**Qsnarksdk使用说明文档**

# 1 使用方法

## 简述：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **通用基础返回属性** | | | | |
| **说明** | 通用的返回属性 | | | |
| **返回数据** | **编号** | **字段** | **属性** | **说明** |
| 1 | code | int | 状态码0为成功，-1为失败 |
| 2 | error | String | 成功时为空失败时为失败的类型 |
| 3 | message | String | 成功时为success，失败时为相应的错误信息 |

每一个接口都有以上返回结果，并封装到返回的对象中以下不在赘述，开发者可以通过code快速知道自己是否成功使用相应的方法。用户通过创建QsnarkAPI对象来使用sdk中的接口，下面我们将给出具体案例。

|  |  |
| --- | --- |
| sdk方法 | 功能 |
| getAccess\_Token | 获取授权令牌 |
| refAccess\_Token | 刷新交易令牌 |
| createAccount | 创建交易地址 |
| queryBlock | 查询指定高度的区块 |
| pageBlocks | 查询指定页的区块 |
| rangeBlocks | 查询指定区间内的区块 |
| nodesChain | 查询所有节点信息 |
| compileContract | 编译合约源码 |
| deployContract | 部署合约 |
| deploysyncContract | 部署合约 |
| deployArgsContract | 部署合约 |
| invokeContract | 调用合约 |
| invokesyncContract | 调用合约 |
| maintainContract | 维护合约 |
| queryContract | 查询链上所有交易量 |
| statusContract | 待完成 |
| countTransaction | 获取链上的交易总数 |
| queryTransaction | 通过hash查询交易 |
| getTxReceipt | 查询指定交易回执信息 |
| discardTransaction | 查询指定时间区间内的无效交易 |

## 1.1 getAccess\_Token接口

### 1.1.1调用api中getAccess\_Token()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| getAccess\_Token | client\_id:<string>httpkey  client\_secret:<string>httpsecret  username: <string> 用户名。  password:<string>密码 | access\_token:<string> 授权令牌  expires\_in:<string>  refresh\_token: <string>刷新token。  scope: <string>范围  token\_type: <string>token类型 |

通过输入client\_id，client\_secret，username,password获取accesstoken，即，QsnarkAPI api = new Qsnark(),然后通过api.getAccesstoken\_Token()方法获取accesstoken。

### 1.1.2方法例子

**Example：**

|  |
| --- |
| // Request  "123" (client\_id)  "123" (client\_secret)  "yeyc" (username)  "hello" (password)  // Result  {  "access\_token": "IEK1DO1ZPAQPBDPPDY7VMQ",  "expires\_in": 7200,  "refresh\_token": "JH\_-7WRJUIS5EQ4U35SD2G",  "scope": "all",  "token\_type": "Bearer"  } |

返回结果封装成GetTokenReturn对象，用户可以通过对象的属性来获取返回结果。在接下来的示例中需要用到以上返回的token 即token\_type+空格+access\_token，本例中即为Bearer IEK1DO1ZPAQPBDPPDY7VMQ;refresh\_token为后面刷新token使用。

## 1.2 refAccess\_Token接口

### 1.2.1调用api中refAccess\_Token()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| refAccess\_Token | client\_id:<string>httpkey  client\_secret:<string>httpsecret  refresh\_token: <string>刷新token。 | access\_token:<string> 授权令牌  expires\_in:<string>  refresh\_token: <string>刷新token。  scope: <string>范围  token\_type: <string>token类型 |

通过输入client\_id，client\_secret,refresh\_token（该值由getAccessToken接口返回）刷新token，即，QsnarkAPI api = new Qsnark(),然后通过api.refAccesstoken\_Token()方法刷新token。

### 1.2.2方法例子

**Example：**

|  |
| --- |
| // Request  "123" (client\_id)  "123" (client\_secret)  "JH\_-7WRJUIS5EQ4U35SD2G" (refresh\_token)  // Result  {  "access\_token":"CD843SDUOQ61RY3NEXZHLA",  "expires\_in":7200,  "refresh\_token":"LAJDJSVYXG-XQTZUVCNOOG",  "scope":"all",  "token\_type":"Bearer"  } |

返回结果封装成ReTokenReturn对象，用户可以通过对象的属性来获取返回结果。在接下来的示例中需要用到以上返回的token 即token\_type+空格+access\_token，本例中即为Bearer CD843SDUOQ61RY3NEXZHLA;refresh\_token为继续刷新token使用。

## 1.3 createAccount接口

### 1.3.1调用api中createAccount()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| createAccount | token: <string>授权令牌。 | status:<string> 调用接口状态  address:<string>账户地址 |

通过输入token,来创建账号，即生成客户账号的地址，通过QsnarkAPI api = new Qsnark(),然后调用createAccount()方法创建地址，该地址为后续的from使用，即调用合约的地址。

