北京大学肖臻老师《区块链技术与应用》公开课笔记

> 以太坊数据结构篇1——状态树2，对应肖老师视频：click here 全系列笔记请见：click here 以太坊数据结构篇1——状态树1请见：click here **About Me:**点击进入我的Personal Page

> 本篇内容从视频中匿名性节44：00左右开始。主要内容为MPT数据结构的讲解。

Merkle Tree 和 Binary Tree： 区块链和链表的区别在于区块链使用哈希指针，链表使用普通指针。 同样，Merkle Tree 相比 Binary Tree，也是普通指针换成了哈希指针。

所以，以太坊系统中可如此，将所有账户组织为一个经过路径压缩和排序的Merkle Tree，其根哈希值存储于block header中。

> BTC系统中只有一个交易组成的Merkle Tree，而以太坊中有三个(三棵树)。 也就是说，在以太坊的block header中，存在有三个根哈希值。

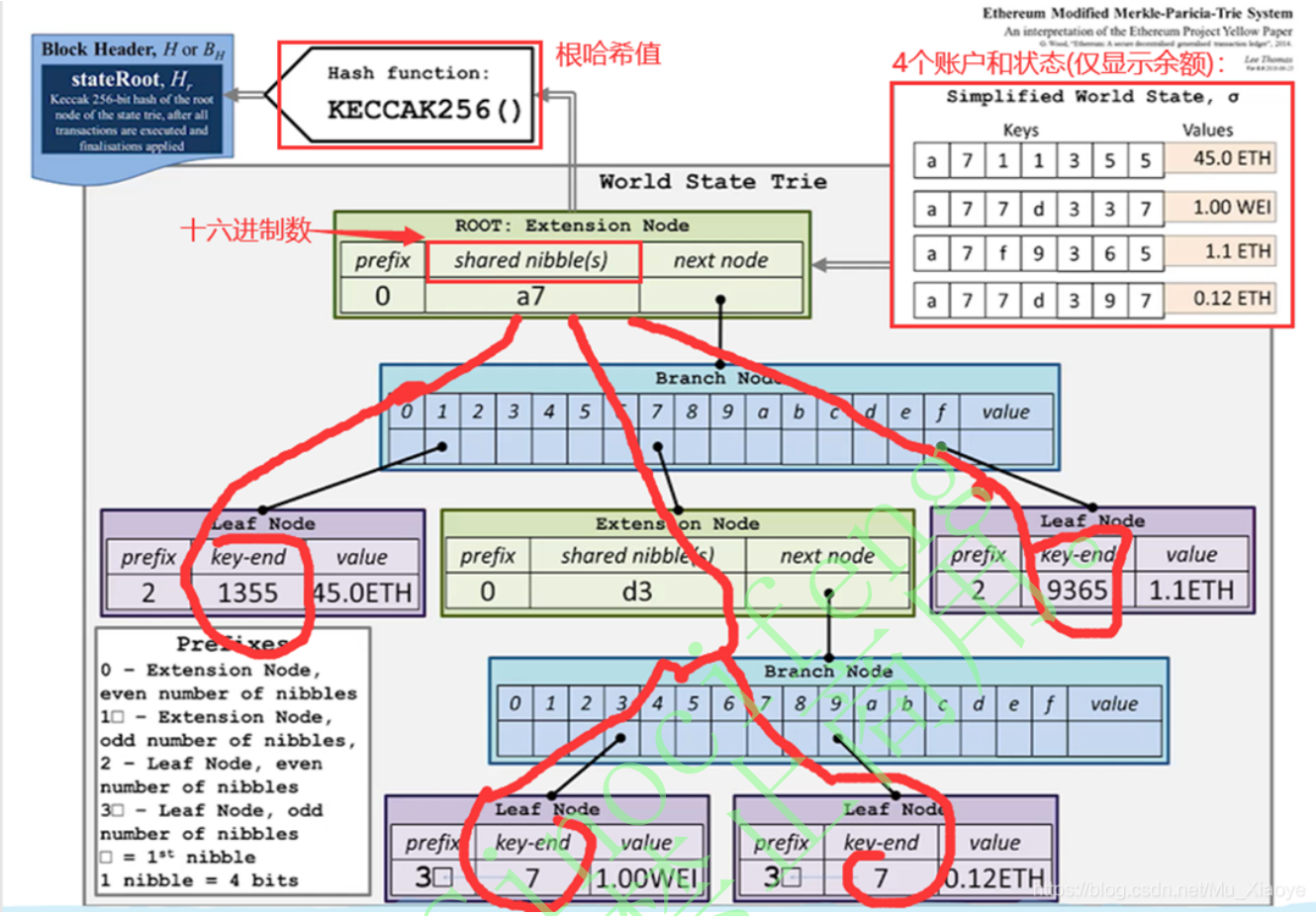
根哈希值的用处：

1. 防止篡改。
2. 提供Merkle proof，可以证明账户余额，轻节点可以进行验证。
3. 证明某个发生了交易的账户是否存在

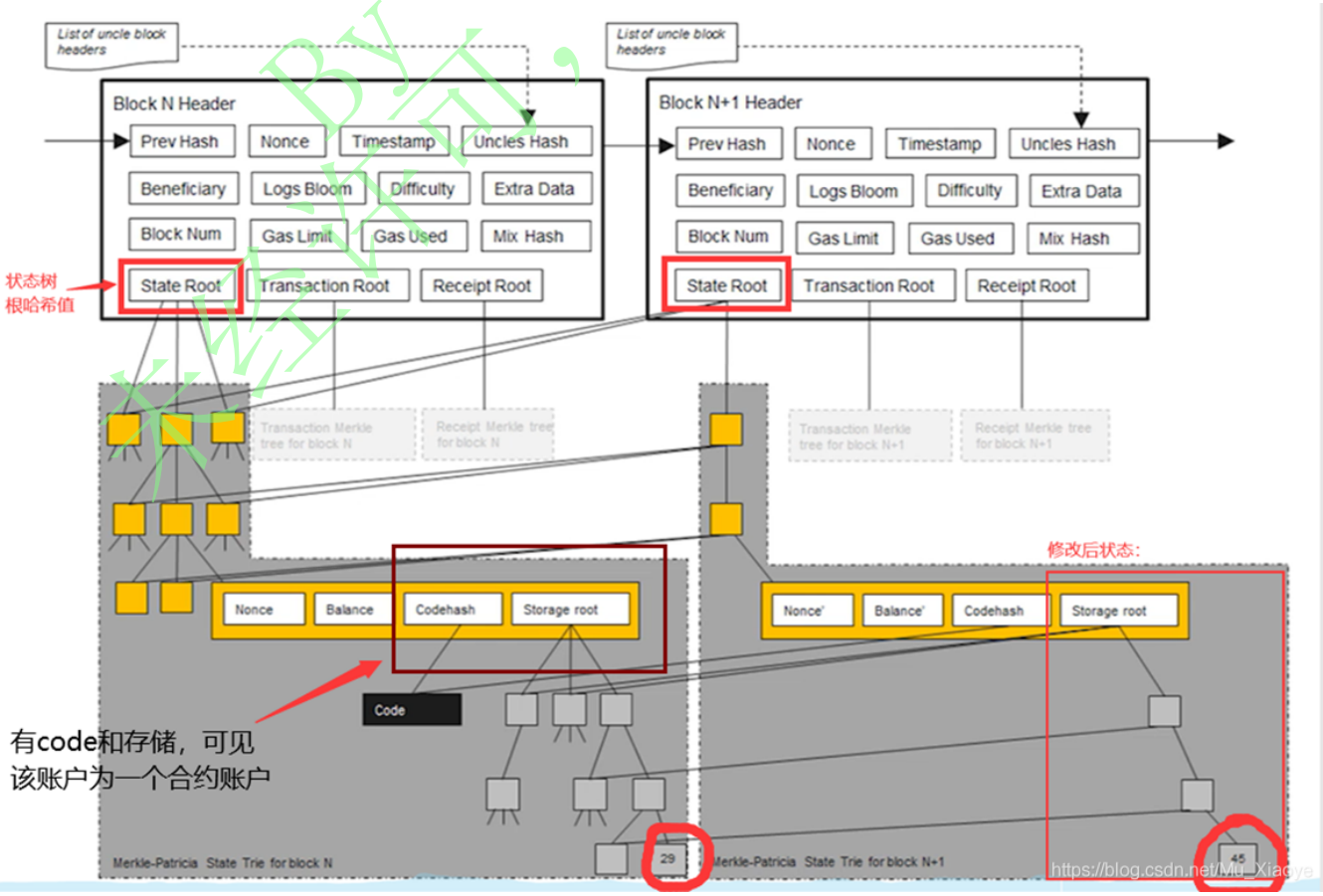
MPT(Modified Patricia tree)

