深度探索以太坊智能合约

钟瑞仙

2018.11.22

钟瑞仙 Rolong

- 早期从事游戏开发，DDoS防御
- 以太坊DAPP开发者
- 以太坊底层实现的研究和应用
- 现任以太零研发团队技术总监

大纲

1、以太坊账户介绍

2、交易数据里data字段的编码规则

3、智能合约属性的索引和存储

3.1、简单属性的索引规则

3.2、map类型的元素索引

3.3、结构体的索引规则

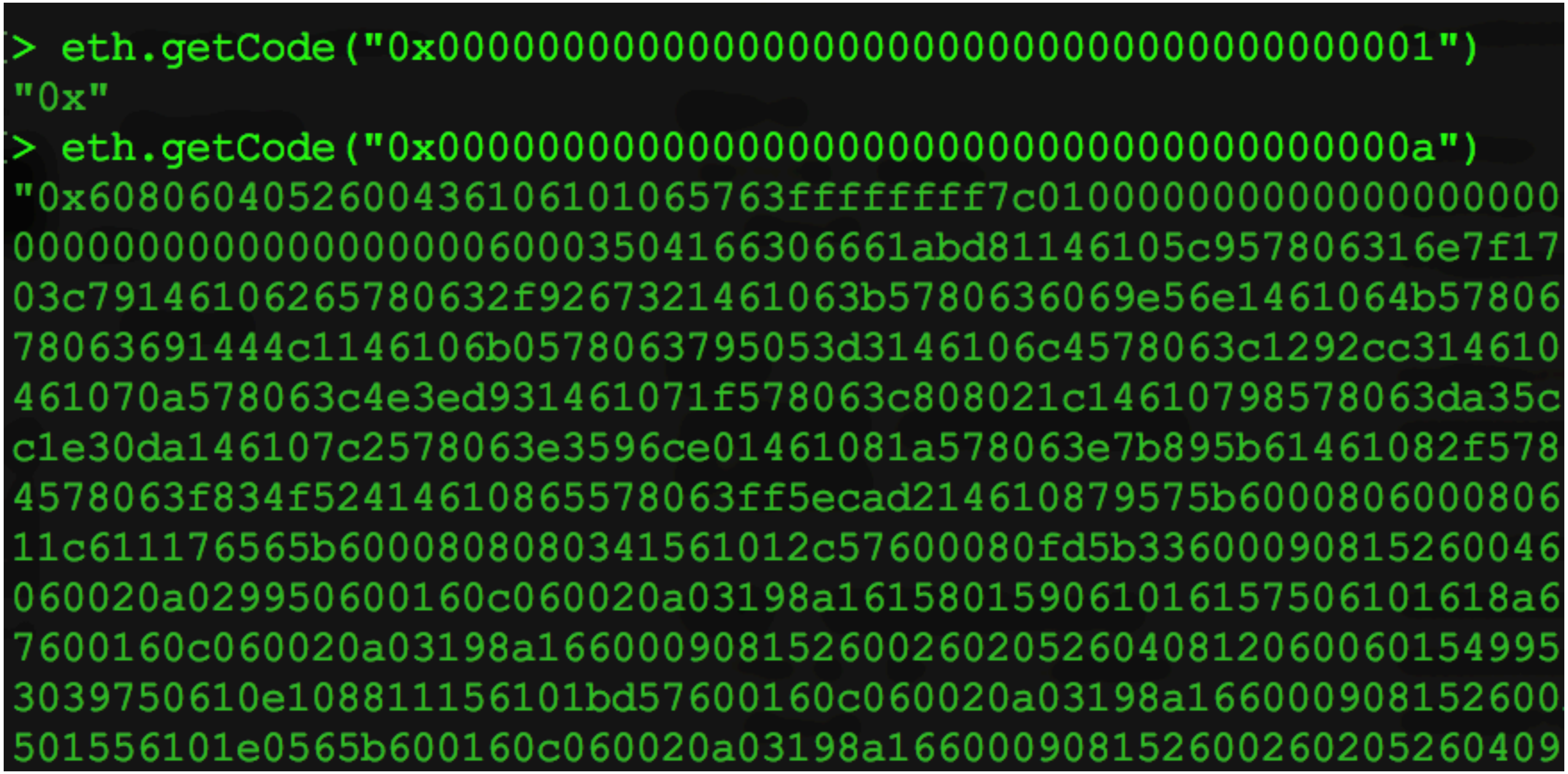
4、预编译合约介绍及汇编调用方法

1、以太坊账户介绍

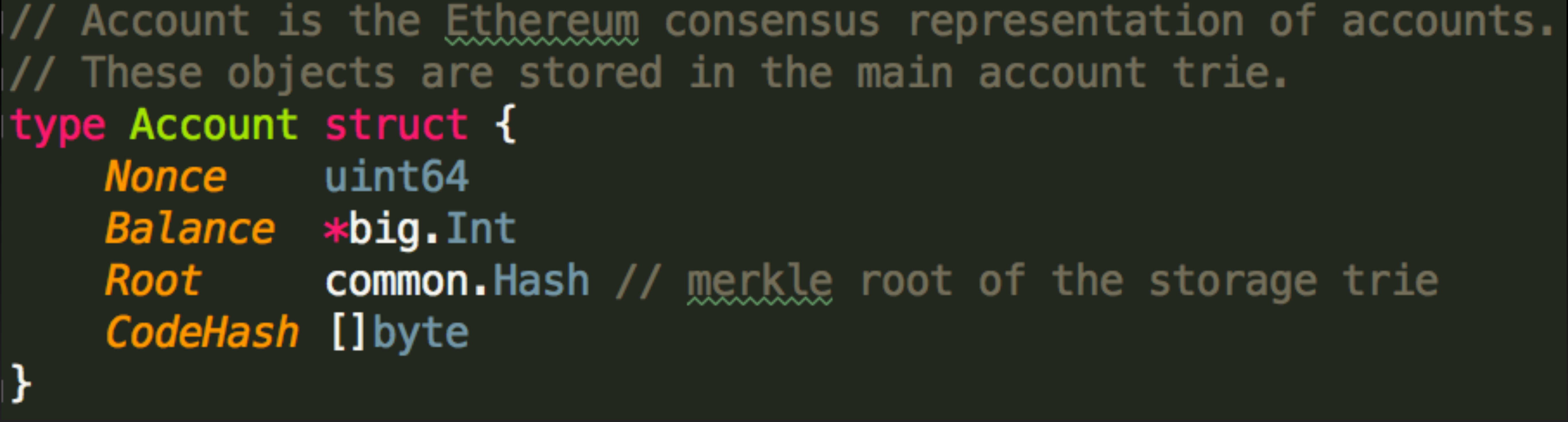
如何判断一个地址是否为合约地址?

eth.getCode("0x0000000000000000000000000000000000000001")

eth.getCode("0x000000000000000000000000000000000000000a")



```
> eth.getCode("0x0000000000000000000000000000000000000001")
"0x"
> eth.getCode("0x000000000000000000000000000000000000000a")
"0x6080604052600436106101065763ffffffff7c010000000000000000000
000000000000000000060003504166306661abd81146105c957806316e7f17
03c79146106265780632f9267321461063b5780636069e56e1461064b57806
78063691444c1146106b0578063795053d3146106c4578063c1292cc314610
461070a578063c4e3ed931461071f578063c808021c14610798578063da35c
c1e30da146107c2578063e3596ce01461081a578063e7b895b61461082f578
4578063f834f52414610865578063ff5ecad214610879575b6000806000806
11c611176565b6000808080341561012c57600080fd5b33600090815260046
060020a029950600160c060020a03198a161580159061016157506101618a6
7600160c060020a03198a16600090815260026020526040812060060154995
3039750610e108811156101bd57600160c060020a03198a166000908152600
501556101e0565b600160c060020a03198a166000908152600260205260409
```



```
// 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
}
```

Root => 保存智能合约属性状态的Merkle树

CodeHash => 保存智能合约静态代码

非合约账户：

Root = 0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421

CodeHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470

- 合约地址的生成规则：

  data = rlp.Encode(创建者地址+当前交易的Nonce值)
  hash = Keccak256(data)
  取hash的后20字节作为合约地址

