



# 井通科技

## jingtum-lib接口说明

V2.0.2

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	吴丹	2018/5/31

<b>1 安装</b>	<b>1</b>
<b>2 项目文件结构</b>	<b>1</b>
<b>3 创建钱包</b>	<b>1</b>
<b>4 REMOTE类</b>	<b>2</b>
4.1 创建REMOTE对象	3
4.2 创建连接	4
4.3 关闭连接	5
4.4 请求底层服务器信息	6
4.5 获取最新账本信息	7
4.6 获取某一账本具体信息	8
4.7 查询某一交易具体信息	11
4.8 请求账号信息	14
4.9 获得账号可接收和发送的货币	16
4.10 获得账号关系	18
4.11 获得账号挂单	22
4.12 获得账号交易列表	24
4.13 获得市场挂单列表	26
4.14 支付	30
4.14.1 创建支付对象	30
4.14.2 传入密钥	30
4.14.3 设置备注	31
4.14.4 提交支付	31
4.15 设置关系	34
4.15.1 创建关系对象	34
4.15.2 传入密钥	34
4.15.3 关系设置	34
4.16 设置账号属性 -----待完善	38
4.16.1 创建属性对象	38
4.16.2 传入密钥	38
4.16.3 属性设置	38

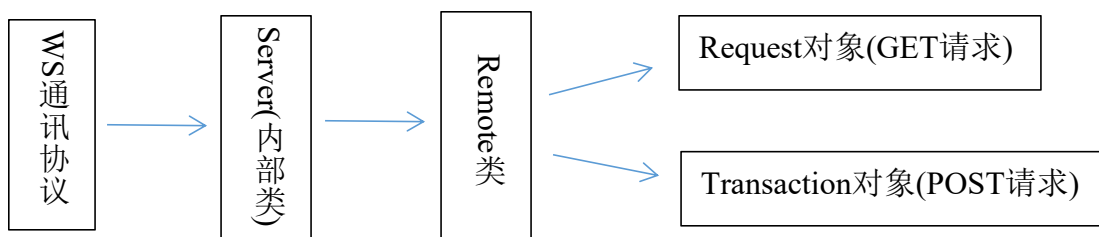
4.17 挂单.....	39
4.17.1 创建挂单对象.....	39
4.17.2 传入密钥.....	40
4.17.3 提交挂单.....	40
4.18 取消挂单.....	44
4.18.1 创建取消挂单对象.....	44
4.18.2 传入密钥.....	44
4.18.3 取消挂单.....	44
4.19 部署合约.....	47
4.19.1 创建部署合约对象.....	47
4.19.2 传入密钥.....	48
4.19.3 部署合约.....	48
4.20 执行合约.....	51
4.20.1 创建执行合约对象.....	51
4.20.2 传入密钥.....	51
4.20.3 执行合约.....	52
4.21 监听事件.....	55
<b>5 REQUEST类.....</b>	<b>56</b>
5.1 指定账本.....	56
5.2 提交请求.....	57
<b>6 TRANSACTION类.....</b>	<b>58</b>
6.1 获得交易账号.....	58
6.2 获得交易类型.....	59
6.3 传入私钥.....	59
6.4 添加备注.....	60
6.5 提交请求.....	61
<b>7 底层常见错误附录.....</b>	<b>61</b>

## 1 安装

```
npm install --save jingtum-lib
```

## 2 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：



## 3 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

#### 4 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)
- \* requestPathFind(options)
- \* createAccountStub()
- \* createOrderBookStub()
- \* buildPaymentTx(options)
- \* buildRelationTx(options)
- \* buildAccountSetTx(options)
- \* buildOfferCreateTx(options)
- \* buildOfferCancelTx(options)
- \* deployContractTx(options)
- \* callContractTx(options)

