井通科技
JINGTUM TECH

井通科技

jingtum-cpp-lib接口说明

V1.0

版本历史

版本	简介	作者	日期
1.0	完成初稿	江	2018.10.28

井通科技
JINGTUM TECH

井通科技
JINGTUM TECH

1. 环境及设置

（1）开发及使用环境：

Jingtum-lib C++库开发环境为Visual Studio Community 2017，同时开发中应用到了BOOST库（V1.68，可于此下载https://dl.bintray.com/boostorg/release/1.68.0/source/boost_1_68_0.zip）及OpenSSL（V1.1，可于此处下载https://www.openssl.org/source/；在上传的lib文件夹中，也已附上相应的lib和dll文件）。下文假设将boost和相应的库文件均放置在D盘。

（2）设置

打开MSVC 2017，新建一个工程，点击进入Project->Properties选项卡

A. 在左侧VC++ Directories里：

1）Include Directorise，添加上boost的include路径，如D:\boost\include\boost-1_68\；

2）Include Directorise，添加上jingtumlib的include路径，如D:\jingtumlib\include\（已包含相关的openssl库文件，如果另行安装openssl，则添加openssl相应的头文件路径）；

3）Library Directories，添加上boost的lib路径，如D:\boost\lib；

4）Library Directories，添加上openssl的lib路径，如D:\jingtumlib\lib\openssl；

B. 在左侧Linker->Input里，添加上相关的lib文件路径：

1）D:\jingtumlib\lib\openssl\libssl.lib；

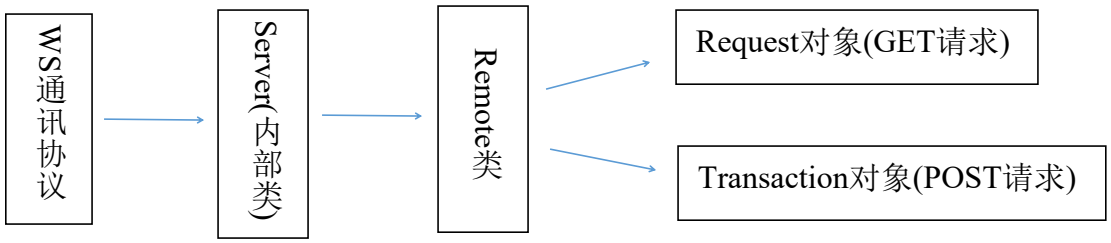
2）D:\jingtumlib\lib\openssl\libcrypto.lib

3）D:\jingtumlib\lib\secp256k1\secp256k1.lib

C. 将D:\jingtumlib\lib下各个文件夹中的dll文件拷贝至自己工程的工程文件同路径下。

2 项目文件结构

jingtum-lib库基于ws协议跟底层交互，其中ws（基于boost/beast）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.from_secret(secret)

参数：

参数	类型	说明
secret	String	井通钱包私钥

例子：

```
#include <jingtum-lib/jingtumlib.h>
#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib")
main(){
	jingtumlib::Wallet wallet;
	//方式一
	std::string res1 = wallet.generate();
	//方式二
	std::string res2 = wallet.fromSecret("ssyA9qRSmdRUfEppFPy5QdPHZwbgU");
}
```

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()
* disConnect()
* requestServerInfo()
* requestLedgerClosed()
* requestLedger(options)
* requestTx(options)
* requestAccountInfo(options)
* requestAccountTums(options)
* requestAccountRelations(options)
* requestAccountOffers(options)
* requestAccountTx(options)
* requestOrderBook(options)
* on(options)
* buildPaymentTx(options)
* buildRelationTx(options)
* buildAccountSetTx(options)
* buildOfferCreateTx (options)
* buildOfferCancelTx(options)
* deployContractTx(options)
* callContractTx(options)

4.1 创建Remote对象

方法：Remote(options)

