智能合约调用接口及编码

openAPI 使用手册

v1.8

github 仓库：https://gitee.com/iwancao_admin/isotope-open-api-example-code/

版本	说明	日期	作者
V1.8	增加了工厂合约案例	2024 年 8 月 21	lwan
V1.7	增加 gas_limit	2024 年 5 月 16 日	Nathan
v1.0	初版		Nathan

1. 链支持

chianid	链	状态
12231	文昌链测试链	支持（测试环境）
1224	文昌链天舟链	支持
1226	文昌链天舟链 Turbo	支持（生产环境）
1	Conflux Core 测试链	支持
1029	Conflux Core 正式链	支持

2. 接入 RPC 节点：

1226 链	https://www.binghetao.com/chain-api/

3. 链上合约部署地址

合约	说明	链上名称	地址
DID 身份	域名解析和身份合约	DID	0x7D5D9e9033dF0939c0fc2CD5CE42667Bc2B31002
存储数据库	Key-value 的共享存储数据库	DDS	0x9E4eE1Cb21DfAA91d513B2BE088338C834DEf001
工厂合约	合约工厂，储存各个合约模	Factory	0x5067CE4dC9a2fb2c3E1898fc24B067cd8d92A000

	版，可以 clone 使用		

4. REST API 列表

API	接口类型	说明
POST /api/v1/chain/create	POST	创建账户
GET /api/v1/chain/queryUser	GET	查询用户在链上账户地址
POST /api/v1/chain/writeCall	POST	动态写入合约方法
GET /api/v1/chain/readCall	GET	动态写入合约方法
GET /api/v1/chain/getTransactionByHash	GET	根据交易 Hash 查询交易成功否信息
POST /api/v1/chain/importAddress	POST	用户导入私钥生成地址

5. 状态码

openAPI 的请求头信息中 content-type 需要统一设置为表单格式:

- content-type:application/x-www-form-urlencoded

请求的返回状态码及定义:

状态码	说明	备注
0	成功	code=0 成功， code >0 失败

403	No permission	没有权限
405	invalid apiKey	API Key 无效
406	signature error	签名错误
408	expire time	请求过期
409	invalid nonce	nonce 无效
301	请求参数错误	请求参数错误

6. 参数编码

智能合约通过 openAPI 调用时，需要把本地参数编码成为 data 内容传入。通常参数的编码要遵循 WEB3J 的编码规范。在 java 的编程中需要用到 web3j 的 DefaultFunctionEncoder 类库，按照每个参数的类型来编码得到二进制的 data。为了验证编码是否正确，可以使用在线工具来验证编码：https://abi.hashex.org/

得到参数编码的 data 后，即可调用 openAPI 做合约调用了。本节的内容在后面合约调用案例 DID（22）访问中有详细说明的 demo。

7. 签名认证

API 请求在通过网络传输的过程中极有可能被篡改，为了确保请求未被更改，私有接口均必须使用您的 API Key 做签名认证，以校验参数或参数值在传输途中是否发生了更改。

- 所有接口都需要进行鉴权，header 参数为 apiKey, timestamp, nonce, encrypt
- imestamp 为当前时间戳（秒级），与服务器时间差正负 10 分钟会被拒绝，nonce 为随机字符串（16 位），不能与上次请求所使用相同
- 签名方法, apiKey, timestamp, nonce，接口参数进行排序连接，使用 md5 方法进行签名

签名步骤

以获取账户地址为例

- 接口
  - GET /api/v3/user
- 示例 API 秘钥
  - apiKey = 7956ca03fe44238ef1d254799de1b556
  - apiSecret = [your secret key]

按照 ASCII 码的顺序对参数名进行排序

- 原始参数顺序为:
  - api_key = 7956ca03fe44238ef1d254799de1b556
  - nonce = 1659936393439541
  - timestamp =1659936393