#### 4.1 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://ts5.jingtum.com:5020', local_sign:  
true});
```

## 4.2 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://ts5.jingtum.com:5020', 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: 'CARE',
  ledger_hash: 'E39CD0668FF2647FD930B1E446E7861EF7B6D194737275A1101B6AF2C4D392EB',
  ledger_index: 9074471,
  ledger_time: 573902570,
  load_base: 256,
  load_factor: 256,
  pubkey_node: 'n9J1qmabAVgRfQ2DPXRzt58odeGsb5FiGCeg2iUCbZ5nRnWuMncc',
```



```
random: 'AD8112448B3B114880D8F65F3309654A4A6FE4590839D10A5C1E8AF602338F5D',
reserve_base: 0,
reserve_inc: 0,
server_status: 'full',
validated_ledgers: '266955-9074471'
}
```

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

### 4.3 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://ts5.jingtum.com:5020', local_sign:
true});
remote.connect(function(err, result) {
  if (err) {
    console.log('err:',err);
  }else{
    remote.disconnect(); //关闭连接
  }
});
```

#### 4.4 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://ts5.jingtum.com:5020', 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:

```
{  
  version: '0.29.60',  
  ledgers: '4685487-8642747',  
  node: 'n94PahMpVKEBUGPn7ymoowELiXD1QWBuMUh7DpyCzJn5hzHMiQoj',  
  state: 'full 122:58:24'  
}
```

返回结果说明:

参数	类型	说明
version	String	服务器部署项目版本
ledgers	String	账本区间
node	String	节点公钥
state	String	服务器状态

#### 4.5 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://ts5.jingtum.com:5020', 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('ledgerInfo:', result);
      }
    });
  }
});
```

返回结果:

```
{
  ledger_hash: '945DA5D61F9AA986C784B6353C7169BC981315CDBC05922C4F34E41B8A45B19F',
  ledger_index: 8642794
}
```

返回结果说明:

参数	类型	说明
ledger_hash	String	账本hash
ledger_index	String	账本高度/区块高度

#### 4.6 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	String	井通区块高度
ledger_hash	String	井通区块hash(与上面ledger_index二选其一)
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://ts5.jingtum.com:5020', 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: '8488670',
    transactions: true
  });
  req.submit(function(err, result) {
    if(err) {console.log('err:',err);}
    else if(result){
      console.log('res:', result);
    }
  });
});
```

返回结果:

```
{
```

```

accepted: true,
account_hash: 'E64648702393BD4AEAAEBFA5144030874070C83D3079A5367FDF828
2C45E1EB',
close_time: 572790860,
close_time_human: '2018-Feb-24 12:34:20',
close_time_resolution: 10,
closed: true,
hash: 'BBFE0C3F25EE707F79A0E4B361A00B2F9254C7DC7AB6B8FED3A804B51F864392
',
ledger_hash: 'BBFE0C3F25EE707F79A0E4B361A00B2F9254C7DC7AB6B8FED3A804B51
F864392',
ledger_index: '8488670',
parent_hash: 'AB78480BDA1B6BA3E328713FD477209D35D3665DC13F5AD38E3AA58CD
9CA0DCA',
seqNum: '8488670',
totalCoins: '599999999996231320',
total_coins: '599999999996231320',
transaction_hash: 'B0E15F21E416E2C720E2F5C4FD2B0B9B6A3EA619ADA3461BDB12
42D022273336',
transactions:
[ '4D09192722D460C29458E401EA7023855521EF3FF8564825474AC6E0EB9CB9A6' ]
}

```

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

#### 4.7 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://ts5.jingtum.com:5020', local_sign: true});
```



```
remote.connect(function(err, result) {
  if (err) {
    return console.log('err:',err);
  }
  var req = remote.requestTx({hash: '084C7823C318B8921A362E39C67A6FB15ADA5BCCD0C7E9A3B13485B1EF2A4313'}));
  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: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',
  Amount: '3000000',
  Destination: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',
  Fee: '10000',
  Flags: 0,
  Memos: [ { Memo: [Object] } ],
  Sequence: 12938,
  SigningPubKey: '021EAF8D79442EF3C27B6398DB39D37F9593F2E5953CF39E57E9A109D4875A7017',
  Timestamp: 572980620,
  TransactionType: 'Payment',
  TxnSignature: '30440220256C52A2C921D1D13FEC048BBFCB1F638470125244F848495E43C990590269FD02206B0D0D2CB465DB367DD3677C8BA3713DFD47BAA94D630573218DA707AA9310A5',
```



```

date: 572980630,
hash: '084C7823C318B8921A362E39C67A6FB15ADA5BCCD0C7E9A3B13485B1EF2A4313',
inLedger: 8507647,
ledger_index: 8507647,
meta:
{ AffectedNodes: [ [Object], [Object], [Object] ],
  TransactionIndex: 0,
  TransactionResult: 'tesSUCCESS' },
validated: true
}