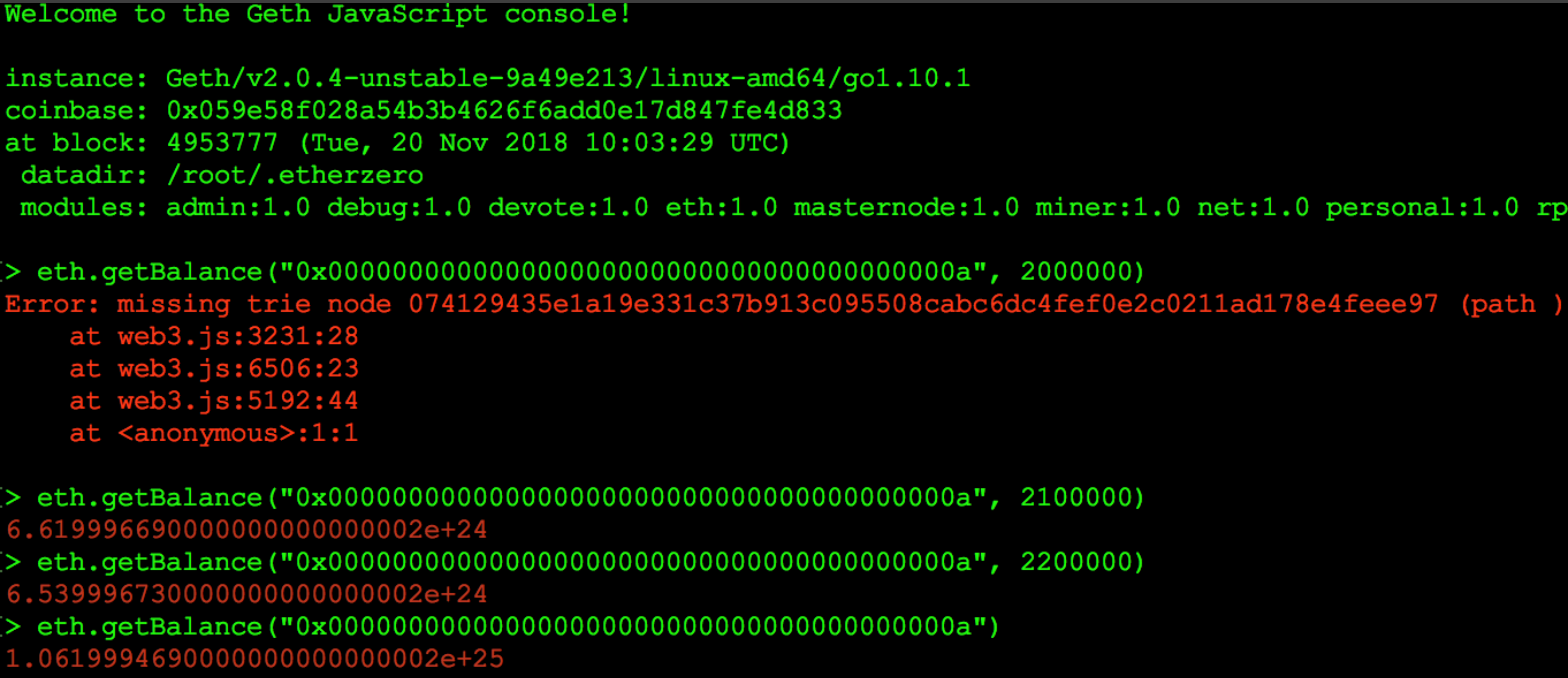
- 个人地址的生成规则：

  由私钥推导出公钥，再由公钥推导出地址

- 两种地址的区别：

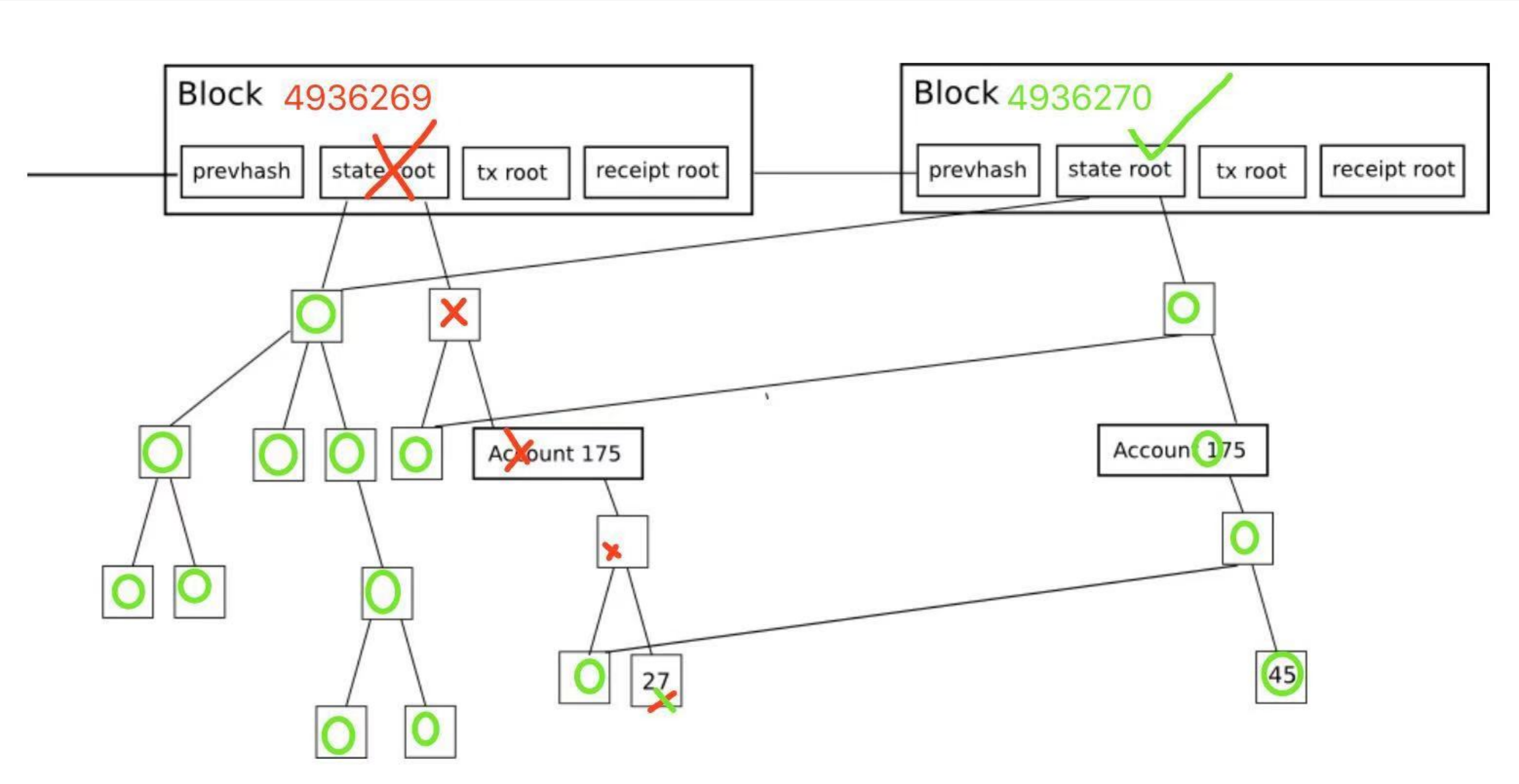
  合约地址和椭圆曲线加密无关，不会生成雷同的地址就可以
  个人地址必须由私钥推导

快速同步的节点缺失了那些数据?



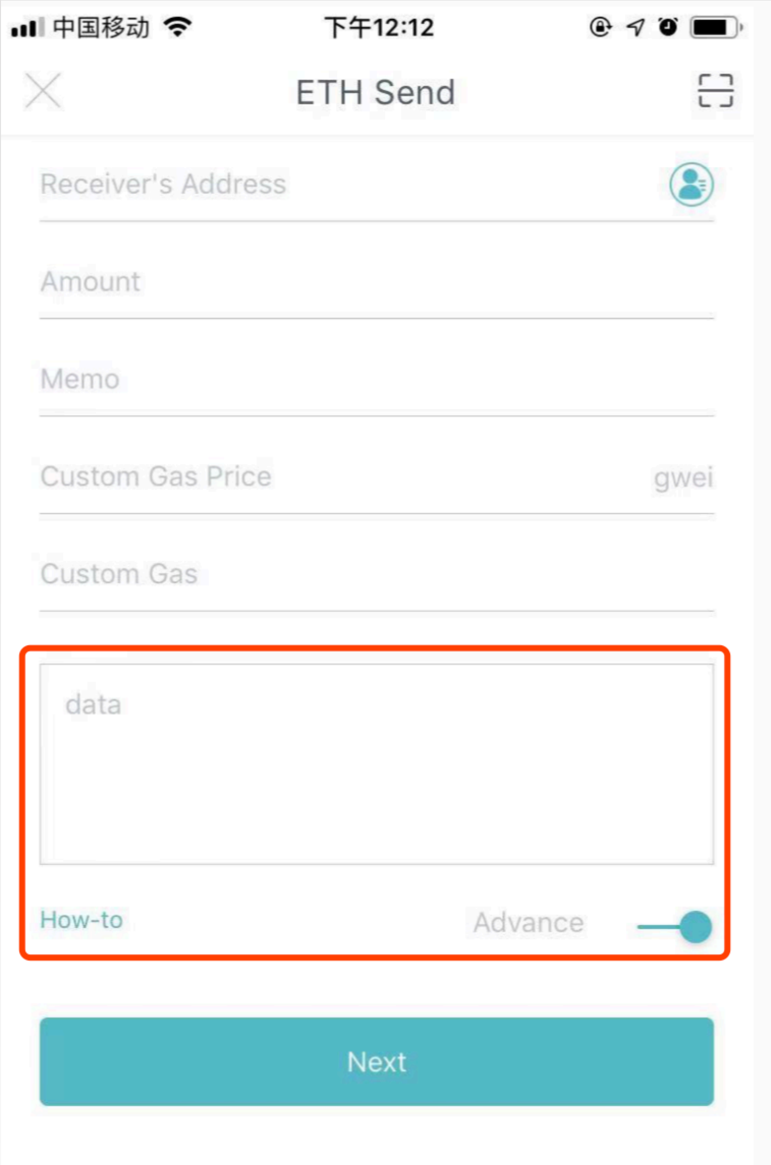
```
Welcome to the Geth JavaScript console!

instance: Geth/v2.0.4-unstable-9a49e213/linux-amd64/go1.10.1
coinbase: 0x059e58f028a54b3b4626f6add0e17d847fe4d833
at block: 4953777 (Tue, 20 Nov 2018 10:03:29 UTC)
 datadir: /root/.etherzero
 modules: admin:1.0 debug:1.0 devote:1.0 eth:1.0 masternode:1.0 miner:1.0 net:1.0 personal:1.0 rp

> eth.getBalance("0x000000000000000000000000000000000000000a", 2000000)
Error: missing trie node 074129435e1a19e331c37b913c095508cabc6dc4fef0e2c0211ad178e4feee97 (path )
    at web3.js:3231:28
    at web3.js:6506:23
    at web3.js:5192:44
    at <anonymous>:1:1

> eth.getBalance("0x000000000000000000000000000000000000000a", 2100000)
6.619996690000000000000002e+24
> eth.getBalance("0x000000000000000000000000000000000000000a", 2200000)
6.539996730000000000000002e+24
> eth.getBalance("0x000000000000000000000000000000000000000a")
1.0619994690000000000000002e+25
```

