

**井通科技**

**jingtum-lib接口说明**

V2.3.4

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **简介** | **作者** | **日期** |
| 1.0.0 | 初版主要接口说明 | 吴丹 | 2018/2/28 |
| 2.0.0 | 整体接口说明 | 吴丹 | 2018/3/2 |
| 2.0.1 | Remote类增加返回结果说明 | 吴丹 | 2018/3/15 |
| 2.0.2 | Remote类增加合约方法(Lua版) | 吴丹 | 2018/5/31 |
| 2.0.3 | Remote类增加设置和获取挂单佣金的方法；挂单可选传入自己的应用来源标记；utils.processTx()方法过滤交易记录时，成交的effect中增加费率，且got是去除挂单手续费的结果； | 吴丹 | 2018/8/15 |
| 2.1.0 | 增加合约类接口(solidity版)，文档最后附solidity ERC20源码； | 吴丹 | 2019/3/5 |
| 2.1.1 | 文档最后增加solidity ERC721源码； | 吴丹 | 2019/5/8 |
| 2.1.2 | solidity调用合约新增amount字段;删除 ERC721源码(版本老); | 吴丹 | 2019/5/14 |
| 2.1.3 | [utils.processTx方法返回结果增加挂单价格及多方撮合数量、价格](http://git.jingtum.com/root/jingtum-lib/commit/407cfcbb1939414b755efaa7fefd1b33bb8b08bf)； | 吴丹 | 2019/5/24 |
| 2.1.4 | 支付amount解决41000变成40999精度不准问题；采用bignumber.js最新版，支持整数+小数20位数字； | 吴丹 | 2019/5/30 |
| 2.1.5 | 新增工具类说明；lua版合约函数名foo改为func | 吴丹 | 2019/06/04 |
| 2.1.6 | 设置挂单佣金接口(buildBrokerageTx)增加收费账号feeAccount字段；获取挂单佣金及获取用户挂单接口返回增加应用来源AppType、收费币种FeeCurrency、手续费分母OfferFeeRateDen、手续费分子OfferFeeRateNum四个字段；processTx方法增加变动币种余额balances、总共收取的挂单手续费totalRate(成交的单子才有此字段)字段；effects中增加应用来源app、挂单费率rate字段； | 吴丹 | 2019/06/12 |
| 2.1.7 | processTx方法中增加balancesPrev字段，显示交易前变动币种余额； | 吴丹 | 2019/06/13 |
| 2.1.8 | 设置挂单佣金接口(buildBrokerageTx)删除app字段；获得挂单佣金信息接口(requestBrokerage)参数改为account；创建挂单(buildOfferCreateTx)手续费标识改为platform字段；processTx方法中balances显示bug修复；effects中app改为platform标识收费平台； | 吴丹 | 2019/06/18 |
| 2.1.9 | processTx过滤交易记录中增加设置手续费(brokerage)类型；文档中增加类型说明； | 吴丹 | 2019/06/19 |
| 2.1.10 | solidity版本合约解析(processTx)中增加合约函数名及合约函数参数；文档重新完善； | 吴丹 | 2019/07/04 |
| 2.2.1 | 增加签名列表和多重签名相关接口； | 吴丹 | 2019/09/02 |
| 增加删除、激活masterkey说明； | 吴丹 | 2019/09/03 |
| 2.2.2 | processTx过滤交易记录时，若有手续费，effect中增加fee字段，显示手续费金额； | 吴丹 | 2019/09/04 |
| 2.3.0 | 增加查询黑名单相关接口；processTx交易记录中增加设置黑名单(setblacklist)类型和移除黑名单(removeblacklist)类型； | 吴丹 | 2019/11/01 |
| 2.3.1 | processTx交易记录中增加ledger\_index字段； 合约bug修复：可显示合约执行的错误结果； | 吴丹 | 2020/04/27 |
| 2.3.2 | processTx文档增加对合约类返回字段的说明； | 吴丹 | 2020/04/28 |
| 2.3.3 | 增加工具类方法eth2Jingtum，将以太坊地址转成jingtum地址； | 吴丹 | 2020/07/21 |
| 2.3.4 | processTx中增加eventLog字段，表示合约调用的日志； | 吴丹 | 2020/07/24 |

[1 说明 1](#_Toc23651)

[2 安装 1](#_Toc13831)

[3 项目文件结构 1](#_Toc3907)

[4 创建钱包 1](#_Toc27594)

[5 Remote类 2](#_Toc3443)

[5.1 创建Remote对象 3](#_Toc29422)

[5.2 创建连接 4](#_Toc21132)

[5.3 关闭连接 6](#_Toc30000)

[5.4 请求底层服务器信息 6](#_Toc13242)

[5.5 获取最新账本信息 8](#_Toc1003)

[5.6 获取某一账本具体信息 9](#_Toc12893)

[5.7 查询某一交易具体信息 11](#_Toc17059)

[5.8 请求账号信息 14](#_Toc5356)

[5.9 获得账号可接收和发送的货币 16](#_Toc21235)

[5.10 获得账号关系 18](#_Toc32679)

[5.11 获得账号挂单 20](#_Toc29923)

[5.12 获得账号交易列表 23](#_Toc21527)

[5.13 获得市场挂单列表 26](#_Toc9968)

[5.14 获得挂单佣金设置信息 30](#_Toc5499)

[5.15 获得签名列表 32](#_Toc11081)

[5.16 获得黑名单列表或账号状态 34](#_Toc16771)

[5.17 支付 36](#_Toc17427)

[5.17.1 创建支付对象 36](#_Toc18451)

[5.17.2 传入密钥 36](#_Toc18101)

[5.17.3 设置备注 37](#_Toc25054)

[5.17.4 提交支付 37](#_Toc26220)

[5.18 设置关系 39](#_Toc17686)

[5.18.1 创建关系对象 40](#_Toc22727)

[5.18.2 传入密钥 40](#_Toc28821)

[5.18.3 关系设置 40](#_Toc19128)

[5.19 设置账号属性 -----------待完善 43](#_Toc18712)

[5.19.1 创建属性对象 44](#_Toc4946)

[5.19.2 传入密钥 44](#_Toc24965)

[5.19.3 属性设置 44](#_Toc14714)

[5.20 挂单 45](#_Toc20064)

[5.20.1 创建挂单对象 45](#_Toc19469)

[5.20.2 传入密钥 46](#_Toc9683)

[5.20.3 提交挂单 46](#_Toc18509)

[5.21 取消挂单 50](#_Toc6383)

[5.21.1 创建取消挂单对象 50](#_Toc26764)

[5.21.2 传入密钥 50](#_Toc14360)

[5.21.3 取消挂单 50](#_Toc17784)

[5.22 部署合约（Lua版） 53](#_Toc17342)

[5.22.1 创建部署合约对象 53](#_Toc18031)

[5.22.2 传入密钥 53](#_Toc13250)

[5.22.3 部署合约 54](#_Toc451)

[5.23 执行合约（Lua版） 57](#_Toc13126)

[5.23.1 创建执行合约对象 57](#_Toc29658)

[5.23.2 传入密钥 57](#_Toc31671)

[5.23.3 执行合约 57](#_Toc27785)

[5.24 设置挂单佣金 60](#_Toc4421)

[5.24.1 创建挂单佣金对象 60](#_Toc19493)

[5.24.2 传入密钥 61](#_Toc13371)

[5.24.3 设置挂单佣金 61](#_Toc13839)

[5.25 部署合约（Solidity版） 64](#_Toc13760)

[5.25.1 创建合约部署对象 64](#_Toc19668)

[5.25.2 传入密钥 65](#_Toc5139)

[5.25.3 部署合约 65](#_Toc27668)

[5.26 调用合约（Solidity版） 82](#_Toc8516)

[5.26.1 创建合约调用对象 82](#_Toc14025)

[5.26.2 传入密钥 82](#_Toc20714)

[5.26.3 调用合约 82](#_Toc17577)

[5.27 废除账号主密钥masterkey 91](#_Toc27762)

[5.27.1 创建签名列表对象 91](#_Toc20408)

[5.27.2 传入密钥 91](#_Toc10997)

[5.27.3 废除masterkey 92](#_Toc2883)

[5.28 激活账号主密钥masterkey 94](#_Toc20667)

[5.29 设置签名列表 98](#_Toc22492)

[5.29.1 创建签名列表对象 98](#_Toc23507)

[5.29.2 传入密钥 98](#_Toc25615)

[5.29.3 设置签名列表 99](#_Toc11879)

[5.30 首个用户签名 102](#_Toc2024)

[5.30.1 创建首个签名对象 102](#_Toc27831)

[5.30.2 传入sequence 103](#_Toc22917)

[5.30.3 提交签名 103](#_Toc13276)

[5.31 后续用户签名 106](#_Toc11073)

[5.31.1 创建后续签名对象 106](#_Toc14280)

[5.31.2 提交签名 106](#_Toc26688)

[5.32 提交多重签名交易 110](#_Toc2589)

[5.32.1 创建多重签名对象 110](#_Toc10858)

[5.32.2 提交多重签名 110](#_Toc6344)

[5.33 通过tx\_json创建Transaction实例 114](#_Toc32013)

[5.34 本地多重签名实现 115](#_Toc10897)

[5.34.1 只有一个用户签名，交易信息由自己创建 115](#_Toc6591)

[5.34.2 多个用户签名，交易信息由其他用户返给 118](#_Toc4340)

[5.35 监听事件 122](#_Toc6466)

[6 Request类 123](#_Toc16902)

[6.1 指定账本 123](#_Toc8295)

[6.2 提交请求 124](#_Toc25132)

[7 Transaction类 125](#_Toc4433)

[7.1 获得交易账号 125](#_Toc17653)

[7.2 获得交易类型 126](#_Toc19332)

[7.3 传入私钥 126](#_Toc25940)

[7.4 添加备注 127](#_Toc6556)

[7.5 设置sequence 128](#_Toc6407)

[7.6 交易多签过程 128](#_Toc8620)

[7.7 交易多签结束 130](#_Toc8186)

[7.8 提交请求 132](#_Toc5515)

[8 工具类 132](#_Toc30681)

[8.1 16进制转字符串 133](#_Toc2717)

[8.2 字符串转16进制 133](#_Toc23865)

[8.3 判断amount是否有效 134](#_Toc23772)

[8.4 判断市场挂单的货币对是否有效 135](#_Toc28412)

[8.5 统一基础货币SWT的amount格式 135](#_Toc24364)

[8.6 判断currency是否合法 136](#_Toc19933)

[8.7 判断hash是否合法 136](#_Toc27451)

[8.8 判断address是否合法 137](#_Toc8513)

[8.9 判断secret是否合法 138](#_Toc13695)

[8.10 以太坊地址转成jingtum地址 138](#_Toc29639)

[8.11 processTx从账号角度解析交易信息 139](#_Toc11795)

[8.12 交易类型type说明 148](#_Toc11571)

[8.13 交易效果effects说明 152](#_Toc15740)

[9 底层常见错误附录 157](#_Toc23700)

[10 Solidity ERC20源码 161](#_Toc25763)

# 说明

jingtum-lib是基于websocket的底层节点公共库，本文旨在说明各个接口的含义与使用，由于节点环境、状态或版本等不一致，可能稍有偏差，例子仅供参考，应以实际结果为准。

常用测试节点地址：ws://ts5.jingtum.com:5020

# 安装

npm install --save jingtum-lib

# 项目文件结构

jingtum-lib库基于ws协议跟底层交互，其中ws封装到Server类中，Server类是一个内部类，不对外开放；Server类封装在Remote类中，Remote类提供对外访问接口并可创建两类对象：Get方式请求的Request对象和Post方式请求的Transaction对象，这两类对象都通过submit()方法提交数据到底层。文件结构图如下：

WS通讯协议

Server(内部类)

Remote类

Request对象(GET请求)

Transaction对象(POST请求)

# 创建钱包

首先引入jingtum-lib库的Wallet对象，然后使用以下两种方法创建钱包

方法1：Wallet.generate();

方法2：Wallet.fromSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

例子：

//创建Wallet对象

var jlib = require('jingtum-lib');

var Wallet = jlib.Wallet;

//方式一

var w1 = Wallet.generate();

console.log(w1);

//方式二

var w2 = Wallet.fromSecret('ss2A7yahPhoduQjmG7z9BHu3uReDk');

console.log(w2);

返回的结果信息：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
|  | secret | String | 井通钱包私钥 |
| address | String | 井通钱包地址 |

# Remote类

Remote是跟井通底层交互最主要的类，它可以组装交易发送到底层、订阅事件及从底层拉取数据。提供以下方法：

\* Remote(options)

\* connect(callback)

\* disconnect()

\* requestServerInfo()

\* requestLedgerClosed()

\* requestLedger(options)

\* requestTx(options)

\* requestAccountInfo(options)

\* requestAccountTums(options)

\* requestAccountRelations(options)

\* requestAccountOffers(options)

\* requestAccountTx(options)

\* requestOrderBook(options)

\* requestBrokerage(options)

\* buildPaymentTx(options)

\* buildRelationTx(options)

\* buildAccountSetTx(options)

\* buildOfferCreateTx(options)

\* buildOfferCancelTx(options)

\* deployContractTx(options)

\* callContractTx(options)

\* buildBrokerageTx(options)

\* initContract(options)

\* invokeContract(options)

\* on('transactions',callback)

\* on('ledger\_closed',callback)

## 创建Remote对象

方法：new Remote(options);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| server | String | 井通底层服务地址 |
| local\_sign | Boolean | 交易是否以本地签名的方式发送给底层 |

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

## 创建连接

每个Remote对象都应该首先手动连接底层，然后才可以请求底层的数据。请求结果在回调函数callback中。

方法：connect(callback)

参数：回调函数callback(err, result)

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 console.log('err:',err);  
 }else{  
 console.log(result);  
 }  
});

返回结果：

{

fee\_base: 10,

fee\_ref: 10,

hostid: 'WALE',

ledger\_hash: '72F60860465AA0345216B5A7BE93F4B3BF206DEBDC8C3D6418DB2F6019597507',

ledger\_index: 3593973,

ledger\_time: 615023760,

load\_base: 256,

load\_factor: 256,

pubkey\_node: 'n9KFgztij6QLsCk4AqDFteyJRJjMFRWV85h75wpaohRm6wVNRmDS',

random: 'F54BE300D48DD7DEAC11C2E35E42C725A04DFFF383DB5D9DA404A9B64BCD765F',

reserve\_base: 10000000,

reserve\_inc: 1000000,

server\_status: 'full',

validated\_ledgers: '2-3593973'

}

返回结果说明：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| fee\_base | Integer | 基础费用(手续费计算公式因子) |
| fee\_ref | Integer | 引用费用(手续费计算公式因子) |
| hostid | String | 主机名 |
| ledger\_hash | String | 账本hash |
| ledger\_index | Integer | 区块高度 |
| ledger\_time | Integer | 账本关闭时间 |
| pubkey\_node | String | 节点公钥 |
| reserve\_base | Integer | 账号保留值 |
| reserve\_inc | Integer | 用户每次挂单或信任冻结数量 |
| server\_status | String | 服务器状态 |
| validated\_ledgers | String | 账本区间 |

## 关闭连接

每个Remote对象可以手动关闭连接。

方法：disconnect()

参数：无

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 console.log('err:',err);  
 }else{  
 remote.disconnect(); //关闭连接  
 }  
});

## 请求底层服务器信息

首先通过本方法返回一个Request对象，然后通过submit方法获得井通底层的服务器信息，包含服务程序版本号version、该服务器缓存的账本区间ledgers、节点公钥node、服务器当前状态state。其中服务器当前状态包含可提供服务状态full和验证节点状态proposing。

方法：requestServerInfo()

参数：无

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {  
 if (err) {  
 console.log('err:', err);  
 } else {  
 var req = remote.requestServerInfo();  
 req.submit(function (err, result) {  
 if (err) {  
 console.log('err:', err);  
 }  
 else {  
 console.log('serverInfo:', result);  
 }  
 });  
 }  
});

返回结果：

{

complete\_ledgers: '2-3594053',

ledger: 'B673D6EB89DDF5D537821B9F5A74486A05EEC302535940CCF3A4ED7623A0A074',

public\_key: 'n9KFgztij6QLsCk4AqDFteyJRJjMFRWV85h75wpaohRm6wVNRmDS',

state: 'full 03:52:35',

peers: 5,

version: 'skywelld-0.29.60'

}

返回结果说明：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| complete\_ledgers | String | 账本区间 |
| ledger | String | 最新账本hash |
| public\_key | String | 节点公钥 |
| state | String | 服务器状态 |
| peers | Number | 节点连接数(不包含自己) |
| version | String | 服务器部署项目版本 |

## 获取最新账本信息

首先通过本方法返回一个Request对象，然后通过submit方法获得最新账本信息，包括区块高度(ledger\_index)与区块hash(ledger\_hash)。

方法：requestLedgerClosed()

参数：无

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {  
 if (err) {  
 console.log('err:', err);  
 } else {  
 var req = remote.requestLedgerClosed();  
 req.submit(function (err, result) {  
 if (err) {  
 console.log('err:', err);  
 }  
 else {  
 console.log(result);  
 }  
 });  
 }  
});

