# 恒生电子股份有限公司

## 证券投资者极速 API

## 开发指南

恒生电子

2024 年 7 月

# 说 明

# 目录

# 1 接口约定

## 1.1 异步调用

- 所有接口均为异步接口。
- 调用异步请求接口返回 0，仅代表请求发送成功，但并不表示服务器收到该笔请求。
- 继承 CHSSecuTradeSpi 接口，并实现回调函数，接受异步响应。

## 1.2 非线程安全性

- API & SPI 均非线程安全，禁止多线程调用相同 API & SPI 实例，如并发对同一个 API 实例调用委托下单接口等。

## 1.3 错误码

- API 的错误码存放 API 的请求接口的返回值中，可通过 GetApiErrorMsg 接口获取具体错误信息。
- 核心的错误信息存放在应答接口的 RspInfo 中。
- 交易所的错误码存放在委托主推接口 OnRtnOrder 的废单错误代码 ErrorNo 中。

# 2 快速上手

本章将以 C++版本的证券极速交易 API 为例编写一个 Linux 环境的证券竞价报单程序，目的在于帮助开发者快速了解证券极速交易 API 的使用方法。

## 2.1 准备工作

（1）准备一个安装 GCC4.8.5 或更高版本的 Linux 环境。

（2）经纪公司提供的测试账号、账号密码、IP 地址、端口、客户端 ID（AppID）、认证码（AuthCode）。

（3）如果经纪公司提供的是 Fens 地址，地址格式规范为：fens://[server-address]:[server-port]，例如：fens://192.168.1.1:1001。如果提供的是前置地址，地址格式规范为：tcp://[server-address]:[server-port] ，比如：tcp://192.168.1.1:1001。

## 2.2 开发包目录

从恒生 NanoExpress（www.nanoexpress.com）下载或者联系恒生客服获取恒生证券极速 API 开发包。恒生证券极速 API 开发包有 Windows、Linux 两种系统版本，支持信创 ARM 平台，信创 Arm 版本程序项则都放置在 09Kunpeng 目录下。

以 Linux 环境下 C++版本的非信创开发包为例。

| 目录 | 文件 | 说明 |
|---|---|---|
| \03API\c++\include | | API 头文件目录 |
| | HSDataType.h | API 数据类型头文件 |
| | HSSecuStruct.h | API 输入输出结构体头文件 |
| | HSSecuTradeApi.h | API 交易接口头文件 |
| | HsTradeApi_Utility.h | API 工具类接口头文件 |
| \03API\c++\lib\linux.x64 | | API linux.x64 动态库目录 |
| | libHSSecuTradeApi.so | API 动态库 so |
| | libldptcpsdk.so | t2sdk 动态库 so（用于读取 Demo 的 ini 文件） |
| | libt2sdk.so | ldptcpsdk 动态库 so（API 动态库依赖 ldptcpsdk） |

## 2.3 接入测试

首先可以使用附录 1 中的接入测试工具进行接入性测试，主要测试经纪公司提供的 IP 地址端口、账号密码等是否可用。如果已经对恒生的系统比较熟悉，则可以跳过该步骤，直接进入下一步。

## 2.4 简单介绍

CHSSecuTradeApi 采用异步调用方式，请求函数定义在 CHSSecuTradeApi 类，回调函数定义在 CHSSecuTradeSpi 类中。请求线程和回调线程不在同一线程，建议定义原子变量传递数据和状态，API 一般调用流程如图 2-4 所示，其中“→”表示请求线程，“←”表示回调线程。

