美图基于以太坊 DPoS 算法实现

林添毅

技术经理

主办方 Geekbang 极客邦科技 InfoQ

区块链前哨
BLOCKCHAIN OUTPOST

正本清源，打造链圈第一技术公众号

掌握前沿区块链资讯
深度分析区块链技术
致力于区块链技术普及

扫码关注区块链前哨

- 简介
- 以太坊基础
- DPoS 实现

工作经历


架构平台，参与分布式
缓存相关服务的研发
2012
2013年
2015
至今
开放平台第三方组件的研发，
如站外的点赞，微博发布框等
2015加入美图，负责分布式缓
存，消息队列，DNS等服务的
研发
以基础服务支撑的角色加入到
区块链领域

我们做了什么

meitu 美图

- ✓ 基于以太坊实现了 **DPoS** 算法并开源到Github
- ✓ 为美图 **贝客钱包** 提供基础服务, 目前支持 ERC20, BTC, EOS
- ✓ **自研公链** 正在落地阶段

代码地址: Github (https://github.com/meitu/go-ethereum)

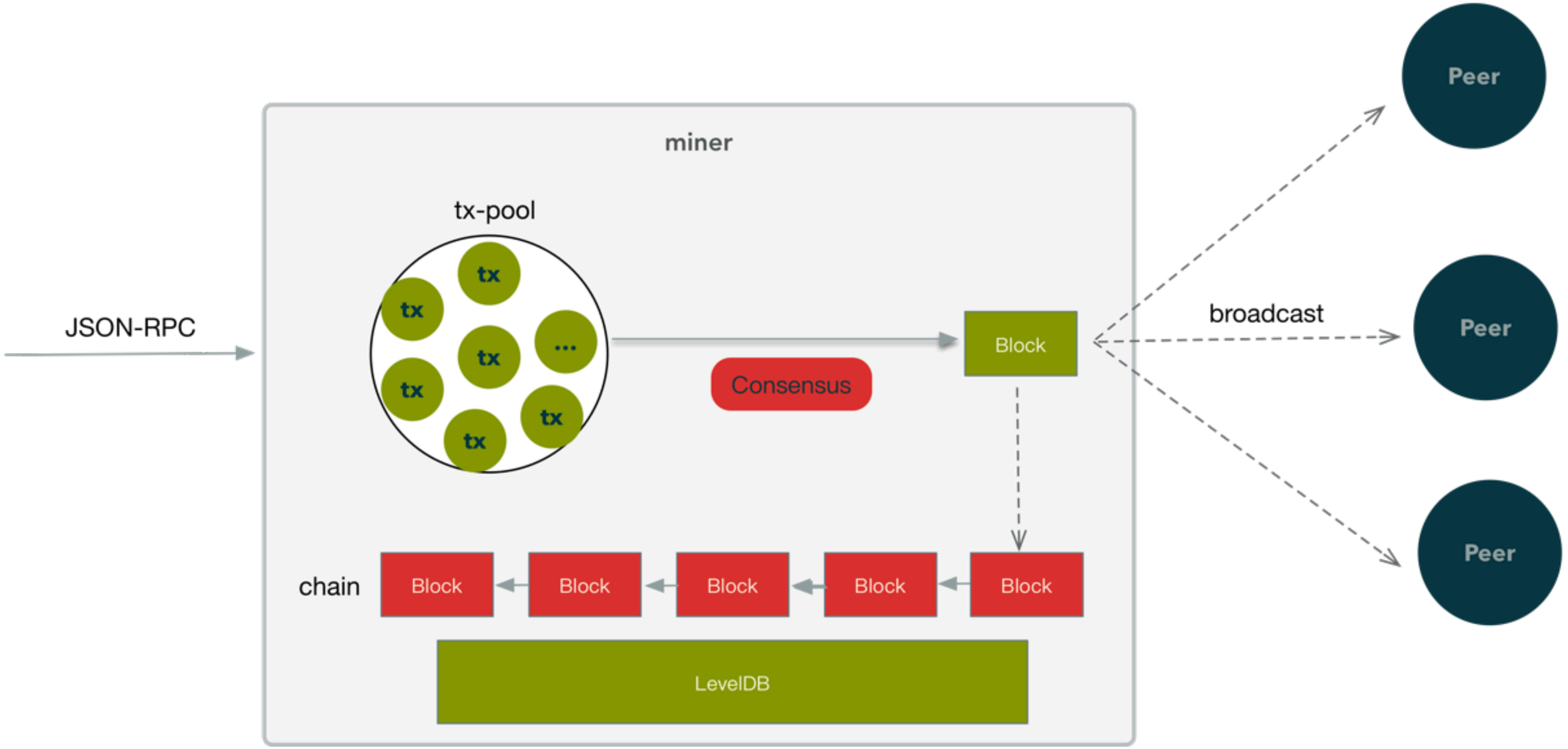
为什么做改造

- ✓ 通过深入源码为美图钱包提供更好的基础服务
- ✓ 技术储备，为公司后续在区块链领域布局做好铺垫
- ✓ 开源代码，回馈社区

meitu 美图

- 简介
- 以太坊基础
- DPoS 实现

整体流程


JSON-RPC
miner
tx-pool
tx
tx
tx
tx
tx
tx
...
Consensus
Block
broadcast
Peer
Peer
Peer
chain
Block
Block
Block
Block
Block
LevelDB