返回结果：

{

ledger\_hash: 'C782A51A3405484B99DD1E0F5962D6D8629845240E973F47D72F6283BB502BF3',

ledger\_index: 3594119

}

返回结果说明：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| ledger\_hash | String | 账本hash |
| ledger\_index | Number | 账本高度/区块高度 |

## 获取某一账本具体信息

首先通过本方法返回一个Request对象，然后通过submit方法获得某一账本的具体信息。

方法：remote.requestLedger({ledger\_index:’8488670’,transactions:true});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| ledger\_index | Number | 井通区块高度 |
| ledger\_hash | String | 井通区块hash(与上面ledger\_index二选一，若两个都写，以本字段为准) |
| transactions | Boolean | 是否返回账本上的交易记录hash，默认false |

注：整体参数是Object类型，当参数都不填时，默认返回最新账本信息。

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }

//var req = remote.requestLedger({});//将默认返回最新账本信息  
 var req = remote.requestLedger({  
 ledger\_index: 3637979,   
 transactions: true   
 });  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log(result);  
 }  
 });  
});

返回结果：

{

accepted: true,

account\_hash: '81A32C91AB405F1761A98088A92F307D40A1F77DEF43DD25DFD2E619B3DF390E',

close\_time: 615463820,

close\_time\_human: '2019-Jul-03 10:10:20',

close\_time\_resolution: 10,

closed: true,

hash: 'FBE16D38BE8B8FAAC139D8162183D6DA4D0BC88DCD267644ADC4A95679458CCB',

ledger\_hash: 'FBE16D38BE8B8FAAC139D8162183D6DA4D0BC88DCD267644ADC4A95679458CCB',

ledger\_index: '3637979',

parent\_hash: 'B0EE6597B5F0AD98A13C83AE7F1EAF1AECE7CAD10E0586E7EED9B7230A00E46DD',

seqNum: '3637979',

totalCoins: '600000000000000000',

total\_coins: '600000000000000000',

transaction\_hash: '05B107D3F0041A23C0D758A39290ED01B3E4BD686B7B978BF78C9461A9117A26',

transactions:

[ 'C79B437A27DEF0761F98421190D4FFDB2F4F9947C7089E6E9AF04D9BF3FCD70A' ]

}

返回结果说明：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| accepted | Boolean | 区块是否已经产生 |
| account\_hash | String | 状态hash树根 |
| close\_time | Integer | 关闭时间 |
| close\_time\_human | String | 关闭时间 |
| close\_time\_resolution | Integer | 关闭周期 |
| closed | Boolean | 账本是否已经关闭 |
| hash | String | 账本hash |
| ledger\_hash | String | 账本hash |
| ledger\_index | String | 账本高度/区块高度 |
| parent\_hash | String | 上一区块hash值 |
| seqNum | String | 账本高度/区块高度 |
| totalCoins | String | swt总量 |
| total\_coins | String | swt总量 |
| transaction\_hash | String | 交易hash树根 |
| transactions | Array | 该账本里的交易列表 |

## 查询某一交易具体信息

首先通过本方法返回一个Request对象，然后通过submit方法获得某一交易的具体信息。

方法：remote.requestTx({hash:’xxx’});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| hash | String | 交易hash |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var req = remote.requestTx({hash: '744A689B030E2F6F1CBFF94B0E52C0F1CDEED5B85D86ACAA0BC76F42C16A2AFC'});  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 var fee = result.Fee/1000000;  
 console.log('关键信息：【 交易费:', fee, '】');  
 }  
 });  
});

返回结果：

{

Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

Amount: '1000000',

Destination: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',

Fee: '10000',

Flags: 0,

Memos: [ { Memo: [Object] } ],

Sequence: 32,

SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',

Timestamp: 615464508,

TransactionType: 'Payment',

TxnSignature: '3044022009294437FD19BC2574C2A193CF72577CA2E0CE09B56A0C506EA1C35E3ECCAAE902204929895243312401161767B22504CFE07D5974BEED79FE8DCC1C121C5865BD03',

date: 615464520,

hash: '744A689B030E2F6F1CBFF94B0E52C0F1CDEED5B85D86ACAA0BC76F42C16A2AFC',

inLedger: 3638049,

ledger\_index: 3638049,

meta:

{ AffectedNodes: [ [Object], [Object], [Object] ],

TransactionIndex: 0,

TransactionResult: 'tesSUCCESS',

delivered\_amount: 'unavailable' },

validated: true

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| Account | | String | 钱包地址 |
| Platform | | String | 平台标识账号（挂单中设置了platform时，此参数才有） |
| Amount | | String/Object | 交易金额 |
| Destination | | String | 交易对家地址 |
| Fee | | String | 交易费 |
| Flags | | Integer | 交易标记 |
| Memos | | Array | 备注 |
| Sequence | | Integer | 自身账号的交易号 |
| SigningPubKey | | String | 签名公钥 |
| Timestamp | | Integer | 交易提交时间戳 |
| TransactionType | | String | 交易类型 |
| TxnSignature | | String | 交易签名 |
| date | | Integer | 交易进账本时间 |
| hash | | String | 交易hash |
| inLedger | | Integer | 交易所在的账本号 |
| ledger\_index | | Integer | 账本高度 |
| meta | | Object | 交易影响的节点 |
|  | AffectedNodes | Array | 受影响的节点 |
| TransactionIndex | Integer | -- |
| TransactionResult | String | 交易结果 |
| delivered\_amount | String | -- |
| validated | | Boolean | 交易是否通过验证 |

## 请求账号信息

首先通过本方法返回一个Request对象，然后通过submit方法获得某一账号的交易信息。

方法：remote.requestAccountInfo({account:’xxx’});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 井通钱包地址 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ'};  
 var req = remote.requestAccountInfo(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

account\_data:

{ Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

Balance: '999926380006',

Flags: 0,

LedgerEntryType: 'AccountRoot',

OwnerCount: 1,

PreviousTxnID: '744A689B030E2F6F1CBFF94B0E52C0F1CDEED5B85D86ACAA0BC76F42C16A2AFC',

PreviousTxnLgrSeq: 3638049,

Sequence: 33,

index: 'E80FF91725E82A623ADC46B458D37FB270651A76A07CD6C8115F1202856342E6' },

ledger\_hash: 'AFF5D2D4A88297F65442F97A1488F476B849617511FF93D49B6E59F84BE7822F',

ledger\_index: 3644878,

validated: true

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| account\_data | | Object | 账号信息 |
|  | Account | String | 钱包地址 |
| Balance | String | swt数量 |
| Domain | String | 域名 |
| Flags | Integer | 属性标志 |
| LedgerEntryType | String | 账本数据结构类型，AccountRoot表示账号类型 |
| MessageKey | String | 公共密钥，用于发送加密的邮件到这个帐户 |
| OwnerCount | Integer | 用户拥有的挂单数和信任线数量的总和 |
| PreviousTxnID | String | 操作该帐号的上一笔交易hash |
| PreviousTxnLgrSeq | Integer | 该帐号上一笔交易所在的账本号 |
| RegularKey | String | RegularKey |
| Sequence | Integer | 账号当前序列号 |
| index | String | 该数据所在索引hash |
| ledger\_hash | | String | 账本hash |
| ledger\_index | | Integer | 账本高度 |
| validated | | Boolean | 交易是否通过验证 |

## 获得账号可接收和发送的货币

首先通过本方法返回一个Request对象，然后通过submit方法获得某一账号可发送和接收的货币种类。

方法：remote.requestAccountTums({account:’xxx’});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 井通钱包地址 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ'};  
 var req = remote.requestAccountTums(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

ledger\_hash: '5C8E1048AEC83C50E550EFE3CF474D011B8BA4FF03D6BD09334753B0FE33A7B2',

ledger\_index: 3645061,

receive\_currencies:

[ 'CNY' ],

send\_currencies:

[ 'CNY' ],

validated: true

}

返回结果说明：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| ledger\_hash | String | 账本hash |
| ledger\_index | Integer | 账本高度 |
| receive\_currencies | Array | 可接收的货币列表 |
| send\_currencies | Array | 可发送的货币列表 |
| validated | Boolean | 交易是否通过验证 |

## 获得账号关系

井通账户之间会建立各种不同的关系。这些关系由井通后台的关系（relations）机制来处理，目前支持以下关系：信任(trust)、授权(authorize)、冻结(freeze)。

首先通过本方法返回一个Request对象，然后通过submit方法获得某一账号指定关系的信息。

方法：remote.requestAccountRelations({account:’xxx’,type:’xxx’});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 井通钱包地址 |
| type | String | 关系类型，固定的三个值：trust、authorize、freeze |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',type:'trust'};  
 var req = remote.requestAccountRelations(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

ledger\_hash: '0A73F0CC75803D62DB29957A31F30778A1606A2946A4A7B71360D81B9697C9C1',

ledger\_index: 3645086,

lines:

[ { account: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',

balance: '100000',

currency: 'CNY',

limit: '1000000000',

limit\_peer: '0',

no\_skywell: true,

quality\_in: 0,

quality\_out: 0 } ],

validated: true

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| account | | String | 钱包地址 |
| ledger\_hash | | String | 账本hash |
| ledger\_index | | Integer | 账本高度 |
| lines | | Array | 该账户的信任线 |
|  | account | String | 信任的银关 |
| balance | String | 余额 |
| currency | String | 货币种类 |
| limit | String | 信任额度 |
| limit\_peer | String | 对方设置的信任额度，默认0 |
| no\_skywell | Boolean | 信任线标记，暂时没用 |
| quality\_in | Integer | 兑换比例，默认0，暂时未用 |
| quality\_out | Integer | 兑换比例，默认0，暂时未用 |
| validated | | Boolean | 交易是否通过验证 |

## 获得账号挂单

首先通过本方法返回一个Request对象，然后通过submit方法获得某一账号的挂单信息。

方法：remote.requestAccountOffers({account:’xxx’});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 井通钱包地址 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ'};  
 var req = remote.requestAccountOffers(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

ledger\_hash: 'A88647AE9459F2E9538190D857D82437003C21769DE22D6A2F041A33F83E2753',

ledger\_index: 8643476,

offers:

[ { FeeCurrency: 'VCC',

OfferFeeRateDen: '1000',

OfferFeeRateNum: '3',

Platform: 'jB77p5n7ekApSzbefoHpuqopPX3k49u71i',

flags: 131072,

seq: 206,

taker\_gets: {

currency: 'USD',

issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',

value: '0.5' },

taker\_pays: '10000000' },

{ FeeCurrency: 'VCC',

OfferFeeRateDen: '1000',

OfferFeeRateNum: '3',

Platform: 'jB77p5n7ekApSzbefoHpuqopPX3k49u71i',

flags: 131072,

seq: 1859,

taker\_gets: {

currency: 'USD',

issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',

value: '0.5' },

taker\_pays: '1000000' } ],

validated: true

}

返回结果说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | | | 类型 | 说明 |
| account | | | String | 钱包地址 |
| ledger\_hash | | | String | 账本hash |
| ledger\_index | | | Integer | 账本高度 |
| offers | | | Array | 该账户的挂单列表 |
|  | Platform | | String | 平台标识账号，设置了才有此字段 |
| FeeCurrency | | String | 货币种类 |
| OfferFeeRateDen | | String | 分母 |
| OfferFeeRateNum | | String | 分子 |
| flags | | Integer | 买卖类型（131072表示卖，否则是买） |
| seq | | String | 余额 |
| taker\_gets | | String | 货币种类 |
|  | value | String | 金额 |
| currency | String | 货币种类 |
| issuer | String | 货币 |
| taket\_pays | | String | 信任额度 |
|  | value | String | 金额 |
| currency | String | 货币种类 |
| issuer | String | 货币 |
| validated | | | Boolean | 交易是否通过验证 |

## 获得账号交易列表

首先通过本方法返回一个Request对象，然后通过submit方法获得某一账号的交易列表信息。

方法：remote.requestAccountTx({account:’xxx’});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 井通钱包地址 |
| limit | Integer | 限定返回多少条记录，默认200 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', limit:2};  
 var req = remote.requestAccountTx(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

ledger\_index\_max: 3645132,

ledger\_index\_min: 3,

limit: 2,

marker: { ledger: 3637979, seq: 0 },

transactions:

[ { date: 1562219430,

hash: '6C357AE19525BBF7EE3324356AFBBE7399E828A5AF1219B8098654DD8A1EE262',

type: 'received',

fee: '0.01',

result: 'tesSUCCESS',

memos: [],

counterparty: 'jL8L5f56zjFfmBBq19nvDTuVRfAQ7n19aJ',

amount: [Object],

effects: [],

balances: [Object],

balancesPrev: [Object] },

{ date: 1562149320,

hash: '744A689B030E2F6F1CBFF94B0E52C0F1CDEED5B85D86ACAA0BC76F42C16A2AFC',

type: 'sent',

fee: '0.01',

result: 'tesSUCCESS',

memos: [Array],

counterparty: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',

amount: [Object],

effects: [],

balances: [Object],

balancesPrev: [Object] } ]

}

返回结果说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | | | 类型 | 说明 |
| account | | | String | 钱包地址 |
| ledger\_index\_max | | | Integer | 当前节点缓存的账本区间最大值 |
| ledger\_index\_min | | | Integer | 当前节点缓存的账本区间最小值 |
| limit | | | Integer | 限定返回的记录条数 |
| marker | | | Object | 查到的当前记录标记 |
| transactions | | | Array | 交易记录列表 |
|  | date | | Integer | 时间戳 |
| hash | | String | 交易hash |
| type | | String | 交易类型 |
| fee | | String | 手续费 |
| result | | String | 交易结果 |
| memos | | Array | 备注 |
| counterparty | | String | 交易对家 |
| amount | | Object | 交易金额对象 |
|  | value | String | 金额 |
| currency | String | 货币种类 |
| issuer | String | 货币 |
| effects | | Array | 交易效果，详见后面effects说明。 |
| balances | | Object | 每笔交易后变动的币种余额 |
| balancesPrev | | Object | 每笔交易后变动前的币种余额 |

## 获得市场挂单列表

首先通过本方法返回一个Request对象，然后通过submit方法获得市场挂单列表信息。

方法：remote.requestOrderBook({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| gets | Object | 对家想要获得的货币信息 |
| pays | Object | 对家想要支付的货币信息 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {

gets: { currency: 'SWT', issuer: '' },

pays: { currency: 'CNY', issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS' }};

var req = remote.requestOrderBook(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

ledger\_current\_index: 8649275,

offers:

[ { Account: 'j9eM8GiBb4QFRZZsrsde6XTPDenXEFnrkm',

BookDirectory: '61EF253552DDD8E9D5A1071C81591086E0132391F92311BB5B03BD5CE35839AA',

BookNode: '0000000000000000',

Flags: 0,

LedgerEntryType: 'Offer',

OwnerNode: '0000000000000000',

PreviousTxnID: '547266B62E7B0931F829C58B89541BCFE94ECB1EA78E712D8DAD071D40A1EDB9',

PreviousTxnLgrSeq: 8574138,

Sequence: 11,

TakerGets: [Object],

TakerPays: '1028266745029520',

index: 'CDD0F0DE4D63DB7F208B6EF5A84D2FE5A8CDD53DB68B876A17E9EE357EFC877B',

owner\_funds: '988270574.4452997',

quality: '1052631.578982826' },

{ Account: 'jKog7BTbU7wzDDDH3FPPmveszWQnZSeP5W',

BookDirectory: '61EF253552DDD8E9D5A1071C81591086E0132391F92311BB5B03C78C2C91CEFB',

BookNode: '0000000000000000',

Flags: 0,

LedgerEntryType: 'Offer',

OwnerNode: '0000000000000000',

PreviousTxnID: 'BBB6A770A818427EC3B3778F94EF1577704EFCAD0C7602740E304207E9FC4943',

PreviousTxnLgrSeq: 1352529,

Sequence: 10,

TakerGets: [Object],

TakerPays: '100000000',

index: '9EDE0EDF25FA1AAC26B11DB38F90095AC8A84351B4B83773162A5B76BFF98674',

owner\_funds: '1001841.261900998',

quality: '1063829.787234043' },

... 200 more items ],

validated: false

}

返回结果说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | | | 类型 | 说明 |
| ledger\_current\_index | | | String | 当前账本号 |
| offers | | | Array | 市场挂单列表 |
|  | Account | | Integer | 账号地址 |
| BookDirectory | | String | -- |
| BookNode | | String | -- |
| Flags | | Integer | 挂单买卖标记 |
| FeeCurrency | | String | 收费币种 |
| OfferFeeRateDen | | String | 分母 |
| OfferFeeRateNum | | String | 分子 |
| Platform | | String | 平台标识账号 |
| LedgerEntryType | | String | 账本数据结构类型，Offer表示挂单类 |
| OwnerNode | | Array | -- |
| PreviousTxnID | | String | 上一笔交易hash |
| PreviousTxnLgrSeq | | Integer | 上一笔交易所在账本号 |
| Sequence | | Integer | 单子序列号 |
| TakerGets | | Object | 对方得到的。（买卖双方，当货币是swt时，数据类型为对象；否则为string） |
|  | value | String | 金额 |
| currency | String | 货币种类 |
| issuer | String | 货币 |
| TakerPays | | String | 对方支付的 |
|  | value | String | 金额 |
| currency | String | 货币种类 |
|  | issuer | String | 货币 |
| index | | String | 该数据所在索引hash |
| owner\_funds | | String | 用户swt资产 |
| quality | | String | 价格或价格的倒数 |
| validated | | | Boolean | 交易是否通过验证 |