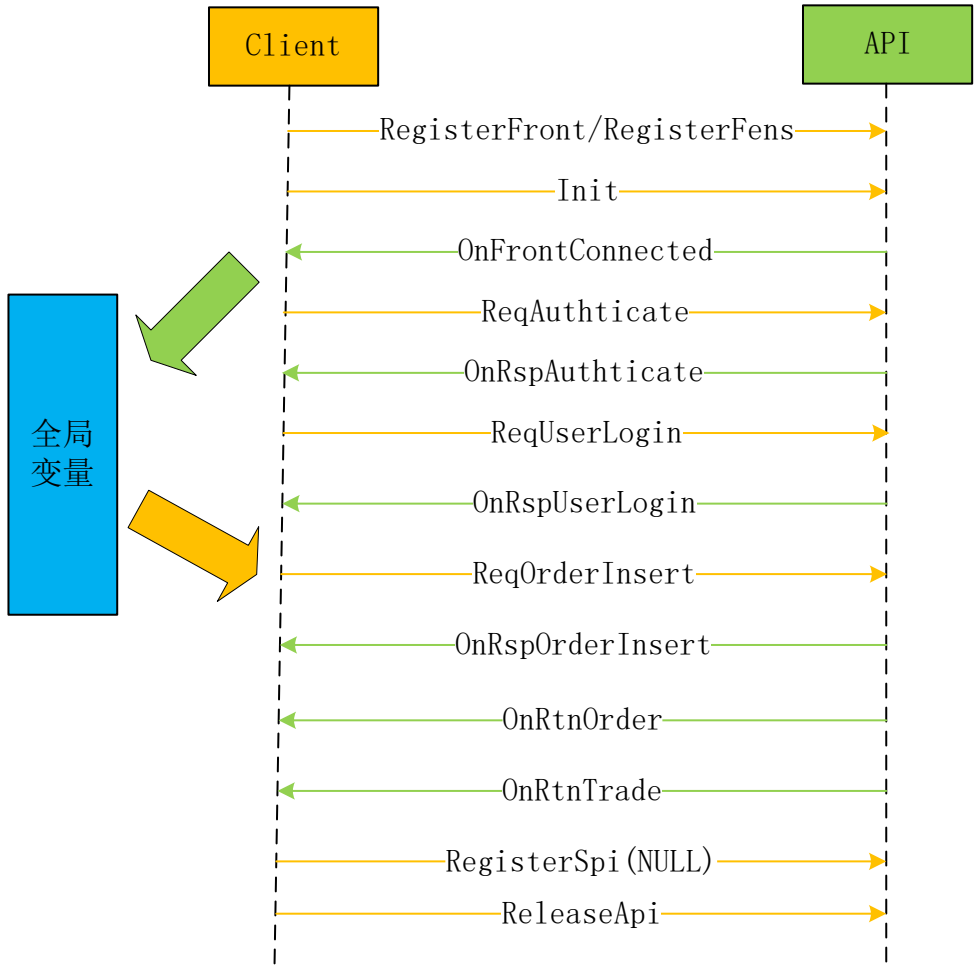



图 2-4

## 2.5 编辑代码

取发布包\03API\c++\demo\src 文件夹放到 Linux 服务器上，src 文件夹目录如下所示。

```
src
-- include
    -- HSDataType.h
    -- HSSecuApi_Utility.h
    -- HSSecuStruct.h
    -- HSSecuTradeApi.h
-- lib
    -- linux.x64
        -- libHSSecuTradeApi.so
        -- libldptcpsdk.so
-- LinuxDemo
    -- Demo.cpp
    -- Demo.x64.gcc
```

根据经纪公司提供的测试账号等信息，修改 Demo.cpp 中的变量值。资金账号、IP 地址端口、客户端编码、认证码、密码、委托方式需要由经纪公司提供，投资者端应用信息和站点信息则有客户填入自己的机器信息。如果是 Fens 接入，则修改 Fens 地址，同时 bIsFens 改为 true。如果是前置接入，则修改前置地址，同时 bIsFens 改为 false。

```
HSAccountID g_FundAccount = "38000200";          // 资金账号
std::string g_FrontAddress = "tcp://10.20.23.131:30572";  // 前置地址
std::string g_FensAddress = "fens://10.20.23.131:33336";  // Fens地址
bool bIsFens = false; // 是否通过fens接入
HSAppID g_AppID = "11";  // 客户端编码
HSAuthCode g_AuthCode = "123456";  // 认证码
HSPassword g_Password = "111111";  // 密码
HSUserApplicationType g_UserApplicationType = '7'; //投资者端应用类别
HSUserApplicationInfo g_UserApplicationInfo = "demo";  // 投资者端应用信息
HSUserStationInfo g_UserStationInfo = "127.0.0.1";  // 站点信息
```