发送交易

```
eth.SendTransaction({
   from: 0x1234,
   to: 0x5678,
   nonce: 1,
   gas: 100000,
   gasPrice: 10000000000,
   value: 100000000000000
})
```

from 转账发起方

to 收款人

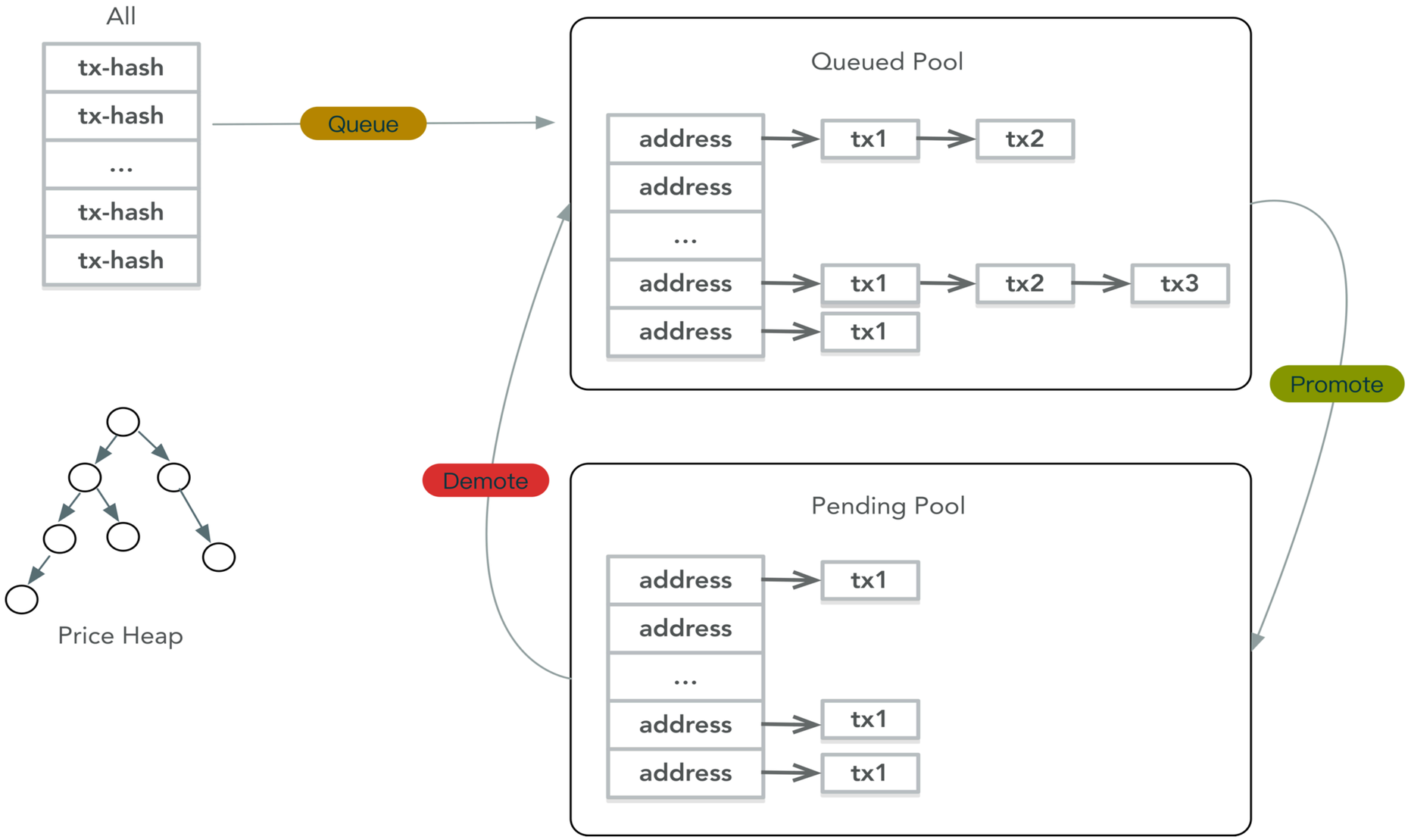
nonce 单调递增计数器, 防止一笔交易被反复执行

gas 允许这笔交易使用的最大 gas 值

gasPrice gas 的单价

value 转账的数目

交易


All
tx-hash
tx-hash
...
tx-hash
tx-hash
Queue
Queued Pool
address
tx1
tx2
address
...
address
tx1
tx2
tx3
address
tx1
Promote
Demote
Price Heap
Pending Pool
address
tx1
address
...
address
tx1
address
tx1

