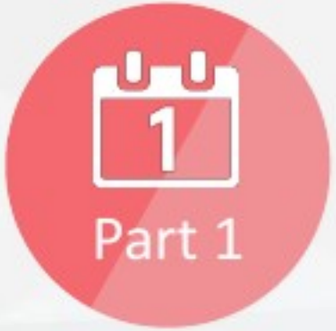
以太坊和智能合约

数据服务事业部

2017-04-10

主要内容

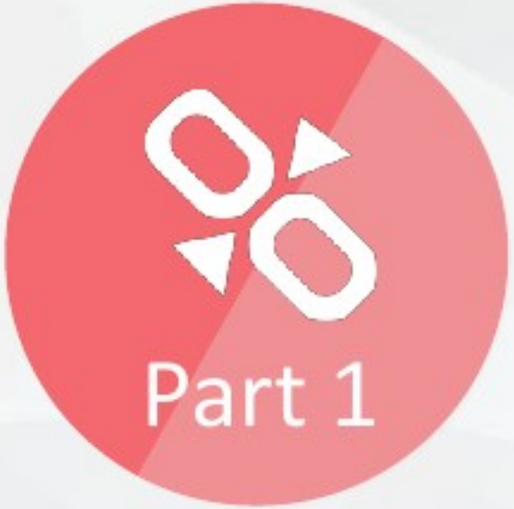


以太坊

智能合约

Solidity

Demo演示



以太坊

比特币的脚本的缺点

缺少图灵完备
不支持循环语句

价值盲
UTXO不可分割

缺少状态
UTXO智能是已花费或未花费

比特币的脚本局限性

以太坊

内置图灵完备有编程语言

以太坊介绍

以太坊通过建立终极的抽象的基础层-内置有图灵完备编程语言的区块链-使得任何人都能够创建合约和去中心化应用，并在其中设立他们自由定义的所有权规则、交易方式和状态转换函数。

以太坊虚拟机（EVM）是以太坊中智能合约的运行环境。它不仅被沙箱封装起来，事实上它被完全隔离，也就是说运行在EVM内部的代码不能接触到网络、文件系统或者其它进程。甚至智能合约与其它智能合约只有有限的接触。

以太坊账户

包含四个部分：

- 随机数，用于确定每笔交易只能被处理一次的计数器，参与生成交易的id
- 账户目前的以太币余额，以太币（Ether）是以太坊内部的主要加密燃料，用于支付交易费用。 以太币的最小单位为Wei， 最大单位为以太，1 Ether=10^18Wei
- 账户的合约代码，如果有的话
- 账户的存储（默认为空）

用户用公钥、私钥控制

相同的地址空间
相同的数据结构
不同的功能

由合约编译后的code控制

交易

外部账户

发送交易

交易包含以下内容：

- 消息的接收者地址
- 用于确认发送者的签名
- 要发送的以太币的数量
- 可选的数据（合约的参数）
- STARTGAS: 用来限制合约最多执行多少次运算
- GASPRICE: 每次计算需要支付的费用

