梧桐链基础版
API 文档

苏州同济区块链研究院
Suzhou Tongji Blockchain Research Institute

一、 读者对象

需要基于梧桐链开发上层应用的开发者或者组织，具有一定的开发能力。

二、 报文协议

Restful base on http(s)

三、 基本操作部分

3.1. 创建交易

3.1.1. Path

/createnewtransaction

3.1.2. Method

POST

3.1.3. Data Format

JSON

3.1.4. Description

通过调用指定的智能合约向链上写入数据，合约内容参考附录

3.1.5. Request parameters

字段	类型	说明
Name	String	合约名字

Version	String	合约版本
Method	String	合约内部定义的方法名
Args	String Array	被调用方法所需要的参数
IsUnique	Bool	值为 True 时，SDK 会检查该笔交易的交易数据是否已提交过

3.1.6. Example

```
{
"Name": "test1",
"Version": "1.0"
"Method": "init"
"Args":  ["a", "100", "b", "200"]
}
```

3.1.7. Result

```
{
"State": 200,
"Data": "3nq6aRvklIrbLhBRKBOLyq2mhByiP4FHDImshQMQXfE%3d%5c",
"Message": "success"
}
```

3.2. 获取交易详情

3.2.1. Path

/gettransaction

3.2.2. Method

GET

3.2.3. Description

根据指定哈希获取交易详情

3.2.4. Request parameters

字段	类型	说明
hashData	String	创建交易返回的 hash code，需要 URL encode

3.2.5. Example

```
/gettransaction?hashData=3nq6aRvklIrbLhBRKBOLyq2mhByiP4FHDImshQMQXfE%3d%5c
```

3.2.6. Result


```
{
   "State": 200,
   "Message": "success",
   "Data": {
      "header": {
         "version": 0,
         "timestamp": 1533711103,
         "transactionHash": "S9PI8xH3TtReFv4ueH/ueMPPIEBVyitfZ8yQy2lLetM="
      },
      "smartContract":
"eyJOYW1lIjoiIiwiVmVyc2lvbiI6IiIsIlR5cGUiOjQsInByaSI6ZmFsc2V9",
      "smartContractArgs": [
         "VVRYTyBUUkFOU0ZFUiBURVNU"
      ],
      "code": 1,
      "message": "Store evidence operation!Always Success!",
      "writes": null,
      "contractResult": {
         "status": 0,
         "payload": ""
      },
      "txRecords": [
         {
```

```
            "From": "4QqVU8DvcZNWQ7mAiuq8SFzZkhKW27PRAgo91Q716KqvK3jYxo",
            "To": "3UycKc8qvVWpVcBr3ipNqDC9oZPd86wj3qSJ6GMbLrVPgeqVwY",
            "TokenType": "ST",
            "Amount": 600
        },
        {
            "From": "4QqVU8DvcZNWQ7mAiuq8SFzZkhKW27PRAgo91Q716KqvK3jYxo",
            "To": "3UycKc8qvVWpVcBr3ipNqDC9oZPd86wj3qSJ6GMbLrVPgeqVwY",
            "TokenType": "TT",
            "Amount": 40
        }
    ]
}
    }
```


3.3. 创建存证交易

3.3.1. Path

/store

3.3.2. Method

POST

3.3.3. Data Format

JSON

3.3.4. Description

创建（隐私）存证交易

3.3.5. Request parameters

字段	类型	说明
Data	String	存证内容

PubKeys	String Array	相关方公钥，可指定多个，公钥格式为 base64 字串

3.3.6. Example

```
{
"Data":  "any data",
"pubkeys":
["LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS1NRmt3RXdZSEtvWkl6ajBDQVFZSUtvR
WN6MVVCZ2kwRFFnQUVYVVo1dG1NdXk0aTl0WVhma1QwTXl6dWI5VnJpDQo0N
VRlUHZWeFFEbWxPN2YxWHY0TzZvYXRyVkJvZjR1dkJPZmVBbVNEMmJ0UGFWNDN
BbUs0WEg1Sy9BPT0NCi0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQ==']
}
```