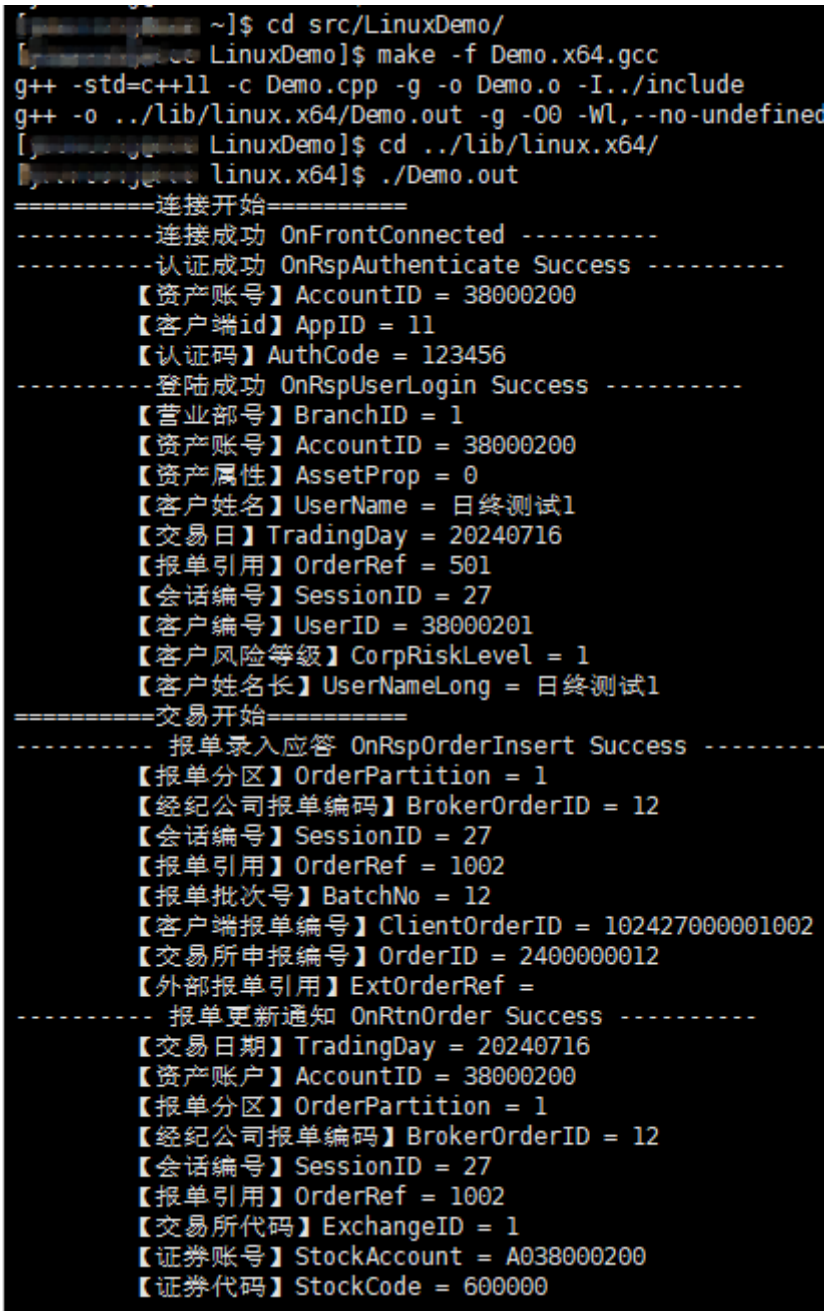
## 2.6 编译运行

切换到 src/LinuxDemo 目录，执行 make –f Demo.x64.gcc 编译代码。编译完成后会在 linux.x64 生成 Demo.out 可执行文件。Demo 源代码及 Makefile 文件见附录 2。

编译完成后，切换到 src/lib/linux.x64 目录，执行以下命令运行。

```
export LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH
./Demo.out
```

运行界面如下：


~]$ cd src/LinuxDemo/
LinuxDemo]$ make -f Demo.x64.gcc
g++ -std=c++11 -c Demo.cpp -g -o Demo.o -I../include
g++ -o ../lib/linux.x64/Demo.out -g -O0 -Wl,--no-undefined
LinuxDemo]$ cd ../lib/linux.x64/
linux.x64]$ ./Demo.out
==========连接开始==========
----------连接成功 OnFrontConnected ----------
----------认证成功 OnRspAuthenticate Success ----------
【资产账号】AccountID = 38000200
【客户端id】AppID = 11
【认证码】AuthCode = 123456
----------登陆成功 OnRspUserLogin Success ----------
【营业部号】BranchID = 1
【资产账号】AccountID = 38000200
【资产属性】AssetProp = 0
【客户姓名】UserName = 日终测试1
【交易日】TradingDay = 20240716
【报单引用】OrderRef = 501
【会话编号】SessionID = 27
【客户编号】UserID = 38000201
【客户风险等级】CorpRiskLevel = 1
【客户姓名长】UserNameLong = 日终测试1
==========交易开始==========
---------- 报单录入应答 OnRspOrderInsert Success ---------
【报单分区】OrderPartition = 1
【经纪公司报单编码】BrokerOrderID = 12
【会话编号】SessionID = 27
【报单引用】OrderRef = 1002
【报单批次号】BatchNo = 12
【客户端报单编号】ClientOrderID = 102427000001002
【交易所申报编号】OrderID = 2400000012
【外部报单引用】ExtOrderRef =
---------- 报单更新通知 OnRtnOrder Success ----------
【交易日期】TradingDay = 20240716
【资产账户】AccountID = 38000200
【报单分区】OrderPartition = 1
【经纪公司报单编码】BrokerOrderID = 12
【会话编号】SessionID = 27
【报单引用】OrderRef = 1002
【交易所代码】ExchangeID = 1
【证券账号】StockAccount = A038000200
【证券代码】StockCode = 600000

