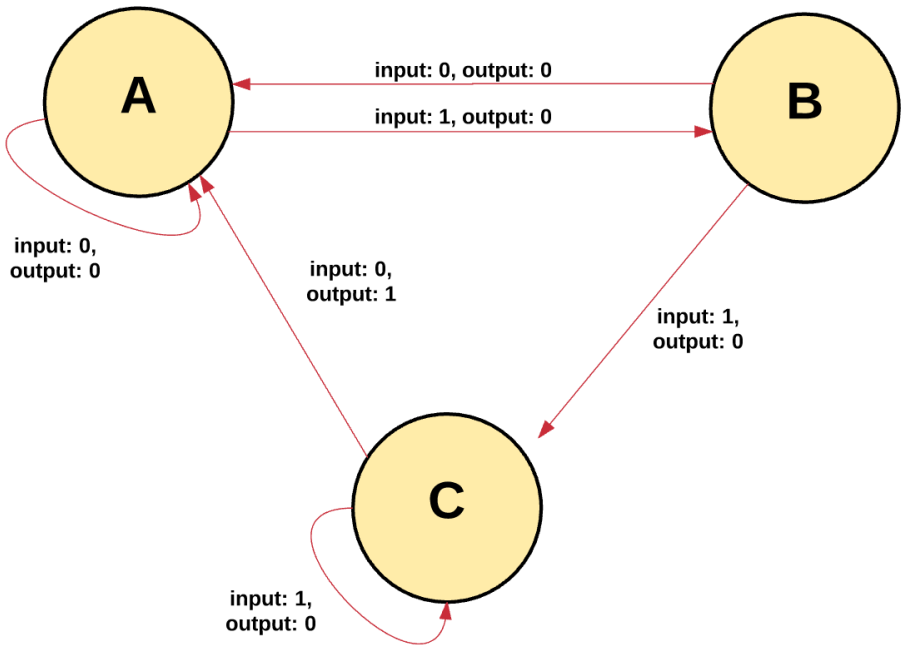
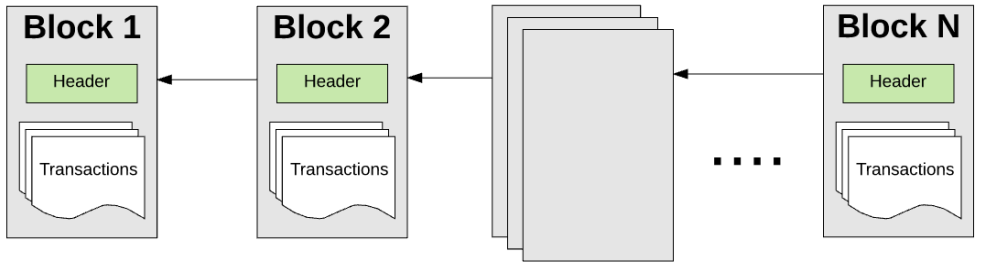
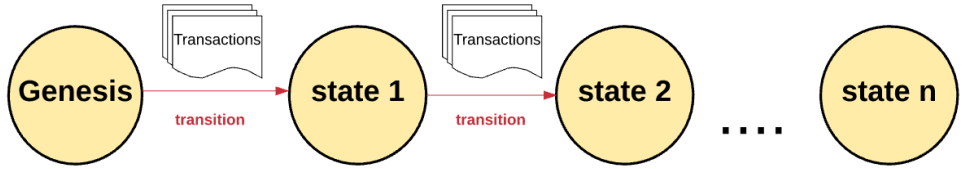
Ethereum State 存储分析