```

返回结果说明:

参数	类型	说明
Account	String	钱包地址
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
validated	Boolean	交易是否通过验证

#### 4.8 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://ts5.jingtum.com:5020', 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: '10884068241',
    Domain: '687474703A2F2F7777772E6A696E6774756D2E636F6D2F',
    Flags: 0,
    LedgerEntryType: 'AccountRoot',
    MessageKey: '736574206D657373616765206B65792074657374',
    OwnerCount: 43,
    PreviousTxnID: 'AE11268454C05B4A2FFFBEC53FFC8FC92118DE387B2911A00879
E6BD76C27D96',
    PreviousTxnLgrSeq: 8642907,
    RegularKey: 'jp53tPyrQLoFriTJhtm8Z9iLUXUDucnwVk',
    Sequence: 1934,
    TransferRate: 1800000000,
    index: 'E80FF91725E82A623ADC46B458D37FB270651A76A07CD6C8115F12028563
42E6' },
  ledger_hash: '79F1CB1EEF303E8928635AA68E4E0319898B7C113AC4C183CD591F274
F0FE2D4',
  ledger_index: 8643224,
  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
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
TransferRate	Integer	手续费汇率
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

#### 4.9 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://ts5.jingtum.com:5020', 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: '870FD292E341D25CEEC919E68FAE4C0E40F6E46045226EED3793B34F08030688',
  ledger_index: 8643336,
  receive_currencies:
  [ 'AAA',
    'CCA',
    'CNY',
    'USD',
    '8100000001000020160022000000000020000001',
    '8100000036000020160622201606300120000002',
```

```

    '8300000001000020160021201600240020000001',
    '830000000B000020160004000000000020000001' ],
  send_currencies:
  [ 'AAA',
    'CCA',
    'CNY',
    'USD',
    '8100000001000020160022000000000020000001',
    '8100000036000020160622201606300120000002',
    '8300000001000020160021201600240020000001',
    '830000000B000020160004000000000020000001' ],
  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

#### 4.10 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://ts5.jingtum.com:5020', 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: 'EFC95414F74F4DFC27703E68FAB28186818A5F596D5CD105EC295B34C7E5A70D',
  ledger_index: 8643385,
  lines:
    [ { account: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',
```

```
    balance: '7.001933333333334',
    currency: 'USD',
    limit: '100',
    limit_peer: '0',
    quality_in: 0,
    quality_out: 0 },
{ account: 'jGRqbsiAf2vgdbet5z31hNLEt8hdBjfPUk',
  balance: '-10',
  currency: 'AAA',
  limit: '100',
  limit_peer: '1000',
  no_skywell_peer: true,
  quality_in: 0,
  quality_out: 0 },
{ account: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',
  balance: '133454.4061014615',
  currency: 'CNY',
  limit: '10000000000',
  limit_peer: '0',
  no_skywell: true,
  quality_in: 0,
  quality_out: 0 },
{ account: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',
  balance: '999984',
  currency: '8100000036000020160622201606300120000002',
  limit: '10000000000',
  limit_peer: '0',
```



```

    no_skywell: true,
    quality_in: 0,
    quality_out: 0 },
{ account: 'jMcCACcfG37xHy7FgqHerzovjLM5FCk7tT',
  balance: '0.04995',
  currency: 'USD',
  limit: '10000000000',
  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	信任的银关
	balance	余额
	currency	货币种类
	limit	信任额度
	limit_peer	对方设置的信任额度，默认0
	quality_in	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