## 获得挂单佣金设置信息

首先通过本方法返回一个Request对象，然后通过submit方法获得挂单佣金设置信息。

方法：remote.requestBrokerage({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 设置挂单佣金的账号 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {

account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh'};

var req = remote.requestBrokerage(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

brokerages:

[ { FeeCurrency: 'AAA',

FeeCurrencyIssuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',

OfferFeeRateDen: '1000',

OfferFeeRateNum: '5',

Platform: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

fee\_account: 'jzTx4CRUZJT1ZsBhGHi7Wqikada63xRVv' } ],

ledger\_hash: 'EEEECA6766A17D4113CDF2FC6DA05C2B735EDFB437EF2546DEDBA127AF63B2BE',

ledger\_index: 1633,

validated: true

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| account | | String | 设置佣金的账号或银关 |
| brokerages | | Array | 设置的所有佣金详情 |
|  | FeeCurrency | String | 收费币种 |
| FeeCurrencyIssuer | String | 货币发行方 |
| OfferFeeRateDen | String | 分母 |
| OfferFeeRateNum | String | 分子 |
| Platform | String | 平台标识账号 |
| fee\_account | String | 收费账号 |
| ledger\_hash | | String | 当前hash |
| ledger\_index | | Integer | 当前账本号 |
| validated | | Boolean | 交易是否通过验证 |

## 获得签名列表

首先通过本方法返回一个Request对象，然后通过submit方法获得市场挂单列表信息。

方法：remote.requestSignerList({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 设置签名列表的源账号 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {

account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh'};

var req = remote.requestSignerList(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

account\_objects:

[ { Flags: 0,

LedgerEntryType: 'SignerList',

OwnerNode: '0000000000000000',

PreviousTxnID: '01D05D192B8AD17A6D1D4627A2778D61A48622392B9B47BB5E840D92BF06E8CA',

PreviousTxnLgrSeq: 26120,

SignerEntries: [ { SignerEntry:

{ Account: 'jpX8SEpM387c9tpdAUfBr2gYTfC2k7RatA',

SignerWeight: 2 } },

{ SignerEntry:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SignerWeight: 2 } },

{ SignerEntry:

{ Account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

SignerWeight: 2 } } ],

SignerQuorum: 3,

index: 'E1C0156713A82509084245A7242E627EB656167D5A4F0C56A68A53FBC17C653E' } ],

ledger\_current\_index: 26510,

validated: false}

返回结果说明：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | | | | | 类型 | 说明 |
| account | | | | | String | 设置签名列表的源账号 |
| account\_objects | | | | | Array | 签名列表相关信息 |
|  | ---- | | | | Object | 签名列表相关信息 |
|  | Flags | | | Integer | 交易标记 |
| LedgerEntryType | | | String | 账本数据结构类型，SignerList表示签名列表类型 |
| OwnerNode | | | String | 列表索引标记 |
| PreviousTxnID | | | String | 上一笔交易hash |
| PreviousTxnLgrSeq | | | Integer | 上一笔交易所在账本号 |
| SignerEntries | | | Array | 签名列表 |
|  | SignerEntry | | Object | 单个签名对象 |
|  | Account | String | 签名者账号地址 |
| SignerWeight | Integer | 该签名者在签名列表中的权重 |
| SignerQuorum | | | Integer | 签名列表阈值 |
| index | | | String | 签名列表id |
| ledger\_current\_index | | | | | String | 当前账本号 |
| validated | | | | | Boolean | 当前账本中，交易是否通过验证 |

## 获得黑名单列表或账号状态

首先通过本方法返回一个Request对象，然后通过submit方法获得市场挂单列表信息。

方法：remote.requestBlacklist({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 可选，查询具体某个账号是否处于黑名单状态 |
| marker | String | 可选，向下查找的标记 |

返回：Request对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var options = {

account: 'jfkgNWMZWYsYD6PzB4braCtQQNSNrstUmi'};

var req = remote.requestBlacklist(options);  
 req.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

account: 'jfkgNWMZWYsYD6PzB4braCtQQNSNrstUmi',

blacklist: true,

ledger\_hash: '199549455126B6674B203BE2310F29E97C9FD02A5C3F347F969BD0060D87AD8C',

ledger\_index: '14275428'}

返回结果说明：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 设置签名列表的源账号 |
| blacklist | Boolean | 是否是黑名单状态 |
| ledger\_hash | String | 账本hash |
| ledger\_index | String | 账本高度 |

## 支付

首先通过buildPaymentTx方法返回一个Transaction对象，然后通过setSecret传入密钥，addMemo添加备注为可选项，最后通过submit方法提交支付信息。

### 创建支付对象

方法：remote.buildPaymentTx({});

参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| account | | String | 发起账号 |
| to | | String | 目标账号 |
| amount | | Object | 支付金额 |
|  | value | String | 支付数量 |
| currency | String | 货币种类，三到六个字母或20字节的自定义货币 |
| issuer | String | 货币发行方 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

### 设置备注

方法：tx.addMemo(memo);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| memo | String | 备注信息 |

### 提交支付

方法：tx.submit(callback);

参数：无

支付完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function(err, result) {  
 if (err) {  
 return console.log('err:',err);  
 }  
 var tx = remote.buildPaymentTx({  
 account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',  
 to: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',  
 amount: {  
 "value": 5,  
 "currency": "SWT",  
 "issuer": ""  
 }  
 });  
 tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');  
 tx.addMemo('支付5swt.');//可选  
 tx.submit(function(err, result) {  
 if(err) {console.log('err:',err);}  
 else if(result){  
 console.log('res:', result);  
 }  
 });  
});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '120000220000000024000000236140000000004C4B40684000000000002710732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD1677446304402206238696F155F863AF941F44373690E9AF4233F9548F05F10B05084B4243640E3022010684A730C9C8E3267AD5A12AF38A0E0B1A52EE2A4023FC6E03350C23CF22483811472F05993EBA9858291D364EBF6EEC3D851BD3792831485B6C98BAD6DBF7805D3C5CCC1B4F989E0CE6749F9EA7D0BE694AFE4BB98357377742EE1F1',

tx\_json:

{ Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

Amount: '5000000',

Destination: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',

Fee: '10000',

Flags: 0,

Memos: [ [Object] ],

Sequence: 35,

SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',

TransactionType: 'Payment',

TxnSignature: '304402206238696F155F863AF941F44373690E9AF4233F9548F05F10B05084B4243640E3022010684A730C9C8E3267AD5A12AF38A0E0B1A52EE2A4023FC6E03350C23CF22483',

hash: '8C2A3212511E0CB84B4958DB8869AD6861C4439BA6487BA17B77AE4F1F738C85' }

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 账号地址 |
| Amount | String | 交易金额 |
| Destination | String | 对家 |
| Fee | String | 交易费 |
| Flags | Integer | 交易标记 |
| Memos | Array | 备注 |
| Sequence | Integer | 单子序列号 |
| SigningPubKey | Object | 签名公钥 |
| TransactionType | String | 交易类型 |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 设置关系

首先通过buildRelationTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法提交支付信息。目前支持的关系类型：信任(trust)、授权(authorize)、冻结(freeze)。

### 创建关系对象

方法：remote.buildRelationTx({});

参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| type | | String | 关系种类 |
| account | | String | 设置关系的源账号 |
| target | | String | 目标账号，授权和冻结才有 |
| limit | | Object | 关系金额 |
|  | value | String | 数量 |
| currency | String | 货币种类，三到六个字母或20字节的自定义货币 |
| issuer | String | 货币发行方 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

### 关系设置

方法：tx.submit(callback);

参数：无

设置关系完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port:5020', local\_sign:true});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var options = {

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

limit:{

currency: 'CNY',

value: '3000000000',

issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS'

},

type:'trust'

};

var tx = remote.buildRelationTx(options);

tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '1200142200000000240000002B63D6CAA87BEE538000000000000000000000000000434E5900000000007478E561645059399B334448F7544F2EF308ED32684000000000002710732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD16774473045022100B1F3D3BC700739D234531096474F604ABE7A8E2499D3978B328984A8F966551A0220493513CB305F2D39E4E685991302CFC44740AC7D6036240D225911C6081828CC811472F05993EBA9858291D364EBF6EEC3D851BD3792',

tx\_json:

{ Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

Fee: '10000',

Flags: 0,

LimitAmount:

{ currency: 'CNY',

issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',

value: '3000000000' },

Sequence: 43,

SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',

TransactionType: 'TrustSet',

TxnSignature: '3045022100B1F3D3BC700739D234531096474F604ABE7A8E2499D3978B328984A8F966551A0220493513CB305F2D39E4E685991302CFC44740AC7D6036240D225911C6081828CC,

hash: 'E9A0DD0110C9400C3E3C1619327D1037FC4377141EF8D79120F00B3FCAB0D914' }

}

返回结果说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | | | 类型 | 说明 |
| engine\_result | | | String | 请求结果 |
| engine\_result\_code | | | Array | 请求结果编码 |
| engine\_result\_message | | | String | 请求结果message信息 |
| tx\_blob | | | String | 16进制签名后的交易 |
| tx\_json | | | Object | 交易内容 |
|  | Account | | String | 账号地址 |
| Fee | | String | 交易费 |
| Flags | | Integer | 交易标记 |
| LimitAmount | | Object | 关系的额度 |
|  | currency | String | 货币 |
| issuer | String | 货币发行方 |
| value | String | 额度 |
| RelationType | | Integer | 关系类型：0信任(信任关系中，该字段不显示)；1授权；3冻结/解冻； |
| Sequence | | Integer | 单子序列号 |
| SigningPubKey | | Object | 签名公钥 |
| Target | | String | 关系对家，有对家才显示 |
| Timestamp | | Integer | 时间戳，签名方式没有该字段 |
| TransactionType | | String | 交易类型：TrustSet信任;RelationDel解冻；RelationSet授权/冻结 |
| TxnSignature | | String | 交易签名 |
| hash | | String | 交易hash |

## 设置账号属性 -----------待完善

首先通过buildAccountSetTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法设置账号属性。目前支持的三类：`property`、 `delegate` 、`signer`。property用于设置账号一般属性；delegate用于某账号设置委托帐户；signer用于设置签名。

### 创建属性对象

方法：remote.buildAccountSetTx({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| type | String | 属性种类 |
| account | String | 设置属性的源账号 |
| set\_flag | String | 属性编号 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

### 属性设置

方法：tx.submit(callback);

参数：无

设置属性完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port'});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var options = {

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

type:'property'

};

var tx = remote.buildAccountSetTx(options);

tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

## 挂单

首先通过buildOfferCreateTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法提交挂单。

### 创建挂单对象

方法：remote.buildOfferCreateTx({});

参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| type | | String | 挂单类型，固定的两个值：Buy、Sell |
| account | | String | 挂单方账号 |
| platform | | String | 平台标识账号，可选 |
| taker\_gets | | Object | 对方得到的，即挂单方支付的 |
|  | value | String | 数量 |
| currency | String | 货币种类 |
| issuer | String | 货币发行方 |
| taker\_pays | | Object | 对方支付的，即挂单方获得的 |
|  | value | String | 数量 |
| currency | String | 货币种类 |
| issuer | String | 货币发行方 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

### 提交挂单

方法：tx.submit(callback);

参数：无

挂单完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var options = {

type: 'Sell',

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

taker\_gets: {

value: '1',

currency: 'CNY',

issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS'

},

taker\_pays: {

value: '200',

currency: 'SWT',

issuer: ''

}

};

var tx = remote.buildOfferCreateTx(options);

tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '120007220008000024000000212F24B0561864400000000BEBC20065D4838D7EA4C68000000000000000000000000000434E5900000000007478E561645059399B334448F7544F2EF308ED32684000000000002710732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD1677446304402207849F557F36C99D1977A5BE84451E9D92098F0DDD23FBB02B43C3ABCCFF5718B02207AF79377A402FACF5BAD79906D53A2CD071AEFBBF02F34AC16BA85F0DECF7F89811472F05993EBA9858291D364EBF6EEC3D851BD3792',

tx\_json:

{ Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

Fee: '10000',

Flags: 524288,

Sequence: 33,

SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',

TakerGets:

{ currency: 'CNY',

issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',

value: '1' },

TakerPays: '200000000',

TransactionType: 'OfferCreate',

TxnSignature: '304402207849F557F36C99D1977A5BE84451E9D92098F0DDD23FBB02B43C3ABCCFF5718B02207AF79377A402FACF5BAD79906D53A2CD071AEFBBF02F34AC16BA85F0DECF7F89',

hash: 'C120ACB1869F20DF9955F4CF9B10AC033C373B7D653A601FEFF164997D1D4DD9' }

}

返回结果说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | | | 类型 | 说明 |
| engine\_result | | | String | 请求结果 |
| engine\_result\_code | | | Array | 请求结果编码 |
| engine\_result\_message | | | String | 请求结果message信息 |
| tx\_blob | | | String | 16进制签名后的交易 |
| tx\_json | | | Object | 交易内容 |
|  | Account | | String | 账号地址 |
| Fee | | String | 交易费 |
| Platform | | String | 平台标记账号，设置了才有此字段 |
| Flags | | Integer | 交易标记 |
| Sequence | | Integer | 单子序列号 |
| SigningPubKey | | String | 签名公钥 |
| TakerGets | | Object | 对家得到的 |
|  | currency | String | 货币 |
| issuer | String | 货币发行方 |
| value | String | 额度 |
| TakerPays | | String | 对家支付的； |
| Timestamp | | Integer | 时间戳，签名方式没有该字段 |
| TransactionType | | String | 交易类型：TrustSet信任;RelationDel解冻；RelationSet授权/冻结 |
| TxnSignature | | String | 交易签名 |
| hash | | String | 交易hash |

## 取消挂单

首先通过buildOfferCancelTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法取消挂单。

### 创建取消挂单对象

方法：remote.buildOfferCancelTx({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 挂单方账号 |
| sequence | Integer | 取消的单子号 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

### 取消挂单

方法：tx.submit(callback);

参数：无

取消挂单完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequence: 33};

var tx = remote.buildOfferCancelTx(options);

tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '1200082200000000240000002C201900000021684000000000002710732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD16774463044022079DEE5E39A5A682E10270E411B95036ACA0F4928119B3A9E5BA58A32BFC392BC022024AA19530FE6D8D61F46875D920DCCB3253AD97D3998E3BF1200F608DBC9A9F3811472F05993EBA9858291D364EBF6EEC3D851BD3792',

tx\_json:

{ Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

Fee: '10000',

Flags: 0,

OfferSequence: 33,

Sequence: 44,

SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',

TransactionType: 'OfferCancel',

TxnSignature: '3044022079DEE5E39A5A682E10270E411B95036ACA0F4928119B3A9E5BA58A32BFC392BC022024AA19530FE6D8D61F46875D920DCCB3253AD97D3998E3BF1200F608DBC9A9F3',

hash: '700894697DB481487599C907872BF721722791718AE704F11FFC3A5E77FDA9FE' }

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 账号地址 |
| Fee | String | 交易费 |
| Flags | Integer | 交易标记 |
| OfferSequence | Integer | 取消的单子号 |
| Sequence | Integer | 单子序列号 |
| SigningPubKey | String | 签名公钥 |
| Timestamp | Integer | 时间戳，签名方式没有该字段 |
| TransactionType | String | 交易类型：OfferCancel取消订单 |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 部署合约（Lua版）

首先通过deployContractTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法部署合约。

### 创建部署合约对象

方法：remote.deployContractTx({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 合约交易源账号 |
| amount | String/Number | 支付金额(最多支持六位小数) |
| payload | String | 智能合约代码(16进制字符串) |

可选参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| params | String | 合约参数 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 源账号私钥 |

### 部署合约

方法：tx.submit(callback);

参数：无

部署合约完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var utils = jlib.utils;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var options = {account: 'jHb9\*\*\*tyTh', amount: 10,payload: utils.stringToHex('result={}; function Init(t) result=scGetAccountBalance(t) return result end; function foo(t) result=scGetAccountBalance(t) return result end'),params: ['jHb9\*\*\*tyTh']};

var tx = remote.deployContractTx(options);

tx.setSecret('s\*\*');

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

返回结果：

{

ContractState: 'jQUzfdE2ZnQLz3AbxuSbyyyQhuoXAxUn3A',

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob

tx\_json:

{ Account: 'jHb9\*\*\*tyTh',

Amount: '10000000',

Args: [ [Object] ],

Fee: '10000',

Flags: 0,

Method: 0,

Payload: '726573756C743D7B7D3B202066756E6374696F6E20496E69742874292020726573756C743D73634765744163636F756E7442616C616E6365287429202072657475726E20726573756C742020656E643B202066756E6374696F6E20666F6F2874292020726573756C743D73634765744163636F756E7442616C616E6365287429202072657475726E20726573756C742020656E64',