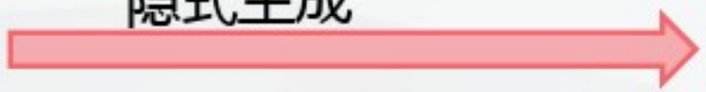
每5个Byte需要1个Gas

防止死循环
交易费用=
STARTGAS* GASPRICE

对计算资源、存储资源、带宽的消耗都用Gas支付

消息

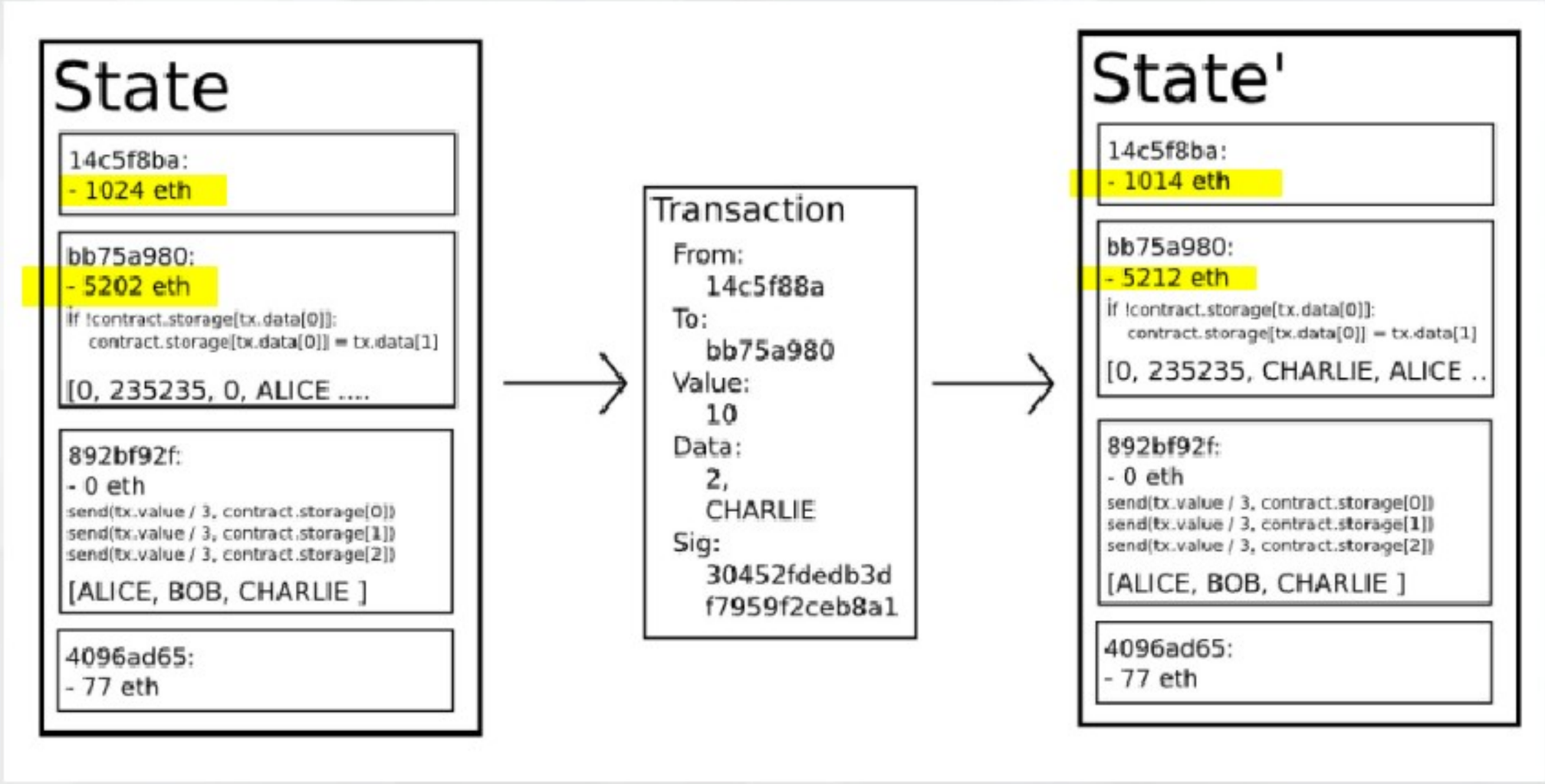
调用合约的
CALL方法时
隐式生成



消息包含五个部分：

- 消息的发送者
- 消息的接受者
- 要发送的以太币的数量
- 可选的数据（合约的参数）
- STARTGAS: 用来限制合约最多执行多少次运算

以太坊状态转移


State
14c5f8ba:
- 1024 eth
bb75a980:
- 5202 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, 0, ALICE ....
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth
Transaction
From:
14c5f88a
To:
bb75a980
Value:
10
Data:
2,
CHARLIE
Sig:
30452fdedb3d
f7959f2ceb8a1
State'
14c5f8ba:
- 1014 eth
bb75a980:
- 5212 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, CHARLIE, ALICE ..
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth

以太坊状态转移函数

以太坊的状态转换函数：APPLY(S,TX) -> S’ ，可以定义如下：

1. 检查交易的格式是否正确（即有正确数值）、签名是否有效和随机数是否与发送者账户的随机数匹配。如否，返回错误。
2. 计算交易费用:fee=STARTGAS * GASPRICE，并从签名中确定发送者的地址。从发送者的账户中减去交易费用和增加发送者的随机数。如果账户余额不足，返回错误。
3. 设定初值GAS = STARTGAS，并根据交易中的字节数减去一定量的燃料值。
4. 从发送者的账户转移价值到接收者账户。如果接收账户还不存在，创建此账户。如果接收账户是一个合约，运行合约的代码，直到代码运行结束或者燃料用完。
5. 如果因为发送者账户足够的钱或者代码执行耗尽燃料导致价值转移失败，生成out-of-gas异常，恢复原来的状态，但是还需要支没有付交易费用，交易费用加至矿工账户。
6. 否则，将所有剩余的燃料归还给发送者，消耗掉的燃料作为交易费用发送给矿工。

