# 一、FEVM部署智能合约和调用

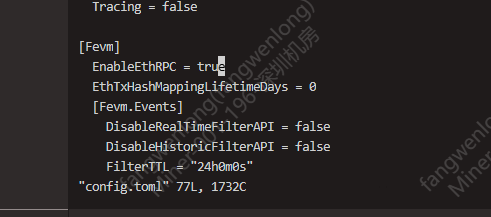
## 1 Lotus智能合约部署

参考地址

<https://lotus.filecoin.io/lotus/developers/local-network/>

需要将lotus的配置目录LOTUS\_PATH下的config.toml文件进行修改

EnableEthRPC = true



然后将lotus重启

创建f4地址

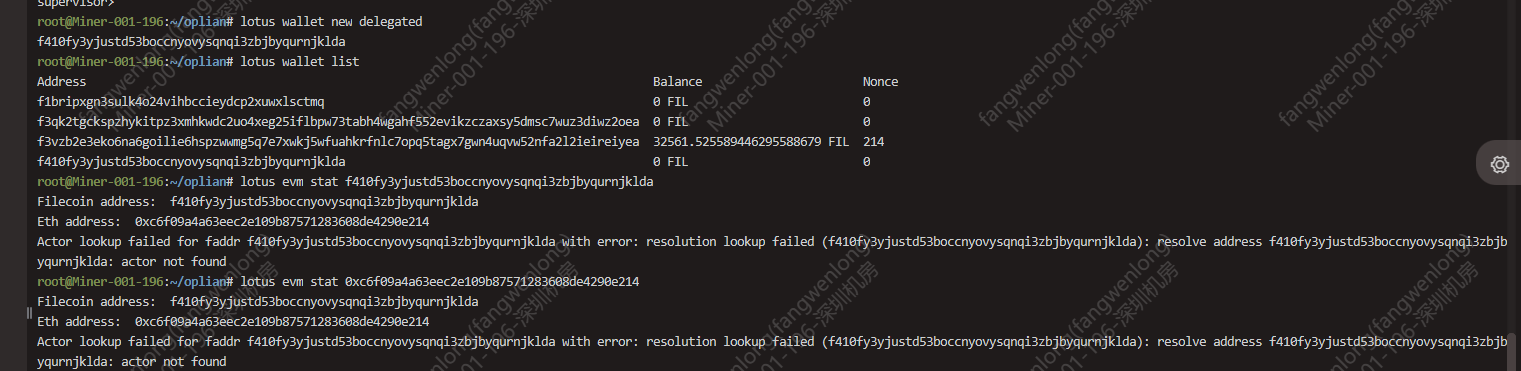
lotus wallet new delegated

查看f4地址对应的eth地址

lotus evm stat f410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda

查看ETH地址对应的f4地址

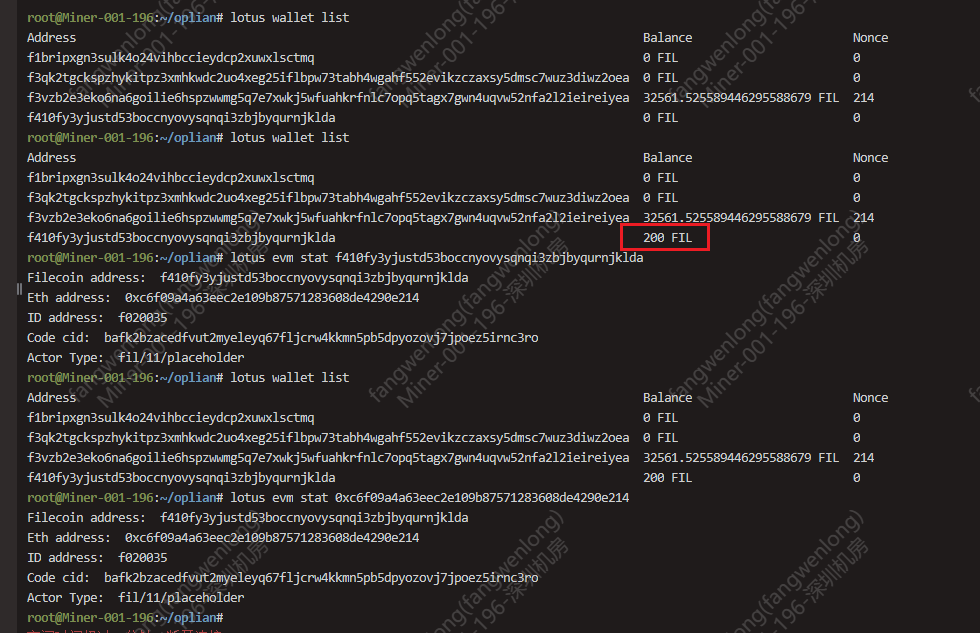
lotus evm stat 0xc6f09a4a63eec2e109b87571283608de4290e214



直接请求会发现会报错误

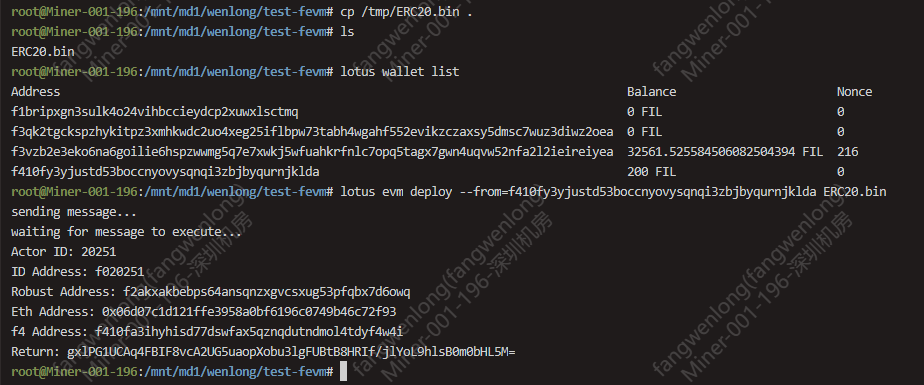
Actor lookup failed for faddr f410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda with error: resolution lookup failed (f410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda): resolve address f410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda: actor not found