共识算法

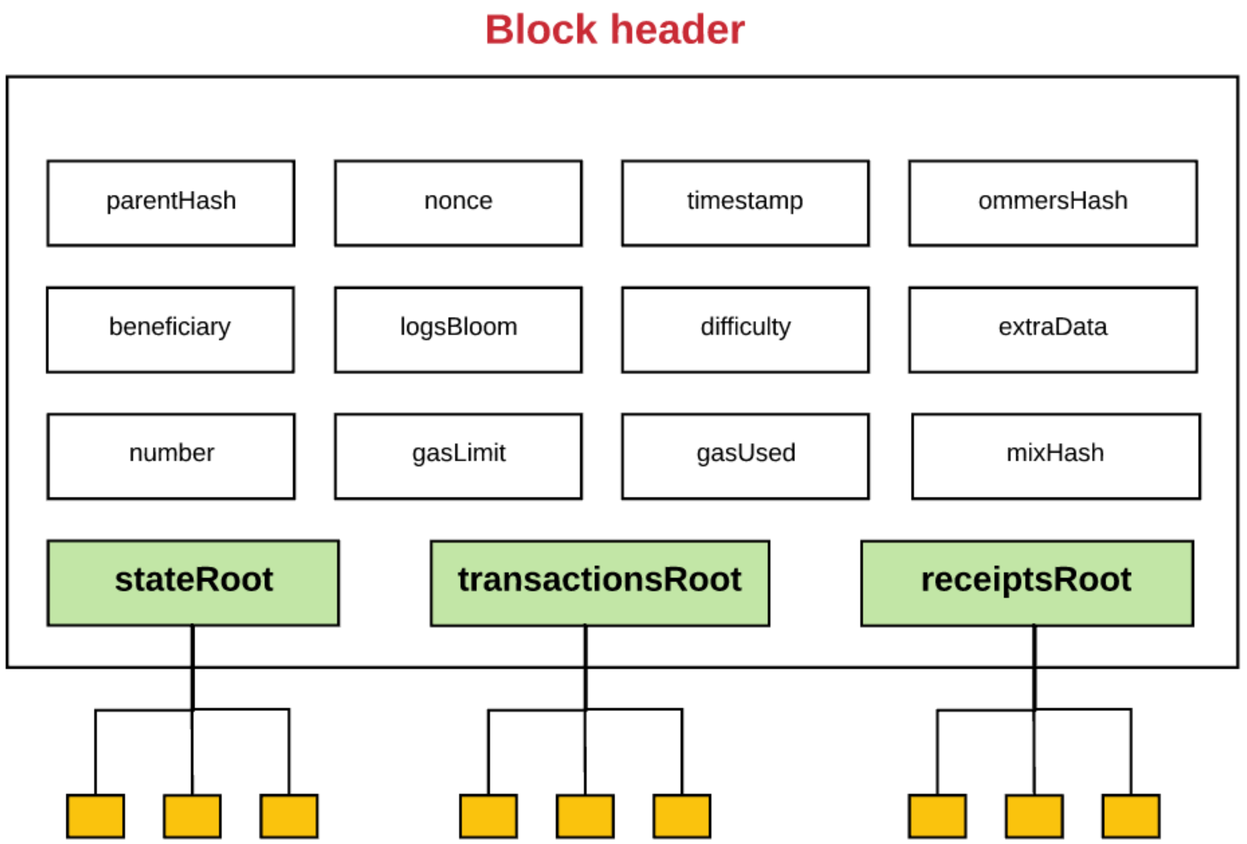
- ✓ PoW(Proof of Work) hash(B) ≤ M/D (1), 其中 D ∈ [1, M], hash = sha256
- ✓ PoS(Proof of Stake) hash(hash(Bprev), A, t) ≤ balance(A) * M/D 其中 hash = sha256
- ✓ DPoS(Delegated Proof of Stake) topN(sort(candidate's votes))

区块

以太坊的区块由三个部分组成：

- ✓ 块头，包含了 PoW 算法的随机数, 难度以及几个用来存储交易树和余额树的根
- ✓ 引用的叔块，以太坊缩短了出块时间, 为了安全性引入奖励叔块的机制
- ✓ 交易，记录当前块所打包的交易，其他节点通过回放交易来更新余额