参数：

参数	类型	说明
server	string	服务器地址
local_sign	Boolean	交易是否以本地签名的方式发送给底层，默认为False

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib")

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

}
```

说明：由于C++本身不直接支持JSON，因此在参数输入时，传入字符串形式表示的JSON数据。下同。

4.2 创建连接

初始化Remote对象应该手动连接底层，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：`remote.connect(callback)`

参数：无

例子：

```
#include <jingtum-lib/jingtumlib.h>
#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib")
void callback_print(std::string, std::string, jingtumlib::Remote *);
int main(){
	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",\"local_sign\":\"true\"}";
	jingtumlib::Remote remote(serverInfo);
	remote.connect(callback_print);
}
void callback_print(std::string ex, std::string msg, jingtumlib::Remote *remote) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		std::cout << "result:" << msg << std::endl;
	}
}
```

4.3 关闭连接

每个Remote对象可以手动关闭连接。

方法：remote.disConnect()

参数：无

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_print(std::string, std::string, jingtumlib::Remote *);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_print);

}

void callback_print(std::string ex, std::string msg, jingtumlib::Remote *r
emote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {

		std::cout << "result:" << msg << std::endl;

		remote->disConnect();

	}

}
```

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requestServerInfo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：

```
#include <jingtum-lib/jingtumlib.h>
#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib")
void callback_requestServerInfo(std::string, std::string, jingtumlib::Remote *);
void callback_print(std::string, std::string);
int main(){
	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",\"local_sign\":\"true\"}";
	jingtumlib::Remote remote(serverInfo);
	remote.connect(callback_requestServerInfo);
}
void callback_requestServerInfo(std::string ex, std::string msg, jingtumlib::Remote *remote) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		jingtumlib::Request req = remote->requestServerInfo();
		req.submit(callback_print);
	}
```

```
}
void callback_print(std::string ex, std::string msg) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		std::cout << "result:" << msg << std::endl;
	}
}
```

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 requestLedgerClosed

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

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_requestLedgerClosed(std::string, std::string, jingtumlib::R
emote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestLedgerClosed);

}

void callback_requestLedgerClosed(std::string ex, std::string msg, jingtum
lib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {

		jingtumlib::Request req = remote->requestLedgerClosed();

		req.submit(callback_print);

	}

}
```

```
void callback_print(std::string ex, std::string msg) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		std::cout << "result:" << msg << std::endl;
	}
}
```

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requestLedger方法获得某一账本的具体信息。

方法：remote.requestLedger(options)

参数：

参数	类型	说明
ledger_index/ ledger_hash	String	井通区块高度/井通区块hash
transactions	Boolean	是否返回账本上的交易记录hash，默认false

注：整体参数是JSON数据，当参数都不填时，默认返回最新账本信息。

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_requestLedger(std::string, std::string, jingtumlib::Remote
*);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestLedger);

}

void callback_requestLedger(std::string ex, std::string msg, jingtumlib::R
emote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {
```

```
                //jingtumlib::Request req = remote->requestLedger("{}");  //
默认返回最新账本信息
                jingtumlib::Request req = remote->requestLedger("{\"ledger_i
ndex\":1000,\"transactions\":\"true\"}");
                req.submit(callback_print);
        }
}
void callback_print(std::string ex, std::string msg) {
        if (ex != "") {
                std::cout << ex << std::endl;
        }
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

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 requestTx 方法获得某一交易的具体信息。

方法：remote.requestTx(options)

参数：

参数	类型	说明
hash	String	交易hash

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_requestTx(std::string, std::string, jingtumlib::Remote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestTx);

}

void callback_requestTx(std::string ex, std::string msg, jingtumlib::Remot
e *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {

		jingtumlib::Request req = remote->requestTx("{\"hash\":\"B0E
15F21E416E2C720E2F5C4FD2B0B9B6A3EA619ADA3461BDB1242D022273336\"}");

		req.submit(callback_print);
```

```
        }
}
void callback_print(std::string ex, std::string msg) {
        if (ex != "") {
                std::cout << ex << std::endl;
        }
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

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 requestAccountInfo 方法获得某一账号的交易信息。

方法：remote.requestAccountInfo(option)

参数：

参数	类型	说明
account	String	井通钱包地址

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_requestAccountInfo(std::string, std::string, jingtumlib::Re
mote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestAccountInfo);

}

void callback_requestAccountInfo(std::string ex, std::string msg, jingtuml
ib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {
```

```
            jingtumlib::Request req = remote->requestAccountInfo("{\"acc
ount\":\"jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c\"}");
            req.submit(callback_print);
      }
}
void callback_print(std::string ex, std::string msg) {
      if (ex != "") {
            std::cout << ex << std::endl;
      }
      else {
            std::cout << "result:" << msg << std::endl;
      }
}
```