# 3 通用规则

## 3.1 命名规则

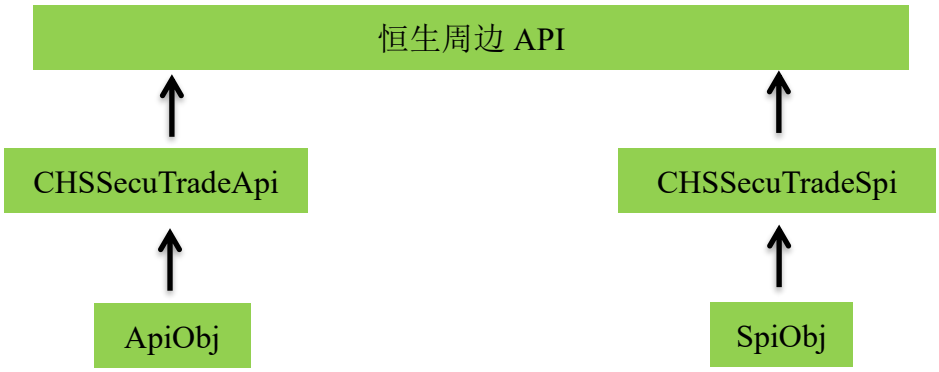
| 消息 | 格式 | 示例 |
| --- | --- | --- |
| 请求 | Req----- | ReqUserLogin |
| 请求应答 | OnRsp----- | OnRspUserLogin |
| 查询 | ReqQry----- | ReqQryTrade |
| 查询应答 | OnRspQry----- | OnRspQryTrade |
| 回报 | OnRtn----- | OnRtnTrade |

## 3.2 接口类

若未特殊说明，后续涉及到的地方都和下面的声明一致。

API 指 CHSSecuTradeApi，包含主动发起请求和订阅的接口函数，开发者直接调用即可。

SPI 指 CHSSecuTradeSpi，包含所有的应答和主推函数，用于接收恒生交易系统发送或交易所发送恒生交易系统转发的信息，开发者需要继承该接口类，并实现其中相应的虚函数。




## 3.3 通用参数

**nRequestID** 客户端发送请求时要为该请求指定一个请求编号。交易接口会在响应或回报中返回与该请求相同的请求编号。当客户端进行频繁操作时，很有可能会造成同一个响应函数被调用多次，这种情况下，能将请求与响应关联起来的纽带就是请求编号。

**bIsLast** 当响应函数需要携带的数据包过大时，该数据包会被分割成数个小的数据包并按顺序逐次发送，这种情况下同一个响应函数就是被调用多次，而参数 bIsLast 就是用于描述当前收到的响应数据包是不是所有数据包中的最后一个。

**RspInfo** 该参数用于描述请求执行过程中是否出现错误。该数据结构中的属性 ErrorID

如果是 0，则说明该请求被交易核心认可通过。否则，该参数描述了交易核心返回的错误信息。

**HSSecuTradeError.xml** 文件中包含 API 所有可能的错误信息。

# 4 开发指南

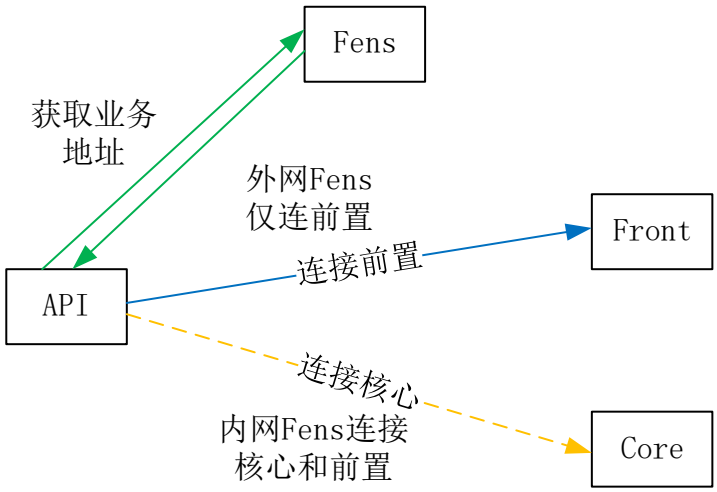
## 4.1 总体流程图

## 4.2 两种连接模式

### 4.2.1 Fens 模式

Fens 是指提供交易业务地址的 Fens 服务器，Fens 模式是指通过 Fens 地址获取交易业务地址从而接入到 LDP 柜台，Fens 地址格式规范为：fens://[server-address]:[server-port]，例如：fens://192.168.1.1:1001，如果有多个 Fens 服务器，可以通过调用多次 RegisterFensServer 设置多个 Fens 服务器地址，最多可设置 4 个不同的 Fens 地址，超过则调用 RegisterFensServer 接口返回错误码-1503。

