**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。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。

## 1.1 getAccess\_Token接口

### 1.1.1调用api中getAccess\_Token()方法

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

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

通过输入user\_id，app\_id,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()方法创建账号

### 1.3.2方法例子(多参数)

**Example：**

|  |
| --- |
| // Request  "Bearer JH\_-7WRJUIS5EQ4U35SD2G" (token)  // Result  {  "Status": "",  "address": "0x7ad4cf715821fefbc2b3d711b0c046c8c66566cf"  } |

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

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，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)  "1" (value)  // Result  {  "Status": "",  "Block": {  "Number": 1,  "Hash": "0x186e71a80e249a9eb7a70a54adc5ec2770952dcdd5293e8b2ff0a3f078c6b7b4",  "ParentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  "WriteTime": 1498639484213423954,  "AvgTime": 38,  "Txcounts": 1,  "MerkleRoot": "0x7568c02ecbcf30af766c4316a74852e3b280823e9720e11196cfbd13562e6680",  "Transactions": [  {  "Version": "1.2",  "Hash": "0x1543bdd417304e31ad218a45a3ad31856c45f7df1cf3a6a8b628e2a1d1d2ec77",  "BlockNumber": 1,  "BlockHash": "0x186e71a80e249a9eb7a70a54adc5ec2770952dcdd5293e8b2ff0a3f078c6b7b4",  "TxIndex": 0,  "From": "0x6201cb0448964ac597faf6fdf1f472edf2a22b89",  "To": "0x0000000000000000000000000000000000000000",  "Amount": 0,  "Timestamp": 1498639484311901767,  "Nonce": 3913557320841421785,  "ExecuteTime": 38,  "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 queryBlocks接口

### 1.5.1调用api中queryBlocks()方法

通过输入token，from，to获取QueryBlocksReturn，即，QsnarkAPI api = new Qsnark(),然后通过api.queryBlocks()方法获取QueryBlocksReturn对象。

### 1.5.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "1" (from)  "2" (to)  // Result  {  "Status": "",  "Blocks": [  {  "Number": 1,  "Hash": "0x186e71a80e249a9eb7a70a54adc5ec2770952dcdd5293e8b2ff0a3f078c6b7b4",  "ParentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  "WriteTime": 1498639484213423954,  "AvgTime": 38,  "Txcounts": 1,  "MerkleRoot": "0x7568c02ecbcf30af766c4316a74852e3b280823e9720e11196cfbd13562e6680",  "Transactions": [  {  "Version": "1.2",  "Hash": "0x1543bdd417304e31ad218a45a3ad31856c45f7df1cf3a6a8b628e2a1d1d2ec77",  "BlockNumber": 1,  "BlockHash": "0x186e71a80e249a9eb7a70a54adc5ec2770952dcdd5293e8b2ff0a3f078c6b7b4",  "TxIndex": 0,  "From": "0x6201cb0448964ac597faf6fdf1f472edf2a22b89",  "To": "0x0000000000000000000000000000000000000000",  "Amount": 0,  "Timestamp": 1498639484311901767,  "Nonce": 3913557320841421785,  "ExecuteTime": 38,  "Payload": "",  "Invalid": false,  "InvalidMsg": ""  }  ]  }  ]  } |

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 nodesChain接口

### 1.6.1调用api中nodesChain()方法

通过输入token获取NodesChainReturn 对象，即，QsnarkAPI api = new Qsnark(),然后通过api.nodesChain()方法获取NodesChainReturn 对象。

### 1.6.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  // Result  {  "Status": "",  "Nodes": [  {  "Status": 0,  "Ip": "172.16.2.4",  "Port": 8201,  "Id": 4,  "Isprimary": false,  "Delay": 1720247  },  {  "Status": 0,  "Ip": "172.16.2.3",  "Port": 8201,  "Id": 3,  "Isprimary": true,  "Delay": 3607448  },  {  "Status": 0,  "Ip": "172.16.2.1",  "Port": 8201,  "Id": 1,  "Isprimary": false,  "Delay": 2695470  },  {  "Status": 0,  "Ip": "172.16.2.1",  "Port": 8201,  "Id": 2,  "Isprimary": false,  "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。

## 1.7 compileContract接口

### 1.7.1调用api中compileContract()方法

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的compileContract()方法。

### 1.7.2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

## 1.8 deployContract接口

### 1.8.1调用api中deployContract()方法

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的deployContract()方法。

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()方法最终返回结果，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deployContract()方法的返回值第二个是getTxReceipt()方法的返回值。