张紫荣

Ethereum: 基于交易的 State machine

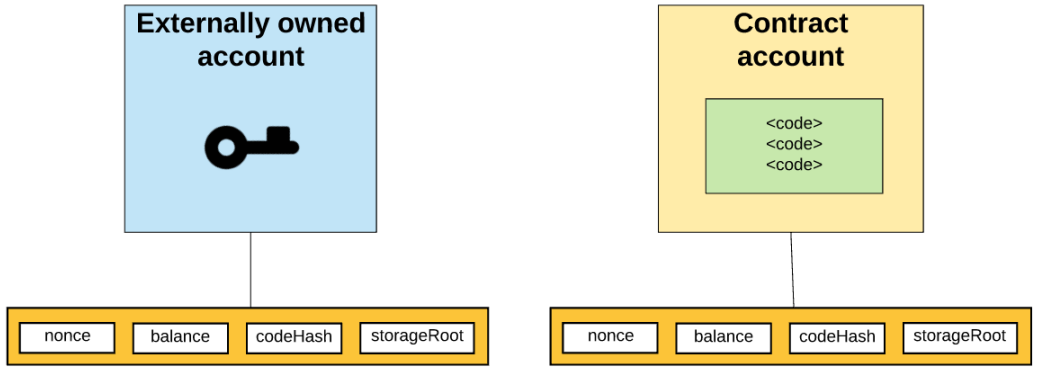






Ethereum 引入 Account 模型

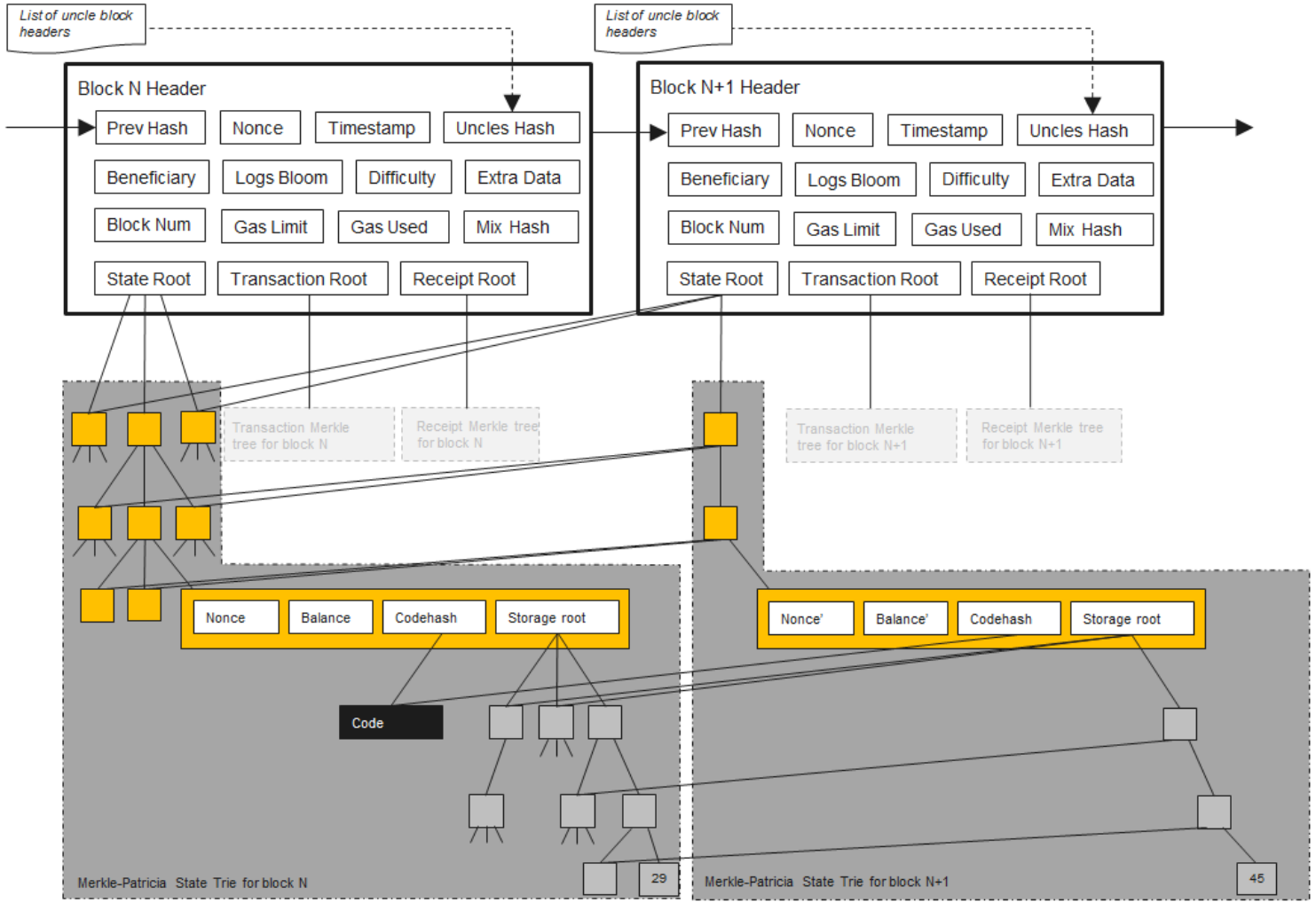
- Account 模型的特点
  - 编码友好，支持智能合约
  - 链上数据少，网络传输和存储效率高
  - 很难实现对来源的追踪和回溯
  - 可以从任一时间点同步区块状态，利于编写轻客户端




```
// Account is the Ethereum consensus representation of accounts.
// These objects are stored in the main account trie.
type Account struct {
    Nonce uint64
    Balance *big.Int
    Root common.Hash // merkle root of the storage trie
    CodeHash []byte
}
```