Fens 地址区分内外网，内网 Fens 一般提供给内部局域网客户，通常会返回核心和前置两个业务地址，从而 API 会建立两个业务连接。交易和常用查询（如查询持仓、资金、委托、成交等）则会通过与核心的连接进行，资金调拨（如银证转账、集中交易资金调拨、多中心资金调拨等）或不常用查询（如查询中签信息、配号信息等）则会通过与前置的连接进行。外网 Fens 一般提供给互联网客户，通常只返回前置地址，从而 API 仅建立与前置的一个连接，所有的业务都通过与前置的连接进行。




### 4.2.2 直连模式

直连模式是指直接通过业务地址接入 LDP 柜台。通过调用 RegisterFront 函数传入需要接入的前置或核心地址进行接入，地址格式规范为：tcp://[server-address]:[server-port] ，比如：tcp://192.168.1.1:1001，如果有多个业务地址，可以调用多次，最多可设置 4 个不同的业务地址，超过则调用 RegisterFront 接口返回错误码-1511。

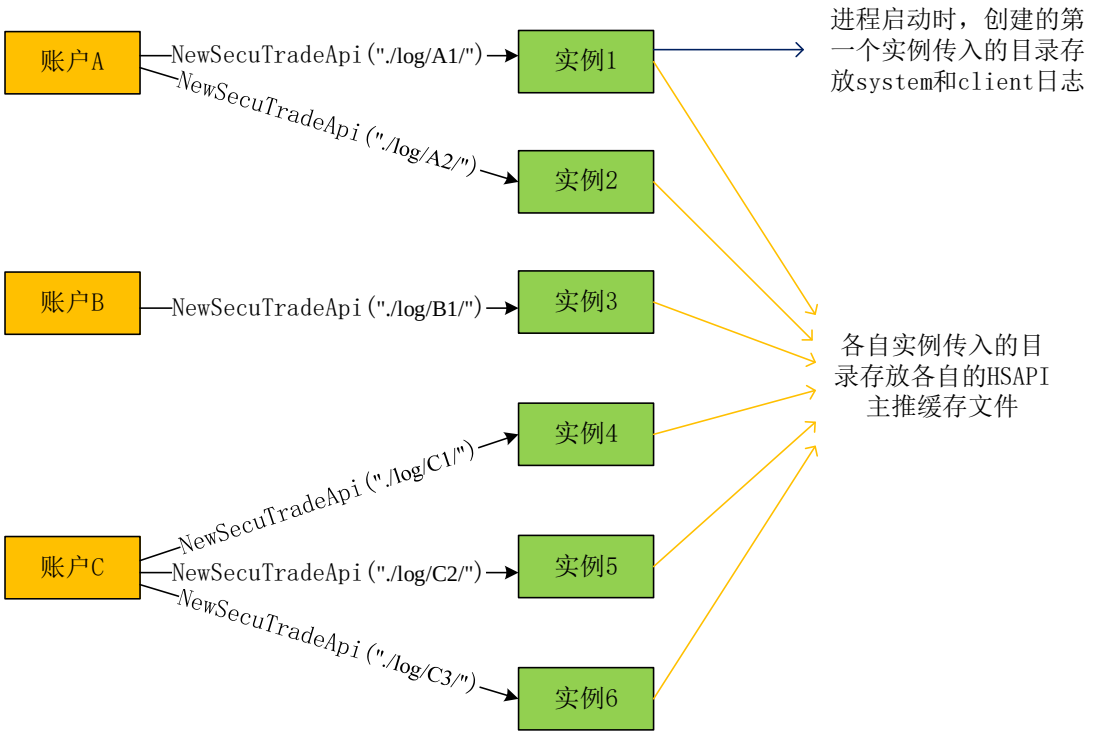
当通过 RegisterFront 接口传入多个可用的业务地址，其中一个业务服务器不能工作，API 会随机寻找一个业务地址进行重连，直到找到一个可用的业务地址，缺点是只能在前期传入的固定的业务地址中寻找可用的业务地址，不能像 fens 那样寻找动态更新的可用业务地址，且不能实现负载均衡。

## 4.3 接口初始化

初始化流程如下:

1. API 通过 NewSecuTradeApi 接口创建 API 实例，入参需要填写缓存目录。工作区路径会存放 API 缓存信息、日志等。

```
/*一个 API 实例只能给一个资产账户使用，如果同一个资产账户创建了多个 API 实例，那么每个实例传入的文件存放目录必须不同(HSAPI 主推缓存文件存放路径由 NewSecuTradeApi 接口传入）。同一个资产账号最多只允许创建 31 个 API 实例，为避免网关闪断等因素导致连接会话被占用，建议 API 实例不要超过 28 个。*/
CHSSecuTradeApi* lpTradeApi = NewSecuTradeApi("./log");
```




2. 创建继承自 SPI 的回调对象实例。

```
DemoSpi* lpDemoSpi = new(std::nothrow) DemoSpi();
```

DemoSpi 实现如下：

```
// 回调类，继承自CHSSecuTradeSpi，可以只实现部分回调函数
class DemoSpi : public CHSSecuTradeSpi
{

    void OnFrontConnected();
```

```
    void OnFrontDisConnected(int nResult);

    void OnRspXXXX(CHSSecuRspXXXXField *pRspXXXX, CHSSecuRspInfoField *pRspInfo,
 int nRequestID, bool bIsLast);
}
```

3. 调用 API 函数 Register 注册 SPI 实例，用于接收 API 的回调消息。

```
lpTradeApi->RegisterSpi(lpDemoSpi);
```

4. 调用 RegisterFensServer 函数设置 Fens 服务器地址，如果不通过 Fens 方式接入，而是通过前置直连接入，那么需要调用 RegisterFront 函数设置前置服务器地址。调用该接口不会跟后台有任何交互，只是 API 本地缓存一下连接地址，后续发起连接时生效。

```
if (bIsFens)
{
    iRet = lpTradeApi->RegisterFensServer("Fens地址", "资产账户");
}
else
{
    iRet = lpTradeApi->RegisterFront("前置地址");
}
```

5. 调用 RegisterSubModel 函数设置回报订阅模式，主要是用于接收历史回报或指定序号开始的回报，如果不需要进行设置，可以跳过该步骤，默认从登录之后最新的回报开始接收。

```
if (SubTertType == 1)
{
    lpTradeApi->RegisterSubModel(HS_TERT_RESUME);//从上次收到的续传
}
else if (SubTertType == 2)
{
    lpTradeApi->RegisterSubModel(HS_TERT_QUICK);//从登录后最新的开始传
}
else if (SubTertType == 3)
{
    lpTradeApi->RegisterSubModel(HS_TERT_DESIGNATED, StartSequenceNumber);//从指定主推序号开始续传
}
else if (SubTertType == 4)
{
    lpTradeApi->RegisterSubModel(HS_TERT_REFUSE);//不接收主推消息
```

```
}
else
{
    lpTradeApi->RegisterSubModel(HS_TERT_RESTART);//从本交易日开始重传
}
```

6. 调用 Init 初始化函数，用于 API 和服务器建立连接。

```
iRet = lpTradeApi->Init("");//对接 ldp 目前不需要填通讯证书
```

7. 等待 OnFrontConnected 响应，表示连接成功，如果连接失败，通过 OnFrontDisconnected 返回错误码(nResult)，详细错误信息可以通过错误码调用 GetApiErrorMsg 函数获取。连接失败时，API 内部会一直尝试重连直至成功。

```
// 连接成功
void OnFrontConnected()
{
    std::cout << "连接成功 OnFrontConnected" << std::endl;
    m_bConnectedFlag.store(true);
};
// 连接失败
void OnFrontDisConnected(int nResult)
{
    std::cout << "连接失败 OnFrontDisConnected ErrorMsg: " <<
lpTradeApi->GetApiErrorMsg(nResult) << std::endl;
    m_bConnectedFlag.store(false);
    m_bLoginedFlag.store(false);
};
```

8. 连接成功后，调用 ReqAuthenticate 接入注册。

```
if (lpDemoSpi->IsConnected())
{
    iRet = lpTradeApi->ReqAuthenticate(&stReqAuthenticate, req_no++);
    if (0 != iRet)
    {
        std::cout << "ReqAuthenticate Failed ErrorID:" << iRet << " ErrorMsg: "
<< lpTradeApi->GetApiErrorMsg(iRet) << std::endl;
    }
}
```

9. 如需绑定会话登录，调用 BindSessionId 设置登录会话号，0 为不绑定，具体可绑定范围需要咨询经纪公司。不需要则跳过该步骤。

```
int iBindSessionId = atoi(SessionID.c_str());
```

```
if (iBindSessionId < 0)
{
    iBindSessionId = 0;
}
lpTradeApi->BindSessionId(iBindSessionId);
```