merkle树索引示意图



2、交易数据里data字段的编码规则

Transaction Data 的用途是什么?
用于调用智能合约非匿名方法



中国移动 下午12:12

ETH Send

Receiver's Address

Amount

Memo

Custom Gas Price gwei

Custom Gas

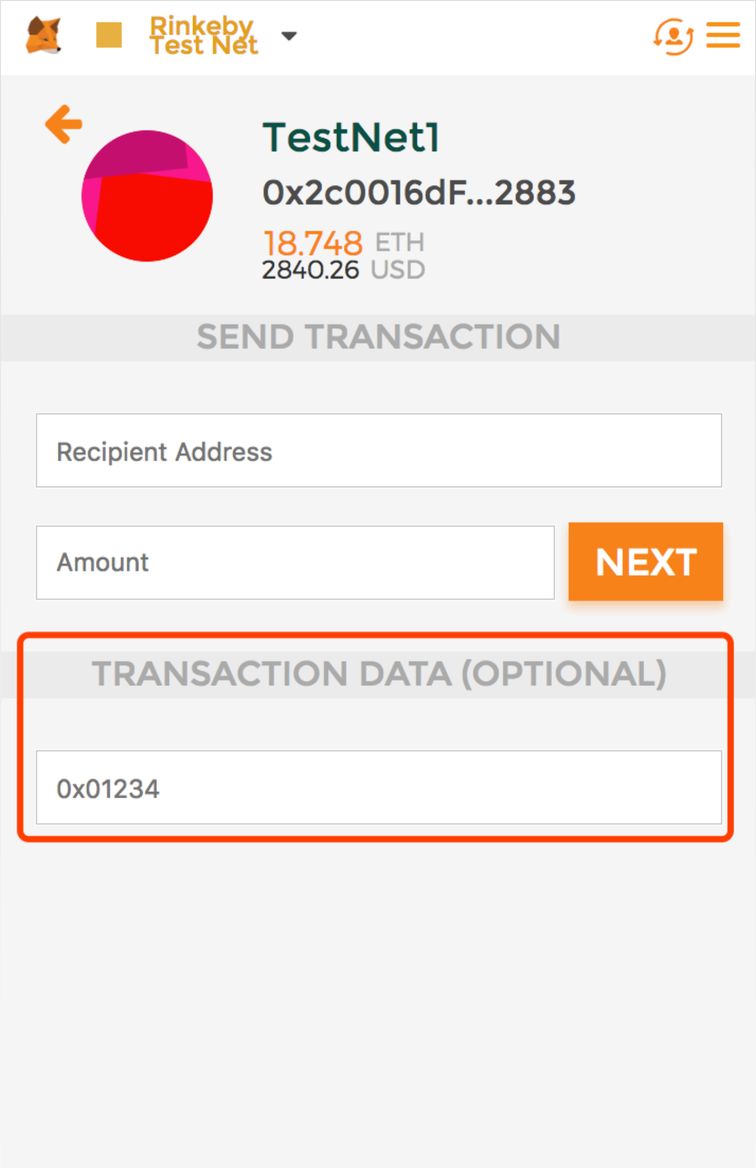
data

How-to

Advance

Next

imToken



Rinkeby Test Net

TestNet1

0x2c0016dF...2883

18.748 ETH

2840.26 USD

SEND TRANSACTION

Recipient Address

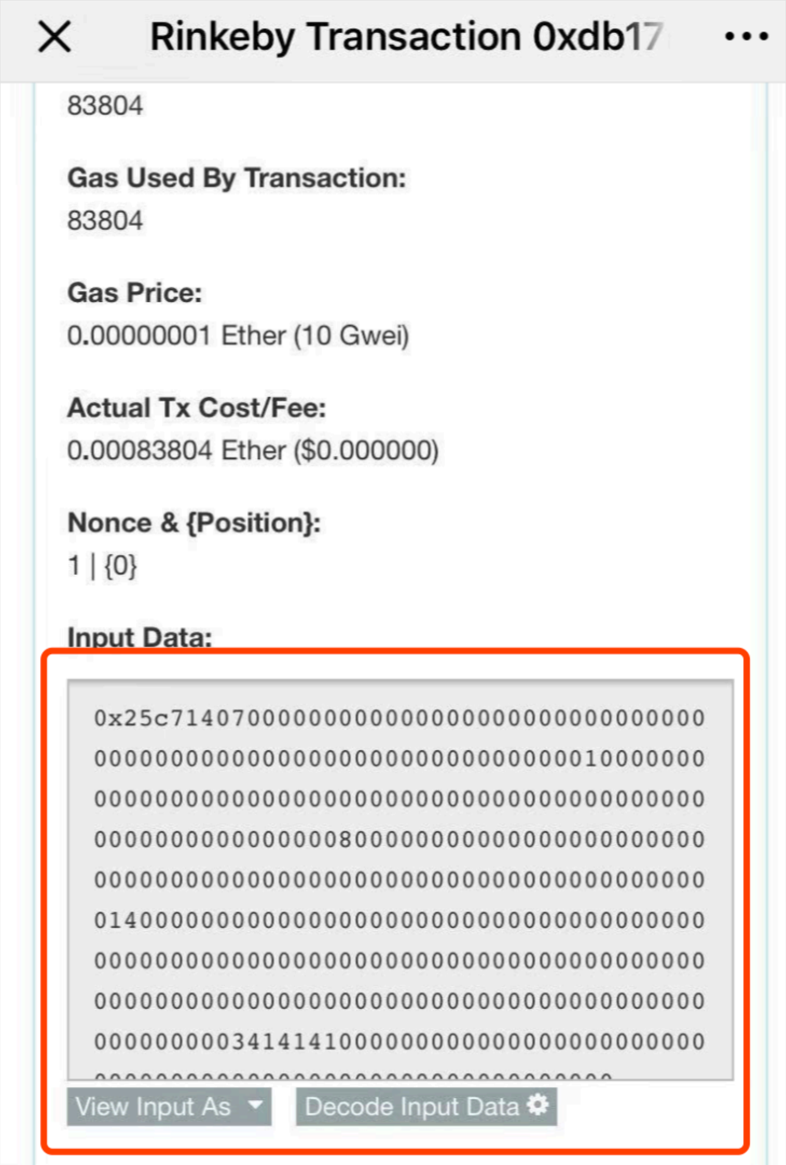
Amount

NEXT

TRANSACTION DATA (OPTIONAL)

0x01234

Metamask



Rinkeby Transaction 0xdb17

83804

Gas Used By Transaction:
83804

Gas Price:
0.00000001 Ether (10 Gwei)

Actual Tx Cost/Fee:
0.00083804 Ether ($0.000000)

Nonce & {Position}:
1 | {0}

Input Data:

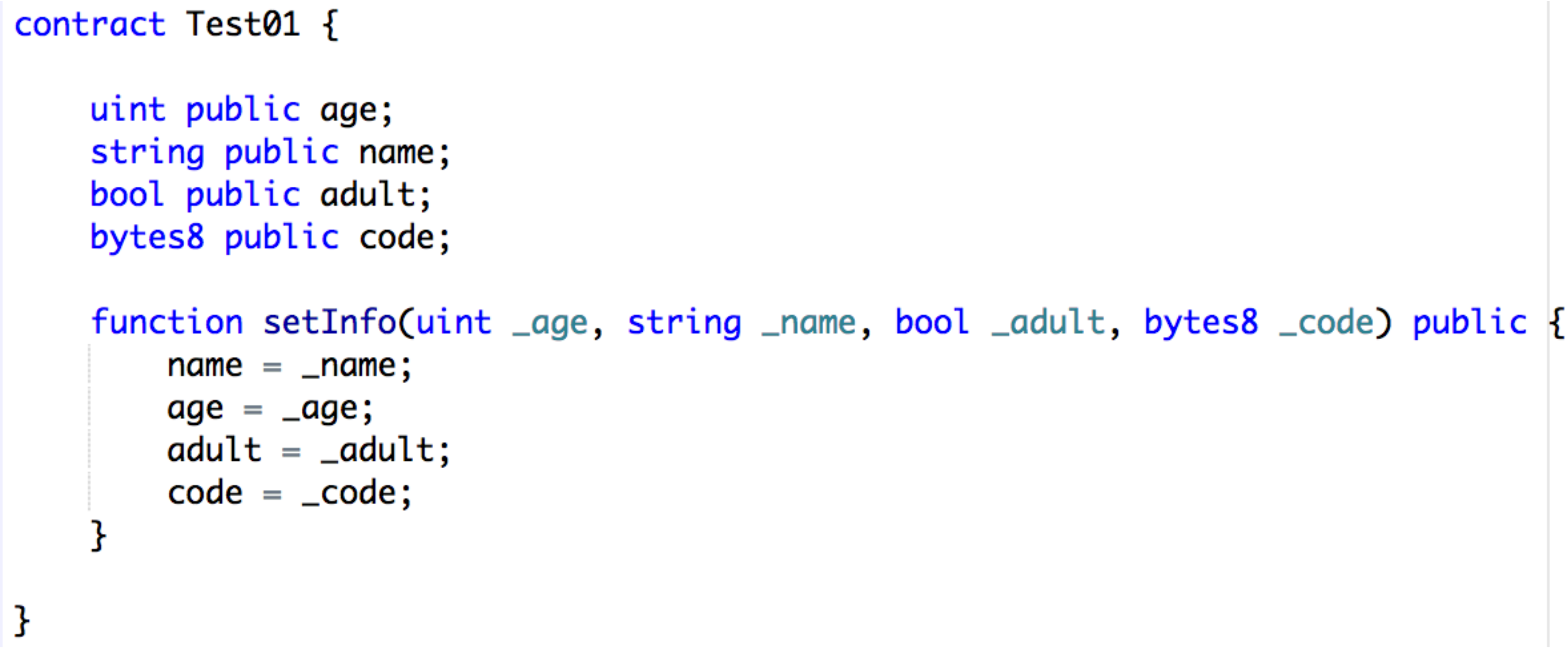
```
0x25c71407000000000000000000000000000000
0000000000000000000000000000000010000000
0000000000000000000000000000000000000000
0000000000000000800000000000000000000000
0000000000000000000000000000000000000000
0140000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000003414141000000000000000000000000
0000000000000000000000000000000000
```

View Input As

Decode Input Data

Etherscan

实例1

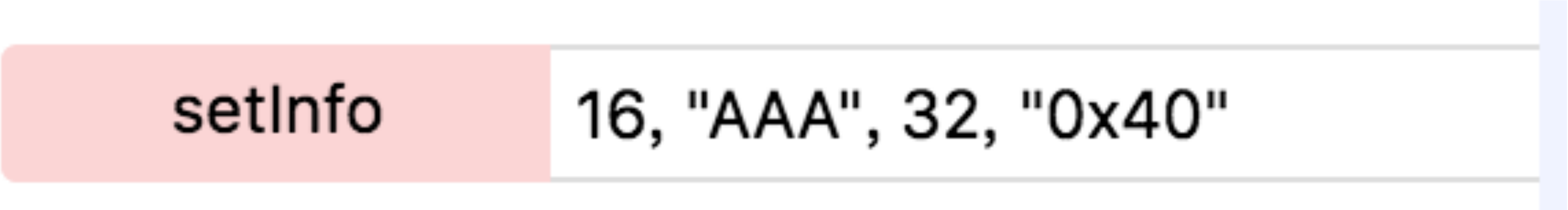


```
contract Test01 {

    uint public age;
    string public name;
    bool public adult;
    bytes8 public code;

    function setInfo(uint _age, string _name, bool _adult, bytes8 _code) public {
        name = _name;
        age = _age;
        adult = _adult;
        code = _code;
    }

}
```

https://rinkeby.etherscan.io/address/0x4bf92828c655b006a52cf476130a295959025cbe

开发者调用模式：

通过Remix发起交易调用setInfo方法：setInfo(16, "AAA", 32, "0x40")