块头

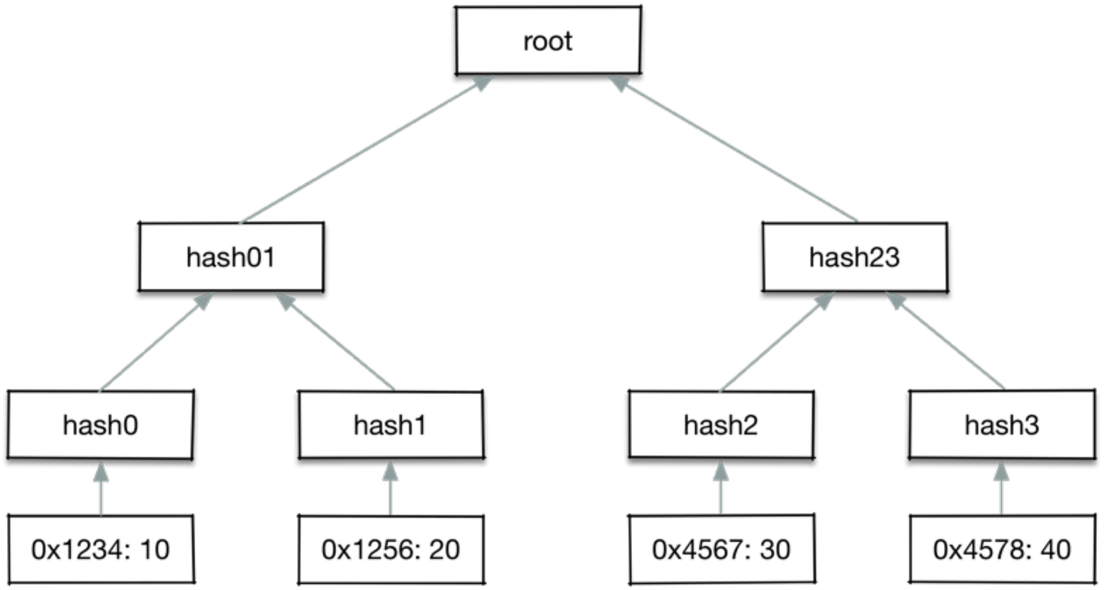


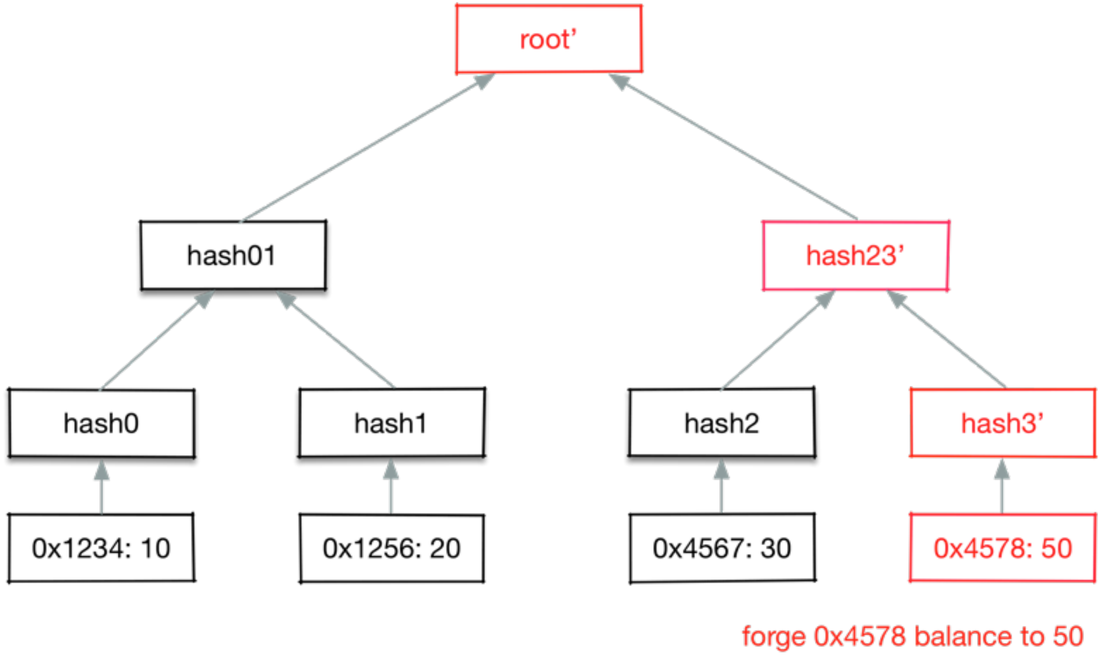
核心几个成员:

- ✓ nonce PoW 算法的随机数
- ✓ stateRoot 全局，存储余额以及合约账户
- ✓ transactionRoot 每个块独有，存放交易
- ✓ ReceiptRoot 每个块独有，存放交易收据

(图片来自: https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369)