#### 4.11 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://ts5.jingtum.com:5020', 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:
[ { flags: 131072,
  seq: 206,
  taker_gets: {
    currency: 'USD',
    issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',
    value: '0.5' },
  taker_pays: '10000000' },
{ 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	
	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	

#### 4.12 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://ts5.jingtum.com:5020', local_sign: true});

remote.connect(function(err, result) {
  if (err) {
    return console.log('err:',err);
  }
  var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ'};
  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: 8643603,
  ledger_index_min: 4685487,
  marker: { ledger: 8478562, seq: 60 },
  transactions:
    [ { date: 1521021610,
        hash: 'EDC7497E0E7FF3ED79F6A3829401B71F4446C3521E17AC40A937DC3DC9023CF6',
        type: 'received',
        fee: '0.01',
        result: 'tesSUCCESS',
        memos: [Array],
        counterparty: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',
        amount: [Object],
        effects: [] },
```

```
... 100 more items ]
}
```

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		
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		

#### 4.13 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
<code>gets</code>	<code>Object</code>	对家想要获得的货币信息
<code>pays</code>	<code>Object</code>	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://ts5.jingtum.com:5020', 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: '61EF253552DDD8E9D5A1071C81591086E0132391F92311BB5B  
03BD5CE35839AA',  
    BookNode: '0000000000000000',  
    Flags: 0,  
    LedgerEntryType: 'Offer',  
    OwnerNode: '0000000000000000',  
    PreviousTxnID: '547266B62E7B0931F829C58B89541BCFE94ECB1EA78E712D8D  
AD071D40A1EDB9',  
    PreviousTxnLgrSeq: 8574138,  
    Sequence: 11,  
    TakerGets: [Object],  
    TakerPays: '1028266745029520',  
    index: 'CDD0F0DE4D63DB7F208B6EF5A84D2FE5A8CDD53DB68B876A17E9EE357E  
FC877B',  
    owner_funds: '988270574.4452997',  
    quality: '1052631.578982826' },  
  { Account: 'jKog7BTbU7wzDDDH3FPPmveszWQnZSeP5W',  
    BookDirectory: '61EF253552DDD8E9D5A1071C81591086E0132391F92311BB5B  
03C78C2C91CEFB',  
    BookNode: '0000000000000000',  
    Flags: 0,  
    LedgerEntryType: 'Offer',  
    OwnerNode: '0000000000000000',  
    PreviousTxnID: 'BBB6A770A818427EC3B3778F94EF1577704EFCAD0C7602740E  
304207E9FC4943',  
    PreviousTxnLgrSeq: 1352529,  
    Sequence: 10,  
    TakerGets: [Object],
```



```

    TakerPays: '100000000',
    index: '9EDE0EDF25FA1AAC26B11DB38F90095AC8A84351B4B83773162A5B76BF
F98674',
    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
LedgerEntryType	String	账本数据结构类型
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
	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

#### 4.14 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。

##### 4.14.1 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

##### 4.14.2 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

#### 4.14.3 设置备注

方法: `tx.addMemo(memo);`

参数:

参数	类型	说明
memo	String	备注信息

#### 4.14.4 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://ts5.jingtum.com:5020', 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": 0.5,
      "currency": "SWT",
      "issuer": ""
    }
  });
  tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');
  tx.addMemo('给jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c支付0.5swt.');//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: '120000220000000024000007926140000000007A12068400000000000271  
0732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD16774  
46304402202EC74EFA5D3AAA663B49610B45E92438C5FD206376CB23323726180656C9EFF  
9022038577D38CA50296A198A4CD2BE06BCFD8916A64E40D68F5C91C12110A92466988114  
72F05993EBA9858291D364EBF6EEC3D851BD3792831485B6C98BAD6DBF7805D3C5CCC1B4F  
989E0CE6749F9EA7D32E7BB996A44556A716F445A4C687A78344443663670765369766A6B  
6A677452455359363263E694AFE4BB98302E357377742EE1F1',  
  tx_json:  
    { Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',  
      Amount: '500000',  
      Destination: 'jDUjqoDZLhxx4DCf6pvSivjkjgtRESY62c',  
      Fee: '10000',  
      Flags: 0,  
      Memos: [ [Object] ],  
      Sequence: 1938,  
      SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68  
BAF3B1C89FD167',  
      TransactionType: 'Payment',
```

```

    TxnSignature: '304402202EC74EFA5D3AAA663B49610B45E92438C5FD206376CB2
3323726180656C9EFF9022038577D38CA50296A198A4CD2BE06BCFD8916A64E40D68F5C91
C12110A9246698',

    hash: 'FA886D470C6B3A96FBB57639325AEFB9104B56D912AE8AE9E385914EA63E
6B6' }
  }