需要我们将对应地址充值一定金额，等待一会儿，即可正常查看

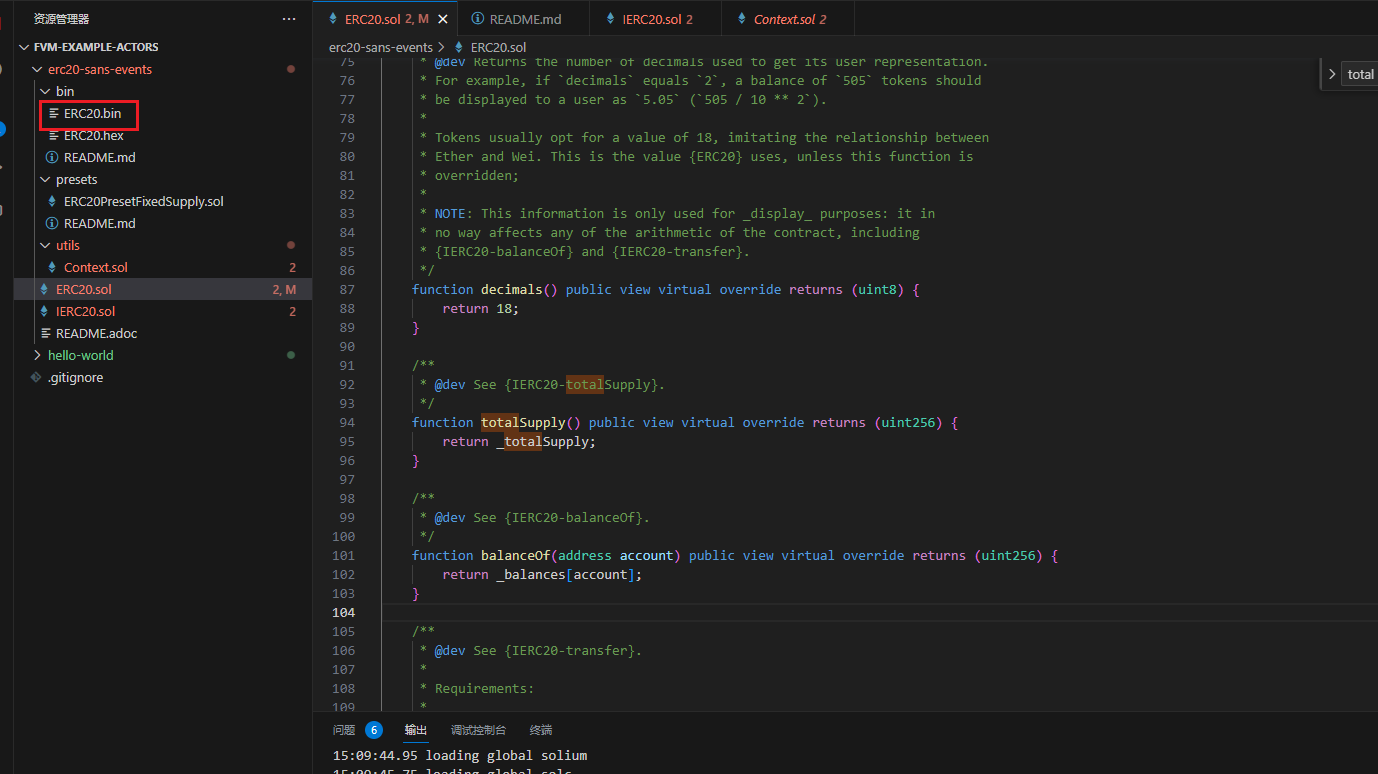


智能合约的部署

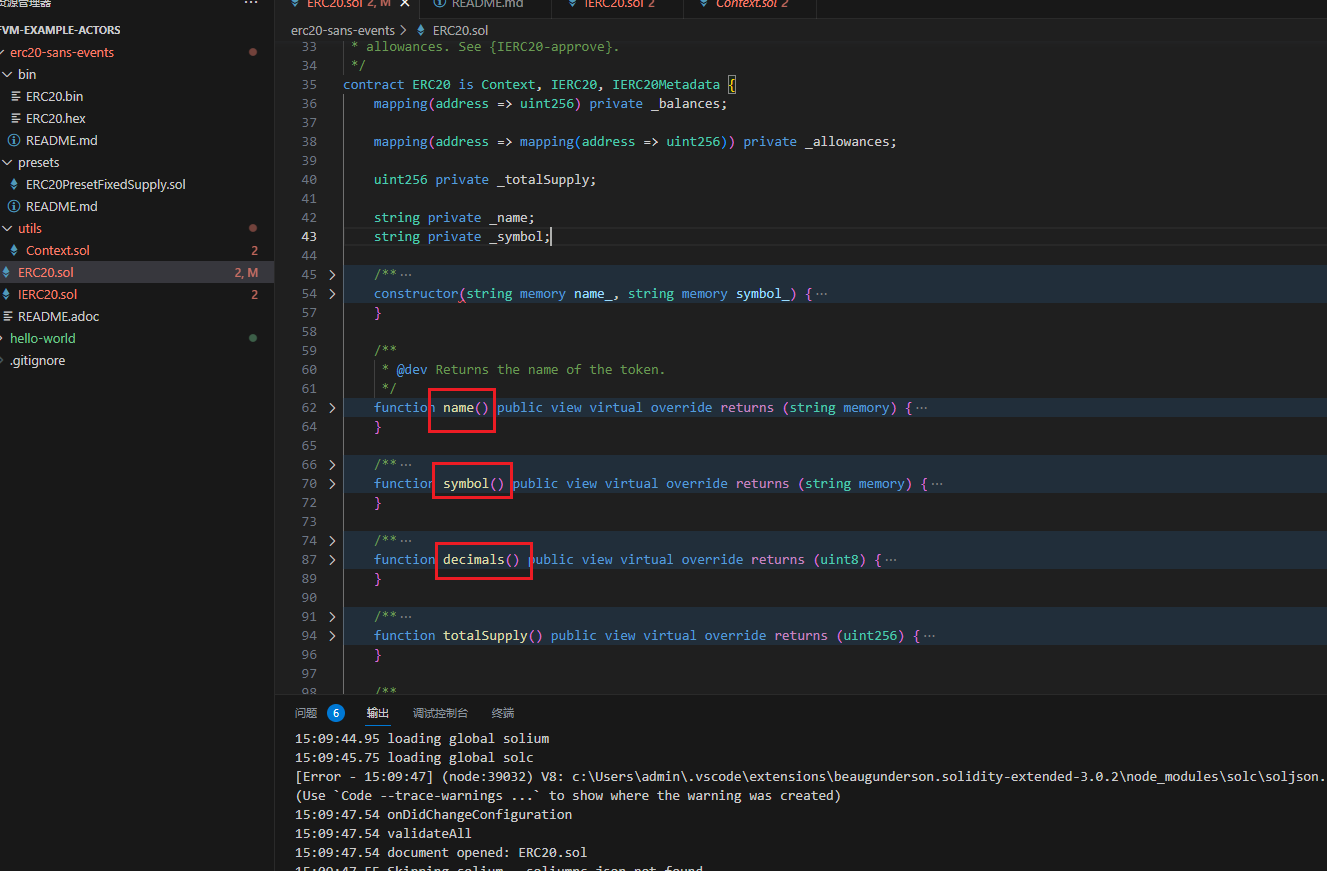
lotus evm deploy --from=f410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda ERC20.bin



其中bin文件为对应基于ERC20生成的文件



其中的方法为：



## 2 Lotus智能合约调用

调用智能合约的指令为：

lotus evm invoke --from=f410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda f020251 06fdde03

--from 拼接对应的f4地址

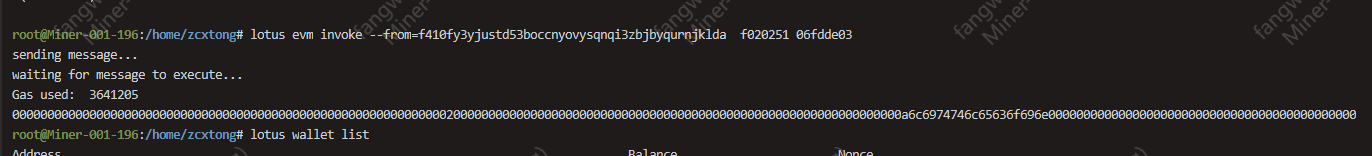
f020251为生成的对应的ID address

06fdde03 为函数ID，对应方法的abi编码后的取前四位



请求对应的方法：

06fdde03 对应name()方法，显示代币名称



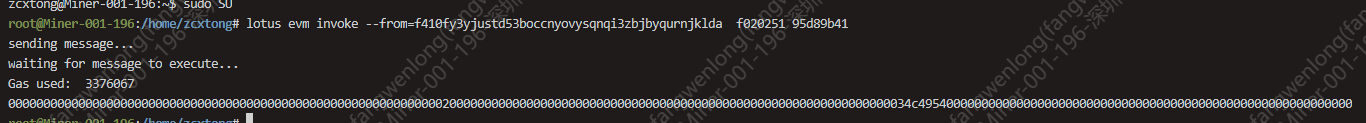
对应返回值为：

0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000a6c6974746c65636f696e00000000000000000000000000000000000000000000

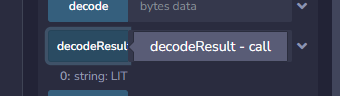
Abi解码结果为



95d89b41 对应symbol()方法,显示代币符号



000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000000034c49540000000000000000000000000000000000000000000000000000000000



Abi编码解码

对应代码:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract ABIEncode{

    function encodeSelector(string memory data) external pure returns(bytes4){

        bytes memory \_ba = bytes(data);

        return bytes4(keccak256(\_ba));

    }

    function decodestring(bytes memory data) public pure returns(string memory) {

        return abi.decode(data, (string));