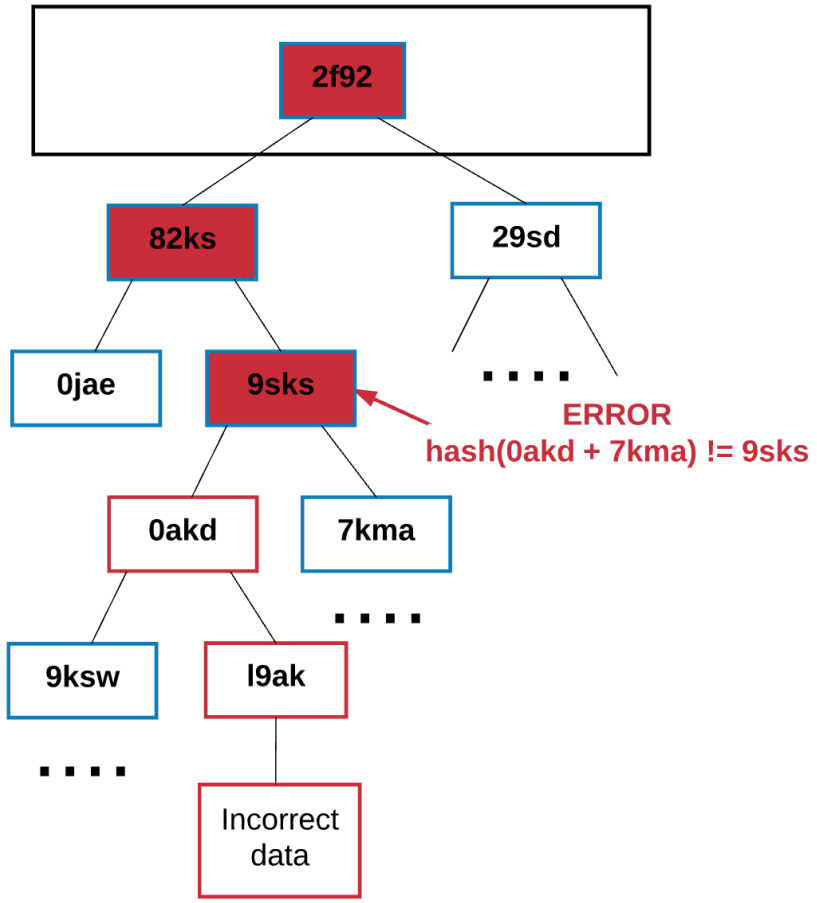
数据组织和存储

- 持久数据：Block 内所有交易执行完才能生成
  - Transaction 树：查询特定交易是否在区块中
  - Receipt 树：查询交易执行记录数据
- 易变数据：每次交易执行都会更新
  - State 树：查询 Account 相关数据



Merkle Patricia Trie

- MPT
  - Radix tree + Merkle tree
- MPT 作用
  - 存储任意长度 key/value 数据
  - 提供快速状态回滚机制
  - 提供快速计算数据集 Hash 标识
  - 提供 Merkle 证明，便于轻节点扩展
- MPT 优势
  - 易搜索、易增量计算 Root Hash、提供 Merkle 证明



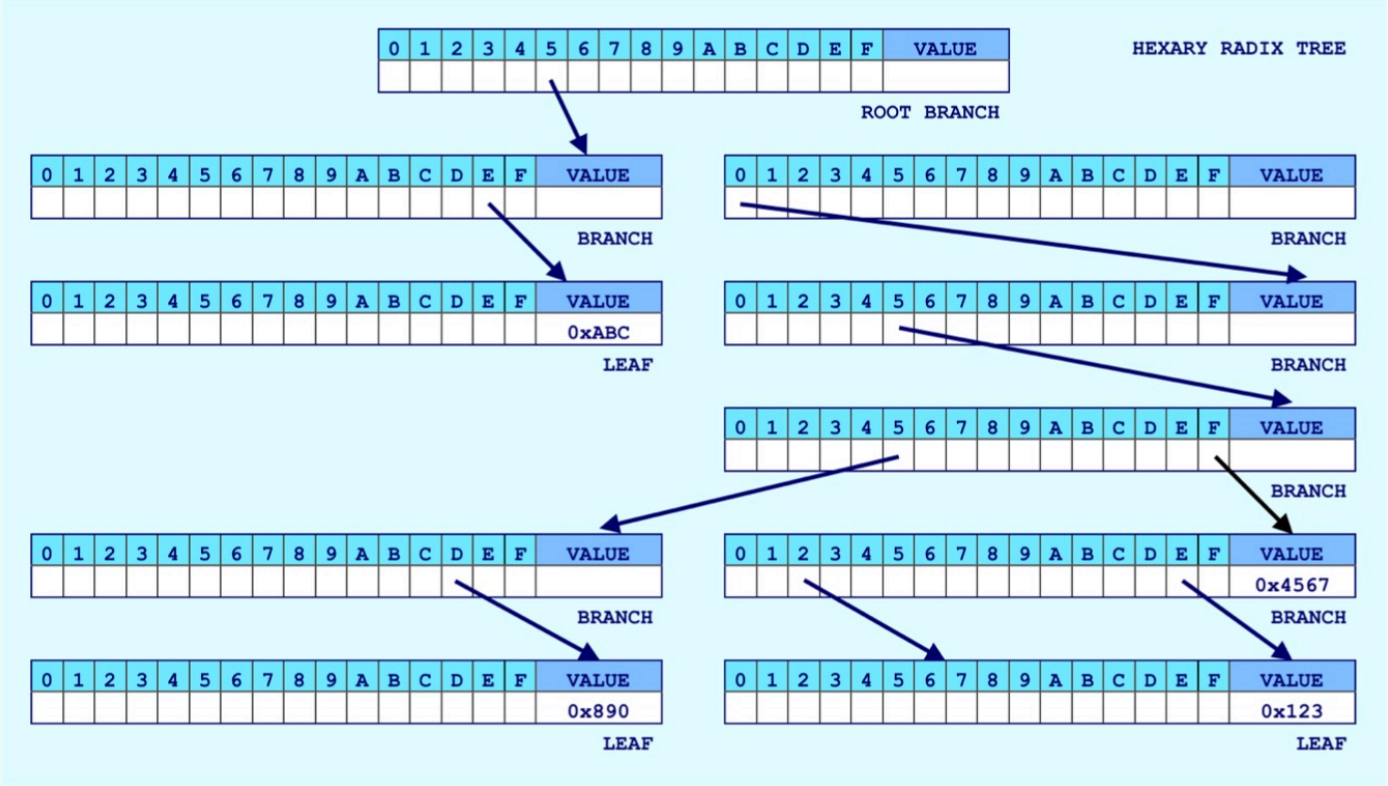
Hexary Radix Tree

- 每个 node 都有 17 个 item
- 前 16 个 item 对应十六进制的半字节 [0 … F]，存储下一个节点的指针
- 最后一个 item 是 node 存储的 value