Sequence: 1274,

SigningPubKey: '0330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD020',

TransactionType: 'ConfigContract',

TxnSignature: '3045022100E53D151ED6EEC46124CB75C0ED0DCACC7D4458B56109252431DAA80D11D1099802207FE4C2987AFBBD7F1FEC6DBFBD9A32C35D2F82CEC44E5AA6EE3DE0E58F1D8B62',

hash: 'D7E40A7164C11BFA81C05117010631EA5BCBD8A1A0B3B2FF343D9FB3F3575936' }

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| ContractState | | String | 生成的合约地址 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 账号地址 |
| Fee | String | 交易费 |
| Flags | Integer | 交易标记 |
| Method | Integer | 合约交易方法：0表示部署；1表示调用 |
| Payload | Integer | 16进制合约代码 |
| Sequence | Integer | 单子序列号 |
| SigningPubKey | String | 签名公钥 |
| TransactionType | String | 交易类型：ConfigContract部署合约 |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 执行合约（Lua版）

首先通过callContractTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法执行合约。

### 创建执行合约对象

方法：remote.callContractTx({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 合约交易源账号 |
| destination | String | 合约地址 |
| func | String | 合约函数名 |

可选参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| params | String | 合约参数 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 源账号私钥 |

### 执行合约

方法：tx.submit(callback);

参数：无

部署合约完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var options = {account: 'jHb9\*\*\*tyTh', destination: 'j4YVQxCxaRRQ6gCVUvi9MoiTfWyPRnHwej',func: 'foo',params: ['jHb9\*\*\*tyTh']};

var tx = remote.callContractTx(options);

tx.setSecret('s\*\*');

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

返回结果：

{

ContractState: '599868797812411271',

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '12001E220000000024000004FC20240000000168400000000000271073210330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD02074473045022100835BEDFD6794F3B95A31971EDBF1865F9190D1A66FFC9515883B15107B368894022052F44E0E16B6A3BC3C8CF2A3881E35687060EB73B6B0503611CEF58781A512EA701103666F6F8114B5F762798A53D543A014CAF8B297CFF8F2F937E88314EC4F411C3B37C3E451D462C27489B0549650572CFAEB7012226A486239434A41577942346A7239315652576E3936446B756B473462776474795468E1F1',

tx\_json:

{ Account: 'jHb9\*\*\*tyTh',

Args: [ [Object] ],

ContractMethod: '666F6F',

Destination: 'j4YVQxCxaRRQ6gCVUvi9MoiTfWyPRnHwej',

Fee: '10000',

Flags: 0,

Method: 1,

Sequence: 1276,

SigningPubKey: '0330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD020',

TransactionType: 'ConfigContract',

TxnSignature: '3045022100835BEDFD6794F3B95A31971EDBF1865F9190D1A66FFC9515883B15107B368894022052F44E0E16B6A3BC3C8CF2A3881E35687060EB73B6B0503611CEF58781A512EA',

hash: '8DE51A01F6FA55F5FDBB196A39A1F4220772E19835C81CE2E8A44B487BCD961C' }

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| ContractState | | String | 调用的合约结果 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 账号地址 |
| Args | Array | 合约传入的参数 |
| ContractMethod | String | 合约函数名 |
| Destination | String | 调用的合约地址 |
| Fee | String | 交易费 |
| Flags | Integer | 交易标记 |
| Method | Integer | 合约交易方法：0表示部署；1表示调用 |
| Sequence | Integer | 单子序列号 |
| SigningPubKey | String | 签名公钥 |
| TransactionType | String | 交易类型：ConfigContract合约类 |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 设置挂单佣金

首先通过buildBrokerageTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法设置平台手续费。

### 创建挂单佣金对象

方法：remote.buildBrokerageTx({});

参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| account | | String | App平台标识账号 |
| feeAccount | | String | 收费账号，必须为激活账号 |
| mol | | Integer | 分子（0和正整数） |
| den | | Integer | 分母（正整数） |
| amount | | Object | 币种对象 |
|  | value | String | 数量，这里只是占位，没有实际意义。 |
| currency | String | 货币种类 |
| issuer | String | 货币发行方 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | App平台标识账号私钥 |

### 设置挂单佣金

方法：tx.submit(callback);

参数：无

设置挂单佣金完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var v = {

secret: 's...UTb',

address: 'j...yTh'

};

var req = remote.buildBrokerageTx({account: v.address, mol: 1, den: 1000, feeAccount:'jzTx4CRUZJT1ZsBhGHi7Wqikada63xRVv',

amount: {

"value": "3",

"currency": "TES",

"issuer": "jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS"

}});

req.setSecret(v.secret);

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '1200CD220000000024000000012025000000013900000000000000013A00000000000003E861D38AA87BEE53800000000000000000000000005445535400000000007478E561645059399B334448F7544F2EF308ED3268400000000000271073210330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD02074473045022100FE513087425A863D1FB0004B29C6656E1B4C237F57F193FC5707317257926787022070AA7125CB5383F5B53000903BDAF9926CBE74F732DF7E09BE3EC581FEF5E7C08114B5F762798A53D543A014CAF8B297CFF8F2F937E8',

tx\_json:

{ Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount:

{ currency: 'TEST',

issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',

value: '3' },

Fee: '10000',

FeeAccountID: 'jzTx4CRUZJT1ZsBhGHi7Wqikada63xRVv',

Flags: 0,

OfferFeeRateDen: '00000000000003E8',

OfferFeeRateNum: '0000000000000001',

Sequence: 1,

SigningPubKey: '0330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD020',

TransactionType: 'Brokerage',

TxnSignature: '3045022100FE513087425A863D1FB0004B29C6656E1B4C237F57F193FC5707317257926787022070AA7125CB5383F5B53000903BDAF9926CBE74F732DF7E09BE3EC581FEF5E7C0',

hash: '3260743D0233EA5B8E2963F00D35B4198BA9EC6BE7638E84C9375A3CBE2172A3' }

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 管理员账号地址 |
| Amount | Object | 收交易手续费的币种信息 |
| Fee | String | 网络费 |
| feeAccountID | String | 收费账号 |
| Flags | Integer | 交易标记 |
| OfferFeeRateDen | String | 分母 |
| OfferFeeRateNum | String | 分子 |
| Sequence | Integer | 单子序列号 |
| SigningPubKey | String | 签名公钥 |
| TransactionType | String | 交易类型：Brokerage设置交易手续费类 |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 部署合约（Solidity版）

首先通过initContract方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法完成合约的部署。

### 创建合约部署对象

方法：remote.initContract({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 合约发布者 |
| amount | Integer | 手续费 |
| payload | String | 合约编译后的16进制字节码 |
| abi | Array | 合约abi |
| params | Array | 可选，合约初始化参数 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 合约发布者账号私钥 |

### 部署合约

方法：tx.submit(callback);

参数：无

部署合约完整例子(solidity源码见文件最后[Solidity ERC20源码])：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var v = {

secret: 's...UTb',

address: 'j...yTh'

};

const abi = [

{

"constant": true,

"inputs": [],

"name": "name",

"outputs": [

{

"name": "",

"type": "string"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "totalSupply",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "decimals",

"outputs": [

{

"name": "",

"type": "uint8"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "",

"type": "address"

}

],

"name": "balanceOf",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "symbol",

"outputs": [

{

"name": "",

"type": "string"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": false,

"inputs": [

{

"name": "\_to",

"type": "address"

},

{

"name": "\_value",

"type": "uint256"

}

],

"name": "transfer",

"outputs": [],

"payable": false,

"stateMutability": "nonpayable",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "",

"type": "address"

},

{

"name": "",

"type": "address"

}

],

"name": "allowance",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"inputs": [

{

"name": "initialSupply",

"type": "uint256"

},

{

"name": "tokenName",

"type": "string"

},

{

"name": "tokenSymbol",

"type": "string"

}

],

"payable": false,

"stateMutability": "nonpayable",

"type": "constructor"

},

{

"payable": false,

"stateMutability": "nonpayable",

"type": "fallback"

}

];

var tx = remote.initContract({

account: v.address,

amount: 10,

payload: '',

abi: abi,

params:[2000, 'TestCurrency', 'TEST1']

});

tx.setSecret(v.secret);

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

返回结果：

{

ContractState: 'jPZ1....9Kkh',

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '',

tx\_json:

{ Account: 'jHb9....tyTh',

Amount: '10000000',

Fee: '10000',

Flags: 0,

Method: 0,

Payload: '36303630363034303532363031323630303236303030363130313030306138313534383136306666303231393136393038333630666631363032313739303535353033343135363130303262353736303030383066643562363034303531363130393237333830333830363130393237383333393831303136303430353238303830353139303630323030313930393139303830353138323031393139303630323030313830353138323031393139303530353036303032363030303930353439303631303130303061393030343630666631363630666631363630306130613833303236303033383139303535353036303033353436303034363030303333373366666666666666666666666666666666666666666666666666666666666666666666666666666666313637336666666666666666666666666666666666666666666666666666666666666666666666666666666631363831353236303230303139303831353236303230303136303030323038313930353535303831363030303930383035313930363032303031393036313030643339323931393036313030663335363562353038303630303139303830353139303630323030313930363130306561393239313930363130306633353635623530353035303530363130313938353635623832383035343630303138313630303131363135363130313030303230333136363030323930303439303630303035323630323036303030323039303630316630313630323039303034383130313932383236303166313036313031333435373830353136306666313931363833383030313137383535353631303136323536356238323830303136303031303138353535383231353631303136323537393138323031356238323831313131353631303136313537383235313832353539313630323030313931393036303031303139303631303134363536356235623530393035303631303136663931393036313031373335363562353039303536356236313031393539313930356238303832313131353631303139313537363030303831363030303930353535303630303130313631303137393536356235303930353635623930353635623631303738303830363130316137363030303339363030306633303036303630363034303532363030343336313036313030383335373630303033353763303130303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303930303436336666666666666666313638303633303666646465303331343631303039333537383036333138313630646464313436313031323135373830363333313363653536373134363130313461353738303633373061303832333131343631303137393537383036333935643839623431313436313031633635373830363361393035396362623134363130323534353738303633646436326564336531343631303239363537356233343135363130303865353736303030383066643562363030303830666435623334313536313030396535373630303038306664356236313030613636313033303235363562363034303531383038303630323030313832383130333832353238333831383135313831353236303230303139313530383035313930363032303031393038303833383336303030356238333831313031353631303065363537383038323031353138313834303135323630323038313031393035303631303063623536356235303530353035303930353039303831303139303630316631363830313536313031313335373830383230333830353136303031383336303230303336313031303030613033313931363831353236303230303139313530356235303932353035303530363034303531383039313033393066333562333431353631303132633537363030303830666435623631303133343631303361303536356236303430353138303832383135323630323030313931353035303630343035313830393130333930663335623334313536313031353535373630303038306664356236313031356436313033613635363562363034303531383038323630666631363630666631363831353236303230303139313530353036303430353138303931303339306633356233343135363130313834353736303030383066643562363130316230363030343830383033353733666666666666666666666666666666666666666666666666666666666666666666666666666666663136393036303230303139303931393035303530363130336239353635623630343035313830383238313532363032303031393135303530363034303531383039313033393066333562333431353631303164313537363030303830666435623631303164393631303364313536356236303430353138303830363032303031383238313033383235323833383138313531383135323630323030313931353038303531393036303230303139303830383338333630303035623833383131303135363130323139353738303832303135313831383430313532363032303831303139303530363130316665353635623530353035303530393035303930383130313930363031663136383031353631303234363537383038323033383035313630303138333630323030333631303130303061303331393136383135323630323030313931353035623530393235303530353036303430353138303931303339306633356233343135363130323566353736303030383066643562363130323934363030343830383033353733666666666666666666666666666666666666666666666666666666666666666666666666666666663136393036303230303139303931393038303335393036303230303139303931393035303530363130343666353635623030356233343135363130326131353736303030383066643562363130326563363030343830383033353733666666666666666666666666666666666666666666666666666666666666666666666666666666663136393036303230303139303931393038303335373366666666666666666666666666666666666666666666666666666666666666666666666666666666313639303630323030313930393139303530353036313034376535363562363034303531383038323831353236303230303139313530353036303430353138303931303339306633356236303030383035343630303138313630303131363135363130313030303230333136363030323930303438303630316630313630323038303931303430323630323030313630343035313930383130313630343035323830393239313930383138313532363032303031383238303534363030313831363030313136313536313031303030323033313636303032393030343830313536313033393835373830363031663130363130333664353736313031303038303833353430343032383335323931363032303031393136313033393835363562383230313931393036303030353236303230363030303230393035623831353438313532393036303031303139303630323030313830383331313631303337623537383239303033363031663136383230313931356235303530353035303530383135363562363030333534383135363562363030323630303039303534393036313031303030613930303436306666313638313536356236303034363032303532383036303030353236303430363030303230363030303931353039303530353438313536356236303031383035343630303138313630303131363135363130313030303230333136363030323930303438303630316630313630323038303931303430323630323030313630343035313930383130313630343035323830393239313930383138313532363032303031383238303534363030313831363030313136313536313031303030323033313636303032393030343830313536313034363735373830363031663130363130343363353736313031303038303833353430343032383335323931363032303031393136313034363735363562383230313931393036303030353236303230363030303230393035623831353438313532393036303031303139303630323030313830383331313631303434613537383239303033363031663136383230313931356235303530353035303530383135363562363130343761333338333833363130346133353635623530353035363562363030353630323035323831363030303532363034303630303032303630323035323830363030303532363034303630303032303630303039313530393135303530353438313536356236303030383038333733666666666666666666666666666666666666666666666666666666666666666666666666666666663136313431353135313536313034636135373630303038306664356238313630303436303030383637336666666666666666666666666666666666666666666666666666666666666666666666666666666631363733666666666666666666666666666666666666666666666666666666666666666666666666666666663136383135323630323030313930383135323630323030313630303032303534313031353135313536313035313835373630303038306664356236303034363030303834373366666666666666666666666666666666666666666666666666666666666666666666666666666666313637336666666666666666666666666666666666666666666666666666666666666666666666666666666631363831353236303230303139303831353236303230303136303030323035343832363030343630303038363733666666666666666666666666666666666666666666666666666666666666666666666666666666663136373366666666666666666666666666666666666666666666666666666666666666666666666666666666313638313532363032303031393038313532363032303031363030303230353430313131313531353631303561363537363030303830666435623630303436303030383437336666666666666666666666666666666666666666666666666666666666666666666666666666666631363733666666666666666666666666666666666666666666666666666666666666666666666666666666663136383135323630323030313930383135323630323030313630303032303534363030343630303038363733666666666666666666666666666666666666666666666666666666666666666666666666666666663136373366666666666666666666666666666666666666666666666666666666666666666666666666666666313638313532363032303031393038313532363032303031363030303230353430313930353038313630303436303030383637336666666666666666666666666666666666666666666666666666666666666666666666666666666631363733666666666666666666666666666666666666666666666666666666666666666666666666666666663136383135323630323030313930383135323630323030313630303032303630303038323832353430333932353035303831393035353530383136303034363030303835373366666666666666666666666666666666666666666666666666666666666666666666666666666666313637336666666666666666666666666666666666666666666666666666666666666666666666666666666631363831353236303230303139303831353236303230303136303030323036303030383238323534303139323530353038313930353535303830363030343630303038353733666666666666666666666666666666666666666666666666666666666666666666666666666666663136373366666666666666666666666666666666666666666666666666666666666666666666666666666666313638313532363032303031393038313532363032303031363030303230353436303034363030303837373366666666666666666666666666666666666666666666666666666666666666666666666666666666313637336666666666666666666666666666666666666666666666666666666666666666666666666666666631363831353236303230303139303831353236303230303136303030323035343031313431353135363130373465353766653562353035303530353035363030613136353632376137613732333035383230623236363564663064386438353232383033613139616336626339386666303130313231653131633136643033343265616365643031643934313030636531383030323930303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030376430303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303036303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030613030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303063353436353733373434333735373237323635366536333739303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303535343435353335343331303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030',

Sequence: 413,

SigningPubKey: '0330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD020',

TransactionType: 'AlethContract',

TxnSignature: '304402206C941CF3355094C5E85C2410FBB131288802094B790C399639A543A9DD64DDB1022059EEA991A946A28EA9695AEB2F5F724831C371BB0F9C72D50DD6855BCDF5C07D',

hash: '7BAAF8543E5E5761500CB21F5BCBCB9CA7A786704E82153481A7944E8F6E97F9'

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| ContractState | | String | 返回合约账号 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 合约发起者账号地址 |
| Amount | Object | 收交易手续费的币种信息 |
| Fee | String | 网络费 |
| Flags | Integer | 交易标记 |
| Method | Integer | 合约方法：0表示部署，1表示调用 |
| Payload | String | 合约编译后的16进制字节码 |
| Sequence | Integer | 单子序列号 |
| SigningPubKey | String | 签名公钥 |
| TransactionType | String | 交易类型：AlethContract合约类 |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 调用合约（Solidity版）