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 Remote 对象，然后通过 requestAccountTums 方法获得某一账号可发送和接收的货币种类。

方法：remote.requestAccountTums (options)

参数：

参数	类型	说明
account	String	井通钱包地址

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_requestAccountTums(std::string, std::string, jingtumlib::Re
mote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestAccountTums);

}

void callback_requestAccountTums(std::string ex, std::string msg, jingtuml
ib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {
```

```
                jingtumlib::Request req = remote->requestAccountTums("{\"acc
ount\":\"jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c\"}");
                req.submit(callback_print);
        }
}
void callback_print(std::string ex, std::string msg) {
        if (ex != "") {
                std::cout << ex << std::endl;
        }
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

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 requestAccountRelations 方法获得某一账号指定关系的信息。

方法：remote.requestAccountRelations (options)

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

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_requestAccountRelations(std::string, std::string, jingtumli
b::Remote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestAccountRelations);

}

void callback_requestAccountRelations(std::string ex, std::string msg, jin
gtumlib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;
```

```
	}
	else {
		jingtumlib::Request req = remote->requestAccountRelations("
{\"account\":\"jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c\",\"type\":\"trust\"}
");
		req.submit(callback_print);
	}
}
void callback_print(std::string ex, std::string msg) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		std::cout << "result:" << msg << std::endl;
	}
}
```

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 requestAccountOffers

方法获得某一账号的挂单信息。

方法：remote.requestAccountOffers (options)

参数：

参数	类型	说明
account	String	井通钱包地址

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_requestAccountOffers(std::string, std::string, jingtumlib::
Remote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestAccountOffers);

}

void callback_requestAccountOffers(std::string ex, std::string msg, jingtu
mlib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {
```

```
            jingtumlib::Request req = remote->requestAccountOffers("{\"a
ccount\":\"jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c\"}");
            req.submit(callback_print);
      }
}
void callback_print(std::string ex, std::string msg) {
      if (ex != "") {
            std::cout << ex << std::endl;
      }
      else {
            std::cout << "result:" << msg << std::endl;
      }
}
```

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 requestAccountTx 方法获得某一账号的交易列表信息。

方法：remote.requestAccountTx (options);

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

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib")

void callback_requestAccountTx(std::string, std::string, jingtumlib::Remote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\", \"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestAccountTx);

}

void callback_requestAccountTx(std::string ex, std::string msg, jingtumlib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {
```