  - id = 15866665555
  - chainid=1

- 按照 ASCII 码顺序对参数名进行排序：
  - api_key = 0816016bb06417f50327e2b557d39aaa
  - chainid = 1
  - id = 15866665555
  - nonce = 1659936589419161
  - timestamp= 1659936589

所有参数按”参数名参数值”格式拼接在一起组成要签名计算的字符串

- apiKey7956ca03fe44238ef1d254799de1b556chainid1id15866665555nonce1659936589419161timestamp1659936589

签名计算的字符串与秘钥(Secret Key)拼接形成最终计算的字符串，使用 32 位 MD5 算法进行计算生成数字签名

- MD5(第二步字符串+秘钥)
- MD5(apiKey7956ca03fe44238ef1d254799de1b556chainid1id15866665555nonce1659936589419161timestamp1659936589bd09139024cdd3136a4f6cf60038c1194e6641063e413c47f517a579fbb158ba)

- 签名结果中字母全部小写:
    - encrypt = bca925c9e774baf35288dc993c160df2

将生成的数字签名加入 header 参数里

header 参数

- api_key = 7956ca03fe44238ef1d254799de1b556
- nonce = 1659936393439541
- timestamp =1659936393
- encrypt = bca925c9e774baf35288dc993c160df2

body 业务参数

- chainid = 1
- id = 15866665555

8. 功能文档

8.1. 创建账户

POST [/api/v1/chain/create]

输入参数:

参数名称	是否必须	数据类型	描述	取值范围
id	true	string	会员手机号	1586666555
chainid	true	int	链 id	1,65535

返回参数:

参数名称	数据类型	描述
code	string	code=0 成功， code >0 失败
success	bool	true：成功 false:失败
data	string	新的地址

返回示例:

```
{
  "success":true,
  "code":"0",
  "data":"0x6F5F61782c1a1f3E715f011051ed1B088b38D2f9"
```

```
}
```

8.2. 查询用户在链上账户地址

GET [/api/v1/chain/queryUser]

输入参数:

参数名称	是否必须	数据类型	描述	取值范围
id	true	string	会员手机号	1586666555
chainid	true	int	链 id	1,65535

返回参数:

参数名称	数据类型	描述
code	string	code=0 成功， code >0 失败
success	bool	true：成功 false:失败
data	数组	地址集合

返回示例:

```
{
  "success":true,
  "code":"0",
```

```
"data":["0x6F5F61782c1a1f3E715f011051ed1B088b38D2f9","cfxtest:aat050mxjwuxz5w
ge6r3xhw224y8zd5s1jrm94b8d1","cfxtest:aas4n7d0f4484ety7p9b399kffd3u8p3cajkdsr
4tn"]
}
```


8.3. 写入合约方法（异步）

POST [/api/v1/chain/writeCall]

输入参数:

参数名称	是否必须	数据类型	描述	取值范围
data	true	string	data	
chainid	true	int	链 id	1,65535
contract	true	string	合约地址	
id	true	string	管理员手机号	会员手机号
fromAddress	true	string	管理员账户地址	会员系统内的个人地址
gas_limit	true	Int	Gas 费用最高上限	0-100000

返回参数:

参数名称	数据类型	描述
code	string	code=0 成功， code >0 失败
success	bool	true：成功 false:失败
data	string	交易 hash

返回示例:

```
{
  "success":true,
  "code":"0",

"data":"0x81065643975146780aea7ed79b393bd3f97d3dd8145f0c78441f1dbfe5510681"
}
```

备注：链上的写操作会改变合约内部资产状态，是异步操作，调用接口会立刻返回一个 hash，凭此 hash 可以查询本交易是否已经成功。但是链上的状态需要单独调用 readCall 去读取。

8.4. 读取合约方法（同步）

GET [/api/v1/chain/readCall]

输入参数:

参数名称	是否必须	数据类型	描述	取值范围
data	true	string	data	
chainid	true	int	链 id	1,65535
contract	true	string	合约地址	
id	true	string	会员手机号 管理员手机号	