插入下面4对 key/value：

(0xA05FE, 0x123), (0xA05F, 0x4567)

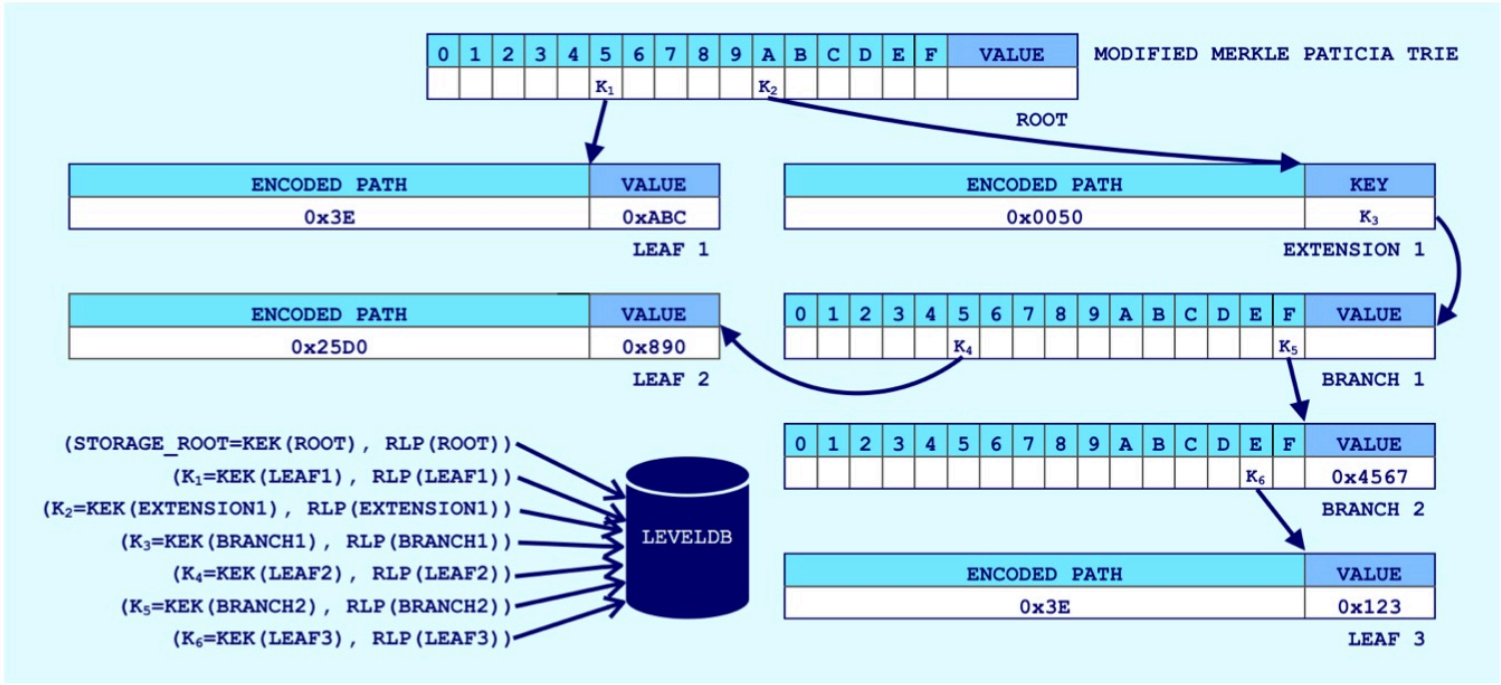
(0xA055D, 0x890), (0x5E, 0xABC)



