**创智链智能合约虚拟机系统**

**用户手册**

**北京新创智链科技有限公司**

目录

[1、引言 2](#_Toc30103823)

[1.1编写目的 2](#_Toc30103824)

[1.2概念解释 3](#_Toc30103825)

[2、GVM与EVM差异说明 3](#_Toc30103826)

[2.1 ABI差异说明 3](#_Toc30103827)

[2.2 其他差异说明 4](#_Toc30103828)

[3、关于GVM虚拟机 4](#_Toc30103829)

[3.1合约部署 4](#_Toc30103830)

[3.1.1合约准备 4](#_Toc30103831)

[3.1.2合约编译&部署 8](#_Toc30103832)

[3.2合约方法的调用 15](#_Toc30103833)

# 1、引言

## 1.1编写目的

本文档提供的对象主要是solidity语言的智能合约项目设计、开发与测试人员、平台维护与运营人员，以及第三方开发者。本文档对GVM虚拟机的操作进行介绍，为用户操作、使用solidity语言开发GVM虚拟机的智能合约提供参考和指导。

## 1.2概念解释

**GVM：**虚拟机

GVM虚拟机，完美兼容了EVM虚拟机，在以太坊上开发的智能合约，可以不用任何更改的移植到GVM中，另外在GVM合约中还可以使用标准通证。

通过单独构建模块化虚拟机—— Virtual Machine（以下简称 GVM）运行智能合约，这样的设计可带来两个好处。一是优化 GVM 性能直接提升合约执行效率，减少系统耦合带来的干扰因素；二是弱化区块链网络与智能合约运行状态的相关性，即便合约执行出现问题，或虚拟 机运行异常，区块链网络的稳定性依然能够保证。

GVM 包含四个模块。合约通过用户端（Console-User）以命令行或RPC的形式进行编写。Cli/RPC是合约的处理模块，负责接收，并将输入传递到中间层，还负责将底层处理完的结果反馈给用户端。RPC（Remote Procedure Call）模块负责接收来自区块链网络的执行请求，并将请求发送到中间层，待合约执行完成之后将结果返回给区块链网络。中间层（Mid-Ware）负责将 Cli/RPC 和 RPC 传来的命令和请求同步传递给底层的编译器和执行器进行编译，执行。并将编译执行结果返回给Cli/RPC或RPC。编译执行器（Compiler & Actuator）负责编译，运行执行环境，接收和执行脚本，并将执行结果反馈给中间层。一个活跃的区块链网络，合约调用非常频繁，为确保合约能够稳定而高效地运行。

# 2、GVM与EVM差异说明

## 2.1 ABI差异说明

1. Blockhash：

GVM：可以获取到当前区块高度hash以及已经产生的任意高度区块的hash

EVM：只能获取到当前区块 -1 至 当前区块 -256 高度的hash；

1. block.coinbase:

GVM：获取到的是当前区块的提案人地址（主链没有挖矿）；

EVM：获取的是当前区块的挖矿地址；

1. block.difficulty:

GVM：链没有挖矿，所以block.difficulty默认设置为0；

EVM：当前区块的出块难度；

## 2.2 其他差异说明

1. 币种精度

GVM：GVM：币种的最小单位为cong(1 = 1\*109)，

为了兼容EVM合约中的 wei 关键字，我们对GVM的账户余额进行了扩展，增加了GVMBalanceMod字段用于记录十八位精度的后九位数值；

EVM：币种精度(1 eth = 1\*1018)；

type AccountInfo struct {

Address types.Address `json:"address"`

Balance bn.Number `json:"balance"`

GVMBalanceMod bn.Number `json:"bbMod,omitempty"` // 十八位精度的后九位

}

1. 部署合约

GVM：在部署合约时，需要指定该合约支持的标准通证；

EVM：在部署合约时，无法指定，默认使用eth。

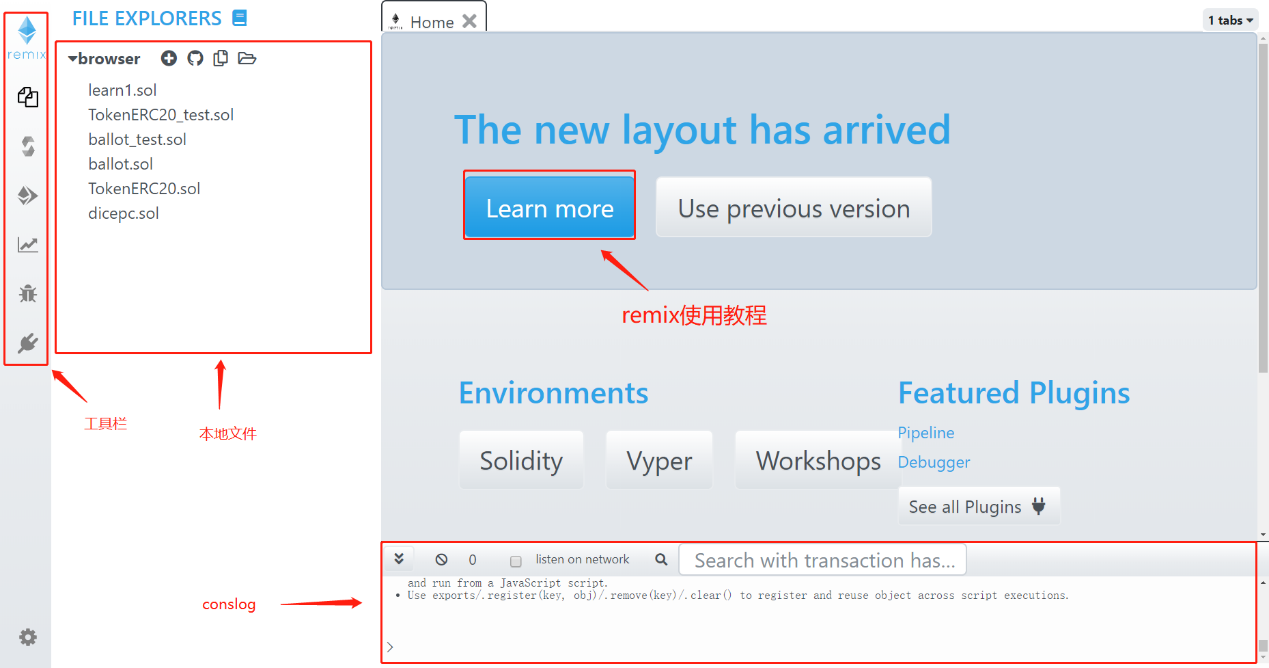
# 3、关于GVM虚拟机

## 3.1合约部署

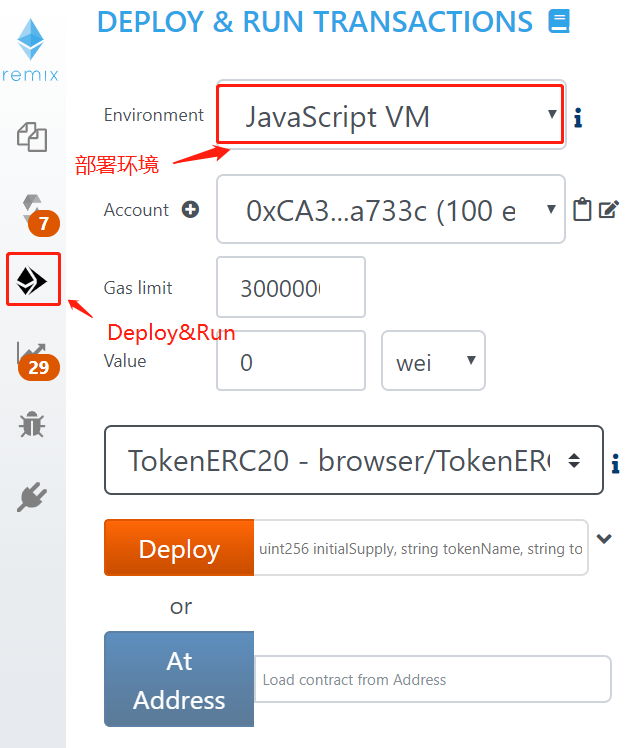
### 3.1.1合约准备

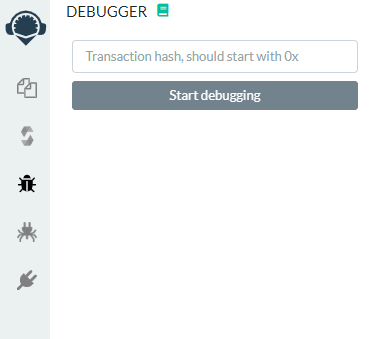
1. 合约的开发
2. 使用网页端编辑器Remix <http://remix.ethereum.org/> 进行开发

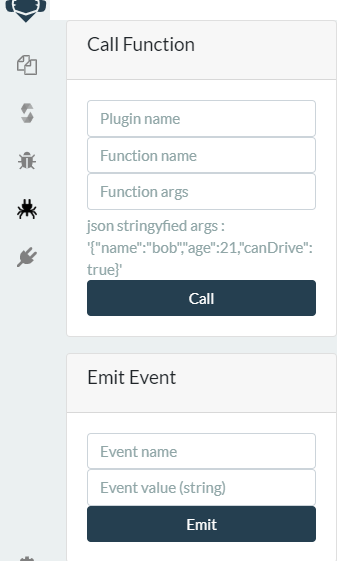
GVM虚拟机支持solidity语言且兼容EVM虚拟机，建议开发者使用 Remix IDE 进行智能合约的开发调试。进入Remix首页后，您将看到下图页面，可以点击 Learn more 学习使用Remix。

