jingtum-lib-go使用说明

V0.1.0

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **简介** | **作者** | **日期** |
| 0.1.0 | 出版主要接口说明 | 杨雪波 | 2018/8/20 |
|  |  |  |  |

本文档介绍jingtum-lib在go平台上的如何使用，另有接口的详细文档。

# 安装

通过go get安装。

go get github.com/swtcpro/jingtum-lib-go

# 项目文件结构

jingtum-lib库基于websocket协议跟底层交互，其中websocket封装到Server类中，Server类是一个内部类；Server类封装在Remote类中，Remote类提供对外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：

Remote类

Request对象（Get请求）

WebSocket

通讯协议

Server

内部类

Transaction对象（Post请求）

# 创建钱包

首先添加jingtum-lib-go引用到项目中，代码中import "jingtumLib"

;

方法一：

jingtumLib.Generate()

方法二：

jingtumLib.FromSecret(secret)

参数：

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| secret | string | 井通钱包私钥 |

# 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。其主要方法如下。

## 创建Remote类

通过构造函数传入井通底层服务地址并制定是否本地签名来创建Remote类。

remote, err := NewRemote("ws://ts5.jingtum.com:5020", true)

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| url | string | 井通底层服务地址 |
| localSign | bool | 交易是否以本地签名的方式发送给底层 |

## 创建连接

每个Remote对象都应该首先通过Connect方法手动连接底层，然后才可以请求底层数据。请求结果在回调函数callback中。

代码示例：

  remote, err := NewRemote("ws://ts5.jingtum.com:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

   conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

## 关闭连接

每个Remote对象可以手动通过Disconnect方法关闭连接。

代码示例：

remote, err := NewRemote("ws://139.129.194.175:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

defer remote.Disconnect()

## 请求底层服务器信息

首先通过RequestServerInfo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。

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    req, err := remote.RequestServerInfo()

    if err != nil {

        t.Fatalf("Fail request server info %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request server info %s", err.Error())

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Success request server info %s", jsonByte)

    })

## 获取最新账本信息

首先通过RequestLedgerClosed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(LedgerIndex)与区块hash(LedgerHash)。

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    req, err := remote.RequestLedgerClosed()

    if err != nil {

        t.Fatalf("Fail request ledger closed %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request ledger closed %s", err.Error())

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Success request ledger closed %s", jsonByte)

    })

## 获取某一账本具体信息

首先通过RequestLedger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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| ledger\_index | int | 井通区块高度 |
| ledger\_hash | string | 井通区块hash(与上面ledger\_index二选其一) |
| transactions | bool | 是否返回账本上的交易记录hash，默认false |

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    options := map[string]interface{}{"transactions": true, "ledger\_index": 969054, "ledger\_hash": "AEE4B16B543D8C8924F09C1DB822C6419780B86019F5F5FF8DC2938E7E0E89D2"}

    req, err := remote.RequestLedger(options)

    if err != nil {

        t.Fatalf("Fail request ledger %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request ledger %s", err.Error())

            wg.Done()

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Success request ledger %s", jsonByte)

    })

## 查询某一交易具体信息

首先通过RequestTx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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| hash | string | 交易hash |

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    hash := "6537F72CE1DBD8043230C3FF64C6E5E95B11F6573D91EF6A13FEADE6940CB71A"

    req, err := remote.RequestTx(hash)

    if err != nil {

        t.Fatalf("Fail request tx %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request tx %s", err.Error())

            return

        }

        // jsonByte, \_ := json.Marshal(result)

        t.Log("Success request tx")

    })

## 请求账号信息

首先通过RequestAccountInfo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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| account | string | 井通钱包地址 |

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    //请求账号信息

    options := map[string]interface{}{"account":"j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk"}

    req, err := remote.RequestAccountInfo(options)

    if err != nil {

        t.Fatalf("RequestAccountInfo fail : %s", err.Error())

        return

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Requst account info : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Success Requst account info result : %s", jsonBytes)

    })

## 获得账号可接收和发送的货币

首先通过RequestAccountTums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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| account | string | 井通钱包地址 |

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    options := map[string]interface{}{"account": "j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk"}

    req, err := remote.RequestAccountTums(options)

    if err != nil {

        t.Fatalf("Fail request Account Tums %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request Account Tums %s", err.Error())

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Success request Account Tums %s", jsonByte)

    })

## 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RequestAccountRelations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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| account | string | 井通钱包地址 |
| type | string | 关系类型，支持trust，authorize，freeze, unfreeze |

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    options := map[string]interface{}{"account": "j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk", "type": "trust"}

    req, err := remote.RequestAccountRelations(options)

    if err != nil {

        t.Fatalf("Fail request account relations %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request account relations %s", err.Error())

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Success request account relations %s", jsonByte)

    })