```
            jingtumlib::Request req = remote->requestAccountTx("{\"accou
nt\":\"jS1DsUTZhngqmgvyeEqChw5kF99M6ynaT\"}");
            req.submit(callback_print);
      }
}
void callback_print(std::string ex, std::string msg) {
      if (ex != "") {
            std::cout << ex << std::endl;
      }
      else {
            std::cout << "result:" << msg << std::endl;
      }
}
```

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 requestOrderBook 方法获得市场挂单列表信息。

方法：remote.requestOrderBook (options)

参数：

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

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib")

void callback_requestOrderBook(std::string, std::string, jingtumlib::Remote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestOrderBook);

}

void callback_requestOrderBook(std::string ex, std::string msg, jingtumlib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {
```

```
            jingtumlib::Request req = remote->requestOrderBook("{\"gets\
":{\"currency\":\"SWT\",\"issuer\":\"\"},\"pays\":{\"currency\":\"CNY\",\
"issuer\":\"jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS\"}}");
            req.submit(callback_print);
      }
}
void callback_print(std::string ex, std::string msg) {
      if (ex != "") {
            std::cout << ex << std::endl;
      }
      else {
            std::cout << "result:" << msg << std::endl;
      }
}
```

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（options)

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

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 添加备注

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( )

参数：无

支付完整例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_buildPaymentTx(std::string, std::string, jingtumlib::Remote
*);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_buildPaymentTx);

}

void callback_buildPaymentTx(std::string ex, std::string msg, jingtumlib::
Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else{

		jingtumlib::Transaction tx = remote->buildPaymentTx("{\"acco
unt\":\"jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ\",\"to\":\"jDUjqoDZLhzx4DCf6pv
SivjkjgtRESY62c\",\"amount\":{\"value\":1,\"currency\":\"SWT\",\"issuer\":
\"\"}}");

		tx.setSecret("sn37nYrQ6KPJvTFmaBYokS3FjXUWd");

		tx.addMemo("给jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c支付1swt.");

		tx.submit(callback_print);
```

```
	}
}
void callback_print(std::string ex, std::string msg) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		std::cout << "result:" << msg << std::endl;
	}
}
```

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 (options)

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		Amount	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

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

设置关系完整例子：

```
#include <jingtum-lib/jingtumlib.h>
```

```
#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_buildRelationTx(std::string, std::string, jingtumlib::Remot
e *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_buildRelationTx);

}

void callback_buildRelationTx(std::string ex, std::string msg, jingtumlib::
Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else{

		std::string options = "{\"account\":\"jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ\",\"target\":\"jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c\",\"limit\
":{\"value\":\"0.01\",\"currency\":\"CCA\",\"issuer\":\"js7M6x28mYDiZVJJt
fJ84ydrv2PthY9W9u\"},\"type\":\"authorize\"}";

		jingtumlib::Transaction tx = remote->buildRelationTx(option
s);

		tx.setSecret("sn37nYrQ6KPJvTFmaBYokS3FjXUWd");

		tx.submit(callback_print);

	}

}

void callback_print(std::string ex, std::string msg) {

	if (ex != "") {

		std::cout << ex << std::endl;
```

```
        }
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

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 (options)

参数：

参数	类型	说明
type	String	属性种类
account	String	设置属性的源账号

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()
参数：无

设置属性完整例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_buildAccountSetTx(std::string, std::string, jingtumlib::Rem
ote *);