3.3.7. Result

```
{
  "State": 200,
  "Message": "success",
  "Data": {
    "Figure": "buS4R85x9pgmui0sPJvN24sMi/IFpCsrGjUE4h8FHb4=",
    "OK": true
  }
}
```

3.4. 查询存证交易

3.4.1. Path

/getstore

3.4.2. Method

GET

3.4.3. Description

查询存证交易

3.4.4. Request parameters

字段	类型	说明
hash	String	创建存证交易接口返回的 hash data

3.4.5. Example

```
/getstore?hash=urlEncode(hash data)
```

3.4.6. Result

```
{
   "State": 200,
   "Message": "success",
   "Data": "any data"
}
```

3.5. 获取隐私存证

3.5.1. Path

/getstore

3.5.2. Method

POST

3.5.3. Description

获取隐私存证信息

3.5.4. Request parameters

字段	类型	说明
Hash	String	创建交易返回的 hash code
PriKey	String	用户私钥，用于解密加密存证数据,base64 编码字串
KeyPwd	String	用户私钥密码

3.5.5. Example


```
{
    "hash":"vYSLGvXTHVSh+V92DneVJq28uMbyVw87qDsObzwUxMI=",
    "PriKey":"LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tTUlHVEFnRUFNQk1HQnlxR1NNNDlBZ0VHQ0NxQkhNOVZBWUl0Qkhrd2R3SUJBUVFnR2hQOGRHUWd3eE9xWUg1bw0KUnRzVVBNaVV3MzRCMW1jS04zaHRWbmFud25DZ0NnWUlLb0VjejFVQmdpMmhSQU5DQUFSZFJubTJZeTdMaUwyMQ0KaGQrUlBRekxPNXYxV3VMamxONCs5WEZBT2FVN3QvVmUvZzdxaHEydFVHaC9pNjhFNTk0Q1pJUUFp1MDlwWGpjQw0KWXJoY2ZrcjgNCi0tLS0tRU5EIFBSSVZBVEUgS0VZLS0tLS0="
}
```


3.5.6. Result

```
{
  "State": 200,
   "Message": "success",
   "Data": "pricacy transaction testing"
}
```

没有权限返回的结果

```
{
   "State": 200,
   "Message": "success",
   "Data":
"\ufffd\u0019\ufffd.\u0010\ufffd\ufffd\ufffd?\ufffd\ufffd\u0000\ufffd\ufffd/\ufffdY
   \ufffd\f\ufffdn\ufffd\u001b'\"\ufffd\ufffd?!\ufffd\ufffd\u0007"
}
```