## 获得账号挂单

首先通过RequestAccountOffers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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| account | string | 井通钱包地址 |

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    options := map[string]interface{}{"account": "j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk"}

    req, err := remote.RequestAccountOffers(options)

    if err != nil {

        t.Fatalf("Fail request account offers %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request account offers %s", err.Error())

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Success request account offers %s", jsonByte)

    })

## 获得账号交易列表

首先通过RequestAccountTx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。

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

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    options := map[string]interface{}{"account": "j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk"}

    req, err := remote.RequestAccountTx(options)

    if err != nil {

        t.Fatalf("Fail request account tx %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request account tx %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Success request account tx : %s", jsonBytes)

    })

## 获得市场挂单列表

首先通过RequestOrderBook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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| gets | Amount | 挂单方获得的货币信息 |
| pays | Amount | 挂单方支付的货币信息 |

代码示例：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    options := make(map[string]interface{})

    gets := constant.Amount{}

    gets.Currency = "SWT"

    pays := constant.Amount{}

    pays.Currency = "CNY"

    pays.Issuer = "jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS"

    options["gets"] = gets

    options["pays"] = pays

    req, err := remote.RequestOrderBook(options)

    if err != nil {

        t.Fatalf("Fail request order book %s", err.Error())

    }

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail request order book %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Success request order book: %s", jsonBytes)

    })

## 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

Remote类的方法：

func (remote \*Remote) BuildPaymentTx(account string, to string, amount constant.Amount) (\*Transaction, error)

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| account | String | 发起账号 |
| to | String | 目标账号 |
| Amount | Amount | 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 |

### 传入密钥

Transaction的方法：

func (tx \*Transaction) SetSecret(secret string)

### 设置备注

Transaction的方法：

func (tx \*Transaction) SetSecret(secret string)

### 提交支付

Transaction的方法：

func (tx \*Transaction) Submit(callback func(err error, result interface{}))

示例代码：

wsurl := "ws://123.57.219.57:5020"

    remote, err := NewRemote(wsurl, true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err)

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Connect to %s success. Result : %s.", wsurl, jsonByte)

    })

    if conErr != nil {

        t.Fatalf("Connect to %s fail : %s", wsurl, conErr.Error())

        return

    }

    defer remote.Disconnect()

    //支付请求

    var v struct {

        account string

        secret string

    }

    v.account = "jGXjV57AKG7dpEv8T6x5H6nmPvNK5tZj72"

    v.secret = "ssc5eiFivvU2otV6bSYmJeZrAsQK3"

    to := "j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk"

    amount := constant.Amount{}

    amount.Currency = "SWT"

    amount.Value = "0.0001"

    tx, err := remote.BuildPaymentTx(v.account, to, amount)

    if err != nil {

        t.Fatalf("Build paymanet tx fail : %s", err.Error())

        return

    }

    tx.SetSecret(v.secret)

    tx.AddMemo("支付0.0001SWT")

    tx.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Fail Payment : %s", err.Error())

            return

        }

        jsonByte, \_ := json.Marshal(result)

        t.Logf("Success Payment result : %s", jsonByte)

    })

## 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

Remote类的方法：

public Transaction<RelationTxResponse>BuildRelationTx(RelationTxOptions options)

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| Account | String | 设置关系的源账号 |
| Target | String | 目标账号，授权和冻结才有 |
| Type | RelationType (enum) | 关系种类 |
| Limit | Amount | 关系金额 |
| 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 |

### 传入密钥

Transaction<T>的方法：

public void SetSecret(string secret)

### 关系设置

Transaction<T>的方法：

public void Submit(MessageCallback<T> callback)

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildRelationTx(new RelationTxOptions {

Account = "j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

Target = "jBKaXuYemkAb5HytZgosAcWgWDZbBvz6KR",

Limit = new Amount

{

Value = "0.01",

Currency = "CNY",

Issuer = " jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS"

},

Type = RelationType.Authorize

});

tx.SetSecret("ssGkkAMnKCBkhGVQd9CNzSQv5zdNi");

tx.Submit(txResult => {

var info = txResult.Result;

});

}

});

## 设置账号属性

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账户；signer用于设置签名。

### 创建属性对象

Remote类的方法：

public Transaction<AccountSetTxResponse>BuildAccountSetTx(AccountSetTxOptions options)

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| Account | String | 设置属性的源账号 |
| Type | AccountSetType (enum) | 属性种类 |
| SetFlag | SetClearFlag (enum) | 属性编号 |

### 传入密钥

Transaction<T>的方法：

public void SetSecret(string secret)

### 属性设置

Transaction<T>的方法：

public void Submit(MessageCallback<T> callback)

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildAccountSetTx(new AccountSetTxOptions