> 以太坊中针对MPT(Merkle Patricia tree)进行了修改，我们称其为MPT(Modified Patricia tree)

下图为以太坊中使用的MPT结构示意图。右上角表示四个账户(为直观，显示较少)和其状态(只显示账户余额)。 (需要注意这里的指针都是哈希指针)

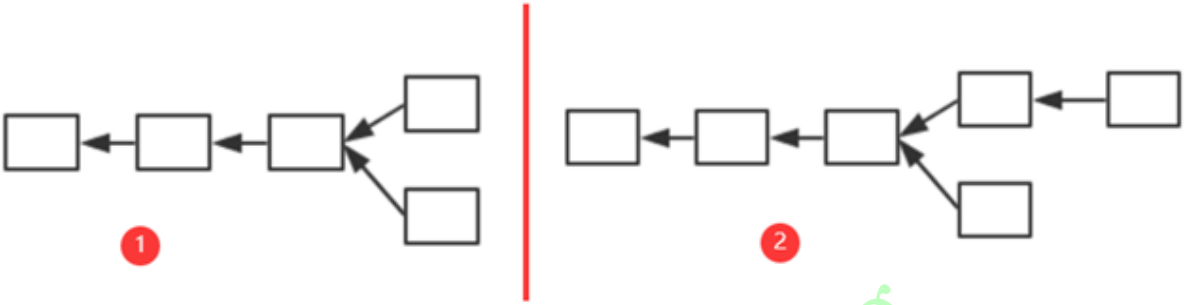


每次发布新区块，状态树中部分节点状态会改变。但改变并非在原地修改，而是新建一些分支，保留原本状态。如下图中，仅仅有新发生改变的节点才需要修改，其他未修改节点直接指向前一个区块中的对应节点。



所以，系统中全节点并非维护一棵MPT，而是每次发布新区块都要新建MPT。只不过大部分节点共享。

> 为什么要保存原本状态？为何不直接修改？ 为了便于回滚。如下1中产生分叉，而后上面节点胜出，变为2中状态。那么，下面节点中状态的修改便需要进行回滚。因此，需要维护这些历史记录。



通过代码看以太坊中的数据结构

1. block header 中的数据结构

```
// Header represents a block header in the Ethereum blockchain.
type Header struct {
        ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
        UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
        Coinbase    common.Address `json:"miner"            gencodec:"required"`
        Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
        TxHash      common.Hash    `json:"transactionsRoot" gencodec:"required"`
        ReceiptHash common.Hash    `json:"receiptsRoot"     gencodec:"required"`
        Bloom       Bloom          `json:"logsBloom"        gencodec:"required"`
        Difficulty  *big.Int       `json:"difficulty"       gencodec:"required"`
        Number      *big.Int       `json:"number"           gencodec:"required"`
        GasLimit    uint64         `json:"gasLimit"         gencodec:"required"`
        GasUsed     uint64         `json:"gasUsed"          gencodec:"required"`
        Time        *big.Int       `json:"timestamp"        gencodec:"required"`
        Extra       []byte         `json:"extraData"        gencodec:"required"`
        MixDigest   common.Hash    `json:"mixHash"          gencodec:"required"`
        Nonce       BlockNonce     `json:"nonce"            gencodec:"required"`
}
```

父区块的哈希(前一个区块的哈希值)
叔父区块的哈希(后续进行详细讲解)
矿工地址
三棵树的根哈希值
布隆过滤器(用于查询，和收据树相关)
挖矿难度
汽油费相关
区块大致产生时间
和挖矿过程相关

2. 区块结构

```
144  // Block represents an entire block in the Ethereum blockchain.
145  type Block struct {
146          header       *Header
147          uncles       []*Header
148          transactions Transactions
149
150          // caches
151          hash atomic.Value
152          size atomic.Value
153
154          // Td is used by package core to store the total difficulty
155          // of the chain up to and including the block.
156          td *big.Int
157
158          // These fields are used by package eth to track
159          // inter-peer block relay.
160          ReceivedAt   time.Time
161          ReceivedFrom interface{}
162  }
```

指向block header 的指针
指向叔父区块的指针
交易列表


```
177  // "external" block encoding. used for eth protocol, etc.
178  type extblock struct {
179          Header *Header
180          Txs    []*Transaction
181          Uncles []*Header
182  }
```

3. 区块在网上真正发布时的信息

> **最后说明** 状态树中保存Key-value对，key就是地址，而value状态通过RLP(Recursive Length Prefix，一种进行序列化的方法)编码序列号之后再进行存储。