返回参数:

参数名称	数据类型	描述
code	string	code=0 成功， code >0 失败
success	bool	true：成功 false:失败
data	string	原生数据，需自己转码

返回示例:

```
{
  "success":true,
  "code":"0",
```

```
"data":"0x0000000000000000000000000000000000000000000000000000000000000000
2000000000000000000000000000000000000000000000000000000000000064e46
5437323100000000000000000000000000000000000000000000000000000000"
}
```


8.5. 根据交易查询交易信息

GET [/api/v1/chain/getTransactionByHash]

输入参数:

参数名称	是否必须	数据类型	描述	取值范围
chainid	true	int	链 id	1,65535
hash	true	string	交易 hash id	

返回参数:

参数名称	数据类型	描述
code	string	code=0 成功， code >0 失败
success	bool	true：成功 false:失败
data	json	交易信息

返回示例:


```
{
   "success":true,
   "code":"0",
   "data":{

"hash":"0x2b3c532af6ce2b15039b6aead1b8e4c59a453ed4b1f6b4e0f5fc04e97e176c90
",
      "nonce":32,

"blockHash":"0xa52f65da9dd716c0598ff01cfc53f78155f409b067a02084875372dcf281a
734",
      "transactionIndex":0,
      "from":"cfxtest:aaj3u3efxtt0yk9jv4hp7egf8r9tee08gy9777z73a",
      "to":"cfxtest:acfpbb9kn2b2z3bev2435dk6j236gzc0kjjj4hwm45",
      "value":0,
      "gasPrice":1000000000,
      "gas":150000,
      "contractCreated":null,

"data":"0xe4452100000000000000000000000000000000000000000000000000000000
```

000000007b000000000000000000000000000000000000000000000000000000000000000000000001",
"storageLimit":0,
"epochHeight":87255643,
"chainId":1,
"status":0,
"v":0,

"r":48244449714512018821650909603302023667601838446831864372152677559202493356686,

"s":76874595916858495443730534422053520028790519188778582761750460123024696307 2}
}

8.6. 用户导入地址

POST [/api/v1/chain/importAddress]

输入参数:

参数名称	是否必须	数据类型	描述	取值范围
chainid	true	int	链 id	1,65535

参数名称	是否必须	数据类型	描述	取值范围
privateKey	true	string	私钥	地址私钥
id	true	string	会员手机号	会员手机号

返回参数:

参数名称	数据类型	描述
code	string	code=0 成功， code >0 失败
success	bool	true：成功 false:失败
data	Bool	True

返回示例:

```
{
  "success":true,
  "code":"0",
  "data":true
}
```

附件一：工厂合约使用的示例

在本节中我们使用 python 作为示例演示从工厂合约 clone 门票合约的使用方法。

9. 第一步：克隆一个门票合约

使用工厂合约，地址：0x5067CE4dC9a2fb2c3E1898fc24B067cd8d92A000

```
# 通过地址加载工厂合约
factory= Factory.at('0x5067CE4dC9a2fb2c3E1898fc24B067cd8d92A000')
# 首先查询自己的链账户地址(此处为 admin)在工厂合约上次克隆编号
id= factory.getContractNumber(admin)
# 使用自己的管理员链账户克隆一个门票合约，名称为 eTicket
factory.deployContract('eTicket', addr(admin))
# 获取克隆的合约地址（注意，id 是递增的，每次加一，唯一对应一个合约地址）
addr= factory.getContractDeployed['uint,address'](id+1, admin)
# 此处也可以简单使用 factory. getLastContractDeployed(admin) 来获取，就不需要自己去管理 id 了。
```