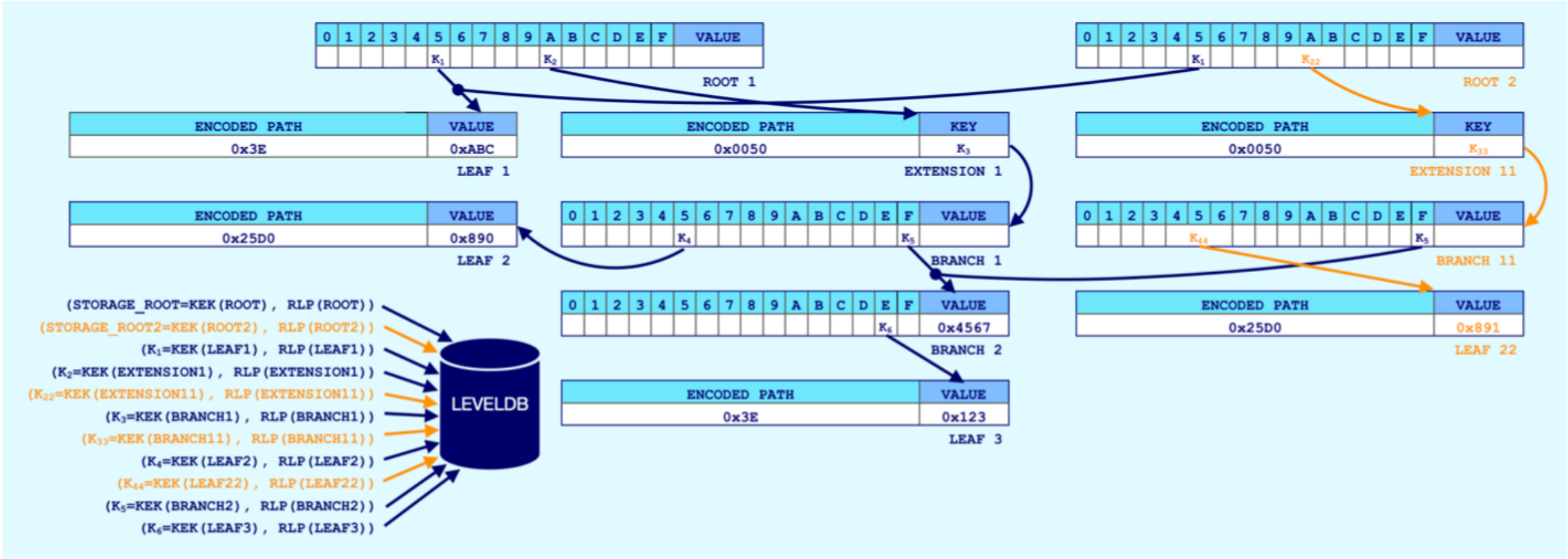
MPT 树优化存储开销

- 分支节点（没value 有分叉 ll 有 value 不管是否分叉）：保有17个item
- 扩展节点（没 value 没分叉）： encoded path + node pointer
- 叶子节点： encoded path + value

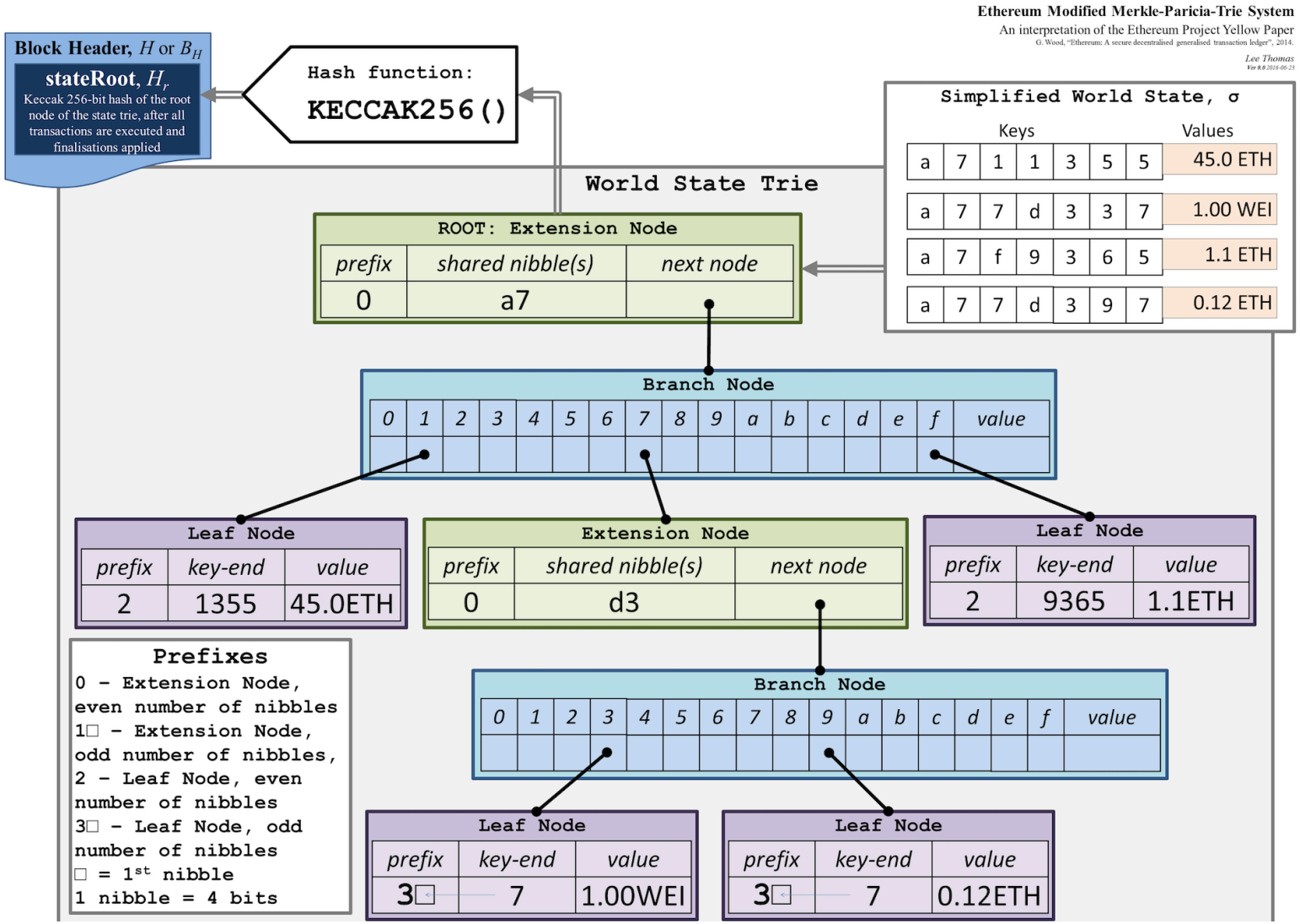
hex char	bits	node type partial	path length
0	0000	extension	even
1	0001	extension	odd
2	0010	terminating (leaf)	even
3	0011	terminating (leaf)	odd