3.6. 获取交易索引

3.6.1. Path

/gettransactionindex

3.6.2. Method

GET

3.6.3. Description

获取指定交易位于块中的索引

3.6.4. Request parameters

字段	类型	说明
hashData	String	创建交易返回的 hash code，需要 URL encode

3.6.5. Example

```
/gettransactionindex?hashData=
3nq6aRvklIrbLhBRKBOLyq2mhByiP4FHDImshQMQXfE%3d%5c
```

3.6.6. Result

```
{
"State": 200,
"Data": 0,
"Message": "success"
}
```

3.7. 获取区块高度

3.7.1. Path

/gettransactionblock

3.7.2. Method

GET

3.7.3. Description

根据交易的哈希获取交易所在块的高度

3.7.4. Request parameters

字段	类型	说明
hashData	String	创建交易返回的 hash code，需要 URL encode

3.7.5. Example

```
/gettransactionblock?hashData=
3nq6aRvklIrbLhBRKBOLyq2mhByiP4FHDImshQMQXfE%3d%5c
```

3.7.6. Result

```
{
"State": 200,
"Data": 7,
"Message": "success"
}
```

3.8. 获取区块链高度

3.8.1. Path

/getheight

3.8.2. Method

GET

3.8.3. Description

返回区块链当前高度

3.8.4. Result


```
{
"State": 200,
"Data": 7,
"Message": "success"
}
```


3.9. 按高度获取区块

3.9.1. Path

/getblockbyheight

3.9.2. Method

GET

3.9.3. Description

根据指定高度获取区块数据

3.9.4. Request parameters

字段	类型	说明
number	Number	高度

3.9.5. Example

```
/getblockbyheight?number=2
```

3.9.6. Result


```
{
"State": 200,
"Data": {
          "header": {
               "height": 2,
               "timestamp": 1520926875,
               "blockHash":
"ZLidpsKEs6QUcpmGtN09AbOOmIWBRacWpcOsva3bZcA=",
               "previousHash":
"K5t0wBA9CBu/Fwdt4/42ENy14BwA1fkI3AaE1CCMcHQ=",
               "worldStateRoot":
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=",
               "transactionRoot":
"f/Eqncx6FHmPjLxWqUwCicrMSacm8J7/0O0DUny1SFI="
          },
          "txs": ["FBk80eO/WM0gv3kchrw7u9l8sUhJdQWdDEZNndfCmRc="]
     },
"Message": "success"
}
```


3.10. 按哈希获取区块

3.10.1. Path

/getblockbyhash

3.10.2. Method

GET

3.10.3. Description

根据区块哈希获取区块详情

3.10.4. Request parameters

字段	类型	说明
hash	String	区块的 block hash，需要 URL encode

3.10.5. Example

```
/getblockbyhash?hash=
ZLidpsKEs6QUcpmGtN09AbOOmIWBRacWpcOsva3bZcA%3d%5c
```

3.10.6. Result


```
{
"State": 200,
"Data": {
          "header": {
               "height": 2,
               "timestamp": 1520926875,
               "blockHash":
"ZLidpsKEs6QUcpmGtN09AbOOmIWBRacWpcOsva3bZcA=",
               "previousHash":
"K5t0wBA9CBu/Fwdt4/42ENy14BwA1fkI3AaE1CCMcHQ=",
               "worldStateRoot":
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=",
               "transactionRoot":
"f/Eqncx6FHmPjLxWqUwCicrMSacm8J7/0O0DUny1SFI="
           },
          "txs": ["FBk80eO/WM0gv3kchrw7u9l8sUhJdQWdDEZNndfCmRc="]
     },
"Message": "success"
}
```

四、 UTXO

4.1. 创建账号地址

4.1.1. Path

/utxo/genaddress

4.1.2. Method

Post

4.1.3. Data Format

JSON

4.1.4. Description

根据公钥生成账号地址

4.1.5. Request parameters

字段	类型	说明
PubKey	String	公钥 pem 格式的 base64 编码

4.1.6. Example

{
"PubKey": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUZrd0V3WUhLb1pJemowQ0
FRWUlLb0VjejFVQmdpMERRZ0FFZnkrUkFjMU02bnpJaUJxSmE4enRtOW41aFQwTApGe
FRXMkhkMWRWdkVseWNHcndVQ2Y4ZmRGeWJiMDJuNmt5d1hTbGFqVjc5dXZoSjF6N3
E0Yk9FdjBBPT0KLS0tLS1FTkQgUFVCTElDIEtFWS0tLS0t"
}

4.1.7. Result

```
{
"State": 200,
"Message": "success",
"Data":{
  "Address":"4WoSi2S7JKqd1p7uW4JnZd2RkxsBA9aH1N4gbjiJo9YYCYtXaW"
}
}
```

4.2. 发行 Token

4.2.1. Path

/utxo/issuetoken

4.2.2. Method

Post

4.2.3. Data Format

JSON

4.2.4. Description

发行数字资产(Token)，通过单签地址发行数字资产，发行成功后，资产会发行到私钥所对应的 UTXO 账号。

4.2.5. Request parameters

字段	类型	说明
PriKey	String	私钥 pem 格式的 base64 编码
TokenType	String	Token 类型, 不同的类型 Token, 数额不能合并为同一输出
Amount	String	发行数量，支持小数点 10 位精度，钱包数据库为 boltdb 时，最大值 18 亿。
Data	String	额外数据，如：发行备注

4.2.6. Example

```
{
"PriKey":"",
"TokenType": "DefaultToken",
"Amount": "3000000000",
"Data”: “第一次发行，Token 数量:30 亿"
}
```