### 1.3.2方法例子(多参数)

**Example：**

|  |
| --- |
| // Request  "Bearer K8YYCZSSMTSVEJY7VA5KTQ" (token)  // Result  {  "Status": "",  "address": "0x7ecc0ffcdd771be370b7cc748cc320d26cef9a55"  } |

返回结果封装成CreteAccountReturn对象，用户可以通过对象的属性来获取返回结果。返回生成的账户地址。

## 1.4 queryBlock接口

### 1.4.1调用api中queryBlock()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| queryBlock | token:<string> 授权令牌  type: <string>可以是number或者hash  value: <object>type为number时可以是int(区块的高度)或latest(最后一块)  type为hash时区块的hash值 | [<QueryBlockReturn>] ­ |

< QueryBlockReturn > ­该对象成员有以下:

{

status: <string> 调用接口状态。

number: <string> 区块的高度。

hash: <string> 区块的哈希值,32字节的十六进制字符串。

parentHash: <string> 父区块哈希值，32字节的十六进制字符串。

writeTime: <long> 区块的生成时间(单位ns)。

avgTime: <long> 当前区块中，交易的平均处理时间（单位ms）。

txCounts: <long> 当前区块中打包的交易数量。

merkleRoot: <string> Merkle树的根哈希。

transactions: [<TransactionResult>] 区块中的交易列表。

}

通过输入token，type，value，其中type和value是可选的type有可选两种类型，分别是number，hash，number对应的value为number值或者latest（最后一个），hash对应的是具体的hash值。返回QueryBlockReturn对象，该方法实现查询指定区块数的block，或者指定hash的block，即，QsnarkAPI api = new Qsnark(),然后通过

api.queryBlock()方法获取QueryBlockReturn对象。

### 1.4.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "number" (type 也可为hash)  1 (可为latest) (value 当type为hash时可为对应的hash )  // Result  {  "Status": "",  "Block": {  "Number": 1,  "Hash": "0x17fb07cb6e6fe9ff00d4de91b82590158eb12ef1faff724abca9ca7ab584daed",  "ParentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  "WriteTime": 1502860899344164151,  "AvgTime": 23,  "Txcounts": 1,  "MerkleRoot": "0xda854e53569a308034da9714d2a9cf16a174e39b96699b0d0727c27d93896f2f",  "Transactions": [  {  "Version": "1.2",  "Hash": "0x4a630908bf78441197c9fc94aa3ebb4f21218cf61dfe82b62184aa1bc7f1dff1",  "BlockNumber": 1,  "BlockHash": "0x17fb07cb6e6fe9ff00d4de91b82590158eb12ef1faff724abca9ca7ab584daed",  "TxIndex": 0,  "From": "0x0aed175e1b70c67aa601a0f65946aee6bc4e5534",  "To": "0x0000000000000000000000000000000000000000",  "Amount": 0,  "Timestamp": 1502860798414577195,  "Nonce": 6651414131918424343,  "ExecuteTime": 23,  "Payload": "",  "Invalid": false,  "InvalidMsg": ""  }  ]  }  } |

返回结果封装成QueryBlockReturn对象，用户可以通过对象的属性来获取返回结果。

## 1.5 pageBlocks接口

### 1.5.1调用api中pageBlocks()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| pageBlocks | token: <string>授权令牌  index: <long>page index 页码  size: <long>page size 每页区块数量 | [<PageBlocksReturn>] ­ |

< PageBlocksReturn > ­该对象成员有以下:

{

status: <string> 调用接口状态。

number: <string> 区块的高度。

hash: <string> 区块的哈希值,32字节的十六进制字符串。

parentHash: <string> 父区块哈希值，32字节的十六进制字符串。

writeTime: <long> 区块的生成时间(单位ns)。

avgTime: <long> 当前区块中，交易的平均处理时间（单位ms）。

txCounts: <long> 当前区块中打包的交易数量。

merkleRoot: <string> Merkle树的根哈希。

transactions: [<TransactionResult>] 区块中的交易列表。

}

通过输入token，index，size获取PageBlocksReturn，该方法实现查询指定页的区块，即，QsnarkAPI api = new Qsnark(),然后通过api.pageBlocks()方法获取PageBlocksReturn对象。

### 1.5.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "1" (index)(**页码**)  "1" (size)(**每页区块数量**)  // Result  {  "Status": "ok",  "List": [  {  "Number": 144,  "Hash": "0x0f91c2be510f101e1d06fec1cfe381bcd8b04e9ad69d99f03521498f1f09c7b6",  "ParentHash": "0x76c785dd4e58762787f39e1e48f57ff0e3d4877f26e8cda53810d31232870d9b",  "WriteTime": 1503322044366500267,  "AvgTime": 24,  "Txcounts": 1,  "MerkleRoot": "0xa49893299fa4f0e2d39d6f49253d3896c9d1724d103913012f765da6083ca3e5",  "Transactions": [  {  "Version": "1.2",  "Hash": "0x5b04a3ebeaa9d39c6d9e001ec4eda5985e02f48b39c8fe7a3b47df69a6054070",  "BlockNumber": 144,  "BlockHash": "0x0f91c2be510f101e1d06fec1cfe381bcd8b04e9ad69d99f03521498f1f09c7b6",  "TxIndex": 0,  "From": "0x76fd78244f6a91d2fbe047a10a9b78e682474e7a",  "To": "0x0000000000000000000000000000000000000000",  "Amount": 0,  "Timestamp": 1503322043820082352,  "Nonce": 5600924393587988459,  "ExecuteTime": 24,  "Payload": "",  "Invalid": false,  "InvalidMsg": ""  }  ]  }  ],  "Count": 144  } |