Merkle Trie

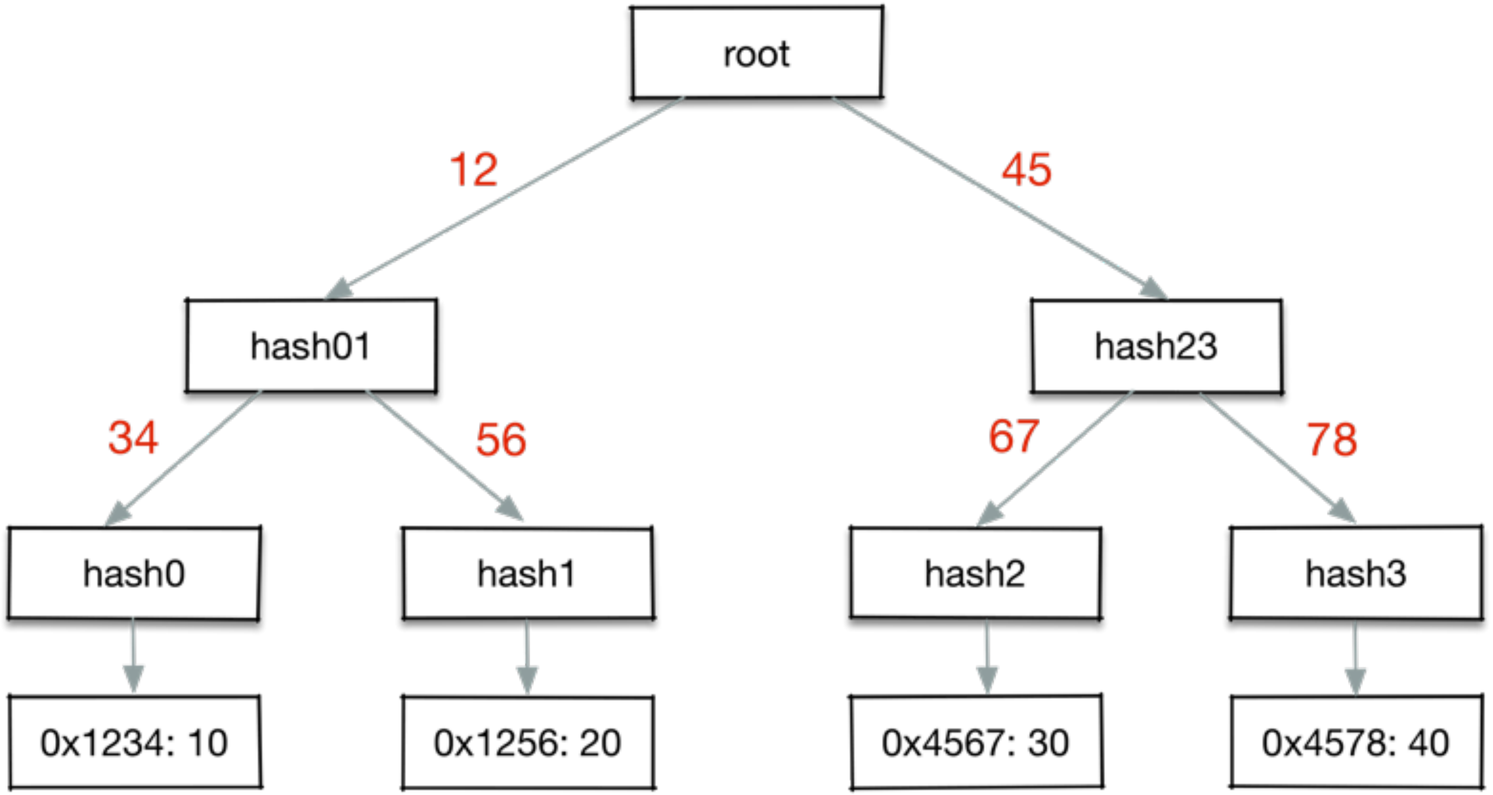




forge 0x4578 balance to 50

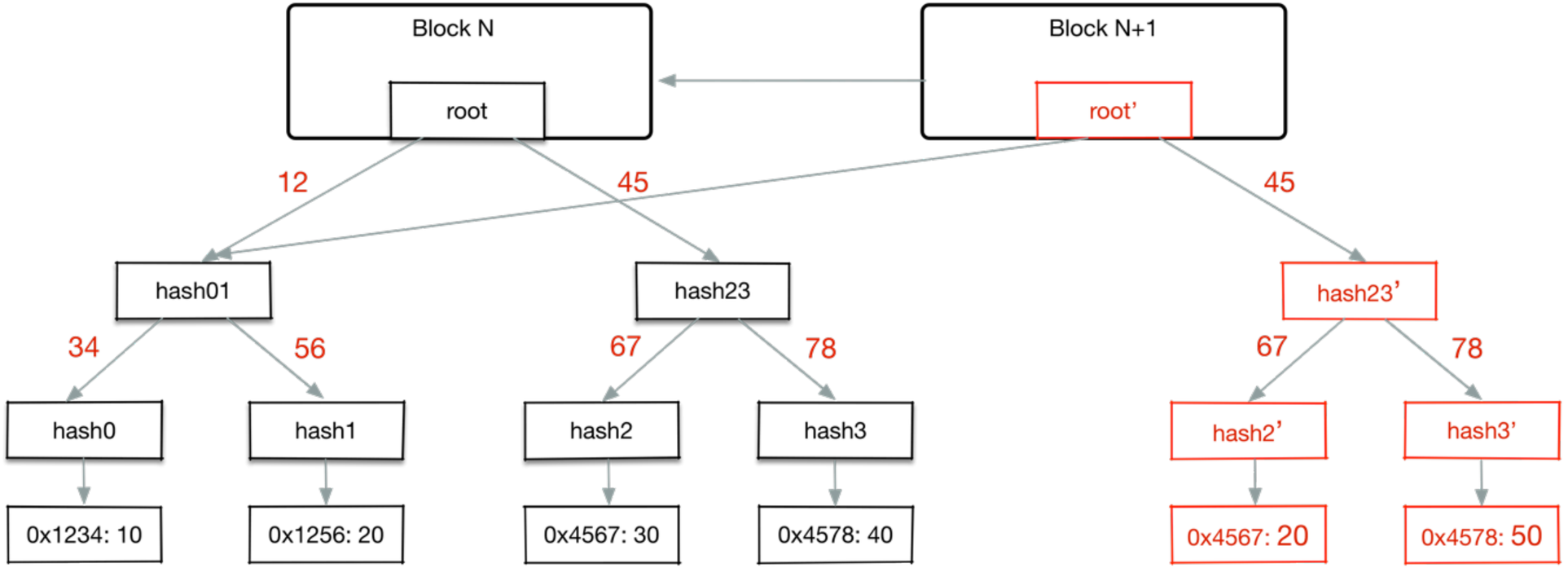
- ✓ 自下而上构造树的节点
- ✓ 修改任意节点会影响到节点对应所有父节点

MPT(Merkle Patrica Trie)