以太坊实现版本

以太坊有Go语言、C++语言、Python语言的版本。

go-ethereum

Official Go implementation of the Ethereum protocol

go ethereum blockchain p2p geth

Go ★ 3,428 1,194 Updated 27 minutes ago

cpp-ethereum

Ethereum C++ client

cpp ethereum ethereum-client

C++ ★ 920 557 Updated 17 hours ago

pyethereum

Next generation cryptocurrency network

Python ★ 748 266 Updated 4 hours ago

web3.js

Ethereum JavaScript API

javascript api ethereum swarm whisper

JavaScript ★ 666 288 Updated 14 hours ago

Etherscan
以太坊区块在线浏览

https://etherscan.io/

TOTAL SUPPLY OF 90,447,075.50 ETHER

$44.83 @ 0.03917 BTC/ETH (-1.41%)

LAST BLOCK

3484756 (14.43s Avg)

TRANSACTIONS

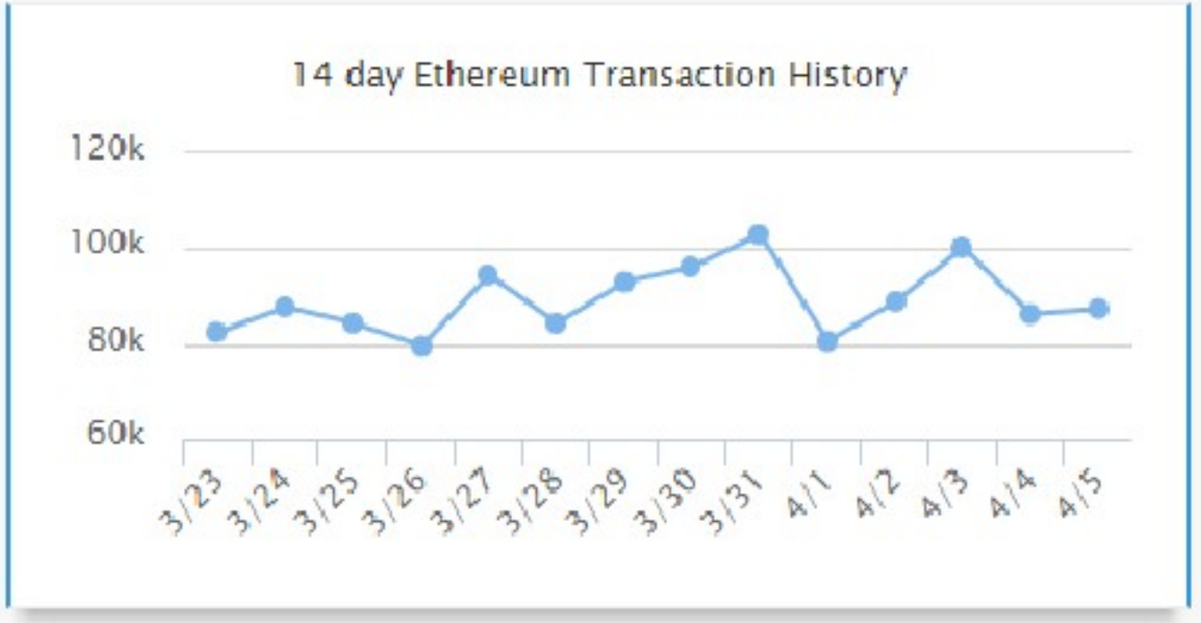
20413879

Hash Rate

17,279.56 GH/s

Network Difficulty

247.76 TH



以太坊的问题

以太坊在不断的改进中，很多机制可能会改变。

智能合约

智能合约的定义

"智能合约" - 根据事先任意制订的规则来自动转移数字资产的系统。

业务人员

开发人员

Smart contracts are pieces of code that live on the blockchain and execute commands exactly how the were told to.

智能合约就是存储在区块链上的代码，用以实现执行特定的功能。

智能合约使用步骤

编译合约

智能合约
hello word

```
contract greeter is mortal { //继承mortal
  string greeting;
  function greeter(string _greeting) public { greeting = _greeting; }
  function greet() constant returns (string) { return greeting; }
}
```

```
var greeterSource = ' contract greeter is mortal {string greeting; .....return greeting;}}'

var greeterCompiled = web3.eth.compile.solidity(greeterSource)
```