返回结果封装成QueryBlocksReturn对象，用户可以通过对象的属性来获取返回结果。

## 1.6 rangeBlocks接口

### 1.6.1调用api中rangeBlocks()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| rangeBlocks | token:<string> 授权令牌。  from: <long> 起始区块号。  to: <object> 终止区块号可以为具体块号或者为latest。 | [<RangeBlocksReturn>] |

< RangeBlocksReturn > ­该对象成员有以下:

{

status: <string> 调用接口状态。

number: <string> 区块的高度。

hash: <string> 区块的哈希值,32字节的十六进制字符串。

parentHash: <string> 父区块哈希值，32字节的十六进制字符串。

writeTime: <long> 区块的生成时间(单位ns)。

avgTime: <long> 当前区块中，交易的平均处理时间（单位ms）。

txCounts: <long> 当前区块中打包的交易数量。

merkleRoot: <string> Merkle树的根哈希。

transactions: [<TransactionResult>] 区块中的交易列表。

}

通过输入token，from，to获取RangeBlocksReturn，该方法实现查询指定区间内的区块，即，QsnarkAPI api = new Qsnark(),然后通过api.rangeBlocks()方法获取RangeBlocksReturn对象。

### 1.6.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "1" (from)  "2" (to)  // Result  {  "Status": "",  "Blocks": [  {  "Number": 2,  "Hash": "0xaa9d9227d307f87d2c19f3b624f6af32bd90784044d1089c9e457fc90b70794f",  "ParentHash": "0x17fb07cb6e6fe9ff00d4de91b82590158eb12ef1faff724abca9ca7ab584daed",  "WriteTime": 1502860905924476359,  "AvgTime": 17,  "Txcounts": 1,  "MerkleRoot": "0xda854e53569a308034da9714d2a9cf16a174e39b96699b0d0727c27d93896f2f",  "Transactions": [  {  "Version": "1.2",  "Hash": "0x612b1b6d10cd1ee55ad3084e8734adaae11f83e9a2fcd062c064c0bc382eb8ba",  "BlockNumber": 2,  "BlockHash": "0xaa9d9227d307f87d2c19f3b624f6af32bd90784044d1089c9e457fc90b70794f",  "TxIndex": 0,  "From": "0x0aed175e1b70c67aa601a0f65946aee6bc4e5534",  "To": "0xceced635239b1b801d972b93d253f628bf1cf9fa",  "Amount": 0,  "Timestamp": 1502860805006583586,  "Nonce": 5944830206637008055,  "ExecuteTime": 17,  "Payload": "0x8da9b772",  "Invalid": false,  "InvalidMsg": ""  }  ]  },  {  "Number": 1,  "Hash": "0x17fb07cb6e6fe9ff00d4de91b82590158eb12ef1faff724abca9ca7ab584daed",  "ParentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  "WriteTime": 1502860899344164151,  "AvgTime": 23,  "Txcounts": 1,  "MerkleRoot": "0xda854e53569a308034da9714d2a9cf16a174e39b96699b0d0727c27d93896f2f",  "Transactions": [  {  "Version": "1.2",  "Hash": "0x4a630908bf78441197c9fc94aa3ebb4f21218cf61dfe82b62184aa1bc7f1dff1",  "BlockNumber": 1,  "BlockHash": "0x17fb07cb6e6fe9ff00d4de91b82590158eb12ef1faff724abca9ca7ab584daed",  "TxIndex": 0,  "From": "0x0aed175e1b70c67aa601a0f65946aee6bc4e5534",  "To": "0x0000000000000000000000000000000000000000",  "Amount": 0,  "Timestamp": 1502860798414577195,  "Nonce": 6651414131918424343,  "ExecuteTime": 23,  "Payload": "",  "Invalid": false,  "InvalidMsg": ""  }  ]  }  ]  } |

返回结果封装成RangeBlocksReturn对象，用户可以通过对象的属性来获取返回结果。

## 1.7 nodesChain接口

### 1.7.1调用api中nodesChain()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| nodesChain | token: <string> 授权令牌。 | [<NodesChainReturn >] |

NodesChainReturn对象成员有以下:

{

status:<string>调用状态

nodes\_status: <long>

ip: <string>

port: <long>

id: <long>

isPrimary: <bool> 表示该节点是否为主节点。

delay: <long> 表示该节点与本节点的延迟时间（单位ns），若为0，则为本节点。

}

说明: nodes\_status 表示节点存活状态,0表示 ALIVE ,1表示 PENDING ,2表示 STOP。

通过输入token获取NodesChainReturn 对象，该方法实现查询所有节点信息，即，QsnarkAPI api = new Qsnark(),然后通过api.nodesChain()方法获取NodesChainReturn 对象。

### 1.7.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  // Result  {  "Status": "",  "Nodes": [  {  "Status": 0,  "Ip": "115.159.156.127:8001",  "Port": 0,  "Id": 3,  "Isprimary": false,  "Delay": 14176901  },  {  "Status": 0,  "Ip": "118.89.111.35:8001",  "Port": 0,  "Id": 4,  "Isprimary": false,  "Delay": 6816348  },  {  "Status": 0,  "Ip": "120.132.30.233:8001",  "Port": 0,  "Id": 5,  "Isprimary": false,  "Delay": 17312812  },  {  "Status": 0,  "Ip": "120.26.96.64:8001",  "Port": 0,  "Id": 2,  "Isprimary": false,  "Delay": 875202  },  {  "Status": 0,  "Ip": "120.26.224.138:8001",  "Port": 0,  "Id": 1,  "Isprimary": true,  "Delay": 0  }  ]  } |

返回结果封装成NodesConReturn 对象，用户可以通过对象的属性来获取返回结果。

## 1.8 compileContract接口

### 1.8.1调用api中compileContract()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| compileContract | token<string> 授权令牌  sourcecode<string> ­合约源码。 | [<compileReturn >] |

< compileReturn > ­对象:

{

status:<string> 调用状态

cts\_status:<string> 以下信息的集合

id: <string>

bin: < string > 合约编译而成的字节码。

abi: < string > 合约源码对应的abi数组。

ok: < boolean >

}

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的compileContract()方法，该方法实现编译合约生成合约的bin，abi，等合约信息。

### 18.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer \_G2BS9FTNUE2SC5SO-NB3Q"  (token 类型String)  "contract Accumulator{ uint32 sum = 0; function increment(){ sum = sum + 1; } function getSum() returns(uint32){ return sum; } function add(uint32 num1,uint32 num2) { sum = sum+num1+num2; } }"  (sourceCode 类型String)  // Result  {  "Status": "ok",  "Cts": [  {  "Status": "",  "Id": 0,  "Bin": "",  "Abi": "[{\"constant\":false,\"inputs\":[{\"name\":\"num1\",\"type\":\"uint32\"},{\"name\":\"num2\",\"type\":\"uint32\"}],\"name\":\"add\",\"outputs\":[],\"payable\":false,\"type\":\"function\"},{\"constant\":false,\"inputs\":[],\"name\":\"getSum\",\"outputs\":[{\"name\":\"\",\"type\":\"uint32\"}],\"payable\":false,\"type\":\"function\"},{\"constant\":false,\"inputs\":[],\"name\":\"increment\",\"outputs\":[],\"payable\":false,\"type\":\"function\"}]",  "OK": true  }  ]  } |

在sdk中我们将返回值进行封装为CompileReturn 类的对象，该对象中包含所有返回结果，通过get方法即可取得返回值，该方法生成的bin，abi，为之后的deploy，invoke，maintain方法使用。

## 1.9 deployContract接口

### 1.9.1调用api中deployContract()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| deployContract | token<string> 授权令牌。  bin: <string>合约编码生成的bin。  from: <string> 合约调用者地址  callback: <string> 用户自定义方法， | status :<string>调用结果的状态  transactionHash:<string> 交易的哈希值,32字节的十六进制字符串。 |

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的deployContract()方法，该方法可以将上一步compile之后的合约部署到区块链上。deployContract方法中间我们进行了一个处理，利用了使用deployContract()方法传入的token以及方法返回回来的txhash，去调用getTxReceipt()方法返回合约的信息，最后用户可以得到ContractAddress,然后通过传递进来的自定义方法去使用address进而实现自己的目的。如下例可以看到方法返回两个值。第一个是deployContract()方法的返回值第二个是getTxReceipt()方法的返回值。

### 1.9.2方法例子

**Example：**

|  |
| --- |
| // Request  "token":"Bearer XUSICYOHONK\_AJA1LMQQLQ",  "Bin":"",  "from": "0x645aaec26e752d2249a7d3df9f8f6f2c10f71ac5",  new DevCallback() {   @Override  public void onCompute(String address) {  System.out.println("onCompute执行了！");  System.out.println("Hash值为" + address);  }  });  (用户自定义方法，可以使用address来进行自己想要运行的方法)  // Result  {  "Status": "ok",  "TxHash": "0x05a267ed9cb3102c6297ecf1585b495bc529245e7d90fd4414bd880f3b7c12b6"  }  {  "Status": "",  "TxHash": "0xe6f62694aa5be2614b24dc3394a79cf1f5923c8dc7dbd81663328ae47aae0613",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0x752bc8552696ba53bb7fddb67294c704b4c6f553",  "Ret": ""  }onCompute执行了！  Hash值为0xf43601f1677db5114626ef1aae7f893019f77733 |