10.第二步：初始化门票合约

```
# 通过合约地址加载门票合约
ticket = eTicket.at(addr)
# 初始化合约 （此处对应门票的 abi 文件说明内容）
ticket.init(["故宫博物院", "故宫博物院成立于 1925 年 10 月 10 日，是以明清两代皇宫和宫廷旧藏文物为基础建立起来的大型综合性古代艺术博物馆，是世界文化遗产地、全国重点文物保护单位和爱国主义教育基地",
                    "https://cctv2024.oss-cn-
```

```
beijing.aliyuncs.com/gugong.png",
                "https://isotop.oss-cn-
shanghai.aliyuncs.com/20240102/2f0c555cd605483b9edd679f4e9d6d27.html",
                    "故宫博物院.did"],
                ["入口",
                    "午门及东西雁翅楼展厅",
                    "永寿宫展厅",
                    "斋宫展厅",
                    "古陶瓷研究中心",
                    "古书画研究中心",
                    "神武门展厅",
                    "入口 Image",
                    "午门及东西雁翅楼展厅 Image",
                    "永寿宫展厅 Image",
                    "斋宫展厅 Image",
                    "古陶瓷研究中心 Image",
                    "古书画研究中心 Image",
                    "神武门展厅 Image"],
                [str_to_hex('S'),    str_to_hex('S'),    str_to_hex('S'),
str_to_hex('S'), str_to_hex('S'), str_to_hex('S'), str_to_hex('S')],
                0,
                1735574400, addr(admin))
# 添加门票库存 10000 张票
ticket.addBank(10000, addr(admin))
# 添加别的售卖渠道管理员（此处可选，因为 admin 创建人也是管理员，需要别的管理员
```

地址时候可以添加

```
ticket.addOperator('0x99345DbE15E083cF93e98Ab447Cc870B968d1bCC',
addr(admin))
```

```
# 给新增管理员添加额度，此处只能买 100 张票
```

```
ticket.setQuota('0x99345DbE15E083cF93e98Ab447Cc870B968d1bCC',100,
addr(admin))
```

附件二：DID 身份合约使用示例

11.合约功能及调用说明-DID

在本节我们使用 java 为例详细说明智能合约 DID 身份的对外接口及调用方式。

11.1. DID 功能 1: 创建/购买 DID

[方法writeCall] create: 创建DID [入参string] _Identity: DID名称 [入参string] _ar_file: arweave的json地址 [入参address] _owner: DID名称归属人 [入参uint256] _expired: 有效期（UINIX时间） [备注] 需要管理员来调用此操作	
create	函数名称 ： new Function（“create”,Arrays.asList(), Collections.emptyList())
[入参string] _owner: DID名称归属人	Arrays.asList() 第一个入参 编码对应web3j 的数据类型是 Utf8String
[入参string] _ar_file: arweave的json地址	Arrays.asList() 第二个入参 编码对应web3j 的数据类型是 Utf8String
[入参address] _owner: DID名称归属人	Arrays.asList() 第三个入参 编码对应web3j 的数据类型是 Address
[入参uint256] _expired: 有效期（UINIX时间)	Arrays.asList() 第四个入参 编码对应web3j 的数据类型是 Uint256
注意事项	1.Arrays.asList()：必须严格按照 API 顺序入参 可参考示例 2.导入web3j包中数据类型,进行编码。

调用此接口的代码示例：

```
import org.web3j.abi.DefaultFunctionEncoder;
import org.web3j.abi.datatypes.Address;
import org.web3j.abi.datatypes.Function;
import org.web3j.abi.datatypes.Utf8String;
import org.web3j.abi.datatypes.generated.Uint256;

public class AbiUtils {
```

```
    public static void main(String[] args)  {
        Utf8String utf8String1 = new Utf8String("123456.did"); //创建名称为
123456.did 的名称
        Utf8String utf8String2 = new Utf8String("json 地址"); //arweave 的 json
地址
        Address address3 = new
Address("0x2bb9645Cc02c9aA8010329ec05C1c8077dd36008"); // 账户地址 这个 did 属于
谁
        Uint256 uint2564 = new Uint256(1715270400L); //截止时间 入参是时间戳
Long 类型
        Function f1 = new Function(
                "create", //函数名称对应 api 方法的名称
                Arrays.asList(utf8String1, utf8String2, address3, uint2564),
                Collections.emptyList()
        );
        String abiEncodedData = DefaultFunctionEncoder.encode(f1);
//abiEncodedData  内容进行编码后的
        System.out.println("abiEncodedData: " + abiEncodedData);
    }
}
```