备注：本节中的所有例子以在Go语言的客户端geth中的操作为例

创建合约

```
var greeterContract =

web3.eth.contract(greeterCompiled["<stdin>:greeter"].info.abiDefinition);
```

```
> greeterCompiled.greeter.info.abiDefinition;
[{
    constant: false,
    inputs: [],
    name: "kill",
    outputs: [],
    type: "function"
}, {
    constant: true,
    inputs: [],
    name: "greet",
    outputs: [{
        name: "",
        type: "string"
    }],
    type: "function"
}, {
    inputs: [{
        name: "_greeting",
        type: "string"
    }],
    type: "constructor"
}]
>
```

abiDefinition是合约的接口说明

老版本中是

greeterCompiled.greeter.info.abiDefinition

新版本（geth-1.5.9）中是

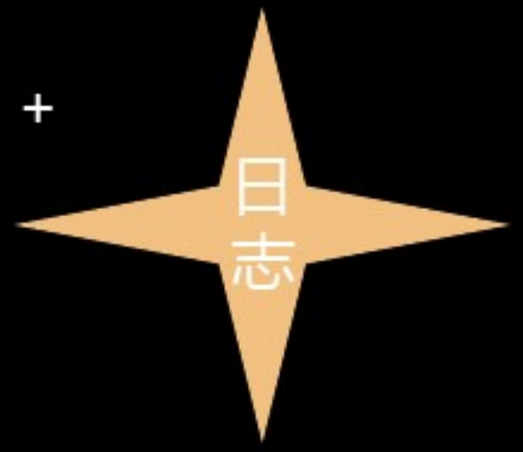
greeterCompiled["<stdin>:greeter"].info.abiDefinition

备注：本节中的所有例子以在Go语言的客户端geth中的操作为例

部署合约

部署合约的时候，需要支付费用，需要用户输入密码，先执行
personal.unlockAccount(eth.accounts[0]);

```
var greeter = greeterContract.new(_greeting, {from: eth.accounts[0],
data: greeterCompiled["<stdin>:greeter"].code, gas: 1000000}, function(e, contract){
  if(!e) {
    if(!contract.address) {
        console.log("Contract transaction send: TransactionHash: " +
        contract.transactionHash +  " waiting to be mined...");
    } else {
      console.log("Contract mined! Address: " + contract.address);
      console.log(contract);
    }
  }
})
```



调用合约

合约部署的Transaction被矿工挖矿之后，可以调用该合约。

```
greeter.greet()
```

合约部署的Transaction被矿工挖矿之后，可以查看该合约的地址。

```
eth.getCode(greeter.address)
```

其它的机器调用合约

其它的机器上没有greeter对象，需要根据合约的地址和abiDefinition来创建，然后调用。

```
var greeter2 =
eth.contract([{constant:false,inputs:[],name:‘kill’,outputs:[],type:‘function’},{const
ant:true,inputs:[],name:‘greet’,outputs:[{name:‘’,type:‘string’}],type:‘function’},{in
puts:[{name:‘_greeting’,type:‘string’}],type:'constructor'}]).at(‘0xcde7cfdf234dfa63
ba4d7c273a’);

greeter2.greet();
```

监听合约

```
var event = token.CoinTransfer({}, '', function(error, result){
  if (!error)
    console.log("Coin transfer: " + result.args.amount + " tokens
});
```

合约部署后，可设置监听

```
contract token {
    mapping (address => uint) public coinBalanceOf;
    event CoinTransfer(address sender, address receiver, uint amount);

  /* Initializes contract with initial supply tokens to the creator of the contract */
  function token(uint supply) {
        coinBalanceOf[msg.sender] = supply;
    }

  /* Very simple trade function */
    function sendCoin(address receiver, uint amount) returns(bool su
        if (coinBalanceOf[msg.sender] < amount) return false;
        coinBalanceOf[msg.sender] -= amount;
        coinBalanceOf[receiver] += amount;
        CoinTransfer(msg.sender, receiver, amount);
        return true;
    }
}
```

```
var event = clientReceipt.Deposit();

// watch for changes
event.watch(function(error, result){
    // result will contain various information
    // including the argumets given to the Deposit
    // call.
    if (!error)
        console.log(result);
});
```

另一种调用方式

销毁合约

```
contract mortal {
    /* Define variable owner of the type address*/
    address owner;

    /* this function is executed at initialization and sets the owner of
    function mortal() { owner = msg.sender; }

    /* Function to recover the funds on the contract */
    function kill() { if (msg.sender == owner) suicide(owner); }
}
```

只有拥有者才能销毁合约
msg存在于上下文中，不是参数中

```
greeter.kill.sendTransaction({from:eth.accounts[0]})
```