{

Account = "j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

Type = AccountSetType.Property，

SetFlag = SetClearFlag.RequireDest

});

tx.SetSecret("ssGkkAMnKCBkhGVQd9CNzSQv5zdNi");

tx.Submit(txResult => {

var info = txResult.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

Remote类的方法：

public Transaction<OfferCreateTxResponse>BuildOfferCreateTx(OfferCreateTxOptions options)

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| Account | String | 挂单方账号 |
| Type | OfferType (enum) | 挂单类型，可选两个值：Buy、Sell |
| TakerGets | Amount | 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 |
| TakerPays | Amount | 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 |

### 传入密钥

Transaction<T>的方法：

public void SetSecret(string secret)

### 提交挂单

Transaction<T>的方法：

public void Submit(MessageCallback<T> callback)

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildOfferCreateTx(new OfferCreateTxOptions

{

Account = " j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

Type = OfferType.Sell,

TakerGets = new Amount

{

Value = "0.01",

Currency = " SWT ",

Issuer = ""

},

TakerPays = new Amount

{

Value = "1",

Currency = "CNY",

Issuer = " jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS "

}

});

tx.SetSecret("ssGkkAMnKCBkhGVQd9CNzSQv5zdNi");

tx.Submit(txResult => {

var info = txResult.Result;

});

}

});

## 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。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| Account | String | 挂单方账号 |
| Sequence | unit | 取消的单子号 |

### 创建取消挂单对象

Remote类的方法：

public Transaction<OfferCancelTxResponse>BuildOfferCancelTx(OfferCancelTxOptions options)

### 传入密钥

Transaction<T>的方法：

public void SetSecret(string secret)

### 取消挂单

Transaction<T>的方法：

public void Submit(MessageCallback<T> callback)

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildOfferCancelTx(new OfferCancelTxOptions

{

Account = "j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

Sequence = 8

});

tx.SetSecret("ssGkkAMnKCBkhGVQd9CNzSQv5zdNi");

tx.Submit(txResult => {

var info = txResult.Result;

});

}

});

## 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方法部署合约。

|  |  |  |
| --- | --- | --- |
| **参数** | **类型** | **说明** |
| account | String | 合约交易源账号 |
| amount | double | 支付金额 |
| payload | string | 智能合约代码 |
| params | String[] | 合约参数 |

### 创建部署合约对象

Remote类的方法：

func (remote \*Remote) DeployContractTx(options map[string]interface{}) (\*Transaction, error)

### 传入密钥

Transaction的方法：

func (tx \*Transaction) SetSecret(secret string)

### 部署合约

Transaction的方法：

func (tx \*Transaction) Submit(callback func(err error, result interface{}))

示例代码：

remote, err := NewRemote("ws://139.129.194.175:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    wg := sync.WaitGroup{}

    wg.Add(1)

    //部署合约

    options := map[string]interface{}{"account": "jHJJXehDxPg8HLYytVuMVvG3Z5RfhtCz7h", "amount": float64(100), "payload": fmt.Sprintf("%X", "result={}; function Init(t) result=scGetAccountBalance(t) return result end; function foo(t) result=scGetAccountBalance(t) return result end"), "params": []string{"jHJJXehDxPg8HLYytVuMVvG3Z5RfhtCz7h"}}

    tx, err := remote.DeployContractTx(options)

    if err != nil {

        t.Fatalf("Fail request deploy contract %s", err.Error())

    } else {

        tx.SetSecret("saNUs41BdTWSwBRqSTbkNdjnAVR8h")

        tx.Submit(func(err error, data interface{}) {

            if err != nil {

                t.Fatalf("Fail request deploy contract %s", err.Error())

            } else {

                jsonBytes, \_ := json.Marshal(data)

                t.Logf("Success deploy contract : %s", string(jsonBytes))

            }

        })

    }

## 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方法执行合约。

|  |  |  |
| --- | --- | --- |
| **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 |
| params | String[] | 合约参数 |

### 创建执行合约对象

Remote类的方法：

func (remote \*Remote) CallContractTx(options map[string]interface{}) (\*Transaction, error)

### 传入密钥

Transaction的方法：

func (tx \*Transaction) SetSecret(secret string)

### 执行合约

Transaction的方法：

func (tx \*Transaction) Submit(callback func(err error, result interface{}))

示例代码：

remote, err := NewRemote("ws://139.129.194.175:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    options := map[string]interface{}{"account": "jHJJXehDxPg8HLYytVuMVvG3Z5RfhtCz7h", "destination": "jGXjV57AKG7dpEv8T6x5H6nmPvNK5tZj72", "foo": "foo", "params": []string{"jHJJXehDxPg8HLYytVuMVvG3Z5RfhtCz7h"}}

    tx, err := remote.CallContractTx(options)

    if err != nil {

        t.Fatalf("Fail request call contract Tx %s", err.Error())

