发.

Genesis的定义可于这里找到:

```
//$GOPATH/src/github.com/tendermint/tendermint/types/genesis.go
type GenesisValidator struct {
    Address Address       `json:"address"`
    PubKey  crypto.PubKey `json:"pub_key"`
    Power   int64         `json:"power"`
    Name    string        `json:"name"`
}
```

```
// GenesisDoc defines the initial conditions for a tendermint blockchain, in particular its
validator set.
type GenesisDoc struct {
    GenesisTime     time.Time          `json:"genesis_time"`
    ChainID         string             `json:"chain_id"`
    ConsensusParams *ConsensusParams   `json:"consensus_params,omitempty"`
    Validators      []GenesisValidator `json:"validators,omitempty"`
    AppHash         cmn.HexBytes       `json:"app_hash"`
    AppState        json.RawMessage    `json:"app_state,omitempty"`
}
```

`config.toml`

关于Tendermint的配置，则放在`$TMHOME/config/config.toml`下。

[config.toml解析]

2. 节点连接(P2P)(待续)

待续

- 多机器
- 单机器

3. 区块链反应器(Blockchain Reactor)(待续)

4. 内存池(MemPoo)(待续)