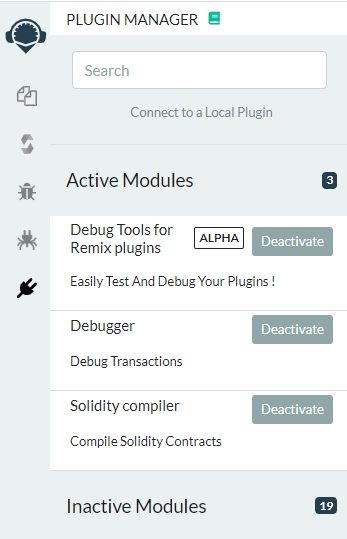


合约编写完成后，在进行模拟部署时，建议选择 JavaScript VM 环境，然后执行Deploy&Run进行部署运行。









1. 使用客户端编辑器Remix-IDE

### 3.1.2合约编译&部署

在合约编译和部署之前，我们默认已经搭建好了最新版本的带有GVM虚拟机的区块链。

GICToolBox工具提供了两种方式对solidity来编译及部署:

Deploy solidity smart contract with information

Usage:

gic solDeploy [flags]

Flags:

-i, --abiFile string path of solidity contract abi file

-b, --binFile string path of solidity contract binaryCode file

-c, --chainid string chainid define blockchain for this invoke

-g, --gasLimit string gas limit for now transaction

-h, --help help for solDeploy

-k, --keystorepath string path of keystore (default ".keystore")

-n, --name string name of wallet

-e, --note string note for tx

-r, --paramsArr stringArray parameters array with construction method

-p, --password string password of wallet

-f, --sourceFile string path of solidity contract source file

-a, --tokenAddr string address of token

-t, --tokenName string name of token

1. 直接对 solidity文件即源码进行编译并部署
2. 主链配置：根据ChainID不同，修改gic.yaml文件的defaultChainID字段，同时修改或添加对应的IP地址

配置文件示例：

# 默认配置

defaultChainID: "devtest"

# 区块链知名入口

urls: !!map

devtest:

- "http://192.168.1.162:46657"

1. 这里使用编写的一个测试合约来举例说明。合约Test.solidity源码如下：

pragma solidity ^0.4.25;

contract Test {

function senders() public view returns (address){

return (msg.sender);

}

function data(address \_data) public pure returns(address){

return \_data;

}

function addmods(uint a,uint b,uint c) public pure returns(uint){

uint num = addmod(a,b,c);

return num;

}

function FFF(uint a,uint32[] b,bytes10 c,bytes d) public view returns (uint, uint32[], bytes10, bytes){

return (a, b, c, d);

}

function sam(bytes a,bool b,uint256[] c) public view returns (bytes ,bool ,uint256[] ){

return (a,b,c);

}

}

1. 编译&部署指令，使用gic的solDeploy命令

./gic solDeploy -f ./Test.solidity -a localHLDhqyoJ1vRuo8re9GxFAFU2Bksw2kNup -n owner -p Ab1@Cd3$ -g 10000000

**Output SUCCESS Example：**

OK

Response: {

"code": 200,

"log": "",

"fee": 712602500,

"txHash": "0x6BBD52055AE7F4918FDB6CF2276F61E74D19FE4D693F853763107249ED6C0363",

"height": 15067,

"data": "local6MY9FxJAkgjmmKcQ3Hy3b49KU34sENVfh"

}

1. 直接部署Remix编辑器编译生成的二进制文件
2. 先通过Remix编辑器编译

**合约编译生成的二进制文件Test.bin的内容如下：**



**合约编译生成的二进制文件Test.abi的内容如下**：

[

{

"constant": true,

"inputs": [

{

"name": "a",

"type": "uint256"

},

{

"name": "b",

"type": "uint256"

},

{

"name": "c",

"type": "uint256"

}

],

"name": "addmods",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "pure",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "senders",

"outputs": [

{

"name": "",

"type": "address"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "a",

"type": "bytes"

},

{

"name": "b",

"type": "bool"

},

{

"name": "c",

"type": "uint256[]"

}

],

"name": "sam",

"outputs": [

{

"name": "",

"type": "bytes"

},

{

"name": "",

"type": "bool"

},

{

"name": "",

"type": "uint256[]"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "\_data",

"type": "address"

}

],

"name": "data",

"outputs": [

{

"name": "",

"type": "address"