合约使用场景

金融类场景

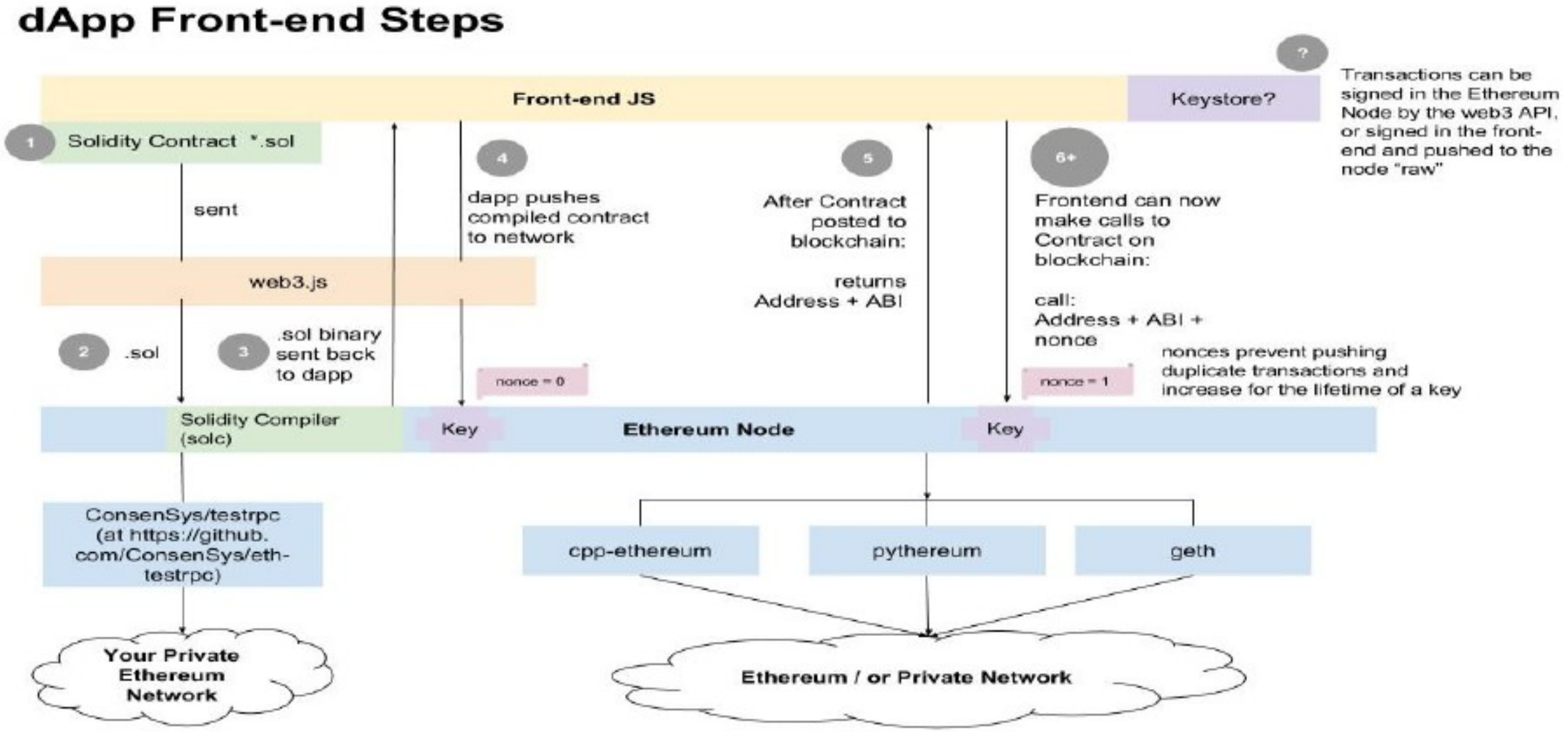
对冲合约、储蓄钱包、遗嘱等

非金融类场景

在线投票、去中心化治理、域名注册

ĐAPPS（去中心化的应用）

http://dapps.ethercasts.com/上面已经有300多个ĐAPPS



A **Contract Creation Transaction** is shown in steps 1-5 at above.

An **Ether Transfer** or **Function Call Transaction** is assumed in step 6.