```

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

## 4.15 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)。

### 4.15.1 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	数量
	currency	货币种类，三到六个字母或20字节的自定义货币
	issuer	货币发行方

返回：Transaction对象

### 4.15.2 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

### 4.15.3 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://ts5.jingtum.com:5020'});
remote.connect(function(err, result) {
  if (err) {
    return console.log('err:',err);
  }
  var options = {
    account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',
    target: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',
    limit:{
      currency: 'CCA',
      value: "0.01",
      issuer: 'js7M6x28mYDiZVJJtfJ84ydrv2PthY9W9u'
    },
    type:'authorize'
  };
  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: '120015220000000024000007932F223CA4FC20230000000163D4038D7EA4C680000000000000000000000000043434100000000001B1CC46616D04D5A8EB8397A7C60B623521336DA6840000000000002710732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD16774463044022064FE815BAB29F691BC5515AB409AF5866A75FA474C0018F33B0D2CF1500170BE02202EA9CA5DAD91922A9785C8837C9C1037E0BEB1E0818EA63FA102F558C6ED126C811472F05993EBA9858291D364EBF6EEC3D851BD3792871485B6C98BAD6DBF7805D3C5CCC1B4F989E0CE6749',
  tx_json: {
    Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',
    Fee: '10000',
    Flags: 0,
    LimitAmount: {
      currency: 'CCA',
      issuer: 'js7M6x28mYDiZVJJtfJ84ydrv2PthY9W9u',
      value: '0.01' },
    RelationType: 1,
    Sequence: 1939,
    SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',
    Target: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',
    Timestamp: 574399740,
    TransactionType: 'RelationSet',
    TxnSignature: '3044022064FE815BAB29F691BC5515AB409AF5866A75FA474C0018F33B0D2CF1500170BE02202EA9CA5DAD91922A9785C8837C9C1037E0BEB1E0818EA63FA102F558C6ED126C',
```



```

    hash: '2BC2051D97999852E515A0E5733F3C0FCC21E0270D93C5CEB43D80BDC1757
35A' }
}

```

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；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	
	Timestamp	Integer	时间戳	
	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
--	------	--------	--------

#### 4.16 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。

##### 4.16.1 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

##### 4.16.2 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

##### 4.16.3 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://ts5.jingtum.com:5020'});
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);
        }
    });
});
```

## 4.17 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。

### 4.17.1 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
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

#### 4.17.2 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

#### 4.17.3 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://ts5.jingtum.com:5020', local_sign:
true});
remote.connect(function (err, result) {
  if (err) {
    return console.log('err:', err);
  }
  var options = {
    type: 'Sell',
    account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',
    taker_gets: {
      value: '0.01',
      currency: 'CNY',
      issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS'
    },
    taker_pays: {
      value: '1',
      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: '120007220008000024000007982F223CAB02644000000000F424065D4038D7EA4C6800000000000000000000000000000434E5900000000007478E561645059399B334448F7544F2EF308ED3268400000000002710732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD1677446304402205E9C26D6F0C4ECC043FAC2D7ADA9740763C89B999857C56047BD3AD3F758976A0220501A401B89560B1F4FA2E9E7727674C0882AA4935748B3A4CBC5BA63F17CCE40811472F05993EBA9858291D364EBF6EEC3D851BD3792',  
  tx_json:  
    { Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',  
      Fee: '10000',  
      Flags: 524288,  
      Sequence: 1944,  
      SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',  
      TakerGets:  
        { currency: 'CNY',  
          issuer: 'jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS',  
          value: '0.01' },  
      TakerPays: '1000000',  
      Timestamp: 574401282,  
      TransactionType: 'OfferCreate',
```