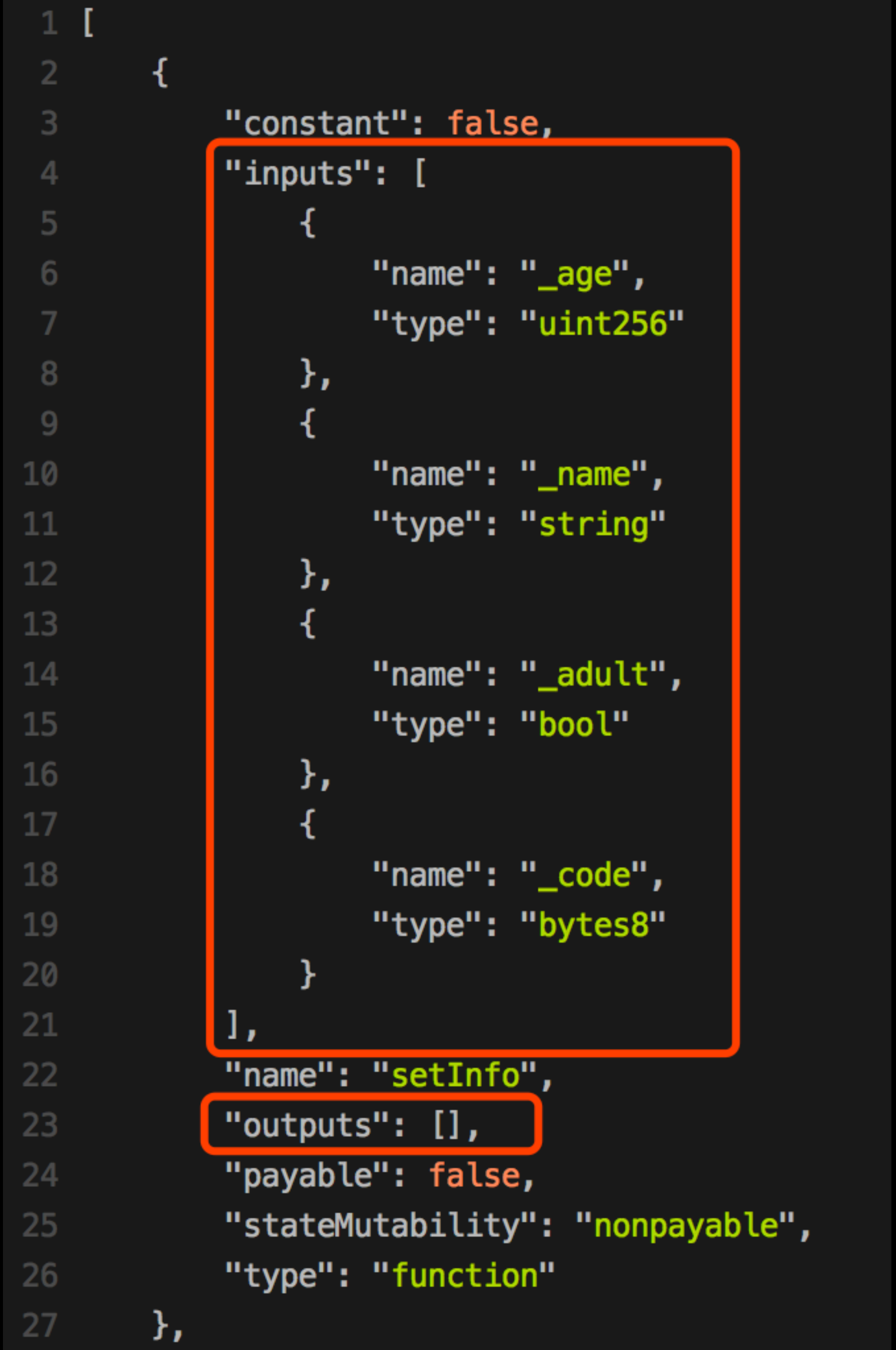
16, "AAA", 32, "0x40"

Dapp用户调用模式：

复制以下DATA粘贴到钱包发送交易

```
0x25c714070000000000000000000000000000000000000000000000000000
00000000001000000000000000000000000000000000000000000000000000
00000000000080000000000000000000000000000000000000000000000000
00000000000000014000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000
00000000000000000003414141000000000000000000000000000000000000
0000000000000000000000
```

https://rinkeby.etherscan.io/tx/0xdb175d4a5b3847bf964a4ea36e3d59d12b029e0777088c92436b45b5d6662a9e



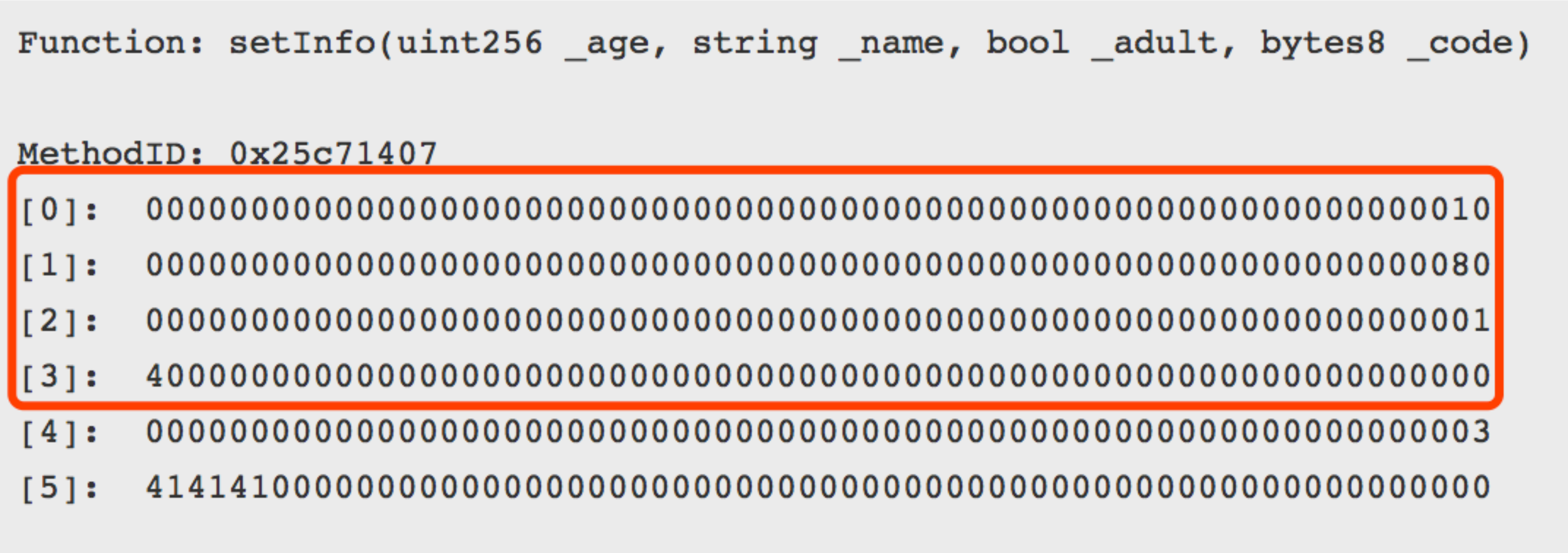
```
[
    {
        "constant": false,
        "inputs": [
            {
                "name": "_age",
                "type": "uint256"
            },
            {
                "name": "_name",
                "type": "string"
            },
            {
                "name": "_adult",
                "type": "bool"
            },
            {
                "name": "_code",
                "type": "bytes8"
            }
        ],
        "name": "setInfo",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    },
```

ABI
描述智能合约接口

- 输入数据的打包规则
- 返回数据的解包规则
- 其他属性。。。

编码规则

0x + 方法ID(4字节) + 参数(32字节对齐)



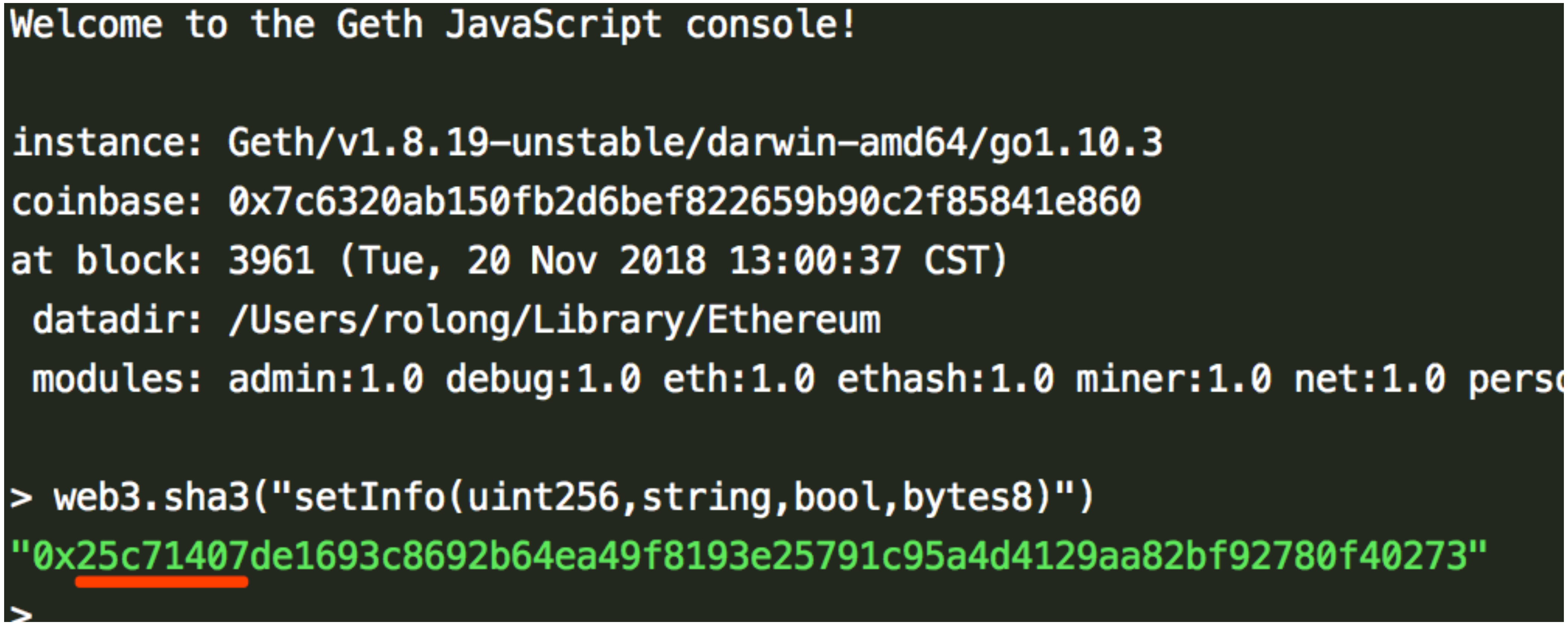
```
Function: setInfo(uint256 _age, string _name, bool _adult, bytes8 _code)

MethodID: 0x25c71407
[0]:  0000000000000000000000000000000000000000000000000000000000000010
[1]:  0000000000000000000000000000000000000000000000000000000000000080
[2]:  0000000000000000000000000000000000000000000000000000000000000001
[3]:  4000000000000000000000000000000000000000000000000000000000000000
[4]:  0000000000000000000000000000000000000000000000000000000000000003
[5]:  4141410000000000000000000000000000000000000000000000000000000000
```