void callback_print(std::string, std::string);
```

```
int main(){
	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";
	jingtumlib::Remote remote(serverInfo);
	remote.connect(callback_buildAccountSetTx);
}
void callback_buildAccountSetTx(std::string ex, std::string msg, jingtumli
b::Remote *remote) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else{
		std::string options = "{\"account\":\"jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ\",\"type\":\"property\"}";
		jingtumlib::Transaction tx = remote->buildAccountSetTx(optio
ns);
		tx.setSecret("sn37nYrQ6KPJvTFmaBYokS3FjXUWd");
		tx.submit(callback_print);
	}
}
void callback_print(std::string ex, std::string msg) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		std::cout << "result:" << msg << std::endl;
	}
}
```

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(options)

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	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
taker_pays	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

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()
参数：无

挂单完整例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_buildOfferCreateTx(std::string, std::string, jingtumlib::Re
mote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_buildOfferCreateTx);

}

void callback_buildOfferCreateTx(std::string ex, std::string msg, jingtuml
ib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else{

		std::string options = "{\"type\":\"Sell\",\"account\":\"jB7r
xgh43ncbTX4WeMoeadiGMfmfqY2xLZ\",\"taker_gets\":{\"value\":\"0.01\",\"cur
rency\":\"CNY\",\"issuer\":\"jBciDE8Q3uJjf111VeiUNM775AMKHEbBLS\"},\"take
r_pays\":{\"value\":\"1\",\"currency\":\"SWT\",\"issuer\":\"\"}}";

		jingtumlib::Transaction tx = remote->buildOfferCreateTx(opti
ons);

		tx.setSecret("sn37nYrQ6KPJvTFmaBYokS3FjXUWd");

		tx.submit(callback_print);
```

```
        }
}
void callback_print(std::string ex, std::string msg) {
        if (ex != "") {
                std::cout << ex << std::endl;
        }
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

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 (options)

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()
参数：无

取消挂单完整例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_buildOfferCancelTx(std::string, std::string, jingtumlib::Re
mote *);

void callback_print(std::string, std::string);

int main(){
```

```
	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";
	jingtumlib::Remote remote(serverInfo);
	remote.connect(callback_buildOfferCancelTx);
}
void callback_buildOfferCancelTx(std::string ex, std::string msg, jingtuml
ib::Remote *remote) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else{
		std::string options = "{\"account\":\"jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ\",\"sequence\":\"2133\"}";
		jingtumlib::Transaction tx = remote->buildOfferCancelTx(opti
ons);
		tx.setSecret("sn37nYrQ6KPJvTFmaBYokS3FjXUWd");
		tx.submit(callback_print);
	}
}
void callback_print(std::string ex, std::string msg) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else {
		std::cout << "result:" << msg << std::endl;
	}
}
```

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 方法部署合约。

4.19.1 创建部署合约对象

方法：remote.deployContractTx (options)

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

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	井通钱包私钥

4.19.3 部署合约

方法：tx.submit()
参数：无

部署合约完整例子：

```
#include <jingtum-lib/jingtumlib.h>
```

```
#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_deployContractTx(std::string, std::string, jingtumlib::Remo
te *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_deployContractTx);

}

void callback_deployContractTx(std::string ex, std::string msg, jingtumli
b::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else{

		std::string options = "{\"account\":\"jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ\",\"amount\":10,\"payload\":\"result={}; function Init(t) re
sult=scGetAccountBalance(t) return result end; function foo(t) result=scGe
tAccountBalance(t) return end\",\"params\":[\"jB7rxgh43ncbTX4WeMoeadiGMfm
fqY2xLZ\"]}";

		jingtumlib::Transaction tx = remote->deployContractTx(option
s);

		tx.setSecret("sn37nYrQ6KPJvTFmaBYokS3FjXUWd");

		tx.submit(callback_print);

	}

}

void callback_print(std::string ex, std::string msg) {

	if (ex != "") {
```

```
                std::cout << ex << std::endl;
        }
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

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 方法部署合约。

4.20.1 创建部署合约对象

方法：remote.callContractTx (options)

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	井通钱包私钥

4.20.3 执行合约

方法：tx.submit()
参数：无

部署合约完整例子：

```
#include <jingtum-lib/jingtumlib.h>
```

```
#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_callContractTx(std::string, std::string, jingtumlib::Remote
*);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_callContractTx);

}

void callback_callContractTx(std::string ex, std::string msg, jingtumlib::
Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else{

		std::string options = "{\"account\":\"jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ\",\"destination\":\"j4YVQxCxaRRQ6gCVUvi9MoiTfWyPRnHwej\",\"f
oo\":\"foo\",\"params\":[\"jB7rxgh43ncbTX4WeMoeadiGMfmfqY2xLZ\"]}";

		jingtumlib::Transaction tx = remote->callContractTx(option
s);

		tx.setSecret("sn37nYrQ6KPJvTFmaBYokS3FjXUWd");

		tx.submit(callback_print);