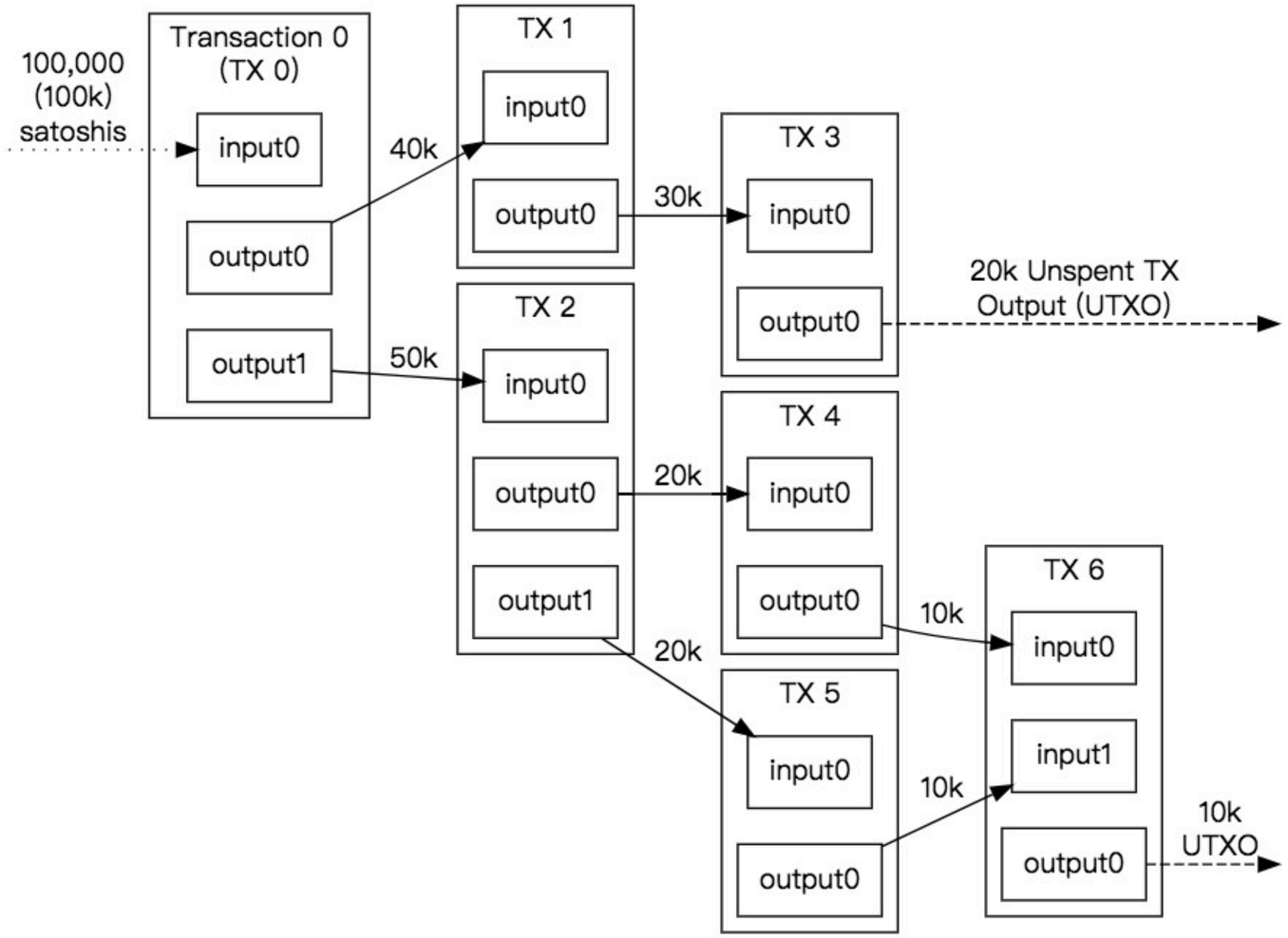
- ✓ 具备 Merkle 树一样的防篡改属性
- ✓ 路径具备索引功能

MPT(Merkle Patrica Trie)



0x4567 transfer 10 ETH to 0x4578 in block N + 1

UTXOs



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

账号模型对比

meitu 美图

UTXOs 的优点:

- ✓ 具备更高的隐私性
- ✓ 并行性更好
- ✓ 防 “双花”

Accounts 的优点:

- ✓ 节省空间
- ✓ 简单且兼容性好
- ✓ 对于智能合约更友好

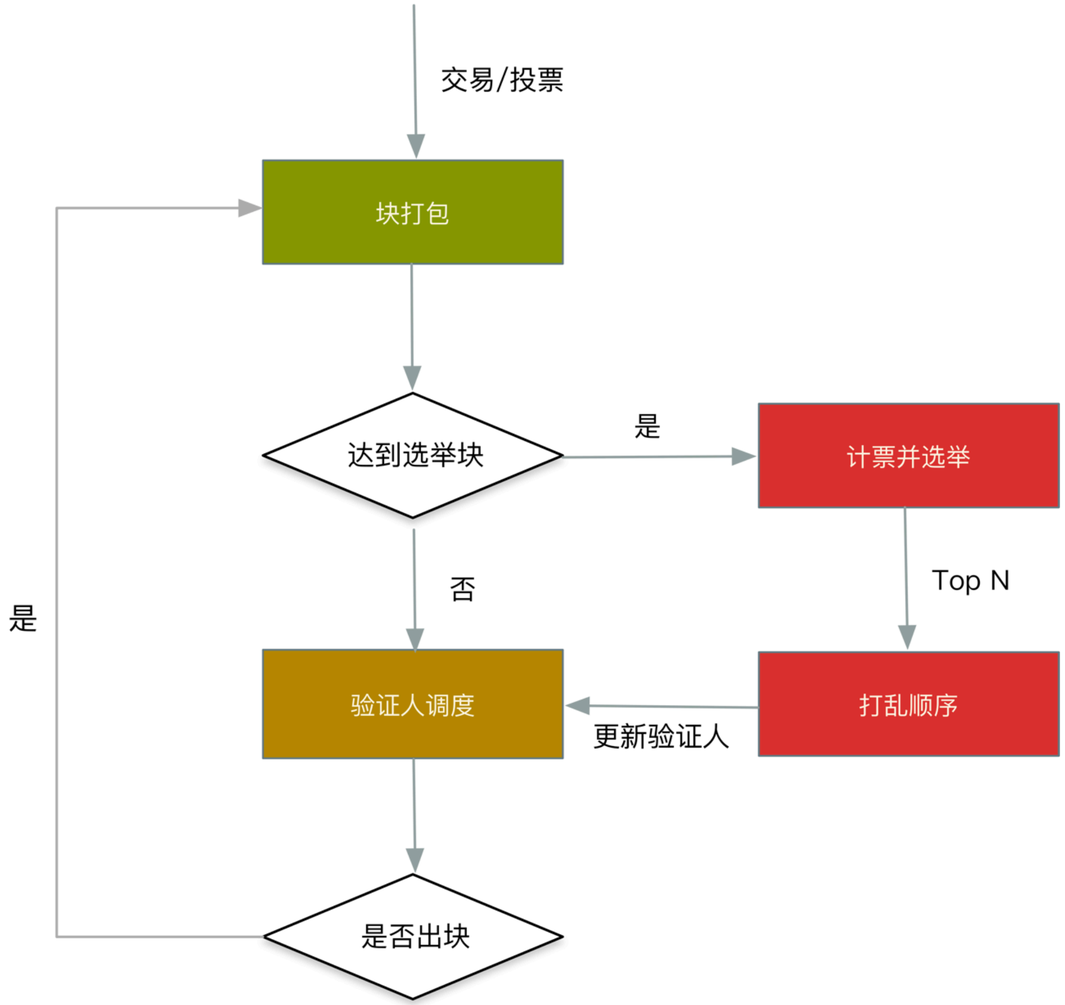
meitu 美图

- 简介
- 以太坊基础
- DPoS 实现

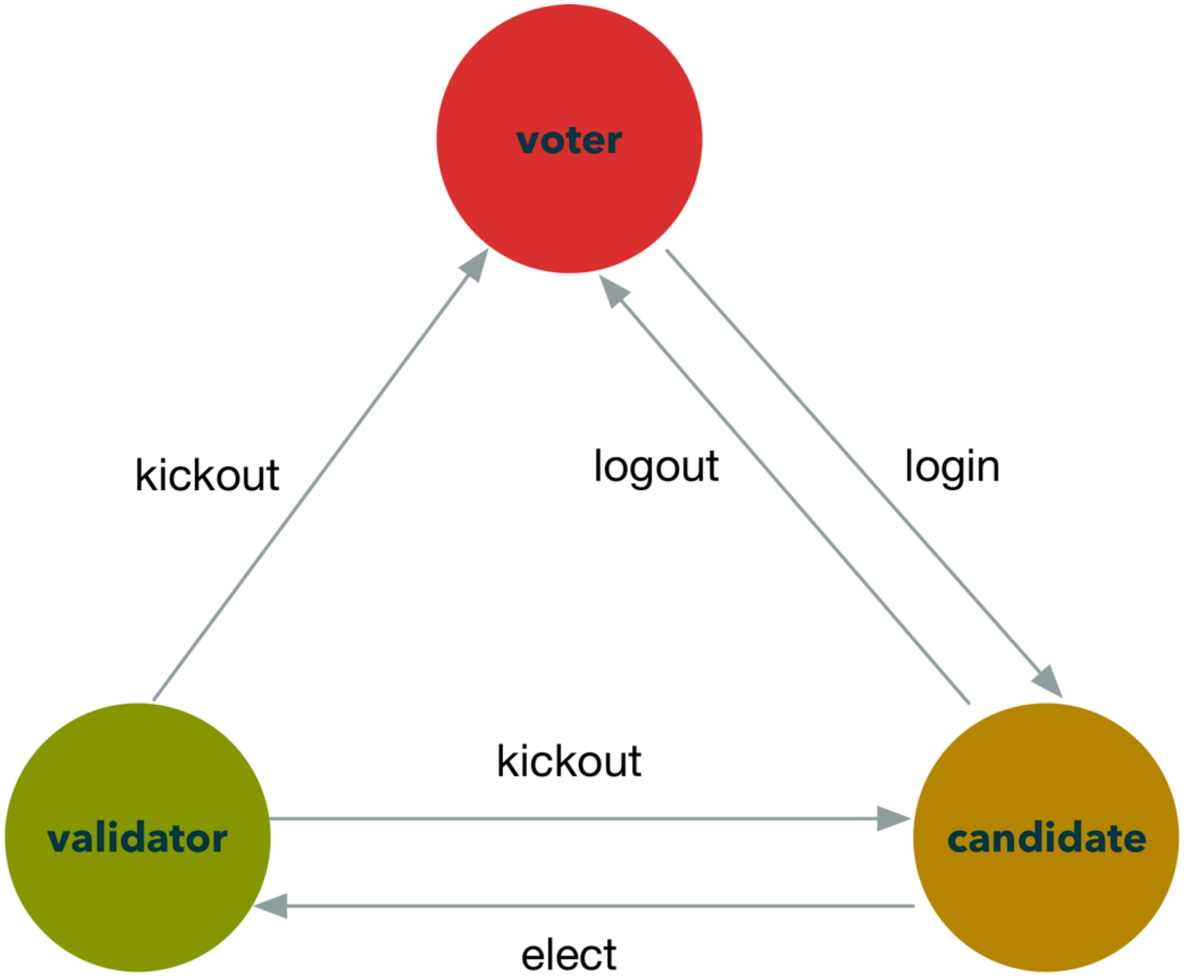
为什么选择 DPoS

- ✓ DPoS 不需要算力成本、简单且性能好, 更加适合 DAPP 落地
- ✓ 以太坊正在往 PoW + PoS 方向演进
- ✓ 为社区提供一种新的选择

整体流程



角色转换


voter
kickout
logout
login
validator
kickout
candidate
elect

投票

```
eth.SendTransaction({
    type: Vote,
    from: 0x123456778,
    to: 0x123566884,
    nonce: 1,
    gas: 100000,
    gasPrice: 10000000000,
    value: 0
})
```