ĐAPPS

Main Resources

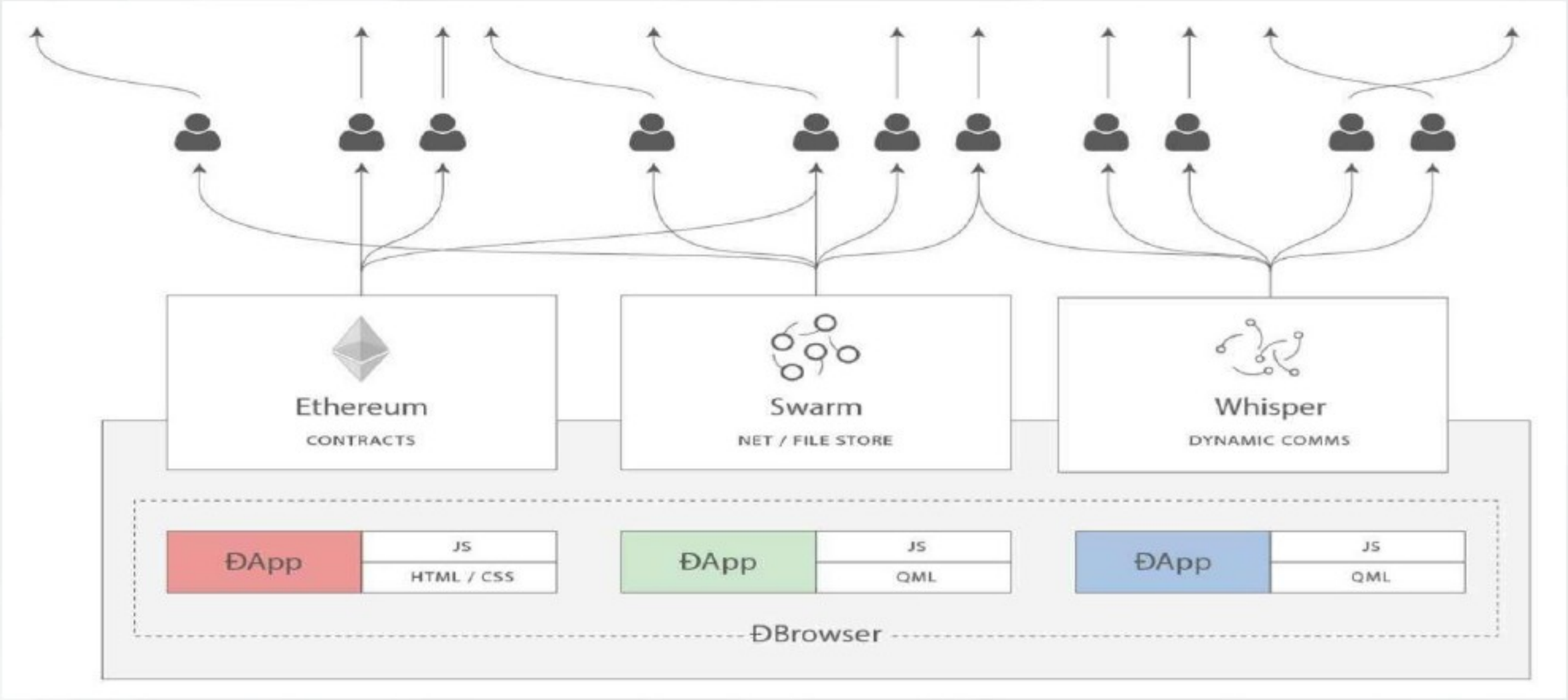
- Web3 JavaScript API - This is the main JavaScript SDK to use when you want to interact with a nodes API
- JSON RPC API - This is the low level JSON RPC 2.0 interface to interface with a node. This API is used by the Web3 JavaScript API.
- Solidity Documentation - Solidity is the Ethereum developed Smart Contract language, which compiles to EVM (Ethereum Virtual Machine) opcodes.

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_syncing","params":[],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": {
    startingBlock: '0x384',
    currentBlock: '0x386',
    highestBlock: '0x454'
  }
}
// Or when not syncing
{
  "id":1,
  "jsonrpc": "2.0",
  "result": false
}
```