在sdk中我们将返回值进行封装为DeployReturn类的对象，该对象中包含所有返回结果。

## 1.10 deploysyncContract接口

### 1.10.1调用api中deploysyncContract()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| deploysyncContract | token<string> 授权令牌。  bin: <string>合约编码生成的bin。  from: <string> 合约调用者地址 | [<GetTxReceiptReturn>] |

< GetTxReceiptReturn >对象

{

status:<string>调用合约状态

txHash:<string> 交易哈希。

postState:<string>

contractAddress:<string> 合约地址。

ret: <string>执行的结果。

}

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的deploysyncContract()方法，该方法可以将上一步compile之后的合约部署到区块链上。

该方法在deploy后直接获取回执getreceipt获得返回的对象，即GetTxReceiptReturn对象。

### 1.10.2方法例子

**Example：**

|  |
| --- |
| // Request  "token":"Bearer XUSICYOHONK\_AJA1LMQQLQ",  "bin":"",  "from": "0x645aaec26e752d2249a7d3df9f8f6f2c10f71ac5"  );  (用户自定义方法，可以使用address来进行自己想要运行的方法)  // Result  {  "Status": "ok",  "TxHash": "0x03ed7e9955bc7aa0a538b7671d4287df7537ce9215319965fcbd451774e3f244",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0x41c018c1f9631bb598218b06810a6df74ad5e07f",  "Ret": "0x60606040525bfe00a165627a7a723058206f3d39eab86ddcfb556e1c6f43a60903e1ab88beb7909c0d49d7a25b1b1a15650029"  } |

在sdk中我们将返回值进行封装为GetTxReceiptReturn类的对象，该对象中包含所有返回结果。

## 1.11 deployArgsContract接口

### 1.11.1调用api中deployArgsContract()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| deployArgsContract | token<string> 授权令牌。  bin: <string>合约编码生成的bin。  from: <string> 合约调用者地址  callback: <string> 用户自定义方法  abi: 合约源码对应的abi数组。  param… <FuncParamReal对象>0个或1个或多个 | status :<string> 调用结果的状态  transactionHash:<string> 交易的哈希值,32字节的十六进制字符串。 |

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的deployArgsContract()方法。该方法使用compile生成的bin以及abi，以及createAccount生成的地址，还有用户自定义方法来实现其功能。

deployArgsContract方法中间我们进行了一个处理，利用了使用deployArgsContract()方法传入的token以及方法返回回来的txhash，去调用getTxReceipt()方法获取调用合约的信息，最后用户可以得到ContractAddress,然后通过传递进来的方法去使用address进而实现自己的目的。如下例可以看到方法返回两个值。第一个是deployArgContract()方法第二个是getTxReceipt()方法。

### 1.11.2方法例子

**Example：**

|  |
| --- |
| FuncParamReal param1 = new FuncParamReal("uint32", 1); FuncParamReal param2 = new FuncParamReal("uint256", 2);  （这里可以创建相应类型以及值的参数）  // Request  "token":"Bearer JQS1VXOTNEANEOUZ79537A",  "bin":"0x60606040523415600b57fe5b604051602080605f83398101604052515b5b505b603380602c6000396000f30060606040525bfe00a165627a7a723058206f3d39eab86ddcfb556e1c6f43a60903e1ab88beb7909c0d49d7a25b1b1a15650029",  "from": "0x885b67411d84f6aa3b1e8e5ee6730c8123423777",  new DevCallback() {   @Override  public void onCompute(String address) {  System.out.println("onCompute执行了！");  System.out.println("Hash值为" + address);  }  },  (用户自定义方法，可以使用address来进行自己想要运行的方法)  "Abistr":"[{"constant":false,"inputs":[{"name":"num1","type":"uint32"},{"name":"num2","type":"uint32"}],"name":"add","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"getSum","outputs":[{"name":"","type":"uint32"}],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"increment","outputs":[],"payable":false,"type":"function"}]",  param...(可为空，一个或多个，此处为空)  }  // Result  {  "Status": "ok",  "TxHash": "0x2c0097fdefc0affc27520a35268d2ac36b64dfca76f4347e140e05d9898a8a98"  }  {  "Status": "",  "TxHash": "0x2c0097fdefc0affc27520a35268d2ac36b64dfca76f4347e140e05d9898a8a98",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0x067758bfa7844813b46b7973d54c241db0ff4b7d",  "Ret": "0x60606040525bfe00a165627a7a723058206f3d39eab86ddcfb556e1c6f43a60903e1ab88beb7909c0d49d7a25b1b1a15650029"  }  onCompute执行了！  Hash值为0x067758bfa7844813b46b7973d54c241db0ff4b7d |