```

    TxnSignature: '304402205E9C26D6F0C4ECC043FAC2D7ADA9740763C89B999857C
56047BD3AD3F758976A0220501A401B89560B1F4FA2E9E7727674C0882AA4935748B3A4CB
C5BA63F17CCE40',

    hash: 'F6B113B1513CE6F5B7CB130028FD06E564E35CC7E83876460E9A9F0038365
424' }
  }
}

```

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	
	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	
	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

## 4.18 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。

### 4.18.1 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

### 4.18.2 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

### 4.18.3 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://ts5.jingtum.com:5020', local_sign:
true});

remote.connect(function (err, result) {
  if (err) {
    return console.log('err:', err);
  }
  var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequenc
e: 688};
  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 va
lidated ledger.',
  tx_blob: '1200082200000000240000079A2F223CADE820190000079868400000000
002710732102FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD
1677446304402207233C0103866BE592885745BDCD7EB501A49B704EAE21DBF5F2EC31062
```

52F12C022003DFB50DD42F0F82BA1532C977853929615690A7261986903301CF15611CEEA0811472F05993EBA9858291D364EBF6EEC3D851BD3792',

tx\_json:

```
{ Account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',
  Fee: '10000',
  Flags: 0,
  OfferSequence: 1944,
  Sequence: 1946,
  SigningPubKey: '02FE64E0C20F0058F22F3742EDC15F49F318C04F88B130742C68BAF3B1C89FD167',
  Timestamp: 574402024,
  TransactionType: 'OfferCancel',
  TxnSignature: '304402207233C0103866BE592885745BDCD7EB501A49B704EAE21DBF5F2EC3106252F12C022003DFB50DD42F0F82BA1532C977853929615690A7261986903301CF15611CEEA0',
  hash: '44BD366DF7BDE52C52FB37B7653564BF685B84B5E51A70E23B0BDCB26D4BA90F' }
```

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
TransactionType	String	交易类型: offerCancel取消订单
TxnSignature	String	交易签名
hash	String	交易hash

#### 4.19 部署合约

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方法取消挂单。

##### 4.19.1 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

#### 4.19.2 传入密钥

方法: `tx.setSecret(secret);`

参数:

参数	类型	说明
secret	String	源账号私钥

#### 4.19.3 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://ts5.jingtum.com:5020', 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: '12001E220000000024000004FA20240000000061400000000989680684000000000271073210330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD02074473045022100E53D151ED6EEC46124CB75C0ED0DCACC7D4458B56109252431DAA80D11D1099802207FE4C2987AFBBD7F1FEC6DBFBD9A32C35D2F82CEC44E5AA6EE3DE0E58F1D8B627F94726573756C743D7B7D3B202066756E6374696F6E20496E69742874292020726573756C743D73634765744163636F756E7442616C616E6365287429202072657475726E20726573756C742020656E643B202066756E6374696F6E20666F6F2874292020726573756C743D73634765744163636F756E7442616C616E6365287429202072657475726E20726573756C742020656E648114B5F762798A53D543A014CAF8B297CFF8F2F937E8FAEB7012226A486239434A41577942346A7239315652576E3936446B756B473462776474795468E1F1',
  tx_json:
    { Account: 'jHb9***tyTh',
      Amount: '10000000',
      Args: [ [Object] ],
      Fee: '10000',
      Flags: 0,
      Method: 0,
    }
}

```