说明:

[入参 string] 对应 web3j : Utf8String

[入参 address] 对应 web3j : Address

[入参 uint256] 对应 web3j : Uint256

```
 Function  f1 = new Function(
                "create",
                Arrays.asList(utf8String1, utf8String2, address3, uint2564),
              Collections.emptyList());
```

注意:

create : 是 API 对应的函数名称

Arrays.asList(utf8String1, utf8String2, address3, uint2564) : 按照 Api 入参的顺序进行入参

String abiEncodedData = DefaultFunctionEncoder.encode(f1)

abiEncodedData 这是内容进行编码后的: 加密数据

DefaultFunctionEncoder.encode(f1) :

导入 import org.web3j.abi.DefaultFunctionEncoder;

11.2. DID 功能 2: 存储 DID 数据

11.3. DID 功能 3: 查询 DID 数据

[方法 readCall] data: 查询 DID 数据 [入参 string] _Identity: DID [出参 tuple]：返回一个结构 structure，分别是 DID 的 tokenID，content, owner, expire [备注]

data	data是函数名称 ： new Function(“data” ,Arrays.asList(), Collections.emptyList())
[入参 string] _Identity: DID	Arrays.asList() 第一个入参 编码对应web3j 的数据类型是 Utf8String
注意事项	1.Arrays.asList()：必须严格按照 API 顺序入参 可参考示例 2.导入web3j包中数据类型,进行编码。

以下为调用接口的代码示例

```
import org.web3j.abi.datatypes.Utf8String;
import org.web3j.abi.datatypes.Function;
import org.web3j.abi.DefaultFunctionEncoder;

public class AbiUtils {
    public static void main(String[] args)  {
        Utf8String utf8String1 = new Utf8String("123456.did"); //创建名称为 123456.did 的名称
                Function f1 = new Function(
                 "data", //函数名称对应 api 方法的名称
                 Arrays.asList(utf8String1),
                 Collections.emptyList()
```

```
        );
        String abiEncodedData = DefaultFunctionEncoder.encode(f1);
        System.out.println("abiEncodedData: " + abiEncodedData);

    }
}
```

注意：123456.did 是创建的 did 名称
Arrays.asList(utf8String1) 请按 API 顺序入参
abiEncodedData 为编码后的内容

11.4. DID 功能 4: 查询 DID 的持有人地址

[方法 readCall] query: 查询 DID 的持有人地址
[入参 string] _Identity:
[出参 address] :

query	query是函数名称 ： new Function(“data” ,Arrays.asList(), Collections.emptyList())
[入参 string] _Identity:	Arrays.asList() 第一个入参 编码对应web3j 的数据类型是 Utf8String
注意事项	1.Arrays.asList()：必须严格按照 API 顺序入参 可参考示例 2.导入web3j包中数据类型,进行编码。

以下为调用接口代码：

```
import org.web3j.abi.datatypes.Function;
import org.web3j.abi.DefaultFunctionEncoder;

public class AbiUtils {
    public static void main(String[] args)  {
        Utf8String utf8String1 = new Utf8String("123456.did"); //查询名称为 123456.did 的持有人地址
                Function f1 = new Function(
                 "query", //函数名称对应 api 方法的名称
                 Arrays.asList(utf8String1),
                 Collections.emptyList()
        );
        String abiEncodedData = DefaultFunctionEncoder.encode(f1);
        System.out.println("abiEncodedData: " + abiEncodedData);
    }
```

}
注意：abiEncodedData 为编码后的内容