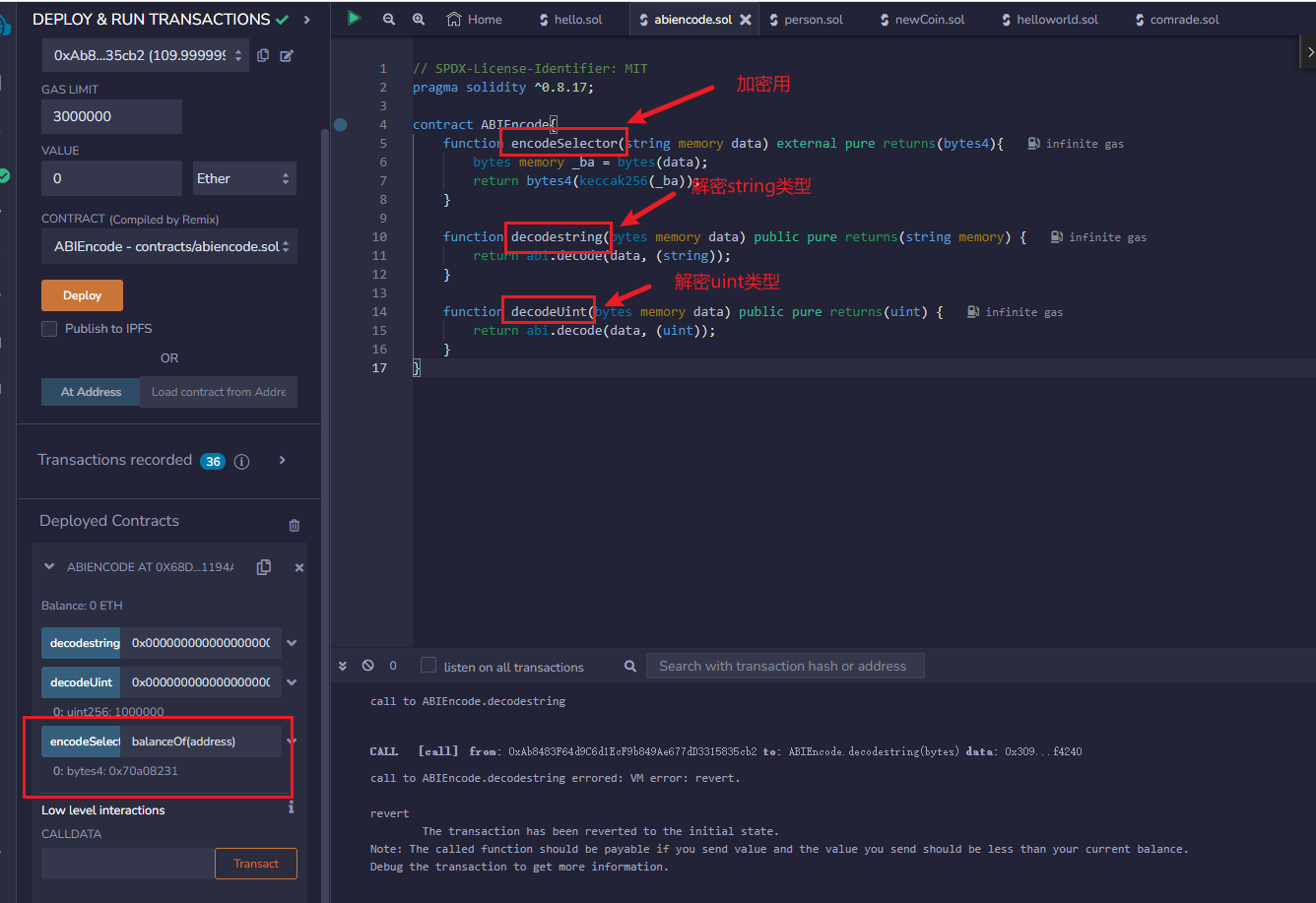
    }

    function decodeUint(bytes memory data) public pure returns(uint) {

        return abi.decode(data, (uint));