            }

    tx.SetSecret("saNUs41BdTWSwBRqSTbkNdjnAVR8h")

    tx.Submit(func(err error, data interface{}) {

        if err != nil {

            t.Fatalf("Fail request call contract Tx %s", err.Error())

        } else {

            jsonBytes, \_ := json.Marshal(data)

            t.Logf("Success call contract Tx : %s", string(jsonBytes))

        }

           })

# 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。可以从回调函数获取错误或者结果。提供以下方法：

SelectLedger指定账本

Submit提交请求

## 指定账本

SelectLedger方法

func (req \*Request) SelectLedger(ledger interface{})

支持int（区块高度），string(账本hash)，账本状态

示例代码：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    //请求账号信息

    options := map[string]interface{}{"account": "j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk"}

    req, err := remote.RequestAccountInfo(options)

    if err != nil {

        t.Fatalf("RequestAccountInfo fail : %s", err.Error())

                return

    }

    req.SelectLedger(1065000)

## 提交请求

提交请求方法：

func (req \*Request) Submit(callback func(err error, data interface{}))

回调函数，包含两个信息：错误信息和结果信息

示例代码：

remote, err := NewRemote("ws://123.57.219.57:5020", true)

    if err != nil {

        t.Fatalf("New remote fail : %s", err.Error())

        return

    }

    conErr := remote.Connect(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("New remote fail : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Connect success : %s", jsonBytes)

    })

    if conErr != nil {

        t.Fatalf("Connect service fail : %s", conErr.Error())

        return

    }

    defer remote.Disconnect()

    //请求账号信息

    options := map[string]interface{}{"account": "j3N35VHut94dD1Y9H1KoWmGZE2kNNRFcVk"}

    req, err := remote.RequestAccountInfo(options)

    if err != nil {

        t.Fatalf("RequestAccountInfo fail : %s", err.Error())

        return

    }

    req.SelectLedger(1065000)

    req.Submit(func(err error, result interface{}) {

        if err != nil {

            t.Fatalf("Requst account info : %s", err.Error())

            return

        }

        jsonBytes, \_ := json.Marshal(result)

        t.Logf("Success Requst account info result : %s", jsonBytes)

            })

# Transaction<T>类

Transaction<T>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。每个回调函数包含错误信息和成功后的结果。提供以下接口：

* Account属性
* TransactionType属性
* SetSecret方法
* AddMemo方法
* SetPath方法
* SetSendMax方法
* SetTransferRate方法
* SetFlags方法
* Submit方法

## 获得交易账号

通过Account属性获取当前交易的账号。

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildOfferCancelTx(new OfferCancelTxOptions

{

Account = " j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

Sequence = 8

});

var account = tx.Account;

Console.Write(account);

}

});

## 获得交易类型

通过TransactionType获取当前交易的类型。

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildOfferCancelTx(new OfferCancelTxOptions

{

Account = "jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ",

Sequence = 688

});

var type = tx.TransactionType;

Console.Write(type);

}

});

## 传入私钥

交易提交之前需要传入私钥，请使用SetSecret方法。

public void SetSecret(string secret)

示例代码：

var remote = new Remote(@"ws://ts5.jingtum.com:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildOfferCancelTx(new OfferCancelTxOptions

{

Account = " j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

Sequence = 8

});

tx.SetSecret("ssGkkAMnKCBkhGVQd9CNzSQv5zdNi");

}

});

## 添加备注

为当前的交易添加备注信息

public void AddMemo(string data)

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildPaymentTx(new PaymentTxOptions {

Account= " j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

To = " jBKaXuYemkAb5HytZgosAcWgWDZbBvz6KR",

Amount = new Amount

{

Value = "0.5",

Currency = "SWT",

Issuer = ""

}

});

tx.SetSecret("ssGkkAMnKCBkhGVQd9CNzSQv5zdNi ");

tx.AddMemo("给jBKaXuYemkAb5HytZgosAcWgWDZbBvz6KR支付0.5swt.");

}

});

## 提交请求

通过Submit方法提交交易请求，参数为回调函数，其中会回传错误信息或者成功的结果。

示例代码：

var remote = new Remote(@"ws://123.57.219.57:5020", true);

remote.Connect(result =>

{

if (result.Exception != null)

{

Console.Write(result.Exception.Message);

}

else

{

var tx = remote.BuildOfferCancelTx(new OfferCancelTxOptions

{

Account = " j9FGhAW9dSzL3RjbXkyW6Z6bHGxFk8cmB1",

Sequence = 8

});

tx.SetSecret("ssGkkAMnKCBkhGVQd9CNzSQv5zdNi ");

tx.Submit(txResult => {

if (txResult.Exception != null)

{

Console.Write(txResult.Exception.Message);

}

else

{

var info = txResult.Result;

}

});

}

});

# 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 private 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\_private\_KEY | private 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. |