10. 发起登录，密码错误需要更换密码重新发起登录。

```
if (lpDemoSpi->IsAuthed())
{
    lpTradeApi->ReqUserLogin(&stReqUserLogin, req_no++);
    if (0 != iRet)
    {
        std::cout << "ReqUserLogin Failed ErrorID:" << iRet << " ErrorMsg: " <<
lpTradeApi->GetApiErrorMsg(iRet) << std::endl;
    }
}
```

11. 查询资金、持仓等。

```
iRet = lpTradeApi->ReqQryFund(&stReqQryFund, req_no++);
if (0 != iRet)
{
    std::cout << "ReqQryFund Failed ErrorID:" << iRet << " ErrorMsg: " <<
lpTradeApi->GetApiErrorMsg(iRet) << std::endl;
}
iRet = lpTradeApi->ReqQryHold(&stReqQryHold, req_no++);
if (0 != iRet)
{
    std::cout << "ReqQryHold Failed ErrorID:" << iRet << " ErrorMsg: " <<
lpTradeApi->GetApiErrorMsg(iRet) << std::endl;
}
```

将持仓和委托数据进行转存。

```
void OnRspQryHold(CHSSecuRspQryHoldField *pRspQryHold, CHSSecuRspInfoField
*pRspInfo, int nRequestID, bool bIsLast)
{
    if (pRspInfo->ErrorID != 0)
    {
        std::cout << "查询失败: "<< pRspInfo->ErrorMsg << std::endl;
    }
    else if (pRspQryHold)
    {
        CHSSecuRspQryHoldField temHold;
```

```
            // 回调函数中的数据是栈上的，需要进行memcpy
            memcpy(&temHold, pRspQryHold, sizeof(CHSSecuRspQryHoldField));
            m_vecHold.push_back(temHold);
        }
        if(bIsLast == true)
        {
            std::cout << "查询完毕" << std::endl;
        }
};
void OnRspQryFund(CHSSecuRspQryFundField *pRspQryFund, CHSSecuRspInfoField
*pRspInfo, int nRequestID, bool bIsLast)
{
    if (pRspInfo->ErrorID != 0)
    {
        std::cout << "查询失败: " << pRspInfo->ErrorMsg << std::endl;
    }
    else if (pRspQryFund)
    {
        m_dAvailableBalance = pRspQryFund->AvailableBalance; // 可用
        m_dFetchBalance = pRspQryFund->FetchBalance; // 可取
        m_dTotalAsset = pRspQryFund->TotalAsset; //总资产
    }
    if (bIsLast == true)
    {
        std::cout << "查询完毕" << std::endl;
    }
};
```

12. 发起下单。

```
iRet = lpTradeApi->ReqOrderInsert(&stReqOrderInsert, 3);
if (0 != iRet)
{
    std::cout << "ReqOrderInsert Failed ErrorID:" << iRet << " ErrorMsg: " <<
lpTradeApi->GetApiErrorMsg(iRet) << std::endl;
}
else
{
    LDPOrderField Order;
    //进行赋值操作
    Order.XXXX = stReqOrderInsert.XXXX;
    m_OrderMap[stReqOrderInsert.ClientOrderID] = Order;
}
```

13. 收取应答和主推

```
void OnRspOrderInsert(CHSSecuOrderField *pRtnOrder)
{
    if (pRspInfo->ErrorID != 0)
    {
        m_OrderMap.erase(pRspOrderInsert->ClientOrderID);
        std::cout << "报单录入应答 OnRspOrderInsert Error ErrorID =  " <<
pRspInfo->ErrorID << "ErrorMsg = " << pRspInfo->ErrorMsg << std::endl;
    }
    else
    {
        if (pRspOrderInsert)
        {
            auto it = m_OrderMap.find(pRtnOrder->ClientOrderID);
            if (it != m_OrderMap.end())
            {
                // 更新订单数据，主要是订单委托号，报单引用等
                it->second.XXXX = pRtnOrder->XXXX;
            }

        }

    }
}
void OnRtnOrder(CHSSecuOrderField *pRtnOrder)
{
    if (pRtnOrder)
    {
        std::map<int64_t, LDPOrderField> m_OrderMap;
        auto it = m_OrderMap.find(pRtnOrder->ClientOrderID);
        if (it != m_OrderMap.end())
        {
            // 更新订单数据，主要是该笔订单成交的汇总数据，如平均成交金额、平均
成交价格
            it->second.XXXX = pRtnOrder->XXXX;
        }

    }

}
void OnRtnTrade(CHSSecuTradeField *pRtnTrade)
{
    if (pRtnTrade)
```

```
    {
        std::map<int64_t, LDPOrderField> m_OrderMap;
        auto it = m_OrderMap.find(pRtnTrade->ClientOrderID);
        if (it != m_OrderMap.end())
        {
            // 更新订单数据,主要是该笔订单成交的实时数据，如实时成交金额、实时成交价格
            it->second.XXXX = pRtnTrade->XXXX;
        }
    }
}
```