4.2.7. Result

```
{
"State":200,
"Message":"success",
"Data":"Z/EyJ4HQjEBkfZi2/Cf8kJTxXMM+1C89AZ9u+7OfWjs="
 }
```

4.3. 转账交易

4.3.1. Path

/utxo/transfer

4.3.2. Method

Post

4.3.3. Data Format

JSON

4.3.4. Description

转移数字资产（Token），支持转入到多个账号地址

4.3.5. Request parameters

字段	类型	说明
PriKey	String	私钥 pem 格式的 base64 编码
Data	String	额外数据，如：转账备注
Token	Object Array	Token 转移信息
– ToAddr	String	接收地址
– TokenType	String	Token 类型
– Amount	String	Token 转移数量，支持小数点 10 位精度

4.3.6. Example

```
{
"PriKey":
"LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JR1RBZ0VBTUJNR0J5cUdTTTQ5QWdFR0NDcU
JITTlWQVlJdEJIa3dkd0lCQVFRZ3FpSHZCN3ptbXZKSllFNUEKc0hTaHplMzlJT3pheVRYU3Erd
jd6enJXaEJtZ0NnWUlLb0VjejFVQmdpMmhSQU5DQUFSL0w1RUJ6VXpxZk1pSQpHb2xyek8
yYjJmbUZQUXNYRk5iWWQzVjFXOFNYSndhdkJRSi94OTBYSnR2VGFmcVRMQmRLVnFOWHY
yNitFblhQCnVyaHM0Uy9RCi0tLS0tRU5EIFBSSVZBVEUgS0VZLS0tLS0K",
"Data": "Transfer Token",
“Token”:[ {"ToAddr":"4VQ5RvNvcqN377bauXsee4ynnsXMM77AepXS8kzE32i1dAeQNP",
 “TokenType”:”Default”,”Amount”:100}]
}
```

4.3.7. Result

```
{
"State": 200,
"Message": "success",
"Data":"vdtqm37uAcMjHtGwpqD6O7+tDoOO3Vzwfk0lGnUu5i4="
 }
```

4.4. 查询余额

4.4.1. Path

/utxo/balance

4.4.2. Method

GET

4.4.3. Data Format

Query string

4.4.4. Description

查询单签账号下的数字资产余额（Token）

4.4.5. Request parameters

字段	类型	说明
key	String	私钥 pem 格式的 base64 编码的 URL 编码
tokentype	String	Token 类型, 如果不指定，则返回所有 Token 类型的余额

4.4.6. Example

```
/utxo/balance?Key=LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1JR1RBZ0VBTUJNR0J5cUdTTTQ5QWdFR0NDcUJITTlWQVlJdEJIa3dkd0lCQVFRZ3FpSHZCN3ptbXZKSllFNUEKc0hTaHplMzlJT3pheVRYU3Erdjd6enJXaEJtZ0NnWUlLb0VjejFVQmdpMmhSQU5DQUFSL0w1RUJ6VXpxZk1pSQpHb2xyek8yYjJmbUZQUXNYRk5iWWQzVjFXOFNYSndhdkJRSi94OTBYSnR2VGFmcVRMQmRLVnFOWHYyNitFblhQCnVyaHM0Uy9RCi0tLS0tRU5EIFBSSVZBVEUgS0VZLS0tLS0K&tokentype=default
```

4.4.7. Result

```
{
"State": 200,
"Message": "success",
"Data":{"Detail":[{“A-Token”:”100”, “B-Token”:”1000”}],”Total”: “1100”}
 }
```

五、 智能合约部分

5.1 安装智能合约

6.1.1. Path

/contract/install

6.1.2. Method

POST

6.1.3. Data Format

JSON

6.1.4. Description

安装合约

6.1.5. Request parameters

字段	类型	说明
Name	String	合约名字
Version	String	合约版本
File	String	合约代码内容 base64 编码格式, 合约代码参考附录