首先通过invokeContract方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法完成合约的调用。

### 创建合约调用对象

方法：remote.invokeContract({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 合约调用者 |
| destination | String | 合约账号 |
| abi | Array | 合约abi |
| func | String | 合约函数名及参数 |
| amount | Number | 可选，合约中msg.value的值。只有payable修饰的合约函数，才可以设置该值，默认为0。 |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 合约调用者账号私钥 |

### 调用合约

方法：tx.submit(callback);

参数：无

调用合约完整例子(solidity源码见文件最后[Solidity ERC20源码])：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var v = {

secret: 's...UTb',

address: 'j...yTh'

};

const abi = [

{

"constant": true,

"inputs": [],

"name": "name",

"outputs": [

{

"name": "",

"type": "string"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "totalSupply",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "decimals",

"outputs": [

{

"name": "",

"type": "uint8"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "",

"type": "address"

}

],

"name": "balanceOf",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": true,

"inputs": [],

"name": "symbol",

"outputs": [

{

"name": "",

"type": "string"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"constant": false,

"inputs": [

{

"name": "\_to",

"type": "address"

},

{

"name": "\_value",

"type": "uint256"

}

],

"name": "transfer",

"outputs": [],

"payable": false,

"stateMutability": "nonpayable",

"type": "function"

},

{

"constant": true,

"inputs": [

{

"name": "",

"type": "address"

},

{

"name": "",

"type": "address"

}

],

"name": "allowance",

"outputs": [

{

"name": "",

"type": "uint256"

}

],

"payable": false,

"stateMutability": "view",

"type": "function"

},

{

"inputs": [

{

"name": "initialSupply",

"type": "uint256"

},

{

"name": "tokenName",

"type": "string"

},

{

"name": "tokenSymbol",

"type": "string"

}

],

"payable": false,

"stateMutability": "nonpayable",

"type": "constructor"

},

{

"payable": false,

"stateMutability": "nonpayable",

"type": "fallback"

}

];

var tx = remote.invokeContract({

account: v.address,

destination: 'jPZ1....9Kkh',

abi: abi,

func:"transfer('jPZ1....9Kkh', 15)"});

tx.setSecret(v.secret);

tx.submit(function (err, result) {

if (err) {

console.log('err:', err);

}

else if (result) {

console.log('res:', result);

}

});

});

返回结果：

{

ContractState: '',

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob

tx\_json:

{ Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Args: [ { Arg:

{ ContractParamsType: 0,

Parameter: '61393035396362623030303030303030303030303030303030303030303030306637386136396463643533303864623330633431373234663034656638366361336163373461623330303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303066' } } ],

Destination: 'jPZ1....9Kkh',

Fee: '10000',

Flags: 0,

Method: 1,

Sequence: 414,

SigningPubKey: '0330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD020',

TransactionType: 'AlethContract',

TxnSignature: '3045022100C44640B21BF99C1764042A87FCAE6877BF1188F9F8DD887969EEF735333E584002204A2158F843021522FF7F93940BA4C1A933E21F03B6B9E5A3F3A54C217428CDDC',

hash: '561190424A8325FE6A93A2E3D1E16852B67CCC46354DA766F63C60A73403377A'

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| ContractState | | String | 调用的合约方法返回值，没有返回值显示空字符串 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 合约发起者账号地址 |
| Args | Array | 合约调用参数信息 |
| Destination | String | 合约账号 |
| Fee | String | 网络费 |
| Flags | Integer | 交易标记 |
| Method | Integer | 合约方法：0表示部署，1表示调用 |
| Sequence | Integer | 单子序列号 |
| SigningPubKey | String | 签名公钥 |
| TransactionType | String | 交易类型：AlethContract合约类 |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 废除账号主密钥masterkey

本功能为禁止某账号做交易而设定，**且只有该账号设置了签名列表才可以废除masterkey成功**。首先通过buildAccountSetTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法提交到底层。

### 创建签名列表对象

方法：remote.buildAccountSetTx({});

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 被废除或激活masterkey的账号地址 |
| type | String | 类型，账号属性类固定为property |
| set\_flag | Integer | 4表示废除masterkey |
| clear\_flag | Integer | 4表示激活masterkey，用于多签中激活masterkey |

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

### 废除masterkey

方法：tx.submit(callback);

参数：无

废除masterkey完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port:5020', local\_sign:true});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var a1 = {address: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb', secret: 'ssR...GT'};

var options = {

account: a1.address,

type:'property',

set\_flag: 4};

var tx = remote.buildAccountSetTx(options);

tx.setSecret(root.secret);

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '120003220000000024000000042F24FF8316202100000004684000000000002710732103B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF74473045022100E08B43B5A1FF871015074C46E8A608AB0AB3F823415F58B4E2546572D07D94900220785786BED6701F2F6ED5F36536337DE90DA12B0C262157F6E21FFCEE9258D04F8114577F22BBCFC872325BB006322692FDC60AE33890',

tx\_json:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '10000',

Flags: 0,

Sequence: 4,

SetFlag: 4,

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TransactionType: 'AccountSet',

TxnSignature: '3045022100E08B43B5A1FF871015074C46E8A608AB0AB3F823415F58B4E2546572D07D94900220785786BED6701F2F6ED5F36536337DE90DA12B0C262157F6E21FFCEE9258D04F',

hash: 'A30EB9C735A9A7D779A9ABAA8E349186B83AA30513C5C8DBDBE4B47ADE145194' }

}

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| engine\_result | | String | 请求结果 |
| engine\_result\_code | | Array | 请求结果编码 |
| engine\_result\_message | | String | 请求结果message信息 |
| tx\_blob | | String | 16进制签名后的交易 |
| tx\_json | | Object | 交易内容 |
|  | Account | String | 交易源账号地址 |
| Fee | String | 交易费 |
| Flags | Integer | 交易标记 |
| Sequence | Integer | 单子序列号 |
| SetFlag | Integer | 账号属性标记 |
| SigningPubKey | String | 签名公钥 |
| TransactionType | String | 交易类型，账号属性类为AccountSet |
| TxnSignature | String | 交易签名 |
| hash | String | 交易hash |

## 激活账号主密钥masterkey

激活通过多签列表中的账号去完成激活，如用账号a2和a3激活，详见下面例子。

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port:5020'});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var a1 = {address: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb', secret: 'ssR...GT'};

var a2 = {address: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz', secret: 'ssL...Sy'};

var a3 = {address: 'jpX8SEpM387c9tpdAUfBr2gYTfC2k7RatA', secret:'ss6...ki'};

var options = {

account: a1.address,

type:'property',

**clear\_flag: 4 //激活**

};

var tx = remote.buildAccountSetTx(options);

tx.setSequence(6);

tx.setFee(100000);

tx.multiSigning({

account: a2.address,

secret: a2.secret

});

tx.multiSigning({

account: a3.address,

secret: a3.secret

});

tx.multiSigned();

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob

tx\_json:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

ClearFlag: 4,

Fee: '100000',

Flags: 0,

Sequence: 6,

Signers: [ { Signer:

{ Account: 'jpX8SEpM387c9tpdAUfBr2gYTfC2k7RatA',

SigningPubKey: '02E10087455D301201958DD926D2D195999E88DB1424A3C2592DE058C8732D56F9',

TxnSignature: '3045022100B955553F2EC08FDEB86C9D45A3D0D5D0A94A5586513BF57D0E830CA65D6F4BAC0220139DC103DE74CA0D1CFB19F98CBCD24AF01E291BD47FF1D615ED406D495CBEDD' } },

{ Signer:

{ Account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

SigningPubKey: '03F54EA6509AD28E8E9AE8762B07CD245B5AB87DC840723DBF6C4C0A621A42B4A1',

TxnSignature: '304502210092480D9BA95B3ED8A2F979A48627D20951D6DD4451C922D7B35DA5A18BAC8E6B02201063A4F3CDEB376C8B089B6628F939CFDE9AEA4AD64A194863C3F5FA181684F3' } } ],

SigningPubKey: '',

TransactionType: 'AccountSet',

hash: '25B0401932307D65AB84066AE7F255794D1BCFFA1D5219A121D102C968ED15FA' }

}

返回结果说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | | | | 类型 | 说明 |
| engine\_result | | | | String | 请求结果 |
| engine\_result\_code | | | | Array | 请求结果编码 |
| engine\_result\_message | | | | String | 请求结果message信息 |
| tx\_blob | | | | String | 16进制签名后的交易 |
| tx\_json | | | | Object | 交易内容 |
|  | Account | | | String | 交易源账号地址 |
| ClearFlag | | | Integer | 账号属性标记 |
| Fee | | | String | 交易费 |
| Flags | | | Integer | 交易标记 |
| Sequence | | | Integer | 单子序列号 |
| Signers | | | Array | 签名列表条目；销毁列表时，没有该字段 |
|  | Signer | | Object | 单个签名条目 |
|  | Account | String | 给该交易签名的账号地址 |
| SigningPubKey | String | 给该交易签名的账号公钥 |
| TxnSignature | String | Account账号给该交易的交易签名 |
| SigningPubKey | | | String | 签名公钥 |
| TransactionType | | | String | 交易类型，账号属性类为AccountSet |
| hash | | | String | 交易hash |

## 设置签名列表

首先通过buildSignerListTx方法返回一个Transaction对象，然后通过setSecret传入密钥，最后通过submit方法提交列表信息。

### 创建签名列表对象

方法：remote.buildSignerListTx({});

参数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | | | 类型 | 说明 |
| account | | | String | 交易源账号 |
| threshold | | | Integer | 阈值 |
| lists | | | Array | 签名列表 |
|  | ---- | | Object | 每个列表对象 |
|  | account | String | 账号 |
| weight | String | 权重 |

**注：不可将交易源账号设为签名列表名单内。**

返回：Transaction对象

### 传入密钥

方法：tx.setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

### 设置签名列表

方法：tx.submit(callback);

参数：无

设置关系完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port:5020', local\_sign:true});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var root = {'secret':'sno...Tb', 'address':'jHb...Th'};

var options = {

account: root.address,

threshold: 3,

lists: [{

account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

weight: 2

},

{

account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

weight: 2

},

{

account: 'jpX8SEpM387c9tpdAUfBr2gYTfC2k7RatA',

weight: 2

}

]};

var tx = remote.buildSignerListTx(options);

tx.setSecret(root.secret);

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob: '1200CF2200000000240000001220260000000368400000000000271073210330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD0207446304402205BAC48B1D53AB7FDF4C907DBFA2A248E779A382A74D91C8EBA50383C02C80ECF0220113DB1AE15B81B089457D189C45DA80EDE859C919C585368BA4ED2479E4A7ECC8114B5F762798A53D543A014CAF8B297CFF8F2F937E8FBEC1300028114577F22BBCFC872325BB006322692FDC60AE33890E1EC1300028114BC51DE21D4591EBF30812B1071C5E1AABBA07DE4E1EC130002811410CA883371FE56DD81D4556C05BCC082C77CF1D3E1F1',

tx\_json:

{ Account: 'jHb...Th',

Fee: '10000',

Flags: 0,

Sequence: 18,

SignerEntries: [ { SignerEntry:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SignerWeight: 2 } },

{ SignerEntry:

{ Account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

SignerWeight: 2 } },

{ SignerEntry:

{ Account: 'jpX8SEpM387c9tpdAUfBr2gYTfC2k7RatA',

SignerWeight: 2 } } ],

SignerQuorum: 3,

SigningPubKey: '0330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD020',

TransactionType: 'SignerListSet',

TxnSignature: '304402205BAC48B1D53AB7FDF4C907DBFA2A248E779A382A74D91C8EBA50383C02C80ECF0220113DB1AE15B81B089457D189C45DA80EDE859C919C585368BA4ED2479E4A7ECC',

hash: 'A080F767A94F57BF79CEC760A25539177C870E5A133D2BAF30419B8BC29550DB' }

}

返回结果说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | | | | 类型 | 说明 |
| engine\_result | | | | String | 请求结果 |
| engine\_result\_code | | | | Array | 请求结果编码 |
| engine\_result\_message | | | | String | 请求结果message信息 |
| tx\_blob | | | | String | 16进制签名后的交易 |
| tx\_json | | | | Object | 交易内容 |
|  | Account | | | String | 交易源账号地址 |
| Fee | | | String | 交易费 |
| Flags | | | Integer | 交易标记 |
| Sequence | | | Integer | 单子序列号 |
| SignerEntries | | | Array | 签名列表条目；销毁列表时，没有该字段 |
|  | SignerEntry | | Object | 单个签名条目 |
|  | Account | String | 签名账号的地址 |
| SignerWeight | String | 该签名在多重签名交易中的权重 |
| SignerQuorum | | | Integer | 多重签名交易通过的阈值，应大于等于零，**零表示销毁签名列表** |
| SigningPubKey | | | String | 签名公钥 |
| TransactionType | | | String | 交易类型，设置签名列表类为SignerListSet |
| TxnSignature | | | String | 交易签名 |
| hash | | | String | 交易hash |

**注：再次设置，会覆盖掉之前的签名列表，以最新设置的为准。**

## 首个用户签名

首先组织好要签名的交易，然后通过buildSignFirstTx方法包装要签名的交易，返回一个Transaction对象。若只有一个用户签名，此时可通过submit方法提交；若有多个用户签名，则需调用下面的后续用户签名接口。

### 创建首个签名对象

方法：remote.buildSignFirstTx({});

参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| options | | Object | 参数对象 |
|  | tx | Object | 创建的Transaction对象 |
| account | String | 签名私钥对应的地址 |
| secret | String | 签名该交易的地址私钥 |

返回：Transaction对象

### 传入sequence

方法：tx.setSequence(sequence);

**注：sequence可通过remote.requestAccountInfo({account: account})方法获取。**

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| sequence | Integer | 账号单子序列号 |

### 提交签名

方法：tx.submit(callback);

参数：无

首个用户签名完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port:5020'});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var root = {'secret':'sno...Tb', 'address':'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh'};

var a1 = {address: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb', secret:'ssR...nGT'};

var tx= remote.buildPaymentTx({account: root.address, to: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb', amount: {

"value": "2",

"currency": "SWT",

"issuer": ""}});

var options = {

account: a1.address,

secret: a1.secret,

tx: tx};

remote.buildSignFirstTx(options);

tx.setSequence(25); //必写，保证sequence统一不变；

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

返回结果：

{

tx\_blob: '120000220000000024000000196140000000001E848068400000000000271073008114B5F762798A53D543A014CAF8B297CFF8F2F937E88314577F22BBCFC872325BB006322692FDC60AE33890FCED732103B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF744630440220550DFCACFF57FEC839D1A371F044BA8182E811F55B31B59177DE6D451CAF8A7A022028974CC56C3FD84FBB485563E24E030A46BB3C773892ADE44FCC22DD4790E6AE8114577F22BBCFC872325BB006322692FDC60AE33890E1F1',

tx\_json:

{ Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '10000',

Flags: 0,

Sequence: 7,

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '304402201E36F24885BD7F5794262C40E676EEF204531897CA2ACBCA8F58D292B5EF4A930220493BE7F5243040C8EB9C75AC795FCA1BFD7EB6E97C97C8AD2719F9F97EDF8397' } } ],

SigningPubKey: '',

TransactionType: 'Payment',

hash: 'FDB0D78C5A6E6D3319788B5F807CD0A84009E625459DBBD3F6FAF23E3243CD36' }

}

返回结果说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | | | | 类型 | 说明 |
| tx\_blob | | | | String | 16进制签名后的交易 |
| tx\_json | | | | Object | 交易内容 |
|  | Account | | | String | 交易源账号地址 |
| Fee | | | String | 交易费 |
| Flags | | | Integer | 交易标记 |
| Sequence | | | Integer | 单子序列号 |
| -- | | | -- | 相关交易对应的其他字段，如Payment类型有Amount、Destination字段，这里不一一列举 |
| Signers | | | Array | 签名列表条目；销毁列表时，没有该字段 |
|  | Signer | | Object | 单个签名条目 |
|  | Account | String | 给该交易签名的账号地址 |
| SigningPubKey | String | 给该交易签名的账号公钥 |
| TxnSignature | String | Account账号给该交易的交易签名 |
| SigningPubKey | | | String | 交易签名公钥，必须为空字符串 |
| TransactionType | | | String | 交易类型 |
| hash | | | String | 交易hash，该hash并没有存链上 |

## 后续用户签名

首先通过buildSignOtherTx方法返回一个Transaction对象，然后通过submit方法提交；若后续还有用户要签名，可继续使用buildSignOtherTx方法并submit提交获取返回结果中的tx\_json。

### 创建后续签名对象

方法：remote.buildSignOtherTx({});

参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| options | | Object | 参数对象 |
|  | tx\_json | Object | 上一个用户签完返回结果中的tx\_json |
| account | String | 签名私钥对应的地址 |
| secret | String | 签名该交易的地址私钥 |

返回：Transaction对象

### 提交签名

方法：tx.submit(callback);