注：1个字为2个HEX字符

方法ID运算规则

web3.sha3("setInfo(uint256,string,bool,bytes8)")

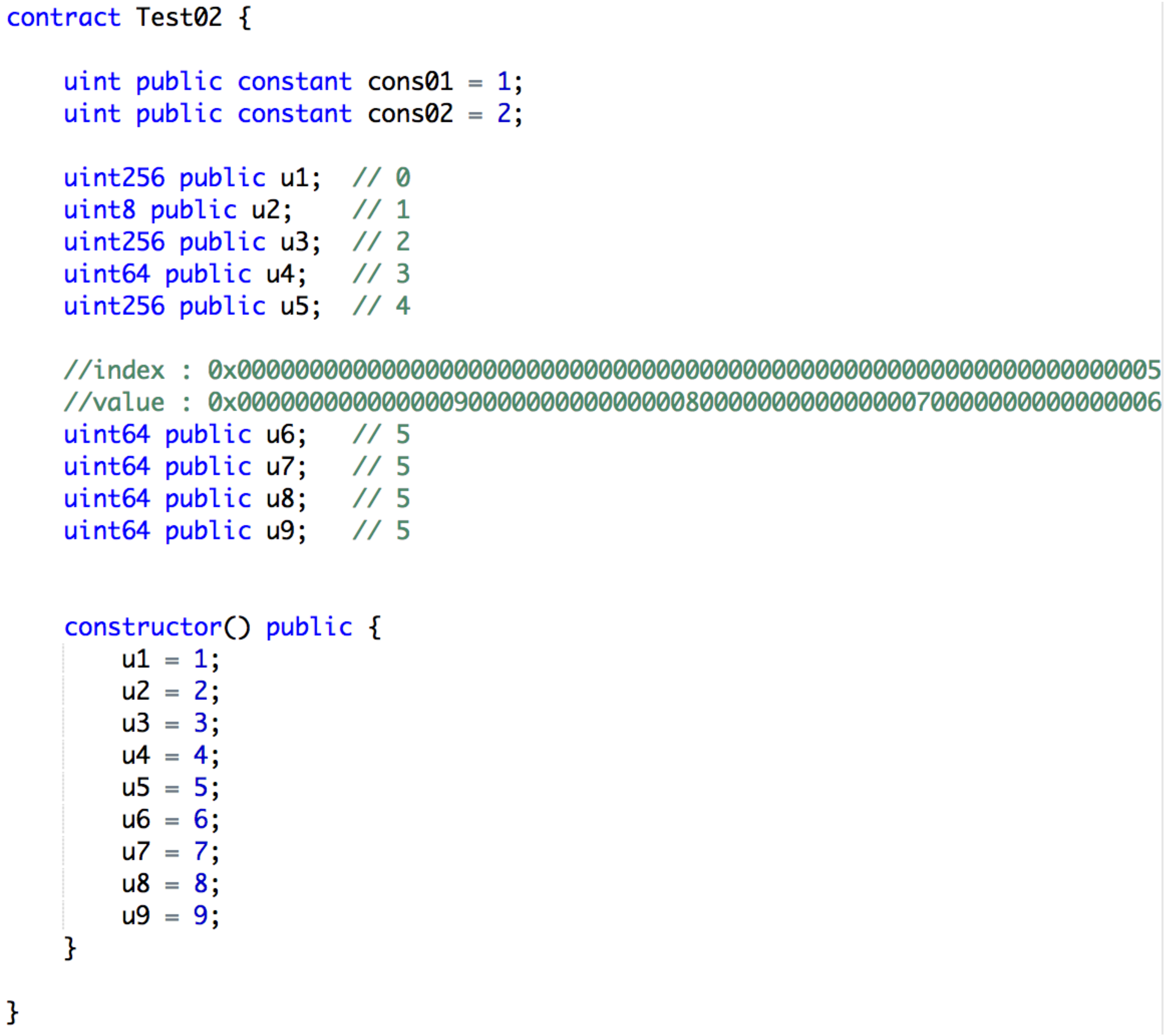
注：没有参数名称，没有空格



```
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.19-unstable/darwin-amd64/go1.10.3
coinbase: 0x7c6320ab150fb2d6bef822659b90c2f85841e860
at block: 3961 (Tue, 20 Nov 2018 13:00:37 CST)
 datadir: /Users/rolong/Library/Ethereum
 modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 pers

> web3.sha3("setInfo(uint256,string,bool,bytes8)")
"0x25c71407de1693c8692b64ea49f8193e25791c95a4d4129aa82bf92780f40273"
>
```

3、智能合约属性的索引和存储

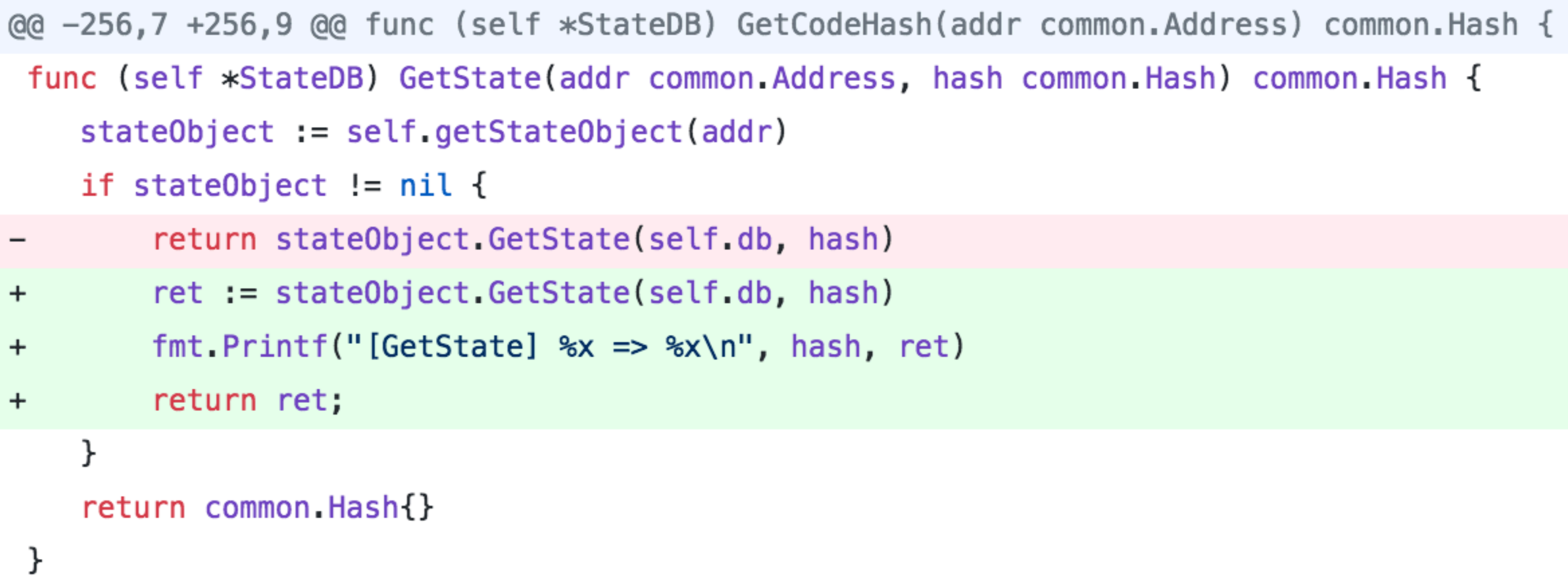
3.1、简单属性的索引规则



```
contract Test02 {

    uint public constant cons01 = 1;
    uint public constant cons02 = 2;

    uint256 public u1;  // 0
    uint8 public u2;    // 1
    uint256 public u3;  // 2
    uint64 public u4;   // 3
    uint256 public u5;  // 4

    //index : 0x0000000000000000000000000000000000000000000000000000000000000005
    //value : 0x0000000000000009000000000000000800000000000000070000000000000006
    uint64 public u6;   // 5
    uint64 public u7;   // 5
    uint64 public u8;   // 5
    uint64 public u9;   // 5


    constructor() public {
        u1 = 1;
        u2 = 2;
        u3 = 3;
        u4 = 4;
        u5 = 5;
        u6 = 6;
        u7 = 7;
        u8 = 8;
        u9 = 9;
    }

}
```