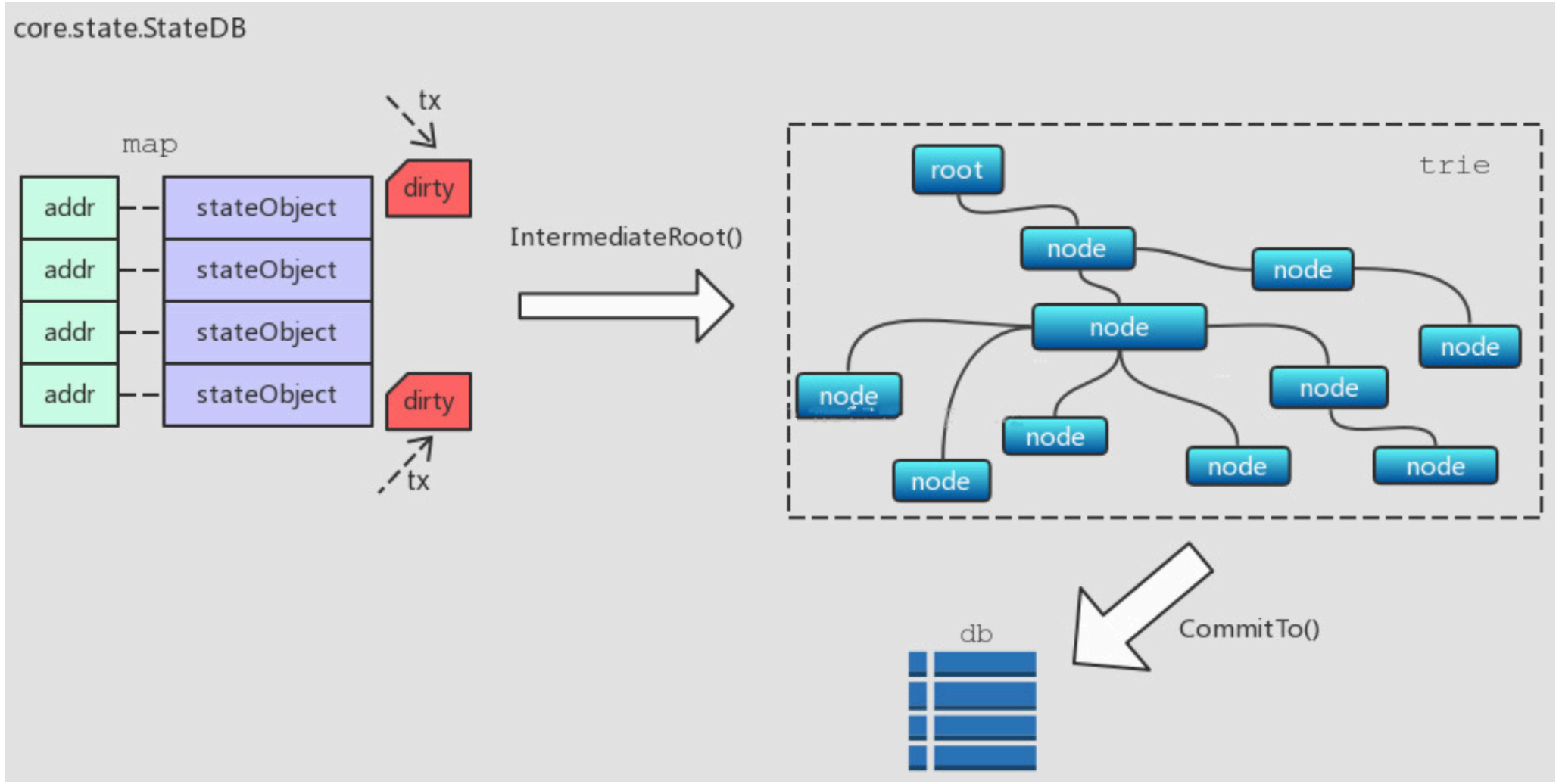
MPT 局部更新特性


0 1 2 3 4 5 6 7 8 9 A B C D E F VALUE
K1 K2
ROOT 1
ENCODED PATH VALUE
0x3E 0xABC
LEAF 1
ENCODED PATH KEY
0x0050 K3
EXTENSION 1
ENCODED PATH VALUE
0x25D0 0x890
LEAF 2
0 1 2 3 4 5 6 7 8 9 A B C D E F VALUE
K4 K5
BRANCH 1
0 1 2 3 4 5 6 7 8 9 A B C D E F VALUE
K6 0x4567
BRANCH 2
ENCODED PATH VALUE
0x3E 0x123
LEAF 3
0 1 2 3 4 5 6 7 8 9 A B C D E F VALUE
K1 K22
ROOT 2
ENCODED PATH KEY
0x0050 K33
EXTENSION 11
0 1 2 3 4 5 6 7 8 9 A B C D E F VALUE
K44 K5
BRANCH 11
ENCODED PATH VALUE
0x25D0 0x891
LEAF 22
(STORAGE_ROOT=KEK(ROOT), RLP(ROOT))
(STORAGE_ROOT2=KEK(ROOT2), RLP(ROOT2))
(K1=KEK(LEAF1), RLP(LEAF1))
(K2=KEK(EXTENSION1), RLP(EXTENSION1))
(K22=KEK(EXTENSION11), RLP(EXTENSION11))
(K3=KEK(BRANCH1), RLP(BRANCH1))
(K33=KEK(BRANCH11), RLP(BRANCH11))
(K4=KEK(LEAF2), RLP(LEAF2))
(K44=KEK(LEAF22), RLP(LEAF22))
(K5=KEK(BRANCH2), RLP(BRANCH2))
(K6=KEK(LEAF3), RLP(LEAF3))
LEVELDB

完整的 MPT


Ethereum Modified Merkle-Paricia-Trie System
An interpretation of the Ethereum Project Yellow Paper
G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", 2014.
Lee Thomas
Ver 0.0 2016-06-23
Block Header, H or B_H
stateRoot, H_r
Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied
Hash function:
KECCAK256()
Simplified World State, σ
Keys
Values
a 7 1 1 3 5 5 45.0 ETH
a 7 7 d 3 3 7 1.00 WEI
a 7 f 9 3 6 5 1.1 ETH
a 7 7 d 3 9 7 0.12 ETH
World State Trie
ROOT: Extension Node
prefix
shared nibble(s)
next node
0
a7
Branch Node
0 1 2 3 4 5 6 7 8 9 a b c d e f value
Leaf Node
prefix
key-end
value
2
1355
45.0ETH
Extension Node
prefix
shared nibble(s)
next node
0
d3
Leaf Node
prefix
key-end
value
2
9365
1.1ETH
Prefixes
0 - Extension Node, even number of nibbles
1□ - Extension Node, odd number of nibbles,
2 - Leaf Node, even number of nibbles
3□ - Leaf Node, odd number of nibbles
□ = 1st nibble
1 nibble = 4 bits
Branch Node
0 1 2 3 4 5 6 7 8 9 a b c d e f value
Leaf Node
prefix
key-end
value
3□
7
1.00WEI
Leaf Node
