**井通智能合约公测说明**

井通solidity版本的ERC20智能合约开发已进入测试阶段，现邀请广大开发者们公开测试，使用说明如下：

公测节点地址：ws://123.57.209.177:5030

整个合约包含编译、部署和调用，步骤如下：

1. **创建solidity合约并编译**

下面我们创建一个发行ERC20代币的合约代码：

pragma solidity ^0.4.19;

contract TokenTest {

string public name;

string public symbol;

uint8 public decimals = 18; // decimals 可以有的小数点个数，最小的代币单位。18 是建议的默认值

uint256 public totalSupply;

// 用mapping保存每个地址对应的余额

mapping (address => uint256) public balanceOf;

// 存储对账号的控制

mapping (address => mapping (address => uint256)) public allowance;

/\*\*

\* 初始化构造

\*/

function TokenTest(uint256 initialSupply, string tokenName, string tokenSymbol) public {

totalSupply = initialSupply \* 10 \*\* uint256(decimals); // 供应的份额，份额跟最小的代币单位有关，份额 = 币数 \* 10 \*\* decimals。

balanceOf[msg.sender] = totalSupply;

name = tokenName; // 代币名称

symbol = tokenSymbol; // 代币符号

}

/\*\*

\* 代币交易转移的内部实现

\*/

function \_transfer(address \_from, address \_to, uint \_value) internal {

// 确保目标地址不为0x0，因为0x0地址代表销毁

require(\_to != 0x0);

// 检查发送者余额

require(balanceOf[\_from] >= \_value);

// 确保转移为正数个

require(balanceOf[\_to] + \_value > balanceOf[\_to]);

// 以下用来检查交易，

uint previousBalances = balanceOf[\_from] + balanceOf[\_to];

// Subtract from the sender

balanceOf[\_from] -= \_value;

// Add the same to the recipient

balanceOf[\_to] += \_value;

// 用assert来检查代码逻辑。

assert(balanceOf[\_from] + balanceOf[\_to] == previousBalances);

}

/\*\*

\* 代币交易转移

\* 从自己（创建交易者）账号发送`\_value`个代币到 `\_to`账号

\* @param \_to 接收者地址

\* @param \_value 转移数额

\*/

function transfer(address \_to, uint256 \_value) public {

\_transfer(msg.sender, \_to, \_value);

}

function() public {

revert();

}

}

编译以上的合约代码(可通过https://remix.ethereum.org/ 在线编译)，生成EVM的bytecode如下：



同时生成的abi如下：

[

{

"constant": true,

"inputs": [],

"name": "name",

"outputs": [

{

"name": "",

"type": "string"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "totalSupply",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "decimals",

"outputs": [

{

"name": "",

"type": "uint8"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "",

"type": "address"

}

],

"name": "balanceOf",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "symbol",

"outputs": [

{

"name": "",

"type": "string"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": false,

"inputs": [

{

"name": "\_to",

"type": "address"

},

{

"name": "\_value",

"type": "uint256"

}

],

"name": "transfer",

"outputs": [],

"payable": false,

"stateMutability": "nonpayable",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "",

"type": "address"

},

{

"name": "",

"type": "address"

}

],

"name": "allowance",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"inputs": [

{

"name": "initialSupply",

"type": "uint256"

},

{

"name": "tokenName",

"type": "string"

},

{

"name": "tokenSymbol",

"type": "string"

}

],

"payable": false,

"stateMutability": "nonpayable",

"type": "constructor"

},

{

"payable": false,

"stateMutability": "nonpayable",

"type": "fallback"

}

]

1. **部署合约**

bytecode和abi生成之后，我们引入jingtum-lib库，通过initContract方法部署合约，代码如下：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://123.57.209.177:5030', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var v = {

secret: 's...UTb',

address: 'j...yTh'

};

var req = remote.initContract({

account: v.address,

amount: 10,

payload: bytecode,//上面solidity编译生成的bytecode

abi: abi,//上面solidity编译生成的abi

params:[2000, 'TestCurrency', 'TEST1']

});

req.setSecret(v.secret);

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

如果成功，结果中会返回一个合约账号，供后面的调用使用。具体接口说明详见接口文档。

1. **调用合约**

调用通部署一样，都是引入jingtum-lib库，然后通过invokeContract方法调用合约，代码如下：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://123.57.209.177:5030', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var v = {

secret: 's...UTb',

address: 'j...yTh'

};

var req = remote.invokeContract({

account: v.address,

destination: 'jPZ1....9Kkh', //部署返回的合约地址

abi: abi,//solidity合约编译生成的abi

func:"transfer('jPZ1....9Kkh', 15)"});//调用合约的某个方法

req.setSecret(v.secret);

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

其中，func参数是具体调用的合约方法，包含合约名及参数，如本例子中给某个账号发币表示为”transfer(jPZ1....9Kkh, 15)”；若无参数则不需要传，但函数的括号必须写全。