    }

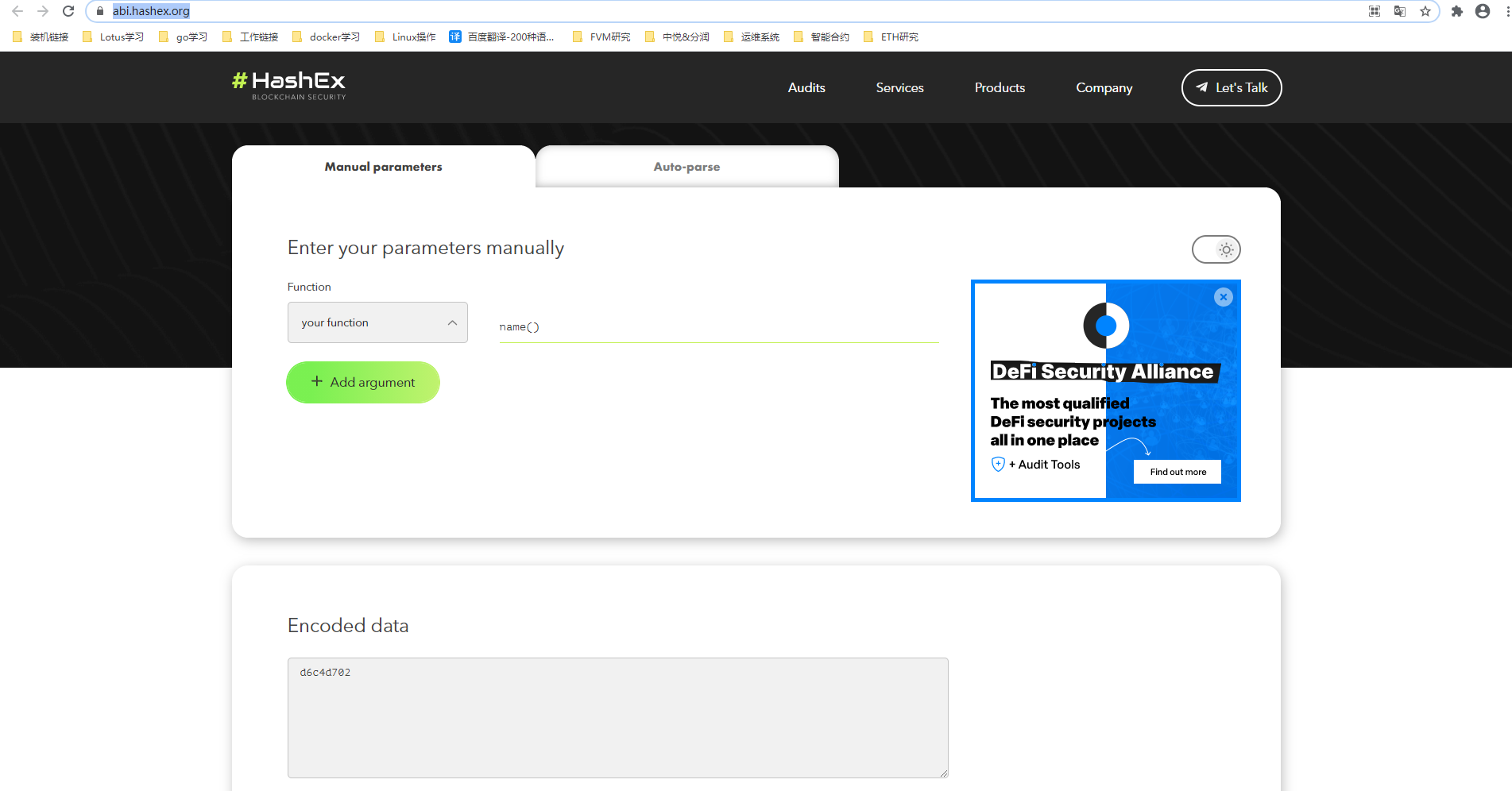
}



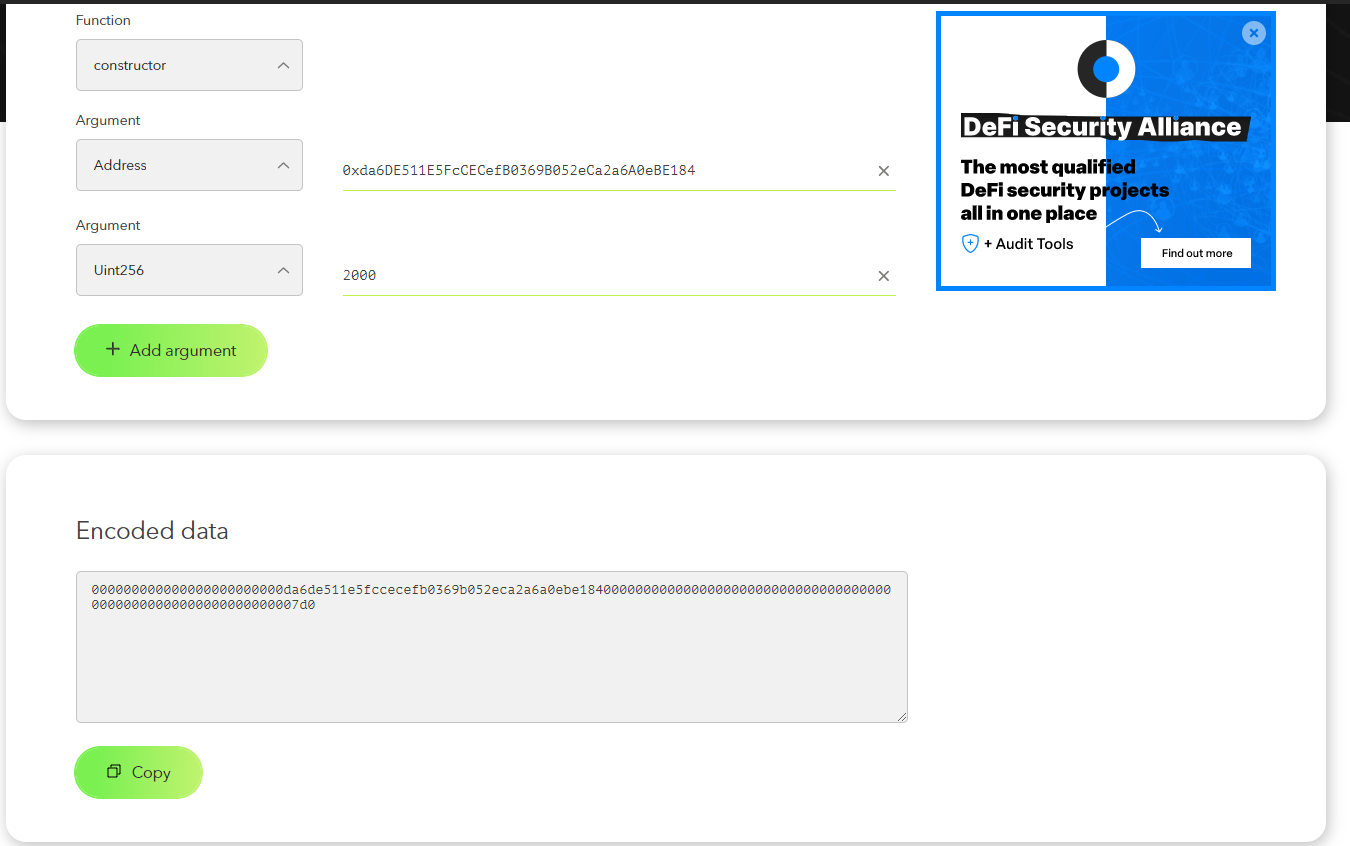
## 3 工具收集

一个方便构建入参的在线工具

<https://abi.hashex.org/>



可以借助录入对应的参数拼接对应的入参，省去很多麻烦



# 二、Go连接lotus智能合约

## 1 Go连接代码

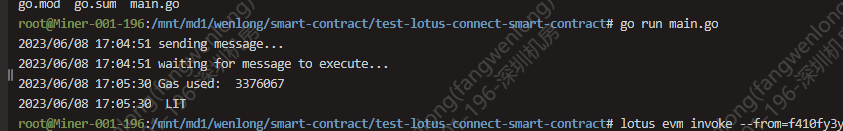
仿照lotus里面的关于evm的写法，借助模仿其传参，即可完成对应的lotus部署的智能合约的连接

package main

import (  
 "bytes"  
 "context"  
 "github.com/filecoin-project/go-address"  
 "github.com/filecoin-project/go-jsonrpc"  
 "github.com/filecoin-project/go-state-types/abi"  
 builtintypes "github.com/filecoin-project/go-state-types/builtin"  
 "github.com/filecoin-project/lotus/api/client"  
 "github.com/filecoin-project/lotus/api/v0api"  
 "github.com/filecoin-project/lotus/chain/types"  
 "github.com/filecoin-project/lotus/chain/types/ethtypes"  
 cbg "github.com/whyrusleeping/cbor-gen"  
 "log"  
 "net/http"  
)  
  
func main() {  
 // 连接rpc  
 fullApi, close1, err := NewLotusApi()  
 if err != nil {  
 return  
 }  
 defer close1()  
  
 var fromAddr address.Address  
 var toAddr address.Address  
 toAddr, err = address.NewFromString("f020801") //对应合约的id  
 if err != nil {  
 return  
 }  
 fromAddr, err = address.NewFromString("f410f3jw6kepf7thm56ydngyff3fcu2qoxymeowrzura") //对应调取的地址  
 if err != nil {  
 return  
 }  
 var calldata []byte  
 calldata, err = ethtypes.DecodeHexStringTrimSpace("95d89b41") //对应方法的abi编码  
 //calldata, err = ethtypes.DecodeHexString(hex.EncodeToString([]byte("symbol()")[:8]))  
 if err != nil {  
 log.Println("decoding hex input data: %w", err)  
 return  
 }  
  
 var buffer bytes.Buffer  
 if err := cbg.WriteByteArray(&buffer, calldata); err != nil {  
 log.Println("failed to encode evm params as cbor: %w", err)  
 return  
 }  
 calldata = buffer.Bytes()  
  
 msg := &types.Message{  
 To: toAddr,  
 From: fromAddr,  
 Value: abi.NewTokenAmount(0),  
 Method: builtintypes.MethodsEVM.InvokeContract,  
 Params: calldata,  
 }  
  
 log.Println("sending message...")  
 smsg, err := fullApi.MpoolPushMessage(context.Background(), msg, nil)  
 if err != nil {  
 log.Println("failed to push message: %w", err)  
 return  
 }  
 log.Println("waiting for message to execute...")  
 wait, err := fullApi.StateWaitMsg(context.Background(), smsg.Cid(), 0)  
 if err != nil {  
 log.Println("error waiting for message: %w", err)  
 return  
 }  
 // check it executed successfully  
 if wait.Receipt.ExitCode != 0 {  
 log.Println("actor execution failed")  
 return  
 }  
  
 log.Println("Gas used: ", wait.Receipt.GasUsed)  
 result, err := cbg.ReadByteArray(bytes.NewBuffer(wait.Receipt.Return), uint64(len(wait.Receipt.Return)))  
 if err != nil {  
 log.Println("evm result not correctly encoded: %w", err)  
 return  
 }  
  
 if len(result) > 0 {  
 log.Println(string(result))  
 } else {  
 log.Println("OK")  
 }  
}

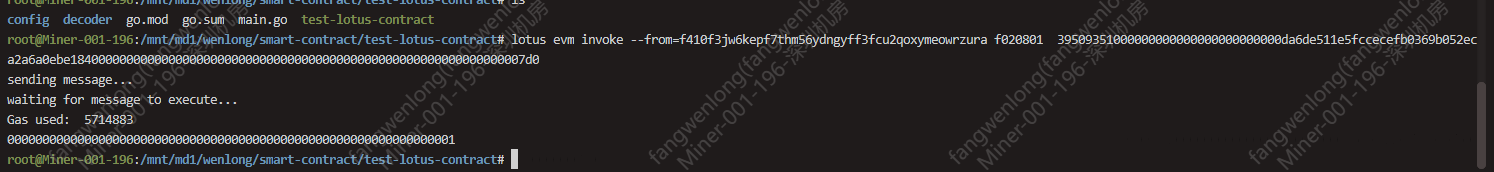
//连接lotus  
func NewLotusApi() (v0api.FullNode, jsonrpc.ClientCloser, error) {  
 headers := http.Header{"Authorization": []string{"Bearer " + "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJBbGxvdyI6WyJyZWFkIiwid3JpdGUiLCJzaWduIiwiYWRtaW4iXX0.L54SnCOpLk\_-U9W30KfHrVZld4j6L3gsrzfoJal2ncU"}}  
 //var err error  
 lapi, cl, err := client.NewFullNodeRPCV0(context.Background(), "ws://"+"10.0.1.196:50002/rpc/v0", headers)  
 if err != nil {  
 log.Println("connecting with lotus failed: ", err.Error())  
 return nil, nil, err  
 }  
 return lapi, cl, nil  
}