prefix
key-end
value
3□
7
0.12ETH

State 存储结构


core.state.StateDB
map
addr
addr
addr
addr
stateObject
stateObject
stateObject
stateObject
tx
dirty
dirty
tx
IntermediateRoot()
root
node
node
node
node
node
node
node
node
node
node
trie
db
CommitTo()

基于 Testnet 分析 State 存储

```
[[redacted] test]# tree chain/
chain/
├── geth
│   ├── chaindata
│   │   ├── 000118.ldb
│   │   ├── 000124.ldb
│   │   ├── 000146.log
│   │   ├── CURRENT
│   │   ├── CURRENT.bak
│   │   ├── LOCK
│   │   ├── LOG
│   │   ├── LOG.old
│   │   └── MANIFEST-000147
│   ├── ethash
│   │   ├── cache-R23-0000000000000000
│   │   └── cache-R23-290decd9548b62a8
│   ├── lightchaindata
│   │   ├── 000001.log
│   │   ├── CURRENT
│   │   ├── LOCK
│   │   ├── LOG
│   │   └── MANIFEST-000000
│   ├── LOCK
│   ├── nodekey
│   └── transactions.rlp
```

```
> eth.blockNumber
278
> eth.getBlock(278)
{
  difficulty: 131072,
  extraData: "0xd783010703846765746887676f312e382e33856c696e7578",
  gasLimit: 4120464,
  gasUsed: 42000,
  hash: "0xfe9ea7f7ddabd1f088627f703645cf82ad0aa58261a62c59addb74bfb1d53274",
  logsBloom: "0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
  miner: "0x1ed4accf9ec0c9a1fa85e2c11d78e8a91af7bab1",
  mixHash: "0xc6e1d4dc9e662894489873c9f9361ece3774ce8b21ea227a8a34cf2ea62cedf2",
  nonce: "0x5ce1589dc617b683",
  number: 278,
  parentHash: "0x2d4a076061397a9e91787ab30d4e494bc136974115b89987f9fdc0faef715c59",
  receiptsRoot: "0xc76ba09e63f53b0ff661f371ae81600f8d211778789c6dc428e9b8cf689b862d",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  size: 762,
  stateRoot: "0x68e96375dd1b202a0b919439d54a96882b4c7d29a3f9f5c69a616817d361fb7e",
  timestamp: 1536388220,
  totalDifficulty: 37557995,
  transactions: ["0x32b8478a08516c7db6d9f505e945a35b7309156949597bcfc724497d38abf0db", "0xe90993098854dcdd57e5a67ed9fdcb8698b40587038d8c049405c60f6d439b09"],
  transactionsRoot: "0x03d91520a77c2e8cbe7363919f415d17d721811c3a52cc4df233e76eb9d10174",
  uncles: []
}
```