参数：无

后续用户签名完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port:5020'});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var a2 = {address: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz', secret:'ss...bSy'};

var tx\_json = { Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '10000',

Flags: 0,

Sequence: 25,

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '30440220550DFCACFF57FEC839D1A371F044BA8182E811F55B31B59177DE6D451CAF8A7A022028974CC56C3FD84FBB485563E24E030A46BB3C773892ADE44FCC22DD4790E6AE' } } ],

SigningPubKey: '',

TransactionType: 'Payment',

hash: 'FDB0D78C5A6E6D3319788B5F807CD0A84009E625459DBBD3F6FAF23E3243CD36' };

var options = {account: a2.address, secret: a2.secret, tx\_json: tx\_json};

remote.buildSignOtherTx(options);

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

返回结果：

{

tx\_blob

tx\_json:

{ Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '10000',

Flags: 0,

Sequence: 25,

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '30440220550DFCACFF57FEC839D1A371F044BA8182E811F55B31B59177DE6D451CAF8A7A022028974CC56C3FD84FBB485563E24E030A46BB3C773892ADE44FCC22DD4790E6AE' } },

{ Signer:

{ Account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

SigningPubKey: '03F54EA6509AD28E8E9AE8762B07CD245B5AB87DC840723DBF6C4C0A621A42B4A1',

TxnSignature: '3045022100B4EE5FE2C3C209793F4893877062F942DFAF43D76225D881EB95ADA868A51B040220346494F45843CC8C28A6C53813C98440A57C5DD9BC7E84C58A3F712B9AEDEEB6' } } ],

SigningPubKey: '',

TransactionType: 'Payment',

hash: '62AF7991C9C5314C7F3D23A5F0FF28E8C13EAE6A13F3E1A6CA06E477223F9B70' }

}

返回结果说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | | | | 类型 | 说明 |
| tx\_blob | | | | String | 16进制签名后的交易 |
| tx\_json | | | | Object | 交易内容 |
|  | Account | | | String | 交易源账号地址 |
| Fee | | | String | 交易费 |
| Flags | | | Integer | 交易标记 |
| Sequence | | | Integer | 单子序列号 |
| -- | | | -- | 相关交易对应的其他字段，如Payment类型有Amount、Destination字段，这里不一一列举 |
| Signers | | | Array | 签名列表条目；销毁列表时，没有该字段 |
|  | Signer | | Object | 单个签名条目 |
|  | Account | String | 给该交易签名的账号地址 |
| SigningPubKey | String | 给该交易签名的账号公钥 |
| TxnSignature | String | Account账号给该交易的交易签名 |
| SigningPubKey | | | String | 交易签名公钥，必须为空字符串 |
| TransactionType | | | String | 交易类型 |
| hash | | | String | 交易hash，该hash并没有存链上 |

## 提交多重签名交易

首先通过buildMultisignedTx方法返回一个Transaction对象，然后通过submit方法完成多重签名**最后**的提交。

### 创建多重签名对象

方法：remote.buildMultisignedTx({});

参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| options | | Object | 参数对象 |
|  | tx\_json | Object | 最后一个用户签完返回结果中的tx\_json |

返回：Transaction对象

### 提交多重签名

方法：tx.submit(callback);

参数：无

后续用户签名完整例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port:5020'});

remote.connect(function(err, result) {

if (err) {

return console.log('err:',err);

}

var tx\_json = { Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '10000',

Flags: 0,

Sequence: 25,

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '30440220550DFCACFF57FEC839D1A371F044BA8182E811F55B31B59177DE6D451CAF8A7A022028974CC56C3FD84FBB485563E24E030A46BB3C773892ADE44FCC22DD4790E6AE' } },

{ Signer:

{ Account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

SigningPubKey: '03F54EA6509AD28E8E9AE8762B07CD245B5AB87DC840723DBF6C4C0A621A42B4A1',

TxnSignature: '3045022100B4EE5FE2C3C209793F4893877062F942DFAF43D76225D881EB95ADA868A51B040220346494F45843CC8C28A6C53813C98440A57C5DD9BC7E84C58A3F712B9AEDEEB6' } } ],

SigningPubKey: '',

TransactionType: 'Payment',

hash: '62AF7991C9C5314C7F3D23A5F0FF28E8C13EAE6A13F3E1A6CA06E477223F9B70' };

remote.buildMultisignedTx(tx\_json);

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob

tx\_json:

{ Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '10000',

Flags: 0,

Sequence: 25,

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '30440220550DFCACFF57FEC839D1A371F044BA8182E811F55B31B59177DE6D451CAF8A7A022028974CC56C3FD84FBB485563E24E030A46BB3C773892ADE44FCC22DD4790E6AE' } },

{ Signer:

{ Account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

SigningPubKey: '03F54EA6509AD28E8E9AE8762B07CD245B5AB87DC840723DBF6C4C0A621A42B4A1',

TxnSignature: '3045022100B4EE5FE2C3C209793F4893877062F942DFAF43D76225D881EB95ADA868A51B040220346494F45843CC8C28A6C53813C98440A57C5DD9BC7E84C58A3F712B9AEDEEB6' } } ],

SigningPubKey: '',

TransactionType: 'Payment',

hash: '62AF7991C9C5314C7F3D23A5F0FF28E8C13EAE6A13F3E1A6CA06E477223F9B70' }

}

返回结果说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | | | | 类型 | 说明 |
| engine\_result | | | | String | 请求结果 |
| engine\_result\_code | | | | Array | 请求结果编码 |
| engine\_result\_message | | | | String | 请求结果message信息 |
| tx\_blob | | | | String | 16进制签名后的交易 |
| tx\_json | | | | Object | 交易内容 |
|  | Account | | | String | 交易源账号地址 |
| Fee | | | String | 交易费 |
| Flags | | | Integer | 交易标记 |
| Sequence | | | Integer | 单子序列号 |
| -- | | | -- | 相关交易对应的其他字段，如Payment类型有Amount、Destination字段，这里不一一列举 |
| Signers | | | Array | 签名列表条目；销毁列表时，没有该字段 |
|  | Signer | | Object | 单个签名条目 |
|  | Account | String | 给该交易签名的账号地址 |
| SigningPubKey | String | 给该交易签名的账号公钥 |
| TxnSignature | String | Account账号给该交易的交易签名 |
| SigningPubKey | | | String | 交易签名公钥，必须为空字符串 |
| TransactionType | | | String | 交易类型 |
| hash | | | String | 交易hash，该hash可在链上查到 |

## 通过tx\_json创建Transaction实例

方法：buildTx(tx\_json)，主要用于多重签名相关；

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| tx\_json | Object | 交易结果返回的tx\_json字段，或者自己组织的tx\_json |

返回：Transaction对象

例子：

var tx\_json = { Flags: 0,

Fee: 100000,

TransactionType: 'Payment',

Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Sequence: 11,

SigningPubKey: '',

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '3045022100DDC408965E9F62626BC6A0603A0D707B64E0157B84299414BCF8C560EDE776970220093CB723375F3725DC41FB5B207B2CEEE158D5D70A90F18BD10D330F637DB9A3' } } ] }

var tx = remote.buildTx(tx\_json);

## 本地多重签名实现

首先组织交易(首个用户通过remote.buildxxxTx方式创建交易；后面的用户拿到上一用户返回结果中的tx\_json，通过remote.buildTx(tx\_json)方式创建交易)返回一个Transaction对象，然后multiSigning方法多重签名，多个用户签完后用multiSigned方法标记结束，最后通过submit方法完成多重签名的提交。由于多签在Transaction类中完成，具体方法说明可在Transaction类中查看，本小节只展示两种具体例子。

### 只有一个用户签名，交易信息由自己创建

var root = {

'secret': 'sn...Tb',

'address': 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh'

};

var a1 = {address: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb', secret: 'ss...GT'};

var tx = remote.buildPaymentTx({

account: root.address, to: a1.address, amount: {

"value": "2",

"currency": "SWT",

"issuer": ""

}

});

// tx.addMemo('支付test');

tx.setSequence(26);//必写

tx.setFee(100000); //可选，这里需补六个零

tx.multiSigning({ //用户签名

account: a1.address,

secret: a1.secret

});

var flag = tx.tx\_json.verifyTx && tx.tx\_json.verifyTx.toString().indexOf('verify failed');//判断签名是否成功

flag ? console.log('verify failed') : console.log('verify success');

// console.log('tx\_json: ', tx.tx\_json);//这里得到的是签名结果

if(!flag){

tx.multiSigned();//多重签名结束

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

}

返回结果：

{

engine\_result: 'tefBAD\_QUORUM',

engine\_result\_code: -184,

engine\_result\_message: 'Signatures provided do not meet the quorum.',

tx\_blob: '1200002200000000240000001A6140000000001E84806840000000000186A073008114B5F762798A53D543A014CAF8B297CFF8F2F937E88314577F22BBCFC872325BB006322692FDC60AE33890FCED732103B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF74473045022100FB82276780F6A049E4B45FCBD4EA815AEB21E7044286D7254611A5CFDE8BF1A3022023DD9C096DD5003F9A994F62BCAE54582B2E6AEA246D053CAE620868A4FE6E118114577F22BBCFC872325BB006322692FDC60AE33890E1F1',

tx\_json:

{ Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '100000',

Flags: 0,

Sequence: 26,

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '3045022100FB82276780F6A049E4B45FCBD4EA815AEB21E7044286D7254611A5CFDE8BF1A3022023DD9C096DD5003F9A994F62BCAE54582B2E6AEA246D053CAE620868A4FE6E11' } }],

SigningPubKey: '',

TransactionType: 'Payment',

hash: '0899437ADAB04B52FA00771BDE80216F19F58BF215CD79CD00A45112E5363080' }

}

返回结果说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | | | | 类型 | 说明 |
| engine\_result | | | | String | 请求结果，tefBAD\_QUORUM表示没达到阈值 |
| engine\_result\_code | | | | Array | 请求结果编码 |
| engine\_result\_message | | | | String | 请求结果message信息 |
| tx\_blob | | | | String | 16进制签名后的交易 |
| tx\_json | | | | Object | 交易内容 |
|  | Account | | | String | 交易源账号地址 |
| Fee | | | String | 交易费 |
| Flags | | | Integer | 交易标记 |
| Sequence | | | Integer | 单子序列号 |
| -- | | | -- | 相关交易对应的其他字段，如Payment类型有Amount、Destination字段，这里不一一列举 |
| Signers | | | Array | 签名列表条目 |
|  | Signer | | Object | 单个签名条目 |
|  | Account | String | 给该交易签名的账号地址 |
| SigningPubKey | String | 给该交易签名的账号公钥 |
| TxnSignature | String | Account账号给该交易的交易签名 |
| SigningPubKey | | | String | 交易签名公钥，必须为空字符串 |
| TransactionType | | | String | 交易类型 |
| hash | | | String | 交易hash，成功可在链上查到 |

### 多个用户签名，交易信息由其他用户返给

var a1 = {address: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb', secret: 'ss...GT'};

var tx\_json = { //其他用户签名返回结果中的tx\_json

Flags: 0,

Fee: 100000,

TransactionType: 'Payment',

Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Sequence: 11,

SigningPubKey: '',

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '3045022100DDC408965E9F62626BC6A0603A0D707B64E0157B84299414BCF8C560EDE776970220093CB723375F3725DC41FB5B207B2CEEE158D5D70A90F18BD10D330F637DB9A3' } } ] };

var tx = remote.buildTx(tx\_json); //包装成Transaction对象

tx.multiSigning({ //用户签名

account: a2.address,

secret: a2.secret

});

var flag = tx.tx\_json.verifyTx && tx.tx\_json.verifyTx.toString().indexOf('verify failed');//判断签名是否成功

flag ? console.log('verify failed') : console.log('verify success');

if(!flag){

tx.multiSigned();//多重签名结束

tx.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

}

返回结果：

{

engine\_result: 'tesSUCCESS',

engine\_result\_code: 0,

engine\_result\_message: 'The transaction was applied. Only final in a validated ledger.',

tx\_blob

tx\_json:

{ Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Fee: '100000',

Flags: 0,

Sequence: 26,

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '3045022100FB82276780F6A049E4B45FCBD4EA815AEB21E7044286D7254611A5CFDE8BF1A3022023DD9C096DD5003F9A994F62BCAE54582B2E6AEA246D053CAE620868A4FE6E11' } },

{ Signer:

{ Account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

SigningPubKey: '03F54EA6509AD28E8E9AE8762B07CD245B5AB87DC840723DBF6C4C0A621A42B4A1',

TxnSignature: '30440220393169B8DA374F03848B6AF1AAC3B9A8F195BD818DAD7BC93E762AD32DA1BD6202203128A6C362C3C423A3B45051CA1329761676BADA5860ED82F6A714B3AFD1DD58' } }],

SigningPubKey: '',

TransactionType: 'Payment',

hash: '413EF2496165852D1863A5AAACBA04DB1CFBD3EFCF506F45B5EB38F3D8742AF7' }

}

返回结果说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | | | | 类型 | 说明 |
| engine\_result | | | | String | 请求结果 |
| engine\_result\_code | | | | Array | 请求结果编码 |
| engine\_result\_message | | | | String | 请求结果message信息 |
| tx\_blob | | | | String | 16进制签名后的交易 |
| tx\_json | | | | Object | 交易内容 |
|  | Account | | | String | 交易源账号地址 |
| Fee | | | String | 交易费 |
| Flags | | | Integer | 交易标记 |
| Sequence | | | Integer | 单子序列号 |
| -- | | | -- | 相关交易对应的其他字段，如Payment类型有Amount、Destination字段，这里不一一列举 |
| Signers | | | Array | 签名列表条目 |
|  | Signer | | Object | 单个签名条目 |
|  | Account | String | 给该交易签名的账号地址 |
| SigningPubKey | String | 给该交易签名的账号公钥 |
| TxnSignature | String | Account账号给该交易的交易签名 |
| SigningPubKey | | | String | 交易签名公钥，必须为空字符串 |
| TransactionType | | | String | 交易类型 |
| hash | | | String | 交易hash，成功可在链上查到 |

## 监听事件

Remote有两个监听事件：监听所有交易(transactions)和监听所有账本(ledger\_closed)，监听结果放到回调函数中，回调中只有一个参数，为监听到的消息。

方法：remote.on('transactions',callback);

方法：remote.on('ledger\_closed',callback);

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

remote.on('transactions', function (msg) {

console.log('tx: ',msg);

});

remote.on('ledger\_closed', function (msg) {

console.log('ledger: ',msg);

});

});

# Request类

Request类主管GET请求，包括获得服务器、账号、挂单、路径等信息。请求时不需要提供密钥，且对所有用户公开。所有的请求是异步的，会提供一个回调函数。每个回调函数有两个参数，一个是错误，另一个是结果。提供以下方法：

\* selectLedger(ledger)

\* submit(callback)

## 指定账本

方法：selectLedger(ledger);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| ledger | String | 账本高度或者账号hash |

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var req = remote.requestAccountInfo({account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ'});

req.selectLedger("8573498");

req.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

## 提交请求

方法：submit(callback);

参数：回调函数，包含两个参数：错误信息和结果信息

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var req = remote.requestAccountInfo({account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ'});

req.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

# Transaction类

Transaction类主管POST请求，包括组装交易和交易参数。请求时需要提供密钥，且交易可以进行本地签名和服务器签名。目前支持服务器签名，本地签名支持主要的交易，还有部分参数不支持。所有的请求是异步的，会提供一个回调函数。每个回调函数有两个参数，一个是错误，另一个是结果。提供以下方法：

\* getAccount()

\* getTransactionType()

\* setSecret(secret)

\* addMemo(memo)

\* setPath(key)

\* setSendMax(amount)

\* setTransferRate(rate)

\* setFlags(flags)

\* submit(callback)

## 获得交易账号

方法：getAccount();

参数：无

返回：账号

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port'});

var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequence: 688};

var tx = remote.buildOfferCancelTx(options);

var account = tx.getAccount();

console.log(account);

## 获得交易类型

方法：getTransactionType();

参数：无

返回：交易类型

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port'});

var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequence: 688};

var tx = remote.buildOfferCancelTx(options);

var type = tx.getTransactionType();

console.log(type);

## 传入私钥

交易提交之前需要传入私钥。

方法：setSecret(secret);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 井通钱包私钥 |

返回：Transaction对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port'});

var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequence: 688};

var tx = remote.buildOfferCancelTx(options);

tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');;

## 添加备注

方法：addMemo(memo);

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| memo | String | 备注信息，不超过2k。 |

返回：Transaction对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port'});

var tx = remote.buildPaymentTx({

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

to: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',

amount: {

"value": 0.5,

"currency": "SWT",

"issuer": ""

}

});

tx.addMemo('给jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c支付0.5swt.');

## 设置sequence

方法：setSequence(sequence);主要用于签名相关。

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| sequence | Integer | 账号当前序列号 |

返回：Transaction对象

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port'});

var tx = remote.buildPaymentTx({

account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',

to: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',

amount: {

"value": 0.5,

"currency": "SWT",

"issuer": ""

}

});

tx.setSequence(10);

## 交易多签过程

方法：multiSigning({})，主要用于多重签名相关；

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| account | String | 给交易签名的账号地址 |
| secret | String | 给交易签名的账号私钥 |