}

],

"payable": false,

"stateMutability": "pure",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "a",

"type": "uint256"

},

{

"name": "b",

"type": "uint32[]"

},

{

"name": "c",

"type": "bytes10"

},

{

"name": "d",

"type": "bytes"

}

],

"name": "FFF",

"outputs": [

{

"name": "",

"type": "uint256"

},

{

"name": "",

"type": "uint32[]"

},

{

"name": "",

"type": "bytes10"

},

{

"name": "",

"type": "bytes"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

}

]

1. 部署指令

gic solDeploy -i ./Test.abi -b ./Test.bin -a localHLDhqyoJ1vRuo8re9GxFAFU2Bksw2kNup -n owner -p Ab1@Cd3$ -g 10000000

**Output SUCCESS Example：**

OK

Response: {

"code": 200,

"log": "",

"fee": 712602500,

"txHash": "0x6BBD52055AE7F4918FDB6CF2276F61E74D19FE4D693F853763107249ED6C0363",

"height": 15067,

"data": "local6MY9FxJAkgjmmKcQ3Hy3b49KU34sENVfh"

}

## **3.2合约方法的调用**

通过gic工具可以对GVM内部署的合约方法进行调用:

Call solidity contract method with require params

Usage:

gic solCall [flags]

Flags:

-i, --abiFile string path of solidity contract abi file

-c, --chainid string chainid define blockchain for this invoke

-a, --contractAddr string address of contract

-g, --gasLimit string gas limit for now transaction

-h, --help help for solCall

-k, --keystorepath string path of keystore (default ".keystore")

-m, --method string method that it invoked with contract

-n, --name string name of wallet

-e, --note string note for tx

-r, --paramsArr stringArray parameters array with call method

-p, --password string password of wallet

-v, --value string value of transfer

1. 调用senders(）方法

./gic solCall -n owner -p Ab1@Cd3$ -a local6MY9FxJAkgjmmKcQ3Hy3b49KU34sENVfh -m senders -i test.abi -g 100000

**Output SUCCESS Example：**

OK

Response: {

"code": 201,

"log": "",

"txHash": "0xAF4B711F058E7CE88AC9C9F8600BEF3AE7A616A151FA3854GIA7D8BF66567DF2",

"data": "eyIiOlsxNDcsMTQ3LDE0NSwyNTUsMTQwLDEyMSwxMzEsNzIsMTcsODAsMjEsNDEsMzUsMTQxLDE3NiwzMCw2Miw0Nyw5MCwzMV19"

}

1. 调用data(address）方法

./gic solCall -n owner -p Ab1@Cd3$ -a local6MY9FxJAkgjmmKcQ3Hy3b49KU34sENVfh -m data -i test.abi -r localNHjSW42A6pqN2mWS3B4kJVUSWubpCvQu6 -g 100000

**Output SUCCESS Example：**

OK

Response: {

"code": 201,

"log": "",

"txHash": "0x549D0096F86B518F9A1EFD1B532AE43EAA1CF424B309571A0C0EE80D64DC77A9",

"data": "eyIiOlsyMzMsMTMyLDE4NSw2OCwxMTgsMjI1LDEwMyw4NiwxMzQsMTg3LDEyMSwyMDMsMTk4LDI0LDExNSwyMDcsMjksNTUsMTQyLDQ1XX0="

}

1. 调用FFF(uint a,uint32[] b,bytes10 c,bytes d)方法

./gic solCall -n owner -p Ab1@Cd3$ -a local6MY9FxJAkgjmmKcQ3Hy3b49KU34sENVfh -m FFF -i test.abi -r 123 -r "[234]" -r "0xabbgi" -r "0xasdfaasdf" -g 100000

**Output SUCCESS Example：**

OK

Response: {

"code": 201,

"log": "",

"txHash": "0xE1005DE5AF7081D4E0DE2D55011AD1C97DCAE89589755322B75410C64554C888",

"data": "eyIiOiJNSGhoYzJSbVlXRnpaR1k9In0="

}

1. 调用sam(bytes a,bool b,uint256[] c)方法

./gic solCall -n owner -p Ab1@Cd3$ -a local6MY9FxJAkgjmmKcQ3Hy3b49KU34sENVfh -m sam -i test.abi -r "0xasdfasd" -r true -r "[123,234]" -g 100000

**Output SUCCESS Example：**

OK

Response: {

"code": 201,

"log": "",

"txHash": "0x3BF198616AAE296AEFA57E28B1CFF4FFCE232C8DA2565D2CE3B9D8424DA04412",

"data": "eyIiOlsxMjMsMjM0XX0="

}

注 : wallet账户、参数构造根据实际情况进行修改。

1.