```
var sync = web3.eth.syncing;
console.log(sync);
/*
{
   startingBlock: 300,
   currentBlock: 312,
   highestBlock: 512
}
*/
```

构建去中心化WEB3.0

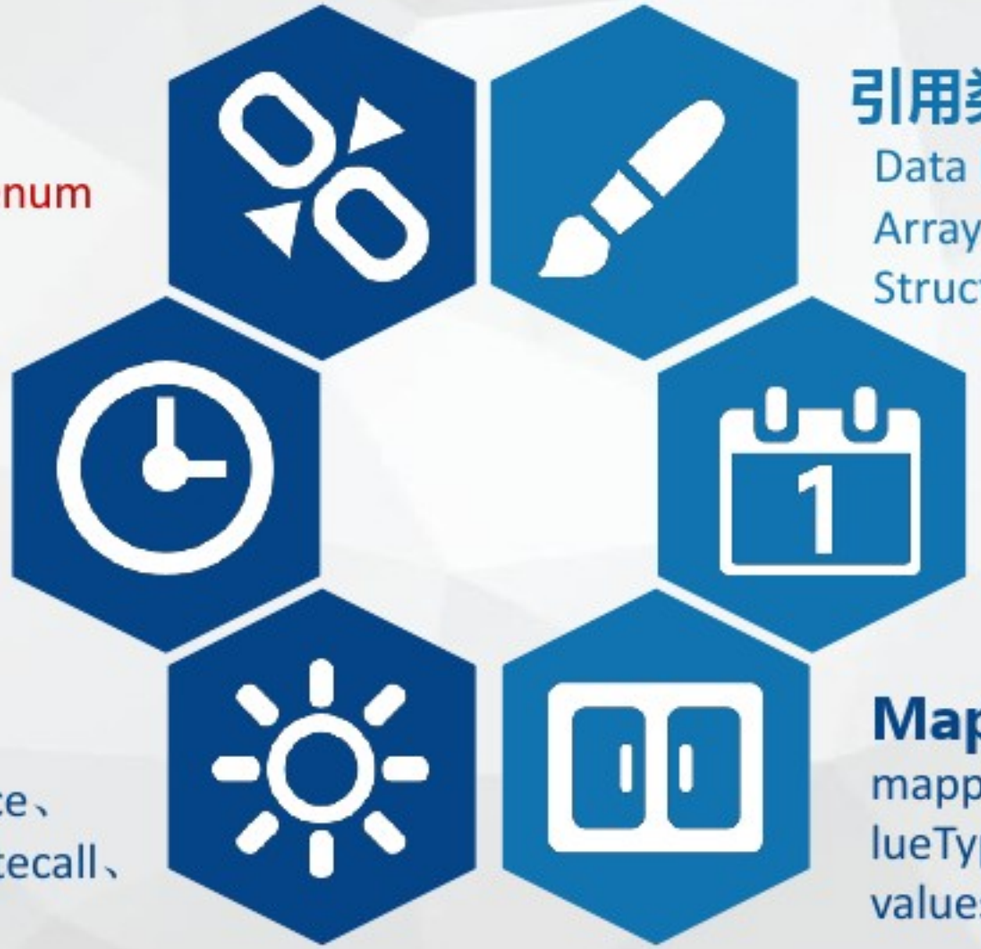


Solidity语言

数据类型

基本类型

boolean, int/uint,byte, string, enum
Int8~int256
bytes1,bytes2,...bytes32
function, fixed point numbers



引用类型

Data location
Arrays
Structs

address

20个字节，主要方法：balance、transfer、send、 call、delegatecall、callcode

Mappings

mapping(_KeyType => _ValueType)不能遍历keys和values

流程控制 和 函数

支持大部分javascript的逻辑控制，包括if, else, while, do, for, break, continue, return, ? : 但是不支持switch 和 goto

function 函数名(**参数**) 权限控制符 **returns** (返回值，可以有多个)

```
function Register(uint initialPrice) { price=initialPrice; }
function utterance() returns (bytes32) { return "miaow"; }
function getData() public returns(uint) { return data; }
```

合约的结构

合约与面向对象编程中的类非常相似。合约由**变量、函数、函数修饰符、事件、struct结构体、枚举类型**构成。合约也可以继承。

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

```
pragma solidity ^0.4.0;

contract Purchase {
    enum State { Created, Locked, Inactive } // Enum
}
```

```
pragma solidity ^0.4.0;

contract SimpleAuction {
    event HighestBidIncreased(address bidder, uint amount); // Event

    function bid() payable {
        // ...
        HighestBidIncreased(msg.sender, msg.value); // Triggering event
    }
}
```

```
pragma solidity ^0.4.0;

contract Ballot {
    struct Voter { // Struct
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }
}
```

函数的修饰符

可见性：
public, private, internal, external。其中public和private用于决定是否可以通过对象直接调用。
internal和external用于决定是否其它的合约可以调用。

另外constant关键词，可以用于修饰常量和常函数。

```
pragma solidity ^0.4.0;

contract C {
    uint private data;

    function f(uint a) private returns(uint b) { return a + 1; }
    function setData(uint a) { data = a; }
    function getData() public returns(uint) { return data; }
    function compute(uint a, uint b) internal returns (uint) { return a+b; }
}


contract D {
    function readData() {
        C c = new C();
        uint local = c.f(7); // error: member "f" is not visible
        c.setData(3);
        local = c.getData();
        local = c.compute(3, 5); // error: member "compute" is not visible
    }
}


contract E is C {
    function g() {
        C c = new C();
        uint val = compute(3, 5);  // acces to internal member (from derivated to parer
    }
}
```

事件Event——非常重要的功能

```
event Deposit( address indexed _from, bytes32 indexed _id, uint _value );
```

以太坊通过Logs实现Events（事件）功能。
日志内容是**交易收据**（Transaction Receipts）的一部分，整个日志内容，包括Receipts的其它内容会生成一个ReceiptsRoot存储在区块的头部。
日志永远存在。
通过设置一些过滤条件，来获取某些特定的事件。
可以用于**追溯合约的历史**状态，设置检查点等。

1. `String|Object` - The string `"latest"` or `"pending"` to watch for changes in the latest block or pending transactions respectively. Or a filter options object as follows:

- `fromBlock`: `Number|String` - The number of the earliest block (`latest` may be given to mean the most recent and `pending` currently mining, block). By default `latest`.
- `toBlock`: `Number|String` - The number of the latest block (`latest` may be given to mean the most recent and `pending` currently mining, block). By default `latest`.
- `address`: `String` - An address or a list of addresses to only get logs from particular account(s).
- `topics`: `Array of Strings` - An array of values which must each appear in the log entries. The order is important, if you want to leave topics out use `null`, e.g. `[null, '0x00...']`. You can also pass another array for each topic with options for that topic e.g. `[null, ['option1', 'option2']]`

继承

继承的关键字是is，支持多重继承。