在sdk中我们将返回值进行封装为deployConReturn类的对象，该对象中包含所有返回结果。

## 1.12 invokeContract接口

### 1.12.1调用api中invokeContract()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| invokeContract | token<string> 授权令牌。  \_const:<Boolean>  from: <string> 合约调用者地址  to:<string>合约地址  abi: <string>合约源码对应的abi数组。  callback: <string> 用户自定义方法  func\_name:<string> 调用合约中的方法名  param… <FuncParamReal对象>0个或1个或多个 | status :<string> 调用结果的状态  transactionHash:<string> 交易的哈希值,32字节的十六进制字符串。 |

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的invokeContract()方法。该方法实现调用合约中的相应方法，要传入对应的方法名以及参数，invokeContract方法中间我们进行了一个处理，利用了使用invokeContract()方法传入的token以及方法返回回来的txhash，去调用getTxReceipt()方法最终返回结果。该方法传依次传入了token,from,to,abi,用户自定义方法,func\_name,param1,....。Invoke Contract 直接接受payload参数，payload打包工作交给SDK,立即返回交易hash，在SDK轮询获取调用结果。

方法名后面的由参数与值构成的对象可以有0个到多个即FuncParamReal param1 = new FuncParamReal("uint32", 1)可以有0个到多个，下面的例子中我们给了两个包装的对象;

该方法返回InvokeConReturn对象，对象中封装了所有返回信息。

### 1.12.2方法例子

**Example：**

|  |
| --- |
| // Request  FuncParamReal param1 = new FuncParamReal("uint32", 1); FuncParamReal param2 = new FuncParamReal("uint32", 2);  "token":"Bearer FWYHAQMTWE6BPQWHPSGFXQ" ,  "const":true ,  "from":"0x9e33ffae1477a33233126c6680d418e0fb1ed219",  "to":"0x4dffc0eecde676583ebde1c29d39a0319bc8b1c7",  "abi":"[{"constant":false,"inputs":[{"name":"num1","type":"uint32"},{"name":"num2","type":"uint32"}],"name":"add","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"getSum","outputs":[{"name":"","type":"uint32"}],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"increment","outputs":[],"payable":false,"type":"function"}]  "，  (用户自定义方法，可以使用address来进行自己想要运行的方法)  new InvCallback() {  @Override  public void onCompute(List ret) {  System.out.println(ret);  } },  "func\_name":"add"，  param1,  param2  // Result  {  "Status": "",  "Data": "",  "TxHash": "0x331c519f6c5c8d33d032e5da33ae55c8ce96d75eefd0f4201f8145ffd584fd1a"  }  {  "Status": "",  "TxHash": "0x331c519f6c5c8d33d032e5da33ae55c8ce96d75eefd0f4201f8145ffd584fd1a",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0x0000000000000000000000000000000000000000",  "Ret": "0x0"  }  0x0 |
|  |

## 1.13 invokesyncContract接口

### 1.13.1调用api中invokesyncContract()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| invokesyncContract | token<string> 授权令牌。  \_const:<Boolean>  from: <string> 合约调用者地址  to:<string>合约地址  abi: <string>合约源码对应的abi数组。  func\_name:<string> 调用合约中的方法名  param… <FuncParamReal对象>0个或1个或多个 | [<GetTxReceiptReturn>] |

< GetTxReceiptReturn >对象

{

status:<string>调用合约状态

txHash:<string> 交易哈希。

postState:<string>

contractAddress:<string> 合约地址。

ret: <string>执行的结果。

}

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的invokesyncContract()方法。该方法实现调用合约中的相应方法，要传入对应的方法名以及参数，方法名后面的由参数与值构成的对象可以有0个到多个即FuncParamReal param1 = new FuncParamReal("uint32", 1)可以有0个到多个，下面的例子中我们给了两个包装的对象;该方法后面直接利用invoke的hash以及token获取回执getReceipt。返回GetTxReceipt对象，封装了回执信息。

### 1.13.2方法例子

**Example：**

|  |
| --- |
| // Request  FuncParamReal param1 = new FuncParamReal("uint32", 1); FuncParamReal param2 = new FuncParamReal("uint32", 2);  "token":"Bearer FWYHAQMTWE6BPQWHPSGFXQ" ,  "const":true ,  "from":"0x9e33ffae1477a33233126c6680d418e0fb1ed219",  "to":"0x4dffc0eecde676583ebde1c29d39a0319bc8b1c7",  "abi":"[{"constant":false,"inputs":[{"name":"num1","type":"uint32"},{"name":"num2","type":"uint32"}],"name":"add","outputs":[],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"getSum","outputs":[{"name":"","type":"uint32"}],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name":"increment","outputs":[],"payable":false,"type":"function"}]  "，  "func\_name":"add"，  param1,  param2  // Result  {  "Status": "",  "TxHash": "0x331c519f6c5c8d33d032e5da33ae55c8ce96d75eefd0f4201f8145ffd584fd1a",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0x0000000000000000000000000000000000000000",  "Ret": "0x0"  } |