30 行代码搞定

```go
package main

import (
    "encoding/hex"
    "fmt"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/core/state"
    "github.com/ethereum/go-ethereum/ethdb"
    "os"
)

func error_exit(err string) {
    fmt.Println(err)
    os.Exit(1)
}

func main() {
    path := "/home/dc2-user/test/chain/geth/chaindata/"
    chainDb, err := ethdb.NewLDBDatabase(path, 0, 0)
    if err != nil {
        error_exit("Open leveldb failed")
    }
    defer chainDb.Close()

    root := "68e96375dd1b202a0b919439d54a96882b4c7d29a3f9f5c69a616817d361fb7e"
    key, err := hex.DecodeString(root)
    if err != nil {
        error_exit("Decode root error")
    }

    state, err := state.New(common.BytesToHash(key), state.NewDatabase(chainDb))
    if err != nil {
        error_exit(fmt.Sprintf("Could not create new state:", err))
    }
    fmt.Printf("%s\n", state.Dump())
}
```

运行结果分析

```
[ read_leveldb]# ./read_leveldb
{
    "root": "68e96375dd1b202a0b919439d54a96882b4c7d29a3f9f5c69a616817d361fb7e",
    "accounts": {
        "1ed4accf9ec0c9a1fa85e2c11d78e8a91af7bab1": {
            "balance": "25000360792000000000",
            "nonce": 1,
            "root": "56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
            "codeHash": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",
            "code": "",
            "storage": {}
        },
        "5552a3bffdd55d497cfaa6742e3c84cd882e2010": {
            "balance": "4000000000000000000",
            "nonce": 2,
            "root": "56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
            "codeHash": "c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",
            "code": "",
            "storage": {}
        },
        "6f5154afd23e2db8750c1e9fe18f6814fc9e80a5": {
            "balance": "0",
            "nonce": 1,
            "root": "56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
            "codeHash": "7511cd9feb97906b9f6b3f7a4c81c75bdc6dc19e0dc1d437966a6f4a18f63b5c",
            "code": "6080604052600436106103f576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff168063165c4a16146044575b600080fd5b348015604f57600080fd5b506076600480360381019080803590602001909291908035906020019092919050505060608c565b60405180828152602001915050604051809103905f35b60008183029050929150505600a165627a7a72305820c638628899b66c161fb3bb11b201b3f9019d87ec542fb95b7c28027ed0e118bb0029",
            "storage": {}
        },
```

参考资料

- https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369
- https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/
- https://hackernoon.com/getting-deep-into-ethereum-how-data-is-stored-in-ethereum-e3f669d96033
- https://github.com/ethereum/wiki/wiki/RLP
- https://github.com/ethereum/wiki/wiki/Patricia-Tree
- https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture
- https://github.com/FISCO-BCOS/Wiki/tree/master/%E6%B5%85%E8%B0%88Ethereum%E7%9A%84%E5%AD%98%E5%82%A8
- https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/
- http://wanderer.github.io/ethereum/nodejs/code/2014/05/21/using-ethereums-tries-with-node/

附录： RLP编码

```
def rlp_encode(input):
    if isinstance(input,str):
        if len(input) == 1 and ord(input) < 0x80: return input
        else: return encode_length(len(input), 0x80) + input
    elif isinstance(input,list):
        output = ''
        for item in input: output += rlp_encode(item)
        return encode_length(len(output), 0xc0) + output

def encode_length(L,offset):
    if L < 56:
         return chr(L + offset)
    elif L < 256**8:
         BL = to_binary(L)
         return chr(len(BL) + offset + 55) + BL
    else:
         raise Exception("input too long")

def to_binary(x):
    if x == 0:
        return ''
    else:
        return to_binary(int(x / 256)) + chr(x % 256)
```

Examples

The string "dog" = [0x83, 'd', 'o', 'g']

The list ["cat", "dog"] = `[ 0xc8, 0x83, 'c', 'a', 't', 0x83, 'd', 'o', 'g' ]`

The empty string ('null') = `[ 0x80 ]`

The empty list = `[ 0xc0 ]`

The integer 0 = `[ 0x80 ]`

The encoded integer 0 ('\x00') = `[ 0x00 ]`

The encoded integer 15 ('\x0f') = `[ 0x0f ]`

The encoded integer 1024 ('\x04\x00') = `[ 0x82, 0x04, 0x00 ]`

The set theoretical representation of three, `[ [], [[]], [ [], [[]] ] ] = [ 0xc7, 0xc0, 0xc1, 0xc0, 0xc3, 0xc0, 0xc1, 0xc0 ]`

The string "Lorem ipsum dolor sit amet, consectetur adipisicing elit" = `[ 0xb8, 0x38, 'L', 'o', 'r', 'e', 'm', ' ', ... , 'e', 'l', 'i', 't' ]`