```
contract Final is Base2, Base1 { }
```

抽象合约

```
pragma solidity ^0.4.0;

contract Feline {
    function utterance() returns (bytes32);
}
```

```
pragma solidity ^0.4.0;

contract Cat is Feline {
    function utterance() returns (bytes32) { return "miaow"; }
}
```

接口

```
interface Token {
    function transfer(address recipient, uint amount);
}
```

1. Cannot inherit other contracts or interfaces.
2. Cannot define constructor.
3. Cannot define variables.
4. Cannot define structs.
5. Cannot define enums.

常用对象和操作

Block and Transaction Properties

- `block.blockhash(uint blockNumber) returns (bytes32)`: hash of the given most recent blocks excluding current
- `block.coinbase` (addr
- `block.difficulty` (ui
- `block.gaslimit` (uint
- `block.number` (uint):
- `block.timestamp` (uin
- `msg.data` (bytes): co
- `msg.gas` (uint): rem
- `msg.sender` (address)
- `msg.sig` (bytes4): fir
- `msg.value` (uint): number of wei sent with the message
- `now` (uint): current block timestamp (alias for `block.timestamp`)
- `tx.gasprice` (uint): gas price of the transaction

Contract Related

`this` **(current contract's type):**
the current contract, explicitly convertible to Address

`selfdestruct(address recipient)`:
destroy the current contract, sending its funds to the given Address

Mathematical and Cryptographic Functions

`assert(bool condition)`:
throws if the condition is not met.

performed with a

on is performed

) hash of the (tigh

`sha256(...) returns (bytes32)`:
compute the SHA-256 hash of the (tightly packed) arguments

编译器和开发环境

编译器是solc

IDE

1、 IntelliJ IDEA plugin

2、 Visual Studio Extension

等其他

Onlie

https://remix.ethereum.org/#version=soljson-v0.4.10+commit.f0d539ae.js

Demo演示

本地区块链网络的部署

编辑genesis.json

```
{ “nonce”: “0x0000000000000042”,
 "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
"difficulty": "0x4000", //难度值，决定块生成的速度
"alloc": { "0x09c7b615a1c5b3016ff6b521723364aa9382ec6e":
        { “balance”: “10000000000000000000” } }, //10个以太币
 "coinbase": "0x0000000000000000000000000000000000000000",
"timestamp": "0x00",
"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
"extraData": "Custem Ethereum Genesis Block",
"gasLimit": "0xffffffff"
}
```

区块链初始化，为账户分配以太币

```
geth  --datadir eth-data --networkid 123 --
nodiscover console init genesis.json
```

keystore存储账户，chaindata存储区块链数据

```
vagrant@ubuntu-1404:~/eth-test/eth-data-node1$ ls
geth  geth.ipc  history  keystore
vagrant@ubuntu-1404:~/eth-test/eth-data-node1/geth$ ls
chaindata  LOCK  nodekey
```

资产竞拍

```
contract Auction {
    event newBid();
    address owner;          资产拥有者
    address public leader;  竞拍领先的人
    address public winner;  最终获得的人
    string public item;     资产名字
    uint public leadingBid;领先的竞拍的金额

    function Auction(string name, uint price) {
        owner = msg.sender;
        item = name;
        leadingBid = price;
    }
    function placeBid() {
        if (msg.value > leadingBid) {
            returnPrevBid();
            leader = msg.sender;
            leadingBid = msg.value;
            newBid();
        }
    }
    function returnPrevBid() {
```

代码解读

简单的资产竞拍的场景。每一次竞拍，维护当前的竞拍的最高价、最高价的拥有者。

```
            if (leader != 0) {
                leader.send(leadingBid);
            }
    }
    function endAuction() {
        if (msg.sender == owner) {
            winner = leader;
            owner.send(leadingBid);
        }
    }
}
```

演示中

参考文档

1. Solidity说明文档： https://solidity.readthedocs.io/en/develop/
2. 以太坊Wiki: https://github.com/ethereum/wiki/wiki
3. 以太坊区块浏览：https://etherscan.io/
4. Dapp社区： http://dapps.ethercasts.com/
5. 实战例子: https://samsclass.info/141/proj/pEth3.htm

Q & A