## 1.14 maintainContract接口

### 1.14.1调用api中maintainContract()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| maintainContract | token<string> 授权令牌。  from: <string> 合约调用者地址  opration:<int>1升级2冻结3解冻  payload:<string>新的合约的bin  to:<string>合约地址 | status : 调用结果的状态  transactionHash:<string> 交易的哈希值,32字节的十六进制字符串。 |

此方法主要是对已经部署的合约进行维护，通过输入token，from，opration,payload,to（合约地址）获取MainTainReturn对象，Maintain Contract[合约升级] opcode: 1:升级，2:冻结，3:解冻即，QsnarkAPI api = new Qsnark(),然后通过api.maintainContract()方法获取MainTainReturn对象。该方法要用到compile方法使用源码改动后生成的bin作为payload，to是合约地址，from为账号地址。

### 1.14.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "0xd4621641bfa08b005c704726d7d2082665f34df4" (from)(调用者地址)  1 (opration)(1代表升级合约 )  0x60606040523415600b57fe5b5b60338060196000396000f30060606040525bfe00a165627a7a72305820d0b1f6bf73919b56bcf02d5db4871de60bd62a9b4ef2200e69b8ade3e6fcd4b90029 (payload)  0xdf9f4f078997b8269f0782bc40cb3d07a6efd069 (to)(合约地址)  // Result  { "Status": "",  "TxHash":"0xb07859c9e33218956ae03941e2c3e03d7102057e64677902f5b8ab69950186bb" } |

返回结果封装成MainTainReturn对象，用户可以通过对象的属性来获取返回结果。

## 1.15 queryContract接口(未实现待完成)

### 1.15.1调用api中queryContract()方法

通过输入token，pindex，psize获取QueryContReturn对象，即，QsnarkAPI api = new Qsnark(),然后通过api.queryContract()方法获取QueryContReturn对象。

### 1.15.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "1" (pindex)  "1" (psize)  // Result  {  "status": "QueryContractList",  "message": "success"  } |

返回结果封装成QueryContReturn对象，用户可以通过对象的属性来获取返回结果。

## 1.16 statusContract接口

### 1.16.1调用api中statusContract()方法

通过输入token，address获取StatusConReturn对象，该方法查询合约状态，即，QsnarkAPI api = new Qsnark(),然后通过api.statusContract()方法获取StatusConReturn对象。

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| statusContract | token:<string>登录令牌  address: <string> 合约地址。 | status:<string> 合约状态。“normal”表示正常状态，“frozen”表示冻结状态，“non-contract”表示非合约，即为普通转账的交易。 |

### 1.16.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "0x9dcaee215a9cbd1207f6d1351a930a804a269892" (contract address  )  // Result  {  "Status": "normal"  } |

返回结果封装成StatusConReturn对象，用户可以通过对象的属性来获取返回结果。

## 1.17 countTransaction接口

### 1.17.1调用api中countTransaction()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象属性 |
| countTrasaction | token:登录令牌 | status:<string> 调用状态。  count:<long> 交易数  timestamp:<long> 时间戳 |

通过输入token获取CountTraReturn对象，该方法获取链上的交易总数，即，QsnarkAPI api = new Qsnark(),然后通过api.countTrasaction()方法获取CountTraReturn对象。

### 1.17.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  // Result  {  "Status": "",  "Count": 175390,  "Timestamp": 1502847160298475207  } |

返回结果封装成CountTraReturn对象，用户可以通过对象的属性来获取返回结果。

## 1.18 queryTransaction接口

### 1.18.1调用api中queryTransaction()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| queryTransaction | token <string> 授权的令牌  transactionHash <string> 交易的哈希值,32字节的十六进制字符串。 | [<QueryTranReturn>] |

QueryTranReturn对象:

{

status: <string> 查询的状态

transaction: <JSONObject> 包含以下全部信息

version: <string> 平台版本号。

hash: <string> 交易的哈希值,32字节的十六进制字符串。

blockNumber: <long> 交易所在的区块高度。

blockHash: <string> 交易所在区块哈希值。

txIndex: <long> 交易在区块中的交易列表的位置。

from: <string> 交易发送方的地址,20字节的十六进制字符串。

to: <string> 交易接收方的地址,20字节的十六进制字符。

amount: <long> 交易量。

timestamp: <long> 交易发生时间(单位 ns)。

nonce: <long> 16位随机数。

executeTime: <string> 交易的处理时间(单位ms)。

payload: <string> 部署合约与调用合约的时候才有这个值，可以通过这个值追朔到合约调用的方法以及调用传入的参数。

invalid: <boolean> 交易是否不合法。

invalidMsg: <string> 交易的不合法信息。

}

通过输入token,hash获取交易信息，该方法通过hash查询交易，即，QsnarkAPI api = new Qsnark(),然后通过api.queryTransaction()方法获取hash对应的交易。