6.1.6. Example

```
{
"Name": "test1",
"Version": "1.0",
"File": base64('合约代码文件内容')
}
```

6.1.7. Result

```
{
"State": 200,
"Data": " SORZFiVQmTKRfn5LoDlaiWdVxSYvlQZUkklFTX1Rb1k=",
"Message": "success"
}
```

6.2. 销毁合约

6.2.1. Path

/contract/destroy

6.2.2. Method

POST

6.2.3. Data Format

JSON

6.2.4. Description

根据指定的合约名字和版本销毁合约

6.2.5. Request parameters

字段	类型	说明

Name	String	合约名字
Version	String	合约版本

附录

合约代码样例

```
package main

import (
	"fmt"
	"strconv"

	shim "github.com/tjfoc/tjfoc/core/chaincode/shim"
	pb "github.com/tjfoc/tjfoc/protos/chaincode"
)

//一份有效的智能合约必须实现 Init 和 Invoke 方法
type MyChaincode struct{}

func (cc MyChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
	_, args := stub.GetFunctionAndParameters()
	var A, B string    // Entities
	var Aval, Bval int // Asset holdings
	var err error
	if len(args) != 4 {
		return shim.Error("Incorrect number of arguments. Expecting 4")
	}

	// Initialize the chaincode
	A = args[0]
	Aval, err = strconv.Atoi(args[1])
	if err != nil {
		return shim.Error("Expecting integer value for asset holding")
	}
	B = args[2]
	Bval, err = strconv.Atoi(args[3])
	if err != nil {
		return shim.Error("Expecting integer value for asset holding")
	}
	fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

	// 向账本写入数据,为账号 A 初始化金额
	err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
```

```
    if err != nil {
        return shim.Error(err.Error())
    }
    // 初始化账号 B 的金额
    err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
    if err != nil {
        return shim.Error(err.Error())
    }
    return shim.Success(nil)
}

func (cc MyChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    function, args := stub.GetFunctionAndParameters()
    if function == "invoke" {
        // Make payment of X units from A to B
        return cc.invoke(stub, args)
    } else if function == "query" {
        // the old "Query" is now implemtned in invoke
        return cc.query(stub, args)
    }

    return shim.Success(nil)
}

// 查询余额
func (cc MyChaincode) query(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
    var A string // Entities
    var err error

    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting name of the
person to query")
    }

    A = args[0]

    // Get the state from the ledger
    Avalbytes, err := stub.GetState(A)
    if err != nil {
        jsonResp := "{\"Error\":\"Failed to get state for " + A + "\"}"
        return shim.Error(jsonResp)
    }
```

```
    if Avalbytes == nil {
        jsonResp := "{\"Error\":\"Nil amount for " + A + "\"}"
        return shim.Error(jsonResp)
    }

    jsonResp := "{\"Name\":\"" + A + "\",\"Amount\":\"" + string(Avalbytes) + "\"}"
    return shim.Success([]byte(jsonResp))
}

// 从账号 A 往账号 B 转账
func (cc MyChaincode) invoke(stub shim.ChaincodeStubInterface, args []string)
pb.Response {
    var A, B string    // Entities
    var Aval, Bval int // Asset holdings
    var X int          // Transaction value
    var err error
    if len(args) != 3 {
        return shim.Error("Incorrect number of arguments. Expecting 3")
    }

    A = args[0]
    B = args[1]

    // Get the state from the ledger
    Avalbytes, err := stub.GetState(A)
    if err != nil {
        return shim.Error("Failed to get state")
    }
    if Avalbytes == nil {
        return shim.Error("Entity not found")
    }
    Aval, _ = strconv.Atoi(string(Avalbytes))
    Bvalbytes, err := stub.GetState(B)
    if err != nil {
        return shim.Error("Failed to get state")
    }
    if Bvalbytes == nil {
        return shim.Error("Entity not found")
    }
    Bval, _ = strconv.Atoi(string(Bvalbytes))

    // Perform the execution
```

```
    X, err = strconv.Atoi(args[2])
    if err != nil {
        return shim.Error("Invalid transaction amount, expecting a integer value")
    }
    Aval = Aval - X
    Bval = Bval + X

    // Write the state back to the ledger
    err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
    if err != nil {
        return shim.Error(err.Error())
    }

    err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
    if err != nil {
        return shim.Error(err.Error())
    }

    return shim.Success(nil)
}

func main() {
    // 注册自定义合约
    err := shim.Start(new(MyChaincode))
    if err != nil {
        fmt.Printf("Error starting my chaincode : %s", err)
    }
}
```