返回：Transaction对象

多重签名过程中会有验签结果，可通过查看Transaction对象中tx\_json.verifyTx字段是否返回错误，若返回错误则失败，需要上一用户重新签名，否则成功。如下判断：

if(tx.tx\_json.verifyTx && tx.tx\_json.verifyTx.toString().indexOf('verify failed')){//验签结果

console.log('verify failed');

}else {

console.log('verify success');

}

例子：

var tx\_json = { Flags: 0,

Fee: 100000,

TransactionType: 'Payment',

Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Sequence: 11,

SigningPubKey: '',

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '3045022100DDC408965E9F62626BC6A0603A0D707B64E0157B84299414BCF8C560EDE776970220093CB723375F3725DC41FB5B207B2CEEE158D5D70A90F18BD10D330F637DB9A3' } } ] }

var tx = remote.buildTx(tx\_json);

tx.multiSigning({

account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

secret: 'ssL...bSy'

});

if(tx.tx\_json.verifyTx && tx.tx\_json.verifyTx.toString().indexOf('verify failed')){//验签结果

console.log('verify failed');

}else {

console.log('verify success');

}

## 交易多签结束

方法：multiSigned()，主要用于多重签名相关；

参数：无

返回：Transaction对象

本接口会有验签结果，可通过查看Transaction对象中tx\_json.verifyTx字段是否返回错误，若返回错误则失败，需要上一用户重新签名；否则成功。如下判断：

if(tx.tx\_json.verifyTx && tx.tx\_json.verifyTx.toString().indexOf('verify failed')){//验签结果

console.log('verify failed');

}else {

console.log('verify success');

}

例子：

var tx\_json = { Flags: 0,

Fee: 100000,

TransactionType: 'Payment',

Account: 'jHb9CJAWyB4jr91VRWn96DkukG4bwdtyTh',

Amount: '2000000',

Destination: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

Sequence: 11,

SigningPubKey: '',

Signers: [ { Signer:

{ Account: 'j3yeaNQUqMmrDb1T1p6Q2qHm9BHaAAmSwb',

SigningPubKey: '03B61B9644843F1781F66D72C99840AD7BE4FAC713EF5AC73F30278B2287B7BBFF',

TxnSignature: '3045022100DDC408965E9F62626BC6A0603A0D707B64E0157B84299414BCF8C560EDE776970220093CB723375F3725DC41FB5B207B2CEEE158D5D70A90F18BD10D330F637DB9A3' } } ] }

var tx = remote.buildTx(tx\_json);

tx.multiSigning({

account: 'jJwkfLEVTkM6u3J7kWoATFd5aauBw5S8Kz',

secret: 'ssL...bSy'

});

tx.multiSigned();//多重签名结束

if(tx.tx\_json.verifyTx && tx.tx\_json.verifyTx.toString().indexOf('verify failed')){//验签结果

console.log('verify failed');

}else {

console.log('verify success');

}

## 提交请求

方法：submit(callback);

参数：回调函数，包含两个参数：错误信息和结果信息

例子：

var jlib = require('jingtum-lib');

var Remote = jlib.Remote;

var remote = new Remote({server: 'ws://xxx:port', local\_sign:true});

remote.connect(function (err, result) {

if (err) {

return console.log('err:', err);

}

var req = remote.requestAccountInfo({account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ'});

req.submit(function(err, result) {

if(err) {console.log('err:',err);}

else if(result){

console.log('res:', result);

}

});

});

# 工具类

Utils类是工具类，提供以下方法：

\* hexToString(hex)

\* stringToHex(str)

\* isValidAmount()

\* isValidAmount0()

\* parseAmount()

\* isValidCurrency()

\* isValidHash()

\* isValidAddress()

\* isValidSecret()

\* processTx()

## 16进制转字符串

方法：hexToString(hex)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| hex | String | 16进制字符串 |

返回：字符串

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.hexToString('6e6968616f'));

返回结果：

nihao

## 字符串转16进制

方法：stringToHex(str)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| str | String | 字符串 |

返回：16进制字符串

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.stringToHex(nihao));

返回结果：

6e6968616f

## 判断amount是否有效

方法：isValidAmount({})

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| value | String | 数值 |
| currency | String | 货币种类 |
| issuer | String | 货币发行方 |

返回：Boolean值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.isValidAmount({value: '20', currency:'SWT', issuer:''}));

返回结果：

true

## 判断市场挂单的货币对是否有效

方法：isValidAmount0({})

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| currency | String | 货币种类 |
| issuer | String | 货币发行方 |

返回：Boolean值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.isValidAmount0({currency:'SWT', issuer:''}));

返回结果：

true

## 统一基础货币SWT的amount格式

方法：parseAmount()

参数：数值/对象；

返回：若为数值，则默认转换成基础货币的amount格式（即{value: ‘数值’,currency:’SWT’,issuer:’’}）返回，**这里value值需乘以1000000**；若为合法amount对象，返回此对象；否则，返回null。

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.parseAmount('2000000'));

返回结果：

{ value: '2', currency: 'SWT', issuer: '' }

## 判断currency是否合法

方法：isValidCurrency(currency)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| currency | String | 货币种类 |

注：currency为3-6位字母(不区分大小写，统一转换成大写)或者由A-F0-9组成的40位16进制字符串。

返回：Boolean值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.isValidCurrency('0CFDA38CC98A630F97B9FFFFB324FC40F2882F97'));

返回结果：

true

## 判断hash是否合法

方法：isValidHash(hash)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| hash | String | 交易hash/账本hash |

返回：Boolean值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.isValidHash('A89A245D311FCCF3C329E865AFF69C59AB9CD66F0808BD0C16D5706103319125'));

返回结果：

true

## 判断address是否合法

方法：isValidAddress(address)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| address | String | 钱包地址 |

注：井通地址以j开头。

返回：Boolean值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.isValidAddress('jsEtk7VtqY8PJ4uSmTbQ1KHkv1u9zhr8pQ'));

返回结果：

true

## 判断secret是否合法

方法：isValidSecret(secret)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| secret | String | 钱包私钥 |

注：井通私钥以s开头。

返回：Boolean值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.isValidSecret('sn5qWH2E18HbnuvfrHeMcnFdENcSu'));

返回结果：

true

## 以太坊地址转成jingtum地址

方法：eth2Jingtum(ethadr)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| ethadr | String | 以太坊地址 |

返回：String值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

console.log(utils.eth2Jingtum('0xa5ccbc99c692a59ffd8b237bf601287142de9b94'));

返回结果：

jGfCmjWeNNqE5d2cVGHBHiNs5NPrbtpRbn

## processTx从账号角度解析交易信息

方法：processTx(tx, account)

参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| tx | Object | 底层返回的交易详情 |
| account | String | 钱包地址 |

注：井通私钥以s开头。

返回：Boolean值

例子：

var jlib = require('jingtum-lib');

var utils= jlib.utils;

var tx = { Account: 'jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR',

Fee: '10000',

Flags: 524288,

Sequence: 650,

SigningPubKey: '03E791056E6B4C62E26C0F1F3BB89317667AB74170B49339972716FC53FFCF007C',

TakerGets: '2000000000',

TakerPays:

{ currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or',

value: '13.46' },

Timestamp: 611552863,

TransactionType: 'OfferCreate',

TxnSignature: '3045022100B342C7159E1AD7FAA13452C1FB01F77A107263AFB06E93F9B4F307EF9DF9F98E0220690C964D89146250E879B9EA5ED311C3317631F1F6450360603E6818AFFB5FAF',

date: 611552870,

hash: 'AB2A25557FF03911A8FC0A412293BE9D9FCB20CDD530EE05957A9859F8467C32',

inLedger: 12839308,

ledger\_index: 12839308,

meta:

{ AffectedNodes: [ { ModifiedNode:

{ FinalFields: { Account: 'j9x4pABowsWxmK1DhhWyK34u3boC6h3LHe',

BookDirectory: '51603377F758E3C8FA007C77312DDA06A737A1395CD5FC435D0547675A0517F6',

BookNode: '0000000000000000',

Flags: 0,

OwnerNode: '0000000000000000',

Sequence: 7031,

TakerGets:

{ currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or',

value: '1148.95233' },

TakerPays: '170721000000' },

LedgerEntryType: 'Offer',

LedgerIndex: '020110B8BED1F151B9D3AF9E5D412D8627CB08232B388ADE1F4B0C68E7608BEC',

PreviousFields: { TakerGets:

{ currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or',

value: '1162.41233' },

TakerPays: '172721000000' },

PreviousTxnID: '9CB6AEFA273C750242D5B8AF4299347E77F9A47C5D0B89EE5F6A4D5577E8C4A0',

PreviousTxnLgrSeq: 12839301 } },

{ ModifiedNode:

{ FinalFields: { Account: 'jEoSyfChhUMzpRDttAJXuie8XhqyoPBYvV',

Balance: '533983297806',

Flags: 0,

OwnerCount: 1,

Sequence: 34380818 },

LedgerEntryType: 'AccountRoot',

LedgerIndex: '109E80FB8CC6D82D4F7F7D77248C2C3C116ECCD4520B3D2A88421FFF94A57B1E',

PreviousFields: { Balance: '533983287806', Sequence: 34380817 },

PreviousTxnID: '756338B8F9D4DCC8D88382B1092B13F75F65F330970278AFC7449496FF9875E9',

PreviousTxnLgrSeq: 12839308 } },

{ ModifiedNode:

{ FinalFields: { Balance:

{ currency: 'CNY',

issuer: 'jjjjjjjjjjjjjjjjjjjjBZbvri',

value: '-6872.222452374449' },

Flags: 2228224,

HighLimit:

{ currency: 'CNY',

issuer: 'jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR',

value: '10000000000' },

HighNode: '0000000000000000',

LowLimit:

{ currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or',

value: '0' },

LowNode: '00000000000012A0' },

LedgerEntryType: 'SkywellState',

LedgerIndex: '2600F8FCB87FEA15F74B0DB785016384C79AEA0730B62F597C1E576801BB813B',

PreviousFields: { Balance:

{ currency: 'CNY',

issuer: 'jjjjjjjjjjjjjjjjjjjjBZbvri',

value: '-6858.762452374449' } },

PreviousTxnID: '9B28F7958E729F0F904410B132D1F81481B38DD9F017790A82168CD38C995331',

PreviousTxnLgrSeq: 12838251 } },

{ ModifiedNode:

{ FinalFields:{ Account: 'j9x4pABowsWxmK1DhhWyK34u3boC6h3LHe',

Balance: '1496144192938',

Flags: 0,

OwnerCount: 8,

Sequence: 7032 },

LedgerEntryType: 'AccountRoot',

LedgerIndex: '40A20BDD3C226C987579F6C821BF84492E1C6B6EFB62311481BA6B8CB1D7775A',

PreviousFields: { Balance: '1494144192938' },

PreviousTxnID: '9CB6AEFA273C750242D5B8AF4299347E77F9A47C5D0B89EE5F6A4D5577E8C4A0',

PreviousTxnLgrSeq: 12839301 } },

{ ModifiedNode:

{ FinalFields: { Account: 'jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR',

Balance: '500538133',

Flags: 0,

OwnerCount: 10,

Sequence: 651 },

LedgerEntryType: 'AccountRoot',

LedgerIndex: 'B39BD926378886F7EF4F81CEF862FC4D1E8E6D1265945AA9EC40FD85132DC629',

PreviousFields: { Balance: '2500548133', Sequence: 650 },

PreviousTxnID: '5BA24DE17EF64EDF942D99F247ED1495F5A61ED9260513FEDCA3E4BADBADFF3E',

PreviousTxnLgrSeq: 12839303 } },

{ ModifiedNode:

{ FinalFields: { Balance:

{ currency: 'CNY',

issuer: 'jjjjjjjjjjjjjjjjjjjjBZbvri',

value: '1148.954817858577' },

Flags: 1114112,

HighLimit:

{ currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or',

value: '0' },

HighNode: '000000000000172A',

LowLimit:

{ currency: 'CNY',

issuer: 'j9x4pABowsWxmK1DhhWyK34u3boC6h3LHe',

value: '10000000000' },

LowNode: '0000000000000000' },

LedgerEntryType: 'SkywellState',

LedgerIndex: 'E3E9FE1827E83B52F7017D3038F8C769F09343801BB073A993DE620756069137',

PreviousFields: { Balance:

{ currency: 'CNY',

issuer: 'jjjjjjjjjjjjjjjjjjjjBZbvri',

value: '1162.414817858577' } },

PreviousTxnID: '9CB6AEFA273C750242D5B8AF4299347E77F9A47C5D0B89EE5F6A4D5577E8C4A0',

PreviousTxnLgrSeq: 12839301 } } ],

TransactionIndex: 3,

TransactionResult: 'tesSUCCESS' },

validated: true };

var newTx = utils.processTx(tx, tx.Account);

console.log(newTx);

返回结果：

{ date: 1558237670,

hash: 'AB2A25557FF03911A8FC0A412293BE9D9FCB20CDD530EE05957A9859F8467C32',

type: 'offernew',

fee: '0.01',

result: 'tesSUCCESS',

memos: [],

ledger\_index: 12839308,

offertype: 'sell',

gets: { value: '2000', currency: 'SWT', issuer: '' },

pays:

{ currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or',

value: '13.46' },

seq: 650,

price: 0.00673,

effects:

[ { effect: 'offer\_bought',

counterparty:

{ account: 'j9x4pABowsWxmK1DhhWyK34u3boC6h3LHe',

seq: 7031,

hash: '9CB6AEFA273C750242D5B8AF4299347E77F9A47C5D0B89EE5F6A4D5577E8C4A0' },

paid: { value: '2000', currency: 'SWT', issuer: '' },

got:

{ value: '13.46',

currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or' },

type: 'sold',

price: '0.00673' } ],

balances: { CNY: 6872.222452374449, SWT: 500.538133 },

balancesPrev: { CNY: 6858.762452374449, SWT: 2500.548133 },

dealGets: { value: '2000', currency: 'SWT', issuer: '' },

dealPays:

{ value: '13.46',

currency: 'CNY',

issuer: 'jGa9J9TkqtBcUoHe2zqhVFFbgUVED6o9or' },

dealPrice: '0.00673',

dealNum: 2 }

返回结果说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| date | | Integer | 交易时间 |
| hash | | String | 交易hash |
| type | | String | 交易类型，详见后面说明 |
| fee | | String | 燃料费 |
| result | | String | 交易结果码，tesSUCCESS表示成功，其他详见底层常见错误附录 |
| memos | | Array | 交易备注 |
| ledger\_index | | Integer | 交易所在区块高度 |
| offertype | | String | 挂单买卖类型，只有挂单类才有此字段 |
| gets | | Object | 委托的对方将要获得的数量，只有挂单类才有此字段 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 发行方 |
| pays | | Object | 委托的对方将要支付的数量，只有挂单类才有此字段 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 发行方 |
| seq | | Integer | 该笔交易对应的该账号的sequence编号 |
| effects | | Array | 交易效果，包括创建挂单、取消挂单、挂单成交详情等，详见后面effects说明 |
| balances | | Object | 变动币种余额 |
| balancesPrev | | Object | 交易前变动币种余额 |
| dealGets | | Object | 实际对方获得的，挂单成交才有此字段 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 发行方 |
| dealPays | | Object | 实际对方支付的，挂单成交才有此字段 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 发行方 |
| totalRate | | Object | 一共收取的挂单手续费，挂单成交才有此字段 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 发行方 |
| dealPrice | | String | 实际成交价，挂单成交才有此字段 |
| dealNum | | Integer | 涉及到的撮合币种，挂单成交才有此字段 |
| signers | | Array | 多重签名类才有，若该笔交易是多重签名方式完成，则此字段显示所有签名账户，否则该字段不显示 |
| threshold | | Integer | 签名列表类才有，阈值 |
| lists | | Array | 签名列表类才有，签名列表 |
| src | | String | 黑名单类才有，设置黑名单的源账号 |
| black | | String | 黑名单类才有，黑名单账号 |
| method | | String | 合约类才有，合约类型（部署合约deploy，调用合约call） |
| payload | | String | 合约类里的部署合约才有，合约编译后的16进制字节码 |
| amount | | Number | 合约类里的调用合约才有，合约中msg.value的值。只有payable修饰的合约函数，才可以设置该值，默认为0。 |
| destination | | String | 合约类的调用合约才有，合约账号 |
| func | | String | 合约类的调用合约才有，合约函数名 |
| func\_parms | | String | 合约类的调用合约才有，合约参数 |
| eventLog | | String | 合约类的调用合约才有，合约调用的eventLog日志 |

## 交易类型type说明

底层返回的交易类型主要有Payment(支付类)、OfferCreate(创建挂单类)、OfferCancel(取消挂单类)、ConfigContract(lua版合约类)、AlethContract(solidity版合约类)、SignerListSet(签名列表类)和Brokerage(手续费设置类)；

processTx方法解析交易信息，主要有如下信息：

\* date： 交易时间UNIXTIME

\* hash：交易hash

\* fee： 燃料费

\* result ：交易结果

\* memos： 交易的备注信息

\* balances： 变动币种余额

\* balancesPrev： 交易前变动币种余额

\* type： 交易类型

type有如下几种：

1. sent，支付类，发送，在交易信息中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| counterparty | | String | 支付对家，即接收方 |
| amount | | Object | 交易数量 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 货币发行方，SWT为空 |
| effects | | Array | []，空 |