调取结果为:



## 2 入参和返回参数abi解析

### 2.1 入参构造



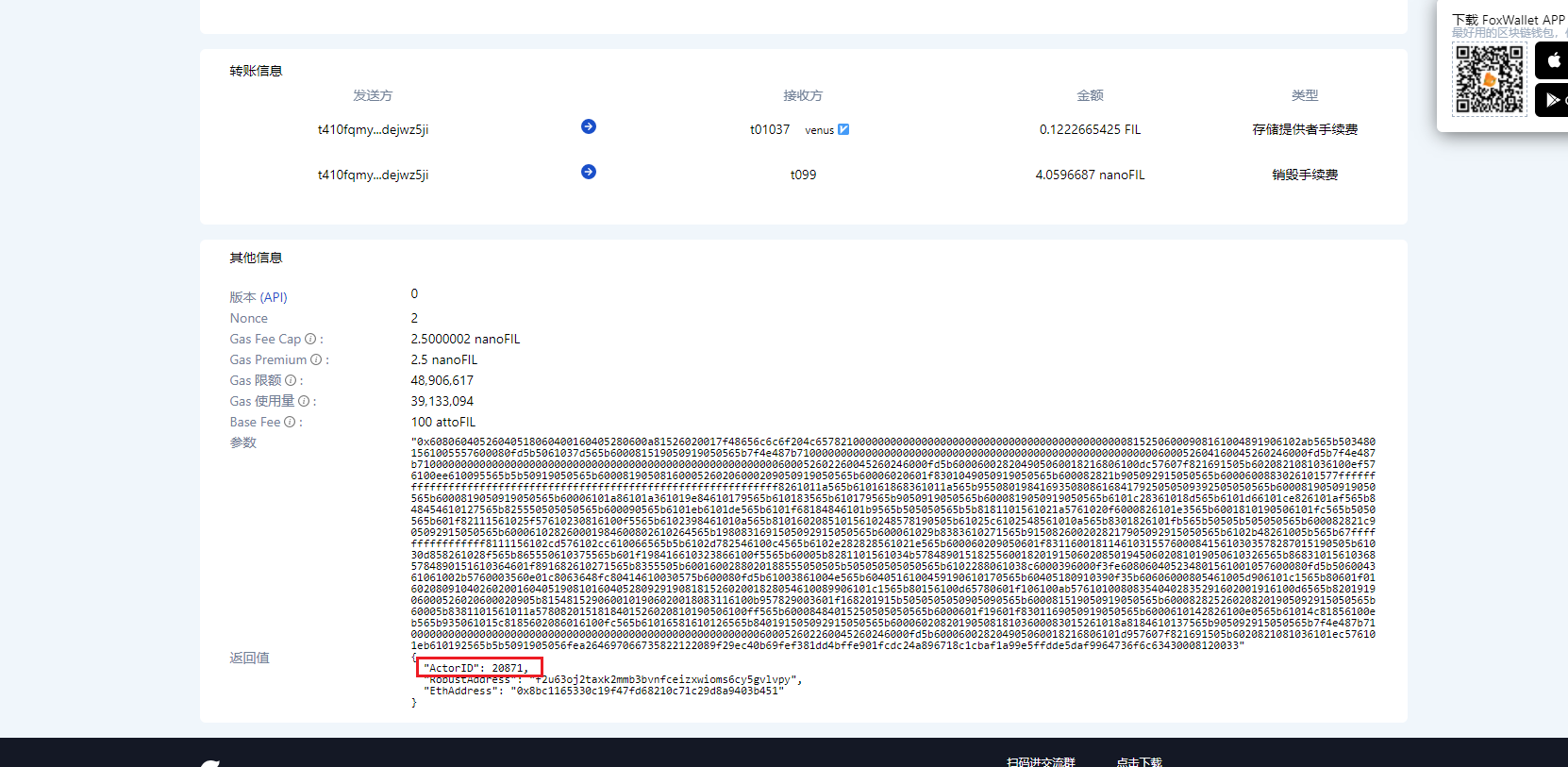
lotus evm invoke --from=f410f3jw6kepf7thm56ydngyff3fcu2qoxymeowrzura f020801 39509351000000000000000000000000da6de511e5fccecefb0369b052eca2a6a0ebe18400000000000000000000000000000000000000000000000000000000000007d0

根据调用的指令：

--from 对应地址，可以选择指定用于发送exec消息的帐户

f020801 为对应的智能合约id

部署后在区块链浏览器可查



39509351000000000000000000000000da6de511e5fccecefb0369b052eca2a6a0ebe18400000000000000000000000000000000000000000000000000000000000007d0

为对应的方法id拼接所要传递的参数

这里的是方法 increaseAllowance(address,uint256)

参数 address 0xda6DE511E5FcCECefB0369B052eCa2a6A0eBE184

uint256 2000

ABI编码方法如下：

package main

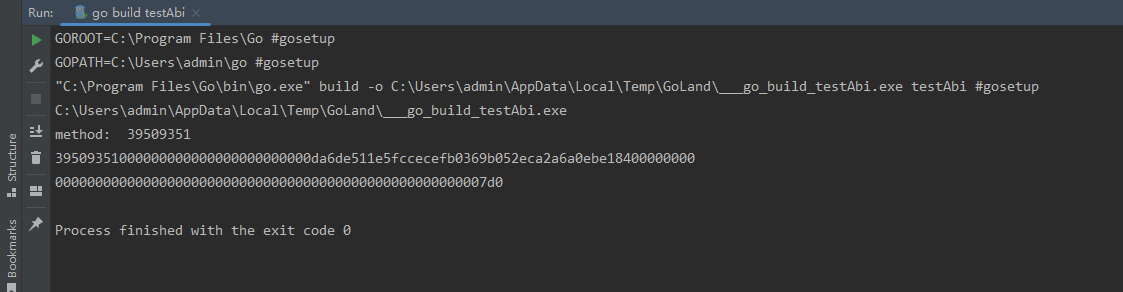
import (  
 "encoding/hex"  
 "fmt"  
 "github.com/ethereum/go-ethereum/common"  
 "github.com/ethereum/go-ethereum/common/hexutil"  
 "github.com/ethereum/go-ethereum/crypto"  
 "math/big"  
)  
  
func main() {  
 method := []byte("increaseAllowance(address,uint256)")  
 method = crypto.Keccak256(method)[:4] //获取方法id  
 fmt.Println("method: ", hexutil.Encode(method)[2:])

data := []byte{}  
 to := common.HexToAddress("0xda6DE511E5FcCECefB0369B052eCa2a6A0eBE184")  
 paddedAddress := common.LeftPadBytes(to.Bytes(), 32) //获取address的编码  
 amount, \_ := new(big.Int).SetString("2000", 10) //10  
 paddedAmount := common.LeftPadBytes(amount.Bytes(), 32) //获取数量的编码  
 //拼接对应方法和参数

data = append(data, method...)   
 data = append(data, paddedAddress...)  
 data = append(data, paddedAmount...)  
  
