PDX Utopia

动态库调用方案阐述以及使用方法

版本 1.1.0

目 录

[1. 设计初衷 1](#_Toc1995974029)

[2. 动态库部署与调用 1](#_Toc493455616)

[2.1 部署 1](#_Toc2074177045)

[2.2调用 1](#_Toc591553564)

[3. 动态库接口以及方法 4](#_Toc809100827)

[3.1 接口与方法 4](#_Toc691146585)

[4. 使用示例 5](#_Toc361607472)

## 

# **1. 设计初衷**

由于原版本Utopia支持运行ChainCode的运行方式为与baap交互信息，ChainCode在baap上运行将运行的结果在返回给Utopia，Utopia将结果再写入账本中。在运行的整个过程中，Utopia与baap之间的通信借助gRPC的方式来实现。而个别ChainCode在运行时，会出现运行超时的现象。究其原因，大部分原因为gRPC交互信息过程中有来无回。因此，将本方案作为gRPC交互信息有来无回的现象的解决方案。即通过调用动态库的方式，将运行ChainCode的方式拓展开来，用户通过实现特定的方法，再将该文件编译成动态库文件，再通过发送交易的方式发送给节点，来实现部署和调用。

# **2. 动态库部署与调用**

## 2.1 部署

在Utopia端新增了一个预编译合约来实现对动态库的部署和调用。合约名称为**CallSOContract**转换成地址信息为：0x6067b1C683c96EDEb4031cA8D75e2902D0dfB9dD，在发送交易的时候to请填充此参数，而交易的payload需要填充一个结构体的编码，结构体内容如下：

|  |
| --- |
| type callSoInfo struct{  SoName string //动态库名字  LookUpClassName string //自定义实现类的变量名称  Args [][]byte //参数  Data []byte //动态库编码  } |

填写相关信息后编码填入交易的payload中，Utopia在接收到交易后会对payload进行解析，下载传输过来的动态库，加载并验证。

对于结构体中Args参数需要着重说明与介绍；由于用户希望调用自己实现的一些方法，并且传递一些参数来实现相关的逻辑。因此函数名以及参数都以[]byte的形式放入Args中。比如用户自己实现的方法如下：

|  |
| --- |
| func queryInfo(input int) ([]byte) {} |

那么Args的构成就是：Args:[][]byte{[]byte(“queryInfo”),intToByte(2)}。

## 2.2调用

当用户想调用动态库的时候，同样需要填充部署时填充的结构体内容参数，其中Data字段无需赋值。Args参数字段填写需要调用的函数名称，以及参数。示例如下：

|  |
| --- |
| package main    import(  “github.com/ethereum/go-ethereum/core/types”  “github.com/ethereum/go-ethereum/crypto”  “context”  “fmt”  “go-eth/eth”  “io/ioutil”  “os”  “pdx-chain/rlp”  )  var (  host = “127.0.0.1:8545”  chainID = “111”  )  type callSoInfo struct{  SoName string  LookUpClassName string  Args [][]byte  Data []byte  }  func main() {  client,err := eth.Connect(host)  if err != nil {  println(“err”,err.Error())  return  }else{  from := crypto.PubkeyToAddress(privKey.PublicKey)  to :=iKeccak256ToAddress(“callso”)  if nonce,err := client.EthClient.NonceAt(context.TODO(),from,nil);err != nil {  fmt.Printf(err.Error())  return  }else{  amount := big.NewInt(0)  gasLimit := uint64(4700000)  gasPrice := big.NewInt(430000000000000)    soPath := “/xxx/xxx/xxx/simple.1.0.so”  soData := readSoFile(soPath)    //部署  soInfo := &callSoInfo{  SoName:”simple”,  LookUpClassName:”simple”,  Data:soData,  }  //调用  /\*\*  soInfo := &callSoInfo{  Args:[][]byte{[]byte(“queryPersonInfo”),[]byte(“json”)},  }  \*\*/  data,\_ := rlp.EncodeToBytes(soInfo)  tx:= types.NewTrasnaction(nonce,to,  amount,gasLimit,gasLimit,gasPrice,data)  //EIP155 signer  signer :=types.NewEIP155Signer(chainID)  signedTx,\_ :=types.SignTx(tx,signer,privKey)  if txHash,err := client.SendRawTransaction(context.TODO(),signedTx);  err != nil {  fmt,Printf(“send raw tx:%s”,err.Error())  }else {  fmt.Printf(“Trasnsaction hash:%s/n”,txHash.String())  }  }  }  }  func readSoFile(path string) []byte{  file,err :=os.Open(path)  if err != nil {  println(“err:”,err.Error())  return []byte{}  }  data,err := ioutil.ReadAll(file)  if err != nil {  println(“err:”,err.Error())  return []byte{}  }  return data  } |

# **动态库接口以及方法**

## 3.1 接口与方法

动态库提供的接口如下：

|  |
| --- |
| type Call interface{  Run(stub interface{}) ([]byte,error)  } |

用户需要实现该接口声明的方法从而使得utopia可以调用，作为动态库与Utopia交互的途径。用户在实现接口声明中的方法时，可以调用动态库so包中提供的函数用以实现自定义的功能。目前动态库中提供的方法如下：

|  |
| --- |
| type StubInterface interface{  //获取所有参数以[]byte的形式返回  GetArgs() [][]byte  //获取所有参数，以string的形式返回  GetStringArgs() []string  //获取函数名称以及参数  GetFunctionAndParameters() (string,[][]byte)  //获取某个key值的历史记录查询范围从start开始到end为止  GetHistoryForKey(key string,start,end uint64) (\*list.Element,error)  //查询某个key对应的值  GetState(key []byte)([]byte,error)  //保存某个键值对  PutState(key []byte,value []byte) error  //删除某个键对应的值  DelState(key []byte)  } |

# **使用示例**

用户想使用之前需要拉取动态库的包，而后用户即可自定义实现自己的动态库了。于项目中创建新的文件simple.go示例如下：

|  |
| --- |
| package main  import (  “errors”  “pdx-chain/so”  )  type Simple struct{  Name string  Age uint64  }  func(s \*Simple) Run(stub interface{}) ([]byte,error){  v,ok := stub.(so.StubInterface)  if !ok {  return nil,nil  }  function,inputs := v.GetFunctionAndParameters()  if function == “queryPersonInfo”{  return s.queryPersonInfo(v,inputs)  }else if function == “savePersonInfo”{  return s.savePersonInfo(v,inputs)  }  return nil,nil  }  func (s \*Simple) queryPersonInfo(stub so.SoStubInterface,args[][]byte) ([]byte,error){  if len(args) != 1 {  return nil,errors.New(“query input error”)  }  key := args[0]  v,err := stub.GetState(key)  if err != nil {  return nil,err  }  return v,nil  }  func (s \*Simple) savePersonInfo(stub so.SoStubInterface,args [][]byte) ([]byte,error){  if len(args) != 2 {  return nil,errors.New(“input error”)  }  key := args[0]  v := args[1]  err := Stub.PutState(key,v)  if err != nil {  return nil,err  }  return v,nil  } |

# 编译成动态库

|  |
| --- |
| go build -buildmode=plugin -o simple.1.0.so ./simple.go |