### 1.18.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer 3VCA4KTPPLCRIQXEWNIWLW" (token)  "0x17e8747903ea0a929cfda41e38e70d20b4205291b3b59af2f3ee0529b8f62825" (hash)  // Result  {  "Status": "",  "Transaction": {  "Version": "1.2",  "Hash": "0x4a630908bf78441197c9fc94aa3ebb4f21218cf61dfe82b62184aa1bc7f1dff1",  "BlockNumber": 1,  "BlockHash": "0x17fb07cb6e6fe9ff00d4de91b82590158eb12ef1faff724abca9ca7ab584daed",  "TxIndex": 0,  "From": "0x0aed175e1b70c67aa601a0f65946aee6bc4e5534",  "To": "0x0000000000000000000000000000000000000000",  "Amount": 0,  "Timestamp": 1502860798414577195,  "Nonce": 6651414131918424343,  "ExecuteTime": 23,  "Payload": "",  "Invalid": false,  "InvalidMsg": ""  }  } |

返回QueryTranReturn对象，用户可通过对象属性获取返回值。

## 1.9 getTxReceipt接口

### 1.9.1调用api中getTxReceipt()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| getTxReceipt | token: <string> 授权令牌。  txhash: <string> 交易hash | [<GetTxReceiptReturn>] |

< GetTxReceiptReturn >对象

{

status:<string>调用合约状态

txHash:<string> 交易哈希。

postState:<string>

contractAddress:<string> 合约地址。

ret: <string>执行的结果。

}

根据交易hash获取交易回执:使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的getTxReceipt()方法。

### 1.9.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer \_AT5GRBAM\_SWXOR63CS7WQ" (token 类型String)  "0xbef684bc8d58d48345e19c1cfa8f8891f7c0cef82181cb73b1b6ad9550e5225d  " (txhash 类型String)  // Result  {  "Status": "",  "TxHash": "0xbef684bc8d58d48345e19c1cfa8f8891f7c0cef82181cb73b1b6ad9550e5225d",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0xa532183e4a75c06aa2d9febfffbc50bef0500a3c",  "Ret": "0x606060405263ffffffff60e060020a6000350416633ad14af381146020575bfe5b3415602757fe5b603c63ffffffff600435811690602435166055565b6040805163ffffffff9092168252519081900360200190f35b8181015b929150505600a165627a7a7230582065c0eab350c9d860178b3851d8b3558d7280114749be7ce9f035c8afbe3ae0350029"  } |

在sdk中我们将返回值进行封装为GetTxReciptReturn类的对象，该对象中包含所有返回结果，通过get方法即可取得返回值。

## 1.20 discardTransaction接口

### 1.20.1调用api中discardTransaction()方法

|  |  |  |
| --- | --- | --- |
| sdk方法 | 参数 | 返回对象 |
| discardTransaction | token: <string> 授权令牌。  start: <timestamp> 开始时间戳  end: <timestamp> 结束时间戳 | [<DiscardConReturn>] |

<DiscardConReturn>对象

{

status: <string> 调用状态。

hash: <string> 交易的哈希值,32字节的十六进制字符串。

blockNumber: <number> 交易所在的区块高度。

blockHash: <string> 交易所在区块哈希值。

txIndex: <number> 交易在区块中的交易列表的位置。

from: <string> 交易发送方的地址,20字节的十六进制字符串。

to: <string> 交易接收方的地址,20字节的十六进制字符。

amount: <number> 交易量。

timestamp: <number> 交易发生时间(单位 ns)。

nonce: <number> 16位随机数。

executeTime: <string> 交易的处理时间(单位ms)。

payload: <string> 部署合约与调用合约的时候才有这个值，可以通过这个值追朔到合约调用的方法以及调用传入的参数。

invalid: <boolean> 交易是否不合法。

invalidMsg: <string> 交易的不合法信息。

}

通过输入token，start，end获取DiscardConReturn对象，即，QsnarkAPI api = new Qsnark(),然后通过api.discardTransaction()方法获取DiscardConReturn对象。

### 1.20.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "1" (start)(timestamp)  "1581776001230590326" (end)(timestamp)  // Result  {  "Status":"",  "Transactions":[{  "Version":"",  "Hash":"0x08dbbaf176ff51b4501d6757efa5c5b7e28fe6b2febf07fec2d373cef12860a6", "BlockNumber":0,  "BlockHash": "",  "TxIndex": 0,  "From":"0x885b67411d84f6aa3b1e8e5ee6730c8123423777",  "To": "0x6a713a318ac303457da2d42e52e7304f33ef310a",  "Amount": 0, "Timestamp": 1503296874051787500,  "Nonce": 894385949183117200,  "ExecuteTime":0,  "Payload":"",  "Invalid": true,  "InvalidMsg": "INVOKE\_CONTRACT\_FAILED"  }]  } |

返回结果封装成DiscardConReturn对象，用户可以通过对象的属性来获取返回结果。