 fmt.Println(hex.EncodeToString(data))  
}

对应输出结果为：

输出的结果是没有换行的，不清楚这个的打印怎么换行了，可能是由于数据太长的缘故。

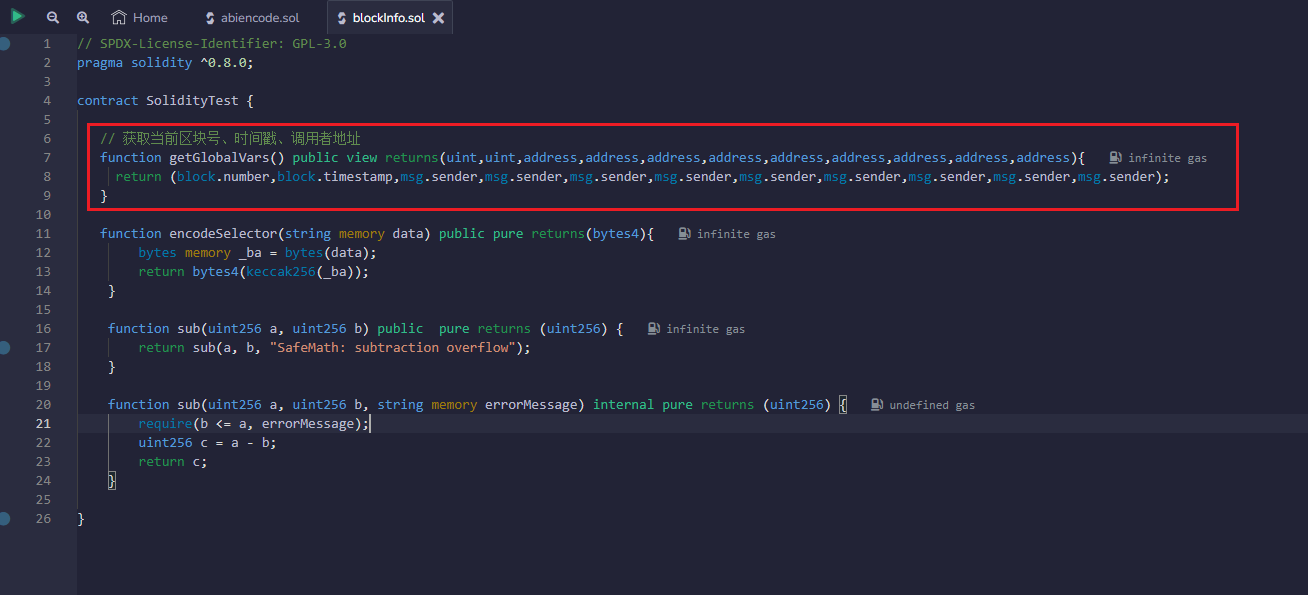


借助这种方式，根据对应的函数传参的顺序进行拼接即可生成对应函数的传参

### 2.2 出参解析

#### 2.2.1 出参展示

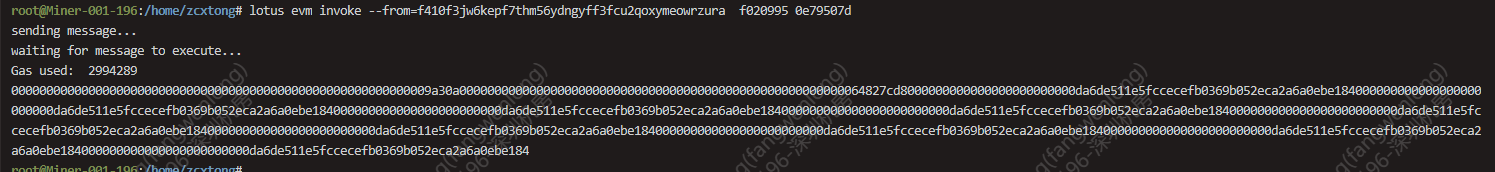
这里构建一个返回信息比较多的智能合约



对应方法为getGlobalVars()

返回信息为包含的参数为：

uint,uint,address,address,address,address,address,address,address,address,address



对应数据：



其中包含两个uint类型和8个address的信息

要获取对应信息，就需要我们对其进行abi解析

#### 2.2.2 go代码展示

对应decoder.go代码

用以进行返回的abi解析

package decoder

import (  
 "encoding/hex"  
 "fmt"  
 "log"  
 "strings"  
  
 "github.com/ethereum/go-ethereum/accounts/abi"  
 "github.com/ethereum/go-ethereum/common"  
 "github.com/ethereum/go-ethereum/core/types"  
)  
  
type DecodedLog struct {  
 Name string  
 Params []Param  
 Address common.Address // contract address  
}  
  
type Param struct {  
 Name string  
 Value interface{}  
 Type string  
}  
type MethodData struct {  
 Name string  
 Params []Param  
}  
  
type OutputData struct {  
 Params []Param  
}  
  
// ABIDecoder ethereum transaction data decoder  
type ABIDecoder struct {  
 myabi abi.ABI  
}  
  
func NewABIDecoder() \*ABIDecoder {  
 return &ABIDecoder{}  
}  
  
func (d \*ABIDecoder) SetABI(contractAbi string) {  
 myabi, err := abi.JSON(strings.NewReader(contractAbi))  
 if err != nil {  
 log.Fatal(err)  
 }  
 d.myabi = myabi  
}  
  
func (d \*ABIDecoder) DecodeMethod(txData string) (MethodData, error) {  
 if strings.HasPrefix(txData, "0x") {  
 txData = txData[2:]  
 }  
  
 decodedSig, err := hex.DecodeString(txData[:8])  
 if err != nil {  
 return MethodData{}, err  
 }  
  
 method, err := d.myabi.MethodById(decodedSig)  
 if err != nil {  
 return MethodData{}, err  
 }  
  
 decodedData, err := hex.DecodeString(txData[8:])  
 if err != nil {  
 return MethodData{}, err  
 }  
  
 inputs, err := method.Inputs.Unpack(decodedData)  
 if err != nil {  
 return MethodData{}, err  
 }  
  
 nonIndexedArgs := method.Inputs.NonIndexed()  
  
 retData := MethodData{}  
 retData.Name = method.Name  
 for i, input := range inputs {  
 arg := nonIndexedArgs[i]  
 param := Param{  
 Name: arg.Name,  
 Value: fmt.Sprintf("%v", input),  
 Type: arg.Type.String(),  
 }  
 retData.Params = append(retData.Params, param)  
 }  
  
 return retData, nil  
}  
  
func (d \*ABIDecoder) DecodeOutPut(methodData, txData string) (OutputData, error) {  
 if strings.HasPrefix(txData, "0x") {  
 txData = txData[2:]  
 }  
  
 decodedSig, err := hex.DecodeString(methodData)  
 if err != nil {  
 return OutputData{}, err  
 }  
  
 method, err := d.myabi.MethodById(decodedSig)  
 if err != nil {  
 return OutputData{}, err  
 }  
  
 decodedData, err := hex.DecodeString(txData)  
 if err != nil {  
 return OutputData{}, err  
 }  
  
 inputs, err := method.Outputs.Unpack(decodedData)  
 if err != nil {  
 return OutputData{}, err  
 }  
  
 nonIndexedArgs := method.Outputs.NonIndexed()  
  
 retData := OutputData{}  
 for i, input := range inputs {  
 arg := nonIndexedArgs[i]  
 param := Param{  
 Name: arg.Name,  
 Value: input,  
 Type: arg.Type.String(),  
 }  
 retData.Params = append(retData.Params, param)  
 }  
  
 return retData, nil  
}  
  
// DecodeLogs decode contract events from log  
func (d \*ABIDecoder) DecodeLogs(logs []\*types.Log) ([]DecodedLog, error) {  
 decodeLogs := make([]DecodedLog, 0, len(logs))  
  
 for \_, logItem := range logs {  
 decodedLog := DecodedLog{}  
 decodedLog.Address = logItem.Address  
  
 event, err := d.myabi.EventByID(logItem.Topics[0])  
 if err != nil {  
 return nil, err  
 }  
 decodedLog.Name = event.Name  
 dataList, err := d.myabi.Unpack(event.Name, logItem.Data)  
 if err != nil {  
 return nil, err  
 }  
  
 params := make([]Param, 0, len(event.Inputs))  
 topicIndex := 1 //indexed value are put in topic  
 dataIndex := 0 // no indexed value are put in data  
 for \_, input := range event.Inputs {  
 param := Param{}  
  
 param.Name = input.Name  
 param.Type = input.Type.String()  
 var value interface{}  
 if input.Indexed {  
 value = logItem.Topics[topicIndex]  
 topicIndex++  
 } else {  
 value = dataList[dataIndex]  
 dataIndex++  
 }  
 param.Value = fmt.Sprintf("%v", value)  
  