	}

}

void callback_print(std::string ex, std::string msg) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}
```

```
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

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(option,callback);`

参数：

参数	类型	说明
option	String	“transactions”/”ledger_closed”
callback		回调函数

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_print(std::string, std::string, jingtumlib::Remote *);

void callback_onLedgerClosed(std::string, std::string, jingtumlib::Remote
*);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_onLedgerClosed);

}

void callback_onLedgerClosed(std::string ex, std::string msg, jingtumlib::
Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;

	}

	else {
```

```
            remote->on("ledger_closed", callback_print);
        }
}
void callback_print(std::string msg) {
        std::cout << "msg:" << msg << std::endl;
}
```

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。

*selectLedger (ledger)

* submit()

5.1 指定账本

方法：select_ledger (ledger)

参数：

参数	类型	说明
ledger	String	账本高度或者账号hash

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib")

void callback_requestAccountInfo(std::string, std::string, jingtumlib::Remote *);

void callback_print(std::string, std::string);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_requestAccountInfo);

}

void callback_requestAccountInfo(std::string ex, std::string msg, jingtumlib::Remote *remote) {

	if (ex != "") {

		std::cout << ex << std::endl;
```

```
        }
        else {
                jingtumlib::Request req = remote->requestAccountInfo("{\"acc
ount\":\"jDUjqoDZLhzx4DCf6pvSivjkjgtRESY62c\"}");
                req.selectLedger("83234498");
                req.submit(callback_print);
        }
}
void callback_print(std::string ex, std::string msg) {
        if (ex != "") {
                std::cout << ex << std::endl;
        }
        else {
                std::cout << "result:" << msg << std::endl;
        }
}
```

5.2 提交请求

方法：submit()

参数：请求参数，返回结果包含错误信息和结果信息

例子见前面所示。

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)

* submit()

6.1 获得交易账号

方法：get_account()

参数：无

返回：账号

例子：

```
#include <jingtum-lib/jingtumlib.h>

#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")

void callback_buildOfferCancelTx(std::string, std::string, jingtumlib::Re
mote *);

int main(){

	std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";

	jingtumlib::Remote remote(serverInfo);

	remote.connect(callback_buildOfferCancelTx);

}

void callback_buildOfferCancelTx(std::string ex, std::string msg, jingtuml
ib::Remote *remote) {

	if (ex != "") {
```

```
                std::cout << ex << std::endl;
        }
        else{
                std::string options = "{\"account\":\"jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ\",\"sequence\":\"2133\"}";
                jingtumlib::Transaction tx = remote->buildOfferCancelTx(opti
ons);
                std:string account = tx.getAccount();
                std:cout << “Account:” << account << std::endl;
        }
}
```

6.2 获得交易类型

方法：getTransactionType（）

参数：无

返回：交易类型

例子：

```
#include <jingtum-lib/jingtumlib.h>
#pragma comment(lib,"../../../lib/jingtumlib/jingtumStaticLib-release.lib
")
void callback_buildOfferCancelTx(std::string, std::string, jingtumlib::Re
mote *);
int main(){
        std::string serverInfo = "{\"server\":\"ws://123.57.219.57:5020\",
\"local_sign\":\"true\"}";
        jingtumlib::Remote remote(serverInfo);
        remote.connect(callback_buildOfferCancelTx);
}
```

```
void callback_buildOfferCancelTx(std::string ex, std::string msg, jingtuml
ib::Remote *remote) {
	if (ex != "") {
		std::cout << ex << std::endl;
	}
	else{
		std::string options = "{\"account\":\"jB7rxgh43ncbTX4WeMoead
iGMfmfqY2xLZ\",\"sequence\":\"2133\"}";
		jingtumlib::Transaction tx = remote->buildOfferCancelTx(opti
ons);
		std:string type = tx.getTransactionType();
		std:cout << “TransactionType:” << type << std::endl;
	}
}
```

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 >= 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.