go-ethereum/core/state/statedb.go

打印合约状态merkle树查询的索引和返回值



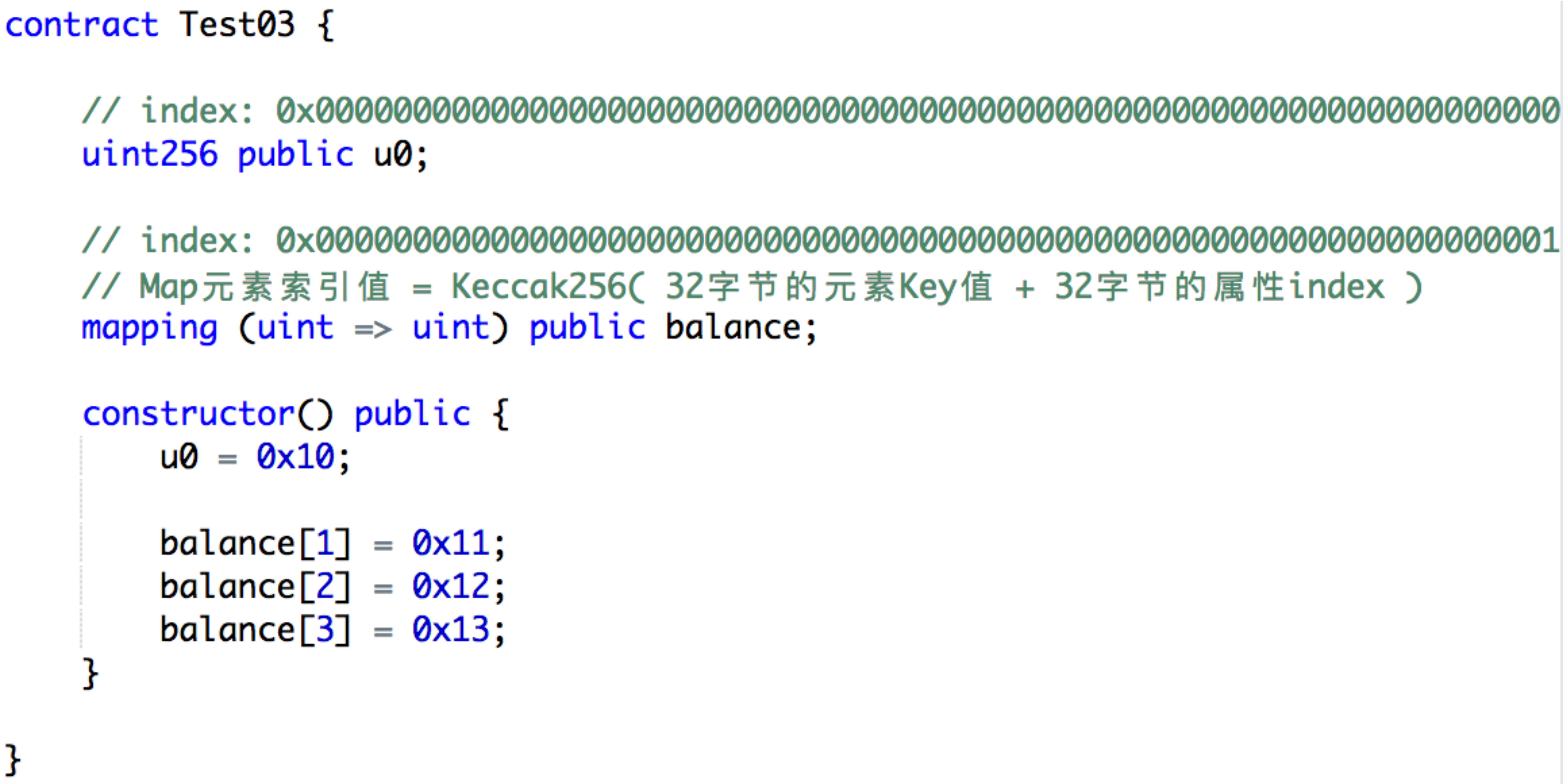
```diff
@@ -256,7 +256,9 @@ func (self *StateDB) GetCodeHash(addr common.Address) common.Hash {
 func (self *StateDB) GetState(addr common.Address, hash common.Hash) common.Hash {
 	stateObject := self.getStateObject(addr)
 	if stateObject != nil {
-		return stateObject.GetState(self.db, hash)
+		ret := stateObject.GetState(self.db, hash)
+		fmt.Printf("[GetState] %x => %x\n", hash, ret)
+		return ret;
 	}
 	return common.Hash{}
 }
```

- 常量属性被写入到静态代码里，不参与编号

- 可写属性从0开始编号

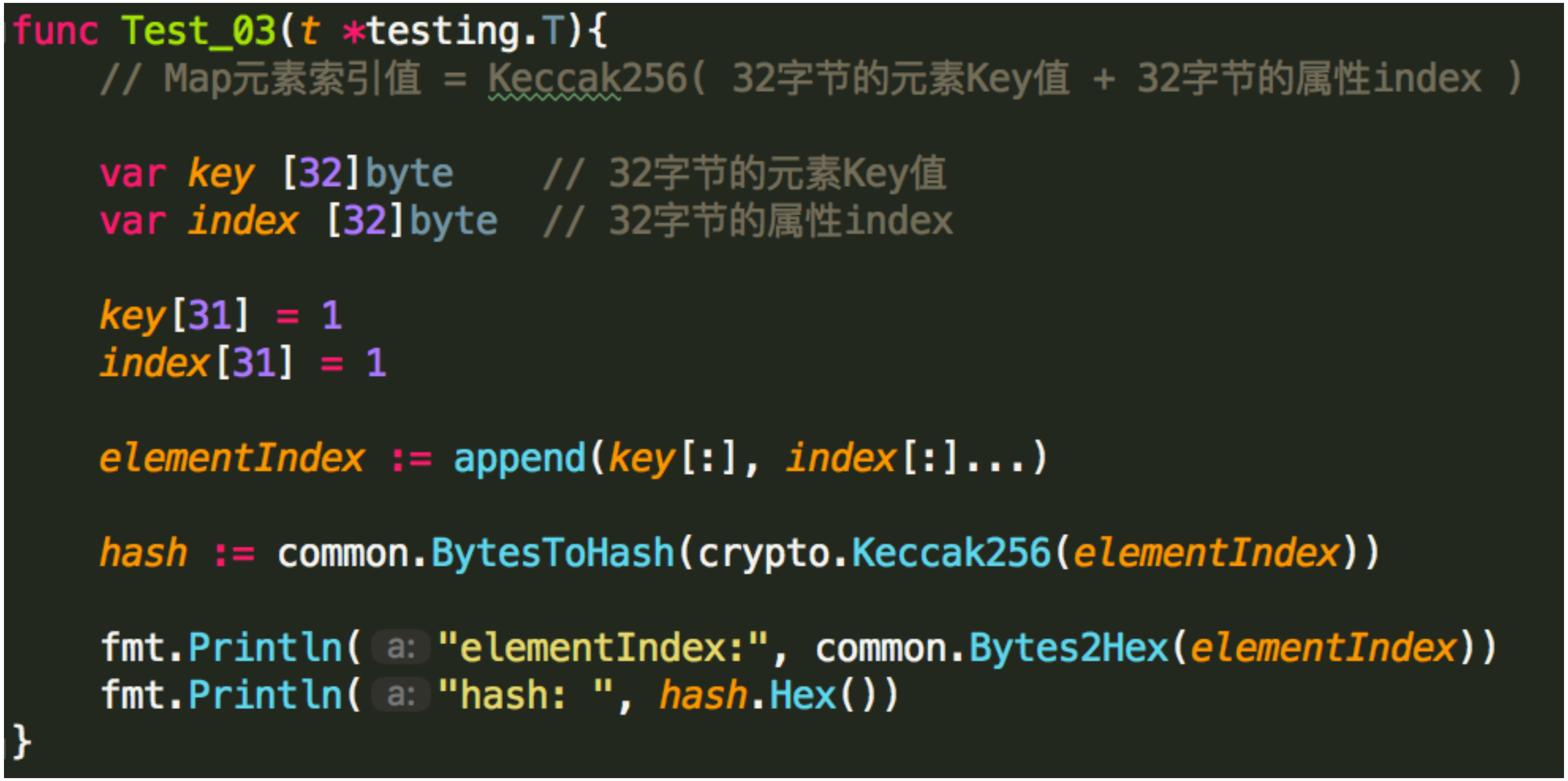
- **相邻属性合并后不超过32字节，会被合并成一个Key-Val**

3.2、map类型的元素索引



```
contract Test03 {

    // index: 0x0000000000000000000000000000000000000000000000000000000000000000
    uint256 public u0;

    // index: 0x0000000000000000000000000000000000000000000000000000000000000001
    // Map元素索引值 = Keccak256( 32字节的元素Key值 + 32字节的属性index )
    mapping (uint => uint) public balance;

    constructor() public {
        u0 = 0x10;

        balance[1] = 0x11;
        balance[2] = 0x12;
        balance[3] = 0x13;
    }

}
```

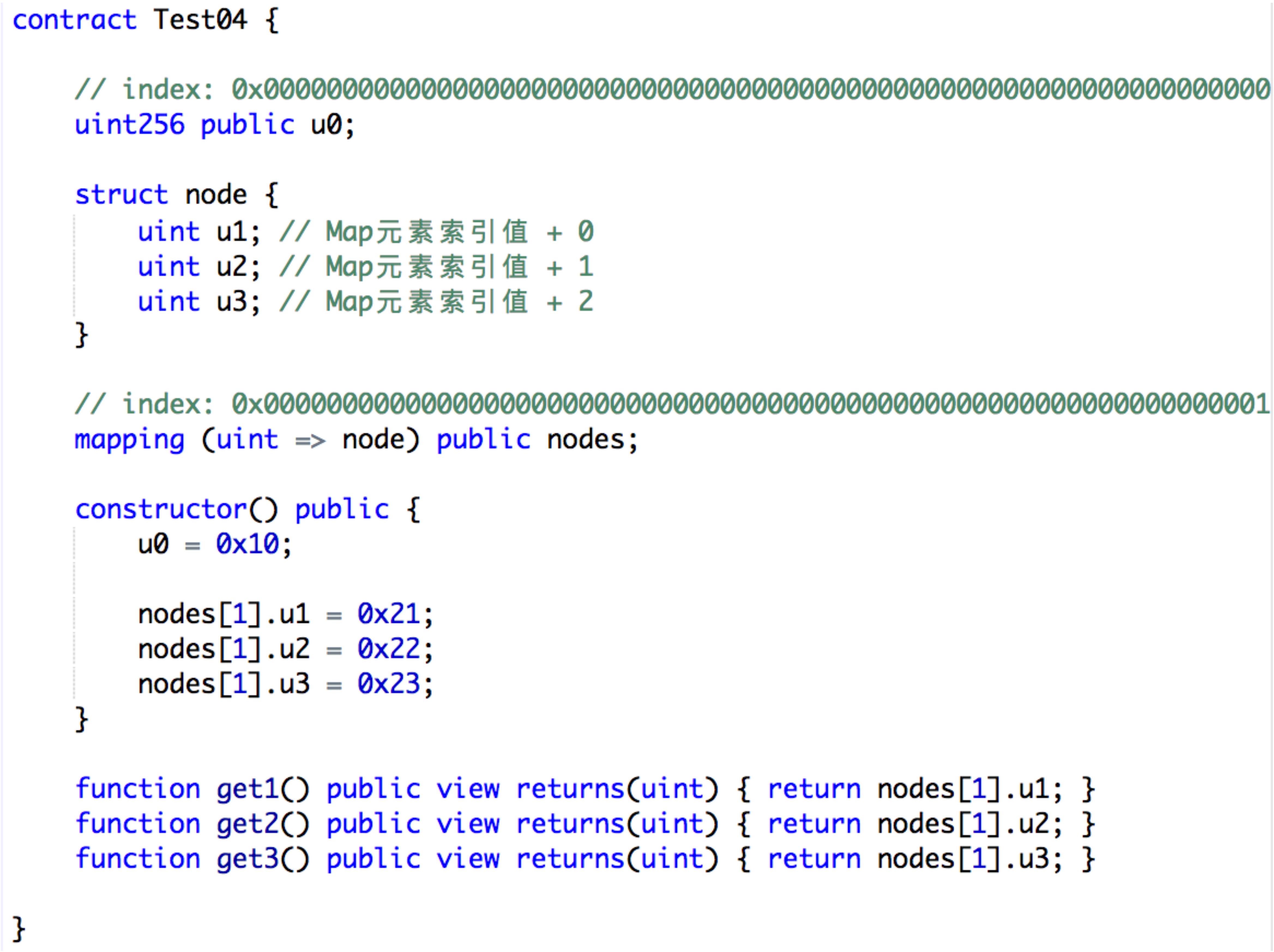
Map元素索引值 = Keccak256(32字节的元素Key值 + 32字节的属性index)



```go
func Test_03(t *testing.T){
    // Map元素索引值 = Keccak256( 32字节的元素Key值 + 32字节的属性index )

    var key [32]byte    // 32字节的元素Key值
    var index [32]byte  // 32字节的属性index

    key[31] = 1
    index[31] = 1

    elementIndex := append(key[:], index[:]...)

    hash := common.BytesToHash(crypto.Keccak256(elementIndex))

    fmt.Println( a: "elementIndex:", common.Bytes2Hex(elementIndex))
    fmt.Println( a: "hash: ", hash.Hex())
}
```

```
elementIndex:
0000000000000000000000000000000000000000000000000000000000000001
0000000000000000000000000000000000000000000000000000000000000001

hash:
0xcc69885fda6bcc1a4ace058b4a62bf5e179ea78fd58a1ccd71c22cc9b688792f
```