1. received，支付类，接收，在交易信息中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| counterparty | | String | 支付对家，即发送方 |
| amount | | Object | 交易数量 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 货币发行方，SWT为空 |
| effects | | Array | []，空 |

1. offernew，创建挂单，在交易信息中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| offertype | | String | 挂单买卖类型，sell或buy |
| gets | | Object | 对方得到的 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 货币发行方，SWT为空 |
| pays | | Object | 对方支付的 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 货币发行方，SWT为空 |
| seq | | Integer | 单子序号 |
| price | | Integer | 价格 |
| effects | | Array | 详见下面的effects解释 |

1. offercancel，取消挂单，在交易信息中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| offerseq | | String | 该笔交易的单子号 |
| gets | | Object | 对方得到的 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 货币发行方，SWT为空 |
| pays | | Object | 对方支付的 |
|  | value | String | 金额 |
| currency | String | 币种 |
| issuer | String | 货币发行方，SWT为空 |
| effects | | Array | 详见下面的effects解释 |

1. brokerage：手续费设置类，在交易信息中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| feeAccount | | String | 收费账号 |
| mol | | Integer | 手续费分子 |
| den | | Integer | 手续费分母 |
| amount | | Object | 收手续费的币种 |
|  | value | String | 金额，占位，没有任何实际含义 |
| currency | String | 币种 |
| issuer | String | 货币发行方，SWT为空 |
| seq | | Integer | 单子序号 |
| effects | | Array | []，空 |

1. alethcontract：solidity版合约类，在交易信息中包含的信息有：

待完善。。。

1. configcontract：lua版合约类，在交易信息中包含的信息有：

待完善。。。

1. offereffect，挂单成交情况，即被动成交的情况，在交易信息中包含的信息有：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| effects | Array | []，空 |

1. signerlistset，签名列表类，在交易信息中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | | 类型 | 说明 |
| threshold | | Integer | 阈值 |
| lists | | Array | 签名列表 |
|  | account | String | 账户 |
| weight | String | 该账户对应权重 |
| effects | | Array | []，空 |

1. setblacklist，设置黑名单类，在交易信息中包含的信息有：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| src | String | 设置黑名单的源账号 |
| black | String | 黑名单账号 |
| effects | Array | []，空 |

1. setblacklist，设置黑名单类，在交易信息中包含的信息有：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| src | String | 设置黑名单的源账号 |
| black | String | 黑名单账号 |
| effects | Array | []，空 |

1. removeblacklist，移除黑名单类，在交易信息中包含的信息有：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 说明 |
| src | String | 设置黑名单的源账号 |
| black | String | 黑名单账号 |
| effects | Array | []，空 |

## 交易效果effects说明

effects是每个用户交易记录信息里面的交易效果，是个JSON数组，数组可以包含多项，每项内容都包含效果类型effect字段，根据effect的不同里面的内容也不同：

1. offer\_funded，被动成交；参考正式环境hash：D9A717FC8388053764908DE867E4FFD72D8D3CD9AA92A53A5FF402B3313D06B0，账号：jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR，其中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | | 类型 | 说明 |
| effect | | String | offer\_funded |
| counterparty | | Object | 对家信息 |
|  | account | String | 对家账号 |
| seq | Integer | 对家单子序号 |
| hash | String | 对家交易hash |
| got | | Object | 用户获得的金额 |
|  | value | String | 金额 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| paid | | Object | 用户付出的金额 |
|  | value | String | 金额 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| type | | String | 交易类型，sold或bought |
| platform | | String | 平台标识账号，设置了才有此字段 |
| rate | | Integer | 手续费汇率，设置了才有此字段 |
| fee | | String | 手续费金额，收取手续费时才有此字段 |
| seq | | Integer | 挂单序号，表示被成交的单子 |
| price | | String | 价格，4位小数 |
| deleted | | Boolean | 单子是否被删除了，被吃了的单子会被删除掉 |

1. offer\_partially\_funded，交易部分成交；参考正式环境交易hash：80A4C0EC32AE6E16A6025A1420DD6AB67DEE2AFED1A618A71A8AD1E37A54FB2D，账号：jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR；其中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | | 类型 | 说明 |
| effect | | String | offer\_partially\_funded |
| counterparty | | Object | 对家信息 |
|  | account | String | 对家账号 |
| seq | Integer | 对家单子序号 |
| hash | String | 对家交易hash |
| remaining | | Boolean | 是否有剩余的单子 |
| gets | | Object | 对方获得的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| pays | | Object | 对方支付的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| paid | | Object | 用户支付的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| got | | Object | 用户获得的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| type | | String | 交易类型，sold或bought |
| platform | | String | 平台标识账号，设置了才有此字段 |
| rate | | Integer | 手续费汇率，设置了才有此字段 |
| fee | | String | 手续费金额，收取手续费时才有此字段 |
| seq | | Integer | 挂单序号，表示被部分成交的单子 |
| price | | String | 挂单的价格，remaining为true才有 |

1. offer\_cancelled，交易单子被取消；参考正式环境交易hash：286C40DB44DF4340B8974E635A49C46475B4D3F3CE0A67E8CB8EC93B399A94B2，账号：jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR；其中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | | 类型 | 说明 |
| effect | | String | offer\_cancelled |
| hash | | String | 被取消单子的hash |
| gets | | Object | 对方获得的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| pays | | Object | 对方支付的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| type | | String | 交易类型，sell或buy |
| platform | | String | 平台标识账号，设置了才有此字段 |
| rate | | Integer | 手续费汇率，设置了才有此字段 |
| seq | | Integer | 被取消单子的序号 |
| price | | String | 挂单的价格 |
| deleted | | Boolean | 单子是否被删除，取消单子为true |

1. offer\_created，交易单子创建；参考正式环境交易hash:55DAB736BB47E940159ADA262B050A110A4EC9C5E917A08649E54DA587E76D9A，账号：jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR，其中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | | 类型 | 说明 |
| effect | | String | offer\_created |
| gets | | Object | 对方获得的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| pays | | Object | 对方支付的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| type | | String | 交易类型，sell或buy |
| platform | | String | 平台标识账号，设置了才有此字段 |
| rate | | Integer | 手续费汇率，设置了才有此字段 |
| seq | | Integer | 新建的单子序号 |
| price | | String | 挂单的价格 |

1. offer\_bought，主动成交；参考正式环境交易hash：F137C081D7B486DA7ACA8F790DE4A231E6B23281E4CA4A5A40CEAF1E43F1390A，账号：jKCQAZwwN2sQG3Mb56GmWVqxkgpLwwAZuR；其中包含的信息有：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | | 类型 | 说明 |
| effect | | String | offer\_bought |
| counterparty | | Object | 对家信息 |
|  | account | String | 对家账号 |
| seq | Integer | 对家单子序号 |
| hash | String | 对家交易hash |
| paid | | Object | 用户支付的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| got | | Object | 用户获得的金额 |
|  | value | String | 数值 |
| currency | String | 货币 |
| issuer | String | 货币发行方，SWT为空字符串 |
| type | | String | 交易类型，sold或bough |
| price | | String | 价格 |
| platform | | String | 平台标识账号，设置了才有此字段 |
| rate | | Integer | 手续费汇率，设置了才有此字段 |
| fee | | String | 手续费金额，收取手续费时才有此字段 |

1. setregularkey，设置代理账号；其中包含的信息有：

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| effect | String | set\_regular\_key |
| type | String | 交易类型，setregularkey |
| account | String | 这是代理的源账号 |
| regularkey | String | 代理账号 |

# 底层常见错误附录

|  |  |
| --- | --- |
| 错误名称 | 说明 |
| tecCLAIM | Fee claimed. Sequence used. No action. |
| tecDIR\_FULL | Can not add entry to full directory. |
| tecFAILED\_PROCESSING | Failed to correctly process transaction. |
| tecINSUF\_RESERVE\_LINE | Insufficient reserve to add trust line. |
| tecINSUF\_RESERVE\_OFFER | Insufficient reserve to create offer. |
| tecNO\_DST | Destination does not exist. Send SWT to create it. |
| tecNO\_DST\_INSUF\_SWT | Destination does not exist. Too little SWT sent to create it. |
| tecNO\_LINE\_INSUF\_RESERVE | No such line. Too little reserve to create it. |
| tecNO\_LINE\_REDUNDANT | Can't set non-existent line to default. |
| tecPATH\_DRY | Path could not send partial amount. |
| tecPATH\_PARTIAL | Path could not send full amount. |
| tecMASTER\_DISABLED | Master key is disabled. |
| tecNO\_REGULAR\_KEY | Regular key is not set. |
| tecUNFUNDED | One of \_ADD, \_OFFER, or \_SEND. Deprecated. |
| tecUNFUNDED\_ADD | Insufficient SWT balance for WalletAdd. |
| tecUNFUNDED\_OFFER | Insufficient balance to fund created offer. |
| tecUNFUNDED\_PAYMENT | Insufficient SWT balance to send. |
| tecOWNERS | Non-zero owner count. |
| tecNO\_ISSUER | Issuer account does not exist. |
| tecNO\_AUTH | Not authorized to hold asset. |
| tecNO\_LINE | No such line. |
| tecINSUFF\_FEE | Insufficient balance to pay fee. |
| tecFROZEN | Asset is frozen. |
| tecNO\_TARGET | Target account does not exist. |
| tecNO\_PERMISSION | No permission to perform requested operation. |
| tecNO\_ENTRY | No matching entry found. |
| tecINSUFFICIENT\_RESERVE | Insufficient reserve to complete requested operation. |
| tecNEED\_MASTER\_KEY | The operation requires the use of the Master Key. |
| tecDST\_TAG\_NEEDED | A destination tag is required. |
| tecINTERNAL | An internal error has occurred during processing. |
| tefALREADY | The exact transaction was already in this ledger. |
| tefBAD\_ADD\_AUTH | Not authorized to add account. |
| tefBAD\_AUTH | Transaction's public key is not authorized. |
| tefBAD\_LEDGER | Ledger in unexpected state. |
| tefCREATED | Can't add an already created account. |
| tefEXCEPTION | Unexpected program state. |
| tefFAILURE | Failed to apply. |
| tefINTERNAL | Internal error. |
| tefMASTER\_DISABLED | Master key is disabled. |
| tefMAX\_LEDGER | Ledger sequence too high. |
| tefNO\_AUTH\_REQUIRED | Auth is not required. |
| tefPAST\_SEQ | This sequence number has already past. |
| tefWRONG\_PRIOR | This previous transaction does not match. |
| telLOCAL\_ERROR | Local failure. |
| telBAD\_DOMAIN | Domain too long. |
| telBAD\_PATH\_COUNT | Malformed: Too many paths. |
| telBAD\_PUBLIC\_KEY | Public key too long. |
| telFAILED\_PROCESSING | Failed to correctly process transaction. |
| telINSUF\_FEE\_P | Fee insufficient. |
| telNO\_DST\_PARTIAL | Partial payment to create account not allowed. |
| telBLKLIST | Tx disable for blacklist. |
| telINSUF\_FUND | Fund insufficient. |
| temMALFORMED | Malformed transaction. |
| temBAD\_AMOUNT | Can only send positive amounts. |
| temBAD\_AUTH\_MASTER | Auth for unclaimed account needs correct master key. |
| temBAD\_CURRENCY | Malformed: Bad currency. |
| temBAD\_EXPIRATION | Malformed: Bad expiration. |
| temBAD\_FEE | Invalid fee, negative or not SWT. |
| temBAD\_ISSUER | Malformed: Bad issuer. |
| temBAD\_LIMIT | Limits must be non-negative. |
| temBAD\_QUORUM | Quorums must be non-negative. |
| temBAD\_WEIGHT | Weights must be non-negative. |
| temBAD\_OFFER | Malformed: Bad offer. |
| temBAD\_PATH | Malformed: Bad path. |
| temBAD\_PATH\_LOOP | Malformed: Loop in path. |
| temBAD\_SEND\_SWT\_LIMIT | Malformed: Limit quality is not allowed for SWT to SWT. |
| temBAD\_SEND\_SWT\_MAX | Malformed: Send max is not allowed for SWT to SWT. |
| temBAD\_SEND\_SWT\_NO\_DIRECT | Malformed: No Skywell direct is not allowed for SWT to SWT. |
| temBAD\_SEND\_SWT\_PARTIAL | Malformed: Partial payment is not allowed for SWT to SWT. |
| temBAD\_SEND\_SWT\_PATHS | Malformed: Paths are not allowed for SWT to SWT. |
| temBAD\_SEQUENCE | Malformed: Sequence is not in the past. |
| temBAD\_SIGNATURE | Malformed: Bad signature. |
| temBAD\_SRC\_ACCOUNT | Malformed: Bad source account. |
| temBAD\_TRANSFER\_RATE | Malformed: Transfer rate must be >= 1.0 |
| temDST\_IS\_SRC | Destination may not be source. |
| temDST\_NEEDED | Destination not specified. |
| temINVALID | The transaction is ill-formed. |
| temINVALID\_FLAG | The transaction has an invalid flag. |
| temREDUNDANT | Sends same currency to self. |
| temREDUNDANTSIGN | Add self as additional sign. |
| temSKYWELL\_EMPTY | PathSet with no paths. |
| temUNCERTAIN | In process of determining result. Never returned. |
| temUNKNOWN | The transaction requires logic that is not implemented yet. |
| temDISABLED | The transaction requires logic that is currently disabled. |
| temMULTIINIT | contract code has multi init function |
| terRETRY | Retry transaction. |
| terFUNDS\_SPENT | Can't set password, password set funds already spent. |
| terINSUF\_FEE\_B | Account balance can't pay fee. |
| terLAST | Process last. |
| terNO\_SKYWELL | Path does not permit rippling. |
| terNO\_ACCOUNT | The source account does not exist. |
| terNO\_AUTH | Not authorized to hold IOUs. |
| terNO\_LINE | No such line. |
| terPRE\_SEQ | Missing/inapplicable prior transaction. |
| terOWNERS | Non-zero owner count. |
| tesSUCCESS | The transaction was applied. Only final in a validated ledger. |
| rpcLOW\_FEE | fee field is lower than transaction needed.(手续费过低) |
| tefBAD\_QUORUM | Signatures provided do not meet the quorum.(权重未达到阈值) |
| tefBAD\_SIGNATURE | A signature is provided for a non-signer.(签名错误) |
| tefNOT\_MULTI\_SIGNING | Account has no appropriate list of multi-signers.(签名列表不存在) |
| tecNO\_ALTERNATIVE\_KEY | The operation would remove the ability to sign transactions with the account. (原因：没有设置签名列表) |

# Solidity ERC20源码

pragma solidity ^0.5.4;

contract TokenTest {

string public name;

string public symbol;

uint8 public decimals = 18; // decimals 可以有的小数点个数，最小的代币单位。18 是建议的默认值

uint256 public totalSupply;

address payable public owner;

// 用mapping保存每个地址对应的余额

mapping (address => uint256) public balanceOf;

// 存储对账号的控制

mapping (address => mapping (address => uint256)) public allowance;

/\*\*

\* 初始化构造

\*/

constructor(uint256 initialSupply, string memory tokenName , string memory tokenSymbol) public {

totalSupply = initialSupply \* 10 \*\* uint256(decimals); // 供应的份额，份额跟最小的代币单位有关，份额 = 币数 \* 10 \*\* decimals。

balanceOf[msg.sender] = totalSupply;

name = tokenName; // 代币名称

symbol = tokenSymbol; // 代币符号

owner = msg.sender;

}

/\*\*

\* 代币交易转移的内部实现

\*/

function \_transfer(address \_from, address \_to, uint \_value) internal{

// 确保目标地址不为0x0，因为0x0地址代表销毁

require(\_to != address(0x0));

// 检查发送者余额

require(balanceOf[\_from] >= \_value);

// 确保转移为正数个

require(balanceOf[\_to] + \_value > balanceOf[\_to]);

// 以下用来检查交易，

uint previousBalances = balanceOf[\_from] + balanceOf[\_to];

// Subtract from the sender

balanceOf[\_from] -= \_value;

// Add the same to the recipient

balanceOf[\_to] += \_value;

// 用assert来检查代码逻辑。

assert(balanceOf[\_from] + balanceOf[\_to] == previousBalances);

}

/\*\*

\* 代币交易转移

\* 从自己（创建交易者）账号发送`\_value`个代币到 `\_to`账号

\* @param \_to 接收者地址

\* @param \_value 转移数额

\*/

function transfer(address \_to, uint256 \_value) public {

\_transfer(msg.sender, \_to, \_value);

}

function withdraw() payable public{

require(msg.sender == owner);

owner.transfer(address(this).balance);

}

function SWTBalance(address \_from) public view returns (uint256) {

return \_from.balance;

}

}