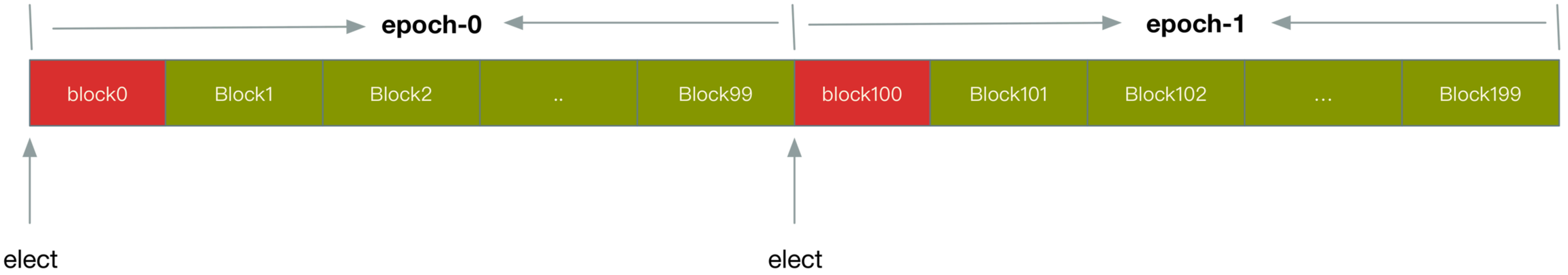
- ✓ 增加一个 **type** 字段来区分交易类型
- ✓ 除转账交易之外其他 **value** 必须为 0

存储

为了快速的选举，需要在块头存储下面几个全局状态树的 root

- ✓ EpochTrie – 记录每个周期的验证人列表
- ✓ VoteTrie – 记录投票人对应投给的候选人
- ✓ CandidateTrie – 记录所有的候选人
- ✓ DelegateTrie – 记录候选人对应所有投票人列表
- ✓ MintCntTrie – 记录每个周期候选人对应的出块数目

选举


epoch-0
epoch-1
block0
Block1
Block2
..
Block99
block100
Block101
Block102
...
Block199
elect
elect

选举

```
prevEpoch := parent.Time.Int64() / epochInterval
currentEpoch := ec.TimeStamp / epochInterval

for i := prevEpoch; i < currentEpoch; i++ {
    votes, err := ec.countVotes()
        if err != nil {
         return err
        }
    }
    sort.Sort(candidates)
  seed := int64(parent.Hash().Bytes()) + i
  r := rand.New(rand.NewSource(seed))
    for i := len(candidates) - 1; i > 0; i-- {
        j := int(r.Int31n(int32(i + 1)))
        candidates[i], candidates[j] = candidates[j], candidates[i]
    }
}
```

- ✓ 根据时间确定是到达选举块
- ✓ 使用父块的哈希值打散候选人
- ✓ 统计候选人的票数并选出 top N

调度

公式: current= block number % epoch interval % validators

比如 13 % 10 % 5 = 3， 那么就是轮到图中的 validator-2 出块

调度

```
func (ec *EpochContext) lookupValidator(now int64) (validator
common.Address, err error) {
   offset := now % epochInterval
   offset /= blockInterval
   validators, err := ec.DposContext.GetValidators()
    if err != nil {
          return common.Address{}, err
    }
     validatorSize := len(validators)
    offset %= int64(validatorSize)
    return validators[offset], nil
}
```

✓ 根据当前时间以及当前验证人来列表来确定是否轮到当前节点出块

出块

```
func (self *worker) mintBlock(now int64) error {
  err := engine.CheckValidator(self.chain.CurrentBlock(), now)
   if err != nil {
            return err
   }
  work, err := self.createNewWork()
   if err != nil {
            return err
    }
    result, err := self.engine.Seal(self.chain, work.Block, self.quitCh)
   if err != nil {
            return err
    }
}
```

- ✓ 查询出块人以及时间
- ✓ 创建块头，打包交易
- ✓ 验证人签名
- ✓ 块广播

测试

```
# 编译
$ cd go-etherem && make geth
# 通过配置文件创建第一个创世块, 第一批验证人通过配置文件指定
$ ./build/bin/geth init --datadir /path/to/datadir dpos_test_genesis.json
# 启动
$ ./build/bin/geth --datadir /path/to/datadir --keystore /path/to/keystore console
# 解锁验证人的账号
$ personal.unlock($validator, $passwd, 0)
# 设置验证人
$ miner.setValidator($validator)
# 开启挖矿
$ miner.start()
```

遇到的一些问题

meitu 美图

- ✓ 节点默认 **fast sync** 模式启动, 这种模式下无法接受新块导致同步一直失败
- ✓ 共识算法的 **单元测试** 和其他模块关系紧密, 修改共识算法需要调整大量单元测试
- ✓ 代码量大，**调试** 是个体力活

References

[1] https://github.com/ethereum/go-ethereum
[2] https://ethereum.github.io/yellowpaper/paper.pdf
[3] https://github.com/ethereum/wiki/wiki/White-Paper
[4] https://github.com/ethereum/wiki/wiki/Design-Rationale
[5] https://bitcoin.org/en/developer-guide
[6] https://eprint.iacr.org/2013/881.pdf
[7] http://bitcoin.org/bitcoin.pdf
[8] http://en.wikipedia.org/wiki/Merkle_tree
[9] http://en.wikipedia.org/wiki/Patricia_tree
[10]https://blog.golemproject.net/how-to-find-10m-by-just-reading-blockchain-6ae9d39fcd95
[11] https://ethfans.org/toya/articles/588

主办方 Geekbang 极客邦科技 InfoQ

THANKS