3.3、结构体的索引规则



```
contract Test04 {

    // index: 0x0000000000000000000000000000000000000000000000000000000000000000
    uint256 public u0;

    struct node {
        uint u1; // Map元素索引值 + 0
        uint u2; // Map元素索引值 + 1
        uint u3; // Map元素索引值 + 2
    }

    // index: 0x0000000000000000000000000000000000000000000000000000000000000001
    mapping (uint => node) public nodes;

    constructor() public {
        u0 = 0x10;

        nodes[1].u1 = 0x21;
        nodes[1].u2 = 0x22;
        nodes[1].u3 = 0x23;
    }

    function get1() public view returns(uint) { return nodes[1].u1; }
    function get2() public view returns(uint) { return nodes[1].u2; }
    function get3() public view returns(uint) { return nodes[1].u3; }

}
```

- 属性不需要初始化，默认是0或空值，包括复杂结构类型
- 逻辑上结构复杂的类型，底层都是扁平化Key-Val存储
- 32字节为一个存储单元，32字节对齐可以优化存储空间
- 尽量使用定长的类型，减少32字节的长度值和解析复杂度

4、预编译合约介绍及汇编调用方法

预编译合约有什么作用?

- 扩展智能合约原生接口
- 智能合约内部和节点交互

为什么要用汇编调用?

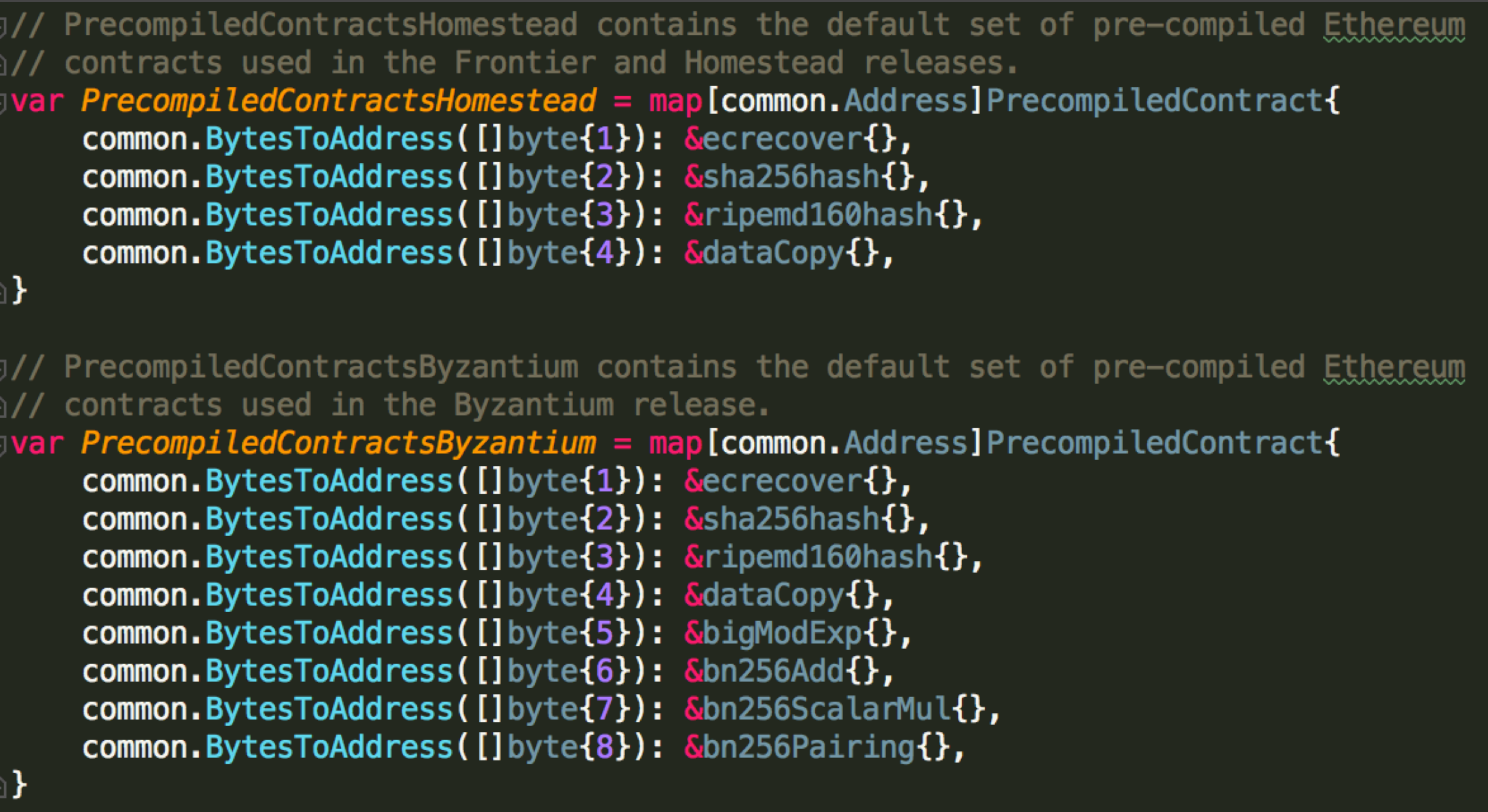
如果我们自己增加了一个原生接口,

现有的solidity编译器不能解析，就会编译错误,

这时，就需要用汇编指令去调用

以太坊在4370000区块高度升级到拜占庭版本后，增加了4个预编译合约

go-ethereum/core/vm/contracts.go



```
// PrecompiledContractsHomestead contains the default set of pre-compiled Ethereum
// contracts used in the Frontier and Homestead releases.
var PrecompiledContractsHomestead = map[common.Address]PrecompiledContract{
    common.BytesToAddress([]byte{1}): &ecrecover{},
    common.BytesToAddress([]byte{2}): &sha256hash{},
    common.BytesToAddress([]byte{3}): &ripemd160hash{},
    common.BytesToAddress([]byte{4}): &dataCopy{},
}

// PrecompiledContractsByzantium contains the default set of pre-compiled Ethereum
// contracts used in the Byzantium release.
var PrecompiledContractsByzantium = map[common.Address]PrecompiledContract{
    common.BytesToAddress([]byte{1}): &ecrecover{},
    common.BytesToAddress([]byte{2}): &sha256hash{},
    common.BytesToAddress([]byte{3}): &ripemd160hash{},
    common.BytesToAddress([]byte{4}): &dataCopy{},
    common.BytesToAddress([]byte{5}): &bigModExp{},
    common.BytesToAddress([]byte{6}): &bn256Add{},
    common.BytesToAddress([]byte{7}): &bn256ScalarMul{},
    common.BytesToAddress([]byte{8}): &bn256Pairing{},
}
```