### 1.8.2方法例子

**Example：**

|  |
| --- |
| // Request  "token":"Bearer XUSICYOHONK\_AJA1LMQQLQ",  "Bin":"",  "from": "0x645aaec26e752d2249a7d3df9f8f6f2c10f71ac5",  new Callback() {   @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": "0x05a267ed9cb3102c6297ecf1585b495bc529245e7d90fd4414bd880f3b7c12b6",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0xf43601f1677db5114626ef1aae7f893019f77733",  "Ret": ""  }  onCompute执行了！  Hash值为0xf43601f1677db5114626ef1aae7f893019f77733 |

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

## 1.9 deployArgsConReturn接口

### 1.9.1调用api中deployArgsContract()方法

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的deployArgsContract()方法。

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()方法最终返回结果，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deployContract()方法第二个是getTxReceipt()方法。

### 1.9.2方法例子

**Example：**

|  |
| --- |
| // Request  "token":"Bearer XUSICYOHONK\_AJA1LMQQLQ",  "Bin":"",  "from": "0x645aaec26e752d2249a7d3df9f8f6f2c10f71ac5",  new Callback() {   @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": "0x05a267ed9cb3102c6297ecf1585b495bc529245e7d90fd4414bd880f3b7c12b6",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0xf43601f1677db5114626ef1aae7f893019f77733",  "Ret": ""  }  onCompute执行了！  Hash值为0xf43601f1677db5114626ef1aae7f893019f77733 |

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

## 1.10 invokeContract接口

### 1.10.1调用api中invokeContract()方法

使用时我们要QsnarkAPI api = new QsnarkAPI();创建QsnarkAPI的对象使用其中的invokeContract()方法。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,....。

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;

### 1.10.2方法例子

**Example：**

|  |
| --- |
| // Request  FuncParamReal param1 = new FuncParamReal("uint32", 1); FuncParamReal param2 = new FuncParamReal("uint32", 2);  "token":"Bearer FWYHAQMTWE6BPQWHPSGFXQ" ,  "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 Callback() {   @Override  public void onCompute(String address) {  System.out.println("onCompute执行了！");  System.out.println("Hash值为" + address);  }  },  "func\_name":"add"，  param1,  param2  // Result  {  "Status": "",  "Data": "",  "TxHash": "0x7e77101b415017f1a65b1a80d4ed1cc71921e066ce5a4ec879d9bfa4eb109962"  }  {  "Status": "",  "TxHash": "0x7e77101b415017f1a65b1a80d4ed1cc71921e066ce5a4ec879d9bfa4eb109962",  "PostState": "0x0000000000000000000000000000000000000000000000000000000000000000",  "ContractAddress": "0x0000000000000000000000000000000000000000",  "Ret": "0x0"  }  onCompute执行了！  Hash值为0x0000000000000000000000000000000000000000 |

## 1.11 mainContract接口

## 1.12 QueryContReturn接口

### 1.12.1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.queryontract()方法获取QueryContReturn对象。

### 1.12.2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。

## 1.13 statusContract接口

### 1.13.1调用api中statusContract()方法

通过输入token，address获取StatusConReturn对象，即，QsnarkAPI api = new Qsnark(),然后通过api.statusContract()方法获取StatusConReturn对象。

### 1.13.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "0x9dcaee215a9cbd1207f6d1351a930a804a269892" (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。

## 1.14 countTransaction接口

### 1.14.1调用api中countTransaction()方法

通过输入token获取CountConReturn对象，即，QsnarkAPI api = new Qsnark(),然后通过api.countConReturn()方法获取CountConReturn对象。

### 1.14.2方法例子

**Example：**

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

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

## 1.15 queryTransactionBy接口

### 1.15.1调用api中queryTransaction()方法

通过输入token,hash获取交易信息，即，QsnarkAPI api = new Qsnark(),然后通过

api.queryTransactionB()方法获取hash对应的交易。

### 1.15.2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。

## 1.6 getTxReceipt接口

### 1.6.1调用api中getTxReceipt()方法

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

### 1.6.2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。

## 1.17 discardTransaction接口

### 1.17.1调用api中discardTransaction()方法

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。

### 1.17.2方法例子

**Example：**

|  |
| --- |
| // Request  "Bearer Y5BFCSV2MCAZUQMZ9LMMSQ" (token)  "20160101" (start)  "20170606" (end)  // Result  {  "Status": "",  "Transactions": []  } |

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。