 params = append(params, param)  
 }  
 decodedLog.Params = params  
  
 decodeLogs = append(decodeLogs, decodedLog)  
 }  
 return decodeLogs, nil  
}  
  
func (d \*ABIDecoder) ABI() abi.ABI {  
 return d.myabi  
}

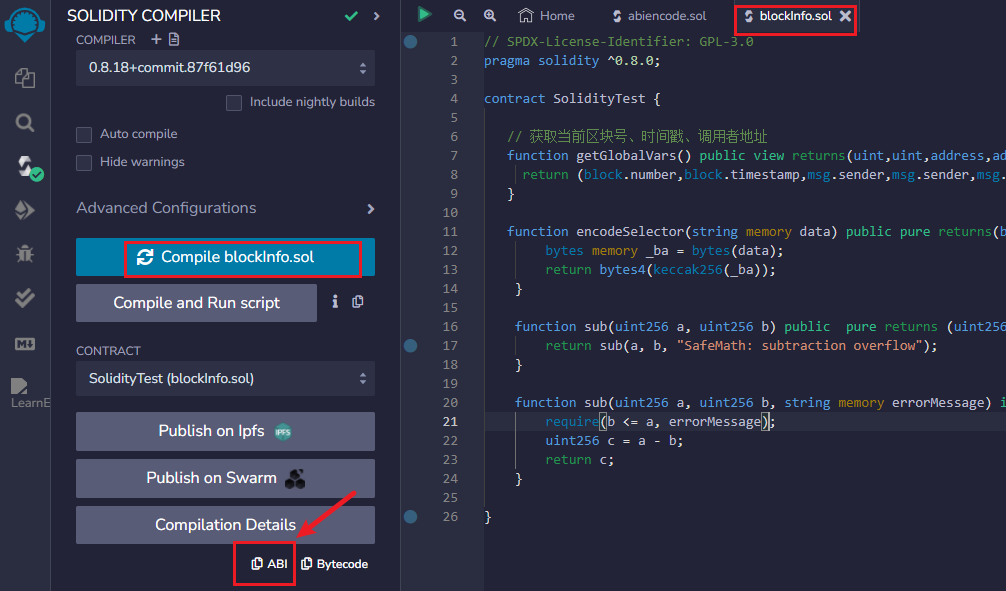
对应的main.go文件

package main

import (  
 "fmt"  
 "log"  
 "reflect"  
 "testAbi/decoder"  
)  
  
var myContractAbi1 = `  
[  
 {  
 "inputs": [  
 {  
 "internalType": "string",  
 "name": "data",  
 "type": "string"  
 }  
 ],  
 "name": "encodeSelector",  
 "outputs": [  
 {  
 "internalType": "bytes4",  
 "name": "",  
 "type": "bytes4"  
 }  
 ],  
 "stateMutability": "pure",  
 "type": "function"  
 },  
 {  
 "inputs": [],  
 "name": "getGlobalVars",  
 "outputs": [  
 {  
 "internalType": "uint256",  
 "name": "",  
 "type": "uint256"  
 },  
 {  
 "internalType": "uint256",  
 "name": "",  
 "type": "uint256"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 },  
 {  
 "internalType": "address",  
 "name": "",  
 "type": "address"  
 }  
 ],  
 "stateMutability": "view",  
 "type": "function"  
 },  
 {  
 "inputs": [  
 {  
 "internalType": "uint256",  
 "name": "a",  
 "type": "uint256"  
 },  
 {  
 "internalType": "uint256",  
 "name": "b",  
 "type": "uint256"  
 }  
 ],  
 "name": "sub",  
 "outputs": [  
 {  
 "internalType": "uint256",  
 "name": "",  
 "type": "uint256"  
 }  
 ],  
 "stateMutability": "pure",  
 "type": "function"  
 }  
]  
`  
  
func main() {  
 txData := ""  
  
 txDataDecoder := decoder.NewABIDecoder()  
 txDataDecoder.SetABI(myContractAbi1)

//借助对应智能合约的abi文件,依照方法id定位对应的方法的返回信息,进行返回信息的解析  
 method, err := txDataDecoder.DecodeOutPut("0e79507d", txData)  
 if err != nil {  
 log.Fatal(err)  
 }  
 for \_, param := range method.Params {  
 fmt.Println(param.Value)  
 fmt.Println(reflect.TypeOf(param.Value))  
 }  
}

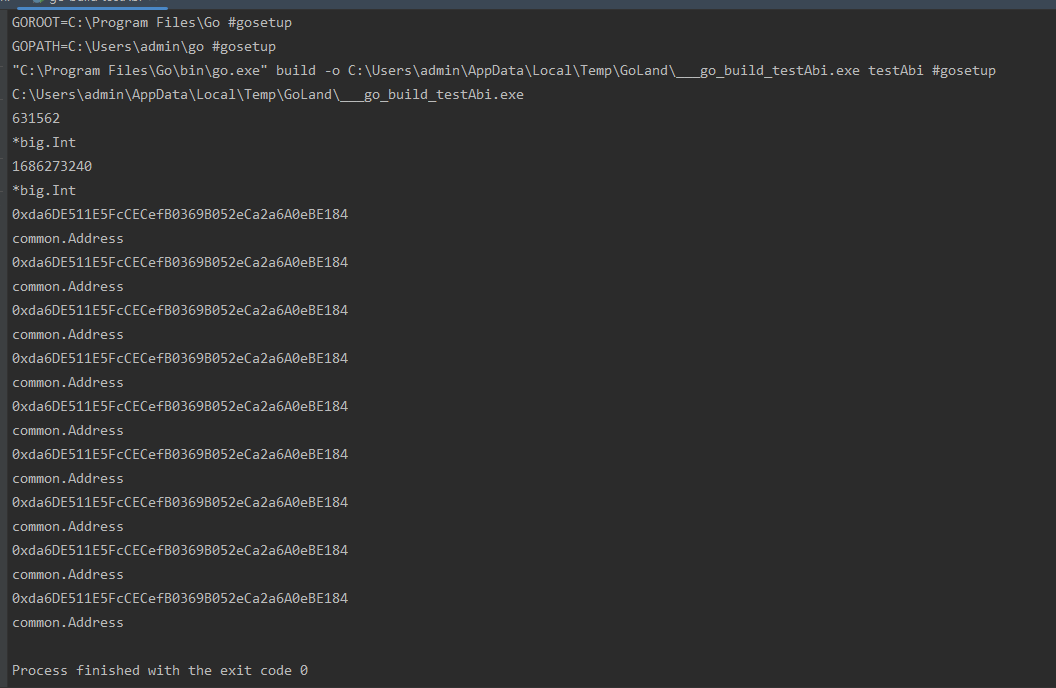
智能合约的abi获取方法:



注：在线remix地址为 <http://remix.ethereum.org/>

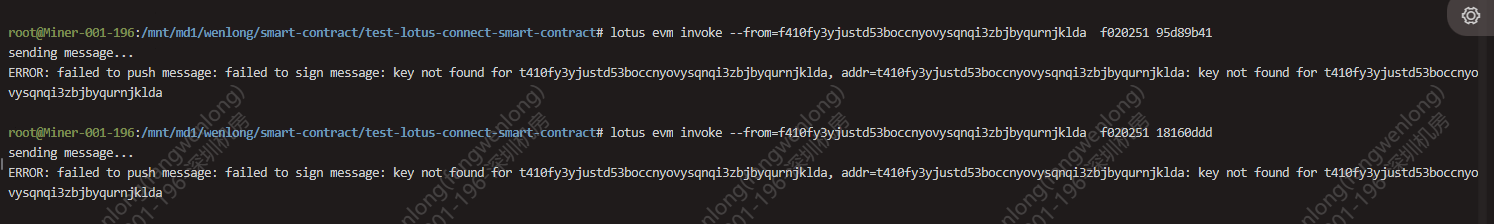
点击复制，粘贴出来即可

返回信息为：



# 报错收集

## 1 使用特定地址调用智能合约报错

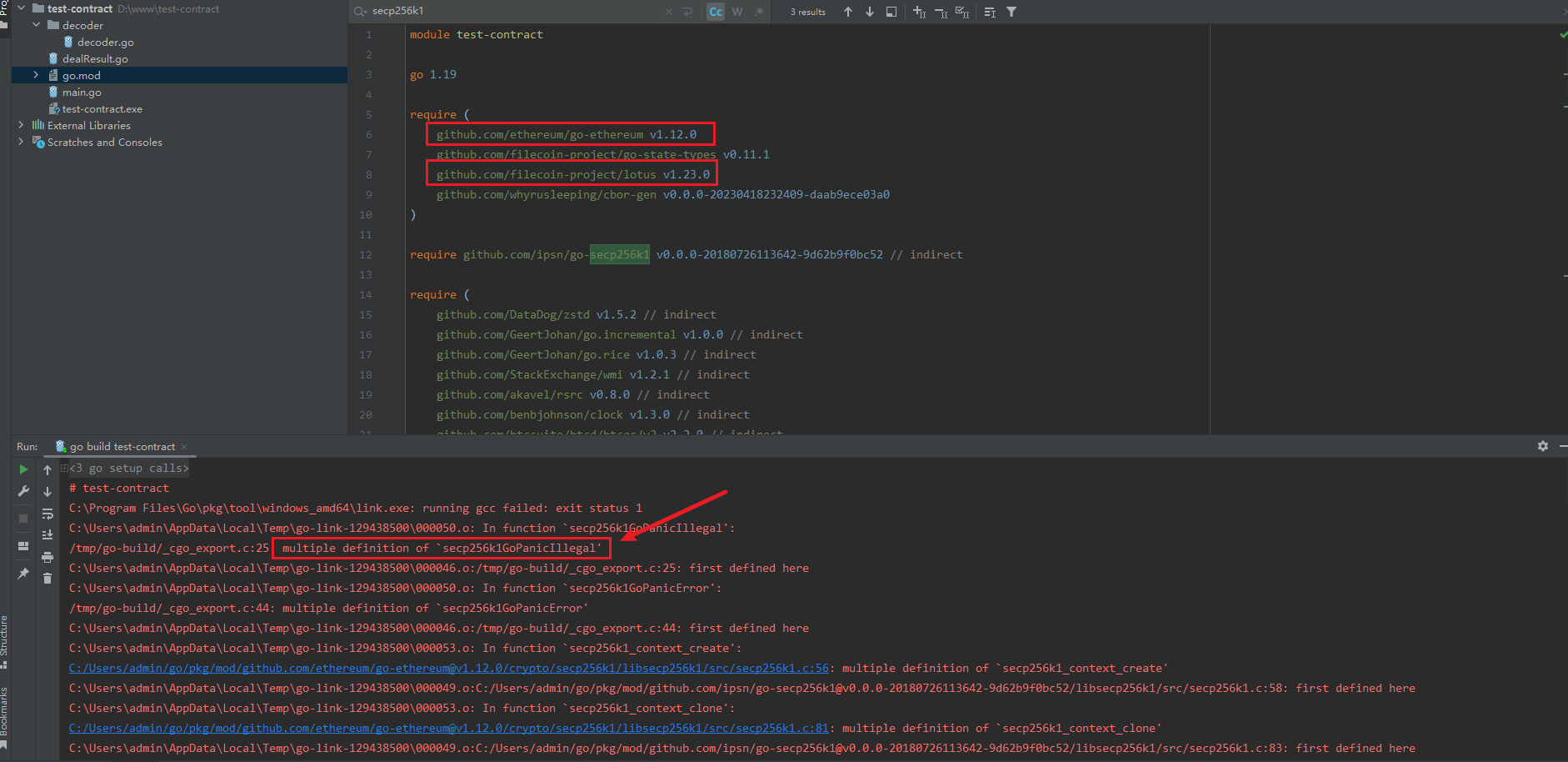


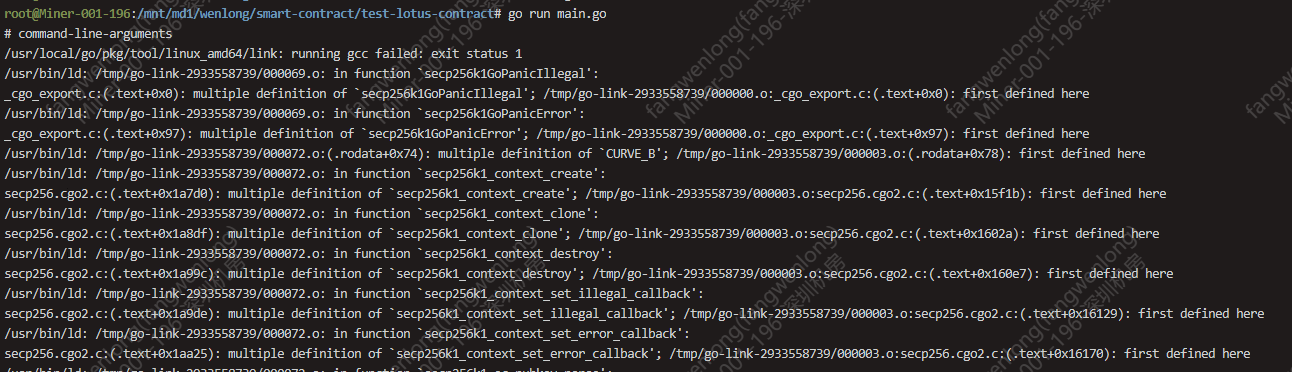
目前遇到请求直接报错，提示：

ERROR: failed to push message: failed to sign message: key not found for t410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda, addr=t410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda: key not found for t410fy3yjustd53boccnyovysqnqi3zbjbyqurnjklda

不清楚是不是由于196节点重装lotus，或者重启lotus导致的地址对应不上，目前报错的原因大概是地址的key文件相关找不到，具体原因待确定

## 2 eth和lotus第三方包一起使用编译报错





原因是因为lotus和eth都是用了crypto包，调用的底层的C语言方法

secp256k1\_context\_create重复定义，因此无论是编译还是执行main函数都会报错。

解决方案，在编译的时候添加命令：

--ldflags '-extldflags "-Wl,--allow-multiple-definition"'

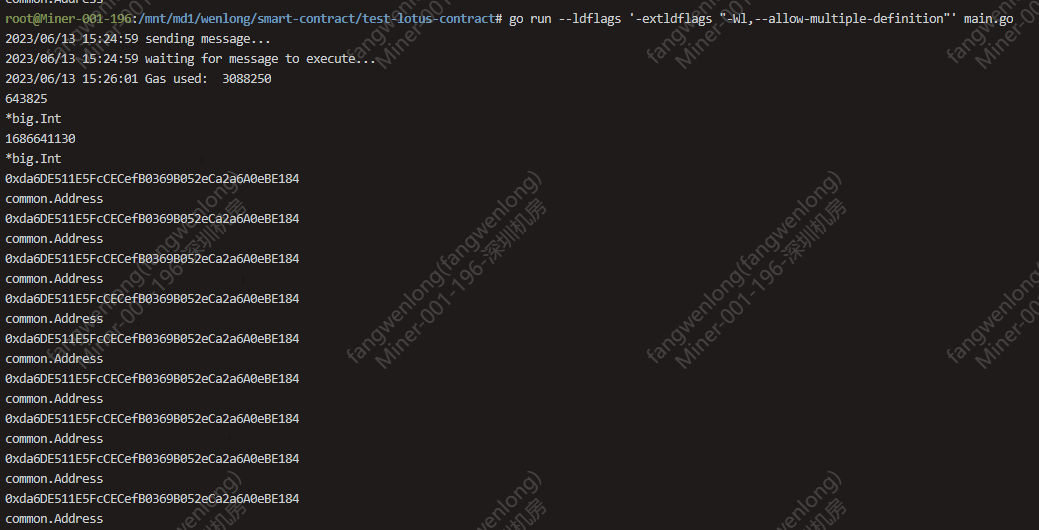
即编译变为：

go build --ldflags '-extldflags "-Wl,--allow-multiple-definition"'

同样的，运行main函数也改为

go run --ldflags '-extldflags "-Wl,--allow-multiple-definition"' main.go

即可正常编译和运行：





参考博客：

<https://stackoverflow.com/questions/56318343/golang-multiple-definition-of-cgo-ported-package>