```

    Payload: '726573756C743D7B7D3B202066756E6374696F6E20496E697428742920
20726573756C743D73634765744163636F756E7442616C616E63652874292020726574757
26E20726573756C742020656E643B202066756E6374696F6E20666F6F2874292020726573
756C743D73634765744163636F756E7442616C616E6365287429202072657475726E20726
573756C742020656E64',

    Sequence: 1274,

    SigningPubKey: '0330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654
A313222F7FD020',

    TransactionType: 'ConfigContract',

    TxnSignature: '3045022100E53D151ED6EEC46124CB75C0ED0DCACC7D4458B5610
9252431DAA80D11D1099802207FE4C2987AFBBD7F1FEC6DBFBD9A32C35D2F82CEC44E5AA6
EE3DE0E58F1D8B62',

    hash: 'D7E40A7164C11BFA81C05117010631EA5BCBD8A1A0B3B2FF343D9FB3F3575
936'  }

}

```

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: <code>ConfigContract</code> 部署合约
TxnSignature	String	交易签名
hash	String	交易hash

## 4.20 执行合约

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`方法取消挂单。

### 4.20.1 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
foo	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

### 4.20.2 传入密钥

方法: `tx.setSecret(secret);`

参数:

参数	类型	说明
----	----	----

secret	String	源账号私钥
--------	--------	-------

### 4.20.3 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://ts5.jingtum.com:5020', local_sign: true});
remote.connect(function (err, result) {
  if (err) {
    return console.log('err:', err);
  }
  var options = {account: 'jHb9***tyTh', destination: 'j4YVQxCxaRRQ6gCVUvi9MoiTfWyPRnHwej', foo: '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: '12001E220000000024000004FC2024000000016840000000000271073210330E7FC9D56BB25D6893BA3F317AE5BCF33B3291BD63DB32654A313222F7FD02074473045022100835BEDFD6794F3B95A31971EDBF1865F9190D1A66FFC9515883B15107B368894022052F44E0E16B6A3BC3C8CF2A3881E35687060EB73B6B0503611CEF58781A512EA70110366F6F8114B5F762798A53D543A014CAF8B297CFF8F2F937E88314EC4F411C3B37C3E451D462C27489B0549650572CFAEB7012226A486239434A41577942346A7239315652576E3936446B756B473462776474795468E1F1',
  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

#### 4.21 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://ts5.jingtum.com:5020', local_sign: true});
remote.connect(function (err, result) {
    if (err) {
        return console.log('err:', err);
    }
    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);
        });
    });
});
```

## 5 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)

### 5.1 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://ts5.jingtum.com:5020', 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);
    }
    });
});
```

## 5.2 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://ts5.jingtum.com:5020', 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);
        }
    });
});
```

## 6 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)

### 6.1 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://ts5.jingtum.com:5020'});
var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequence: 688};
var tx = remote.buildOfferCancelTx(options);
var account = tx.getAccount();
console.log(account);
```

## 6.2 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://ts5.jingtum.com:5020'});
var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequence: 688};
var tx = remote.buildOfferCancelTx(options);
var type = tx.getTransactionType();
console.log(type);
```

## 6.3 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://ts5.jingtum.com:5020'});
```

```
var options = {account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ', sequence: 688};  
var tx = remote.buildOfferCancelTx(options);  
tx.setSecret('sn37nYrQ6KPJvTFmaBYokS3FjXUWd');
```

## 6.4 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://ts5.jingtum.com:5020'});  
var tx = remote.buildPaymentTx({  
  account: 'jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ',  
  to: 'jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c',  
  amount: {  
    "value": 0.5,  
    "currency": "SWT",  
    "issuer": ""  
  }  
});  
tx.addMemo('给jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c支付0.5swt.');
```



## 6.5 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://ts5.jingtum.com:5020', local_sign:
true});
remote.connect(function (err, result) {
    if (err) {
        return console.log('err:', err);
    }
    var req = remote.requestAccountInfo({account: 'jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ'}));
    req.submit(function(err, result) {
        if(err) {console.log('err:',err);}
        else if(result){
            console.log('res:', result);
        }
    });
});
```

## 7 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 $\geq 1.0$
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.