实例5

1、把节点私钥导入到钱包恢复地址

2、通过智能合约恢复节点公钥对应的地址

3、以上两种方式恢复出来的地址是一致的

恢复（推导）地址的三种途径：

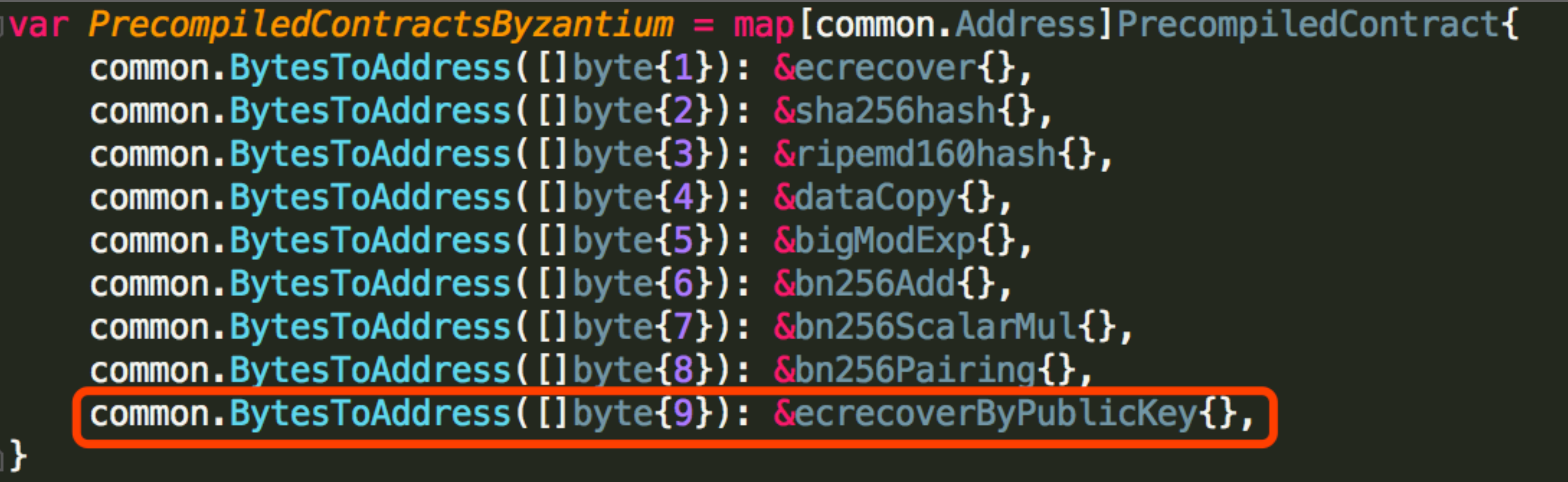
1、私钥 -> 公钥 -> 地址

2、私钥签名 -> 公钥 -> 地址

3、公钥 -> 地址

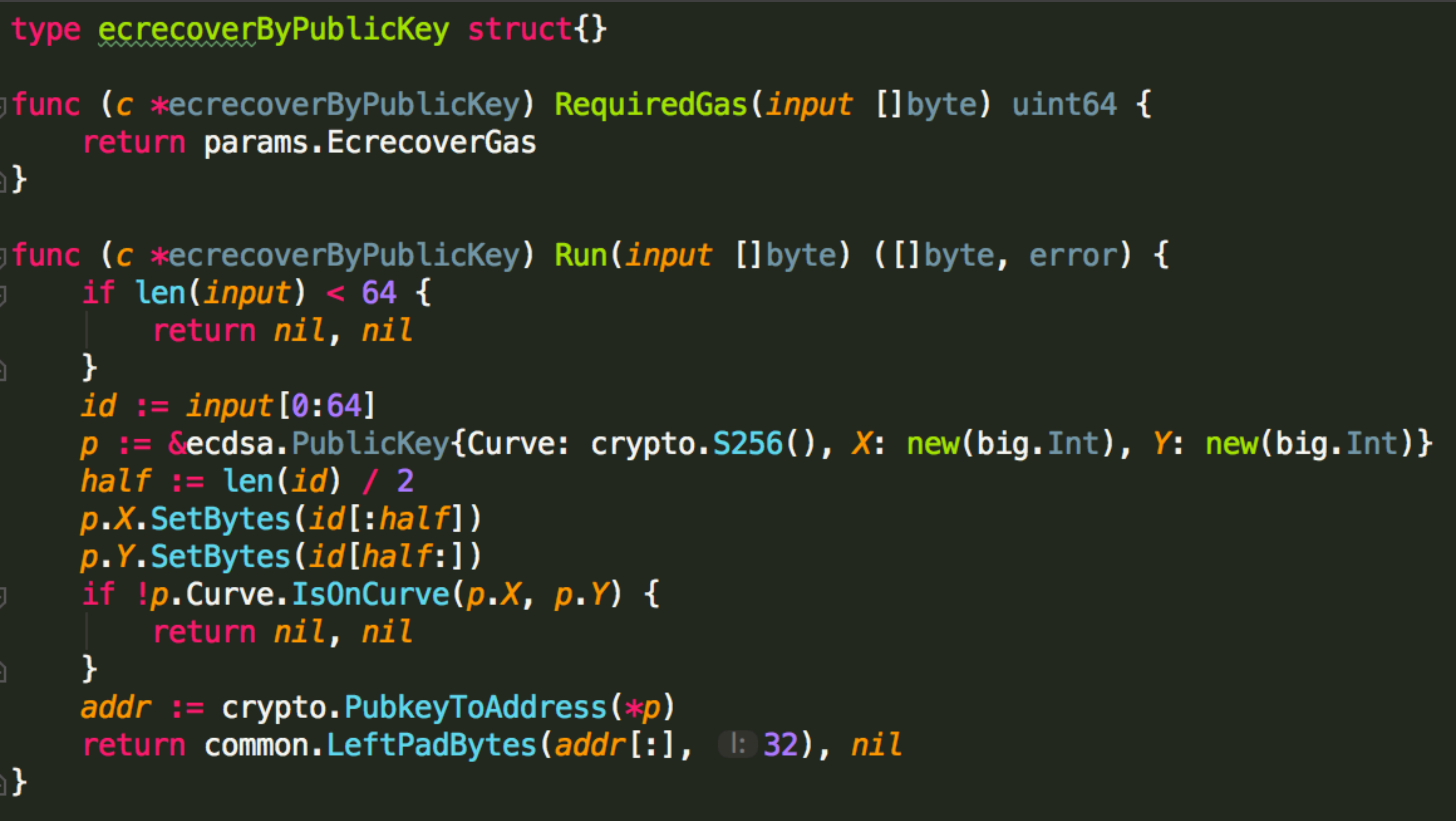
增加一个预编译合约

功能：在智能合约里支持用公钥恢复地址



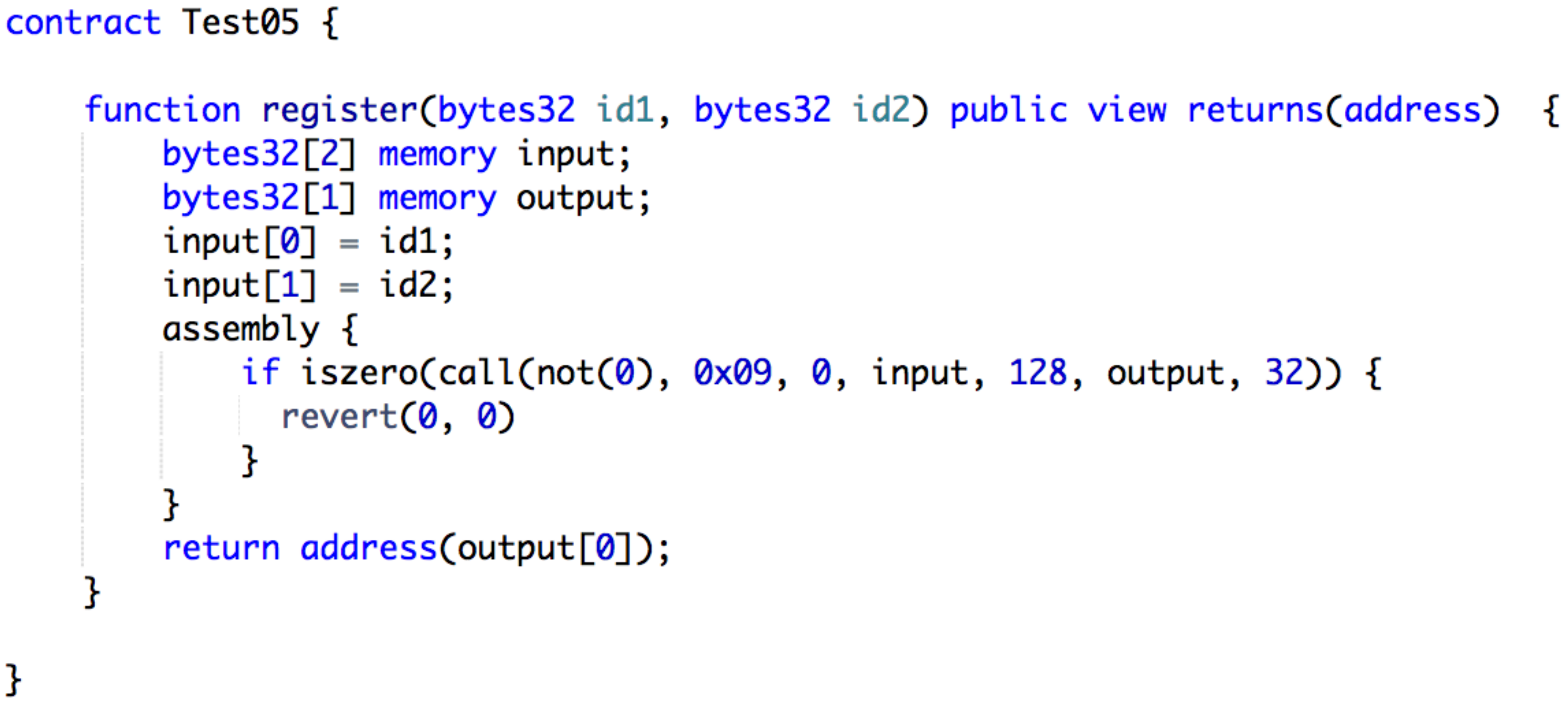
```
var PrecompiledContractsByzantium = map[common.Address]PrecompiledContract{
	common.BytesToAddress([]byte{1}): &ecrecover{},
	common.BytesToAddress([]byte{2}): &sha256hash{},
	common.BytesToAddress([]byte{3}): &ripemd160hash{},
	common.BytesToAddress([]byte{4}): &dataCopy{},
	common.BytesToAddress([]byte{5}): &bigModExp{},
	common.BytesToAddress([]byte{6}): &bn256Add{},
	common.BytesToAddress([]byte{7}): &bn256ScalarMul{},
	common.BytesToAddress([]byte{8}): &bn256Pairing{},
	common.BytesToAddress([]byte{9}): &ecrecoverByPublicKey{},
}
```

ecrecoverByPublicKey的实现



```go
type ecrecoverByPublicKey struct{}

func (c *ecrecoverByPublicKey) RequiredGas(input []byte) uint64 {
    return params.EcrecoverGas
}

func (c *ecrecoverByPublicKey) Run(input []byte) ([]byte, error) {
    if len(input) < 64 {
        return nil, nil
    }
    id := input[0:64]
    p := &ecdsa.PublicKey{Curve: crypto.S256(), X: new(big.Int), Y: new(big.Int)}
    half := len(id) / 2
    p.X.SetBytes(id[:half])
    p.Y.SetBytes(id[half:])
    if !p.Curve.IsOnCurve(p.X, p.Y) {
        return nil, nil
    }
    addr := crypto.PubkeyToAddress(*p)
    return common.LeftPadBytes(addr[:], l: 32), nil
}
```

在智能合约里调用ecrecoverByPublicKey



```
contract Test05 {

    function register(bytes32 id1, bytes32 id2) public view returns(address)  {
        bytes32[2] memory input;
        bytes32[1] memory output;
        input[0] = id1;
        input[1] = id2;
        assembly {
            if iszero(call(not(0), 0x09, 0, input, 128, output, 32)) {
              revert(0, 0)
            }
        }
        return address(output[0]);
    }

}
```

Thank You!

演示代码：

https://github.com/rolong/go-ethereum/tree/topic20181122