14. 释放 API 实例

```
lpTradeApi->RegisterSpi(NULL);
lpTradeApi->ReleaseApi();
```

## 4.4 代码模块

### 4.4.1 导出函数

第三方开发者通过 GetSecuTradeApiVersion 函数可以获取当前 API 的版本，通过 NewSecuTradeApi 创建一个 API 实例，入参为缓存目录。通过 NewApiUtility 可以创建一个工具类实例，入参 API 实例对象可选。

```
    /// Description: 删除接口对象本身
    virtual void ReleaseApi() = 0;

    /// Description: 初始化连接
    ///                 pszLicFile          通讯证书
    ///                 pszSafeLevel        安全级别
    ///                 pszPwd              通讯密码
    ///                 pszSslFile          SSL证书
    ///                 pszSslPwd           SSL密码
    virtual int Init(const char *pszLicFile, const char *pszSafeLevel = "", const char *pszPwd = "", const char *pszSslFile = "", const char *pszSslPwd = "") = 0;

    /// Description: API和交易柜台建立连接，连接建立成功后，主线程会等待用户操作子线程退出
```

```
    /// Return      : int 0表示连接建立成功，否则表示失败，通过调用GetApiErrorMsg可以获取详细错误信息
    virtual int Join() = 0;

    /// Description: 注册回调接口
    /// Input       : pSpi                派生自回调接口类的实例
    virtual void RegisterSpi(CHSSecuTradeSpi *pSpi) = 0;

    ///注册前置机网络地址
    ///@param pszFrontAddress：前置机网络地址。
    virtual int RegisterFront(const char *pszFrontAddress) = 0;

    /// Description: 注册订阅模式
    /// Input       : 订阅方式
    ///                     HS_TERT_RESTART://从本交易日开始重传
    ///                     HS_TERT_RESUME: //从上次收到的续传
    ///                     HS_TERT_QUICK:  //从登录后最新的开始传
    ///                     HS_TERT_DESIGNATED: //从指定序号开始续传，nSequenceNumber作为续传起始号
    ///                     HS_TERT_REFUSE:  //不接收主推消息
    virtual int RegisterSubModel(const char *pszSubModel, int nSequenceNumber = 0) = 0;

    /// Description: 获得错误详细信息
    virtual const char* GetApiErrorMsg(int nErrorCode) = 0;
```

---

4.4.2 SPI 实现

以下是 SPI 的函数,子类只需要继承该回调类，并重载其中的函数即可正常使用。

---

```
    /// Description: 当客户端与交易后台开始建立通信连接，连接成功后此方法被回调。
    virtual void OnFrontConnected(){};

    /// Description:当客户端与交易后台通信连接异常时，该方法被调用。
    /// Others      :通过GetApiErrorMsg(nResult)获取详细错误信息。
    virtual void OnFrontDisconnected(int nResult){};

    /// Description:客户登录
    virtual void OnRspUserLogin(CHSSecuRspUserLoginField *pRspUserLogin, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};
```

```
/// Description:委托录入
virtual void OnRspOrderInsert(CHSSecuRspOrderInsertField *pRspOrderInsert, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};

/// Description:委托撤单
virtual void OnRspOrderAction(CHSSecuRspOrderActionField *pRspOrderAction, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};

/// Description:持仓查询
virtual void OnRspQryHold(CHSSecuRspQryHoldField *pRspQryHold, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};

/// Description:资金查询
virtual void OnRspQryFund(CHSSecuRspQryFundField *pRspQryHold, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};

/// Description:委托查询
virtual void OnRspQryOrder(CHSSecuOrderField *pRspQryOrder, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};

/// Description:成交查询
virtual void OnRspQryTrade(CHSSecuTradeField *pRspQryTrade, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};

/// Description:新股申购额度查询
virtual void OnRspQryEquity(CHSSecuRspQryEquityField *pRspQryEquity, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {};

/// Description:证券代码查询
virtual void OnRspQryStkcode(CHSSecuRspQryStkcodeField *pRspQryStkcode, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast){};

/// Description:主推-委托回报
virtual void OnRtnOrder(CHSSecuOrderField *pRtnOrder) {};

/// Description:主推-成交回报
virtual void OnRtnTrade(CHSSecuTradeField *pRtnTrade) {};
```

# 5 功能介绍

本章节介绍证券极速交易 API 的特色功能。这些功能将针对投资者不同的需求场景进行灵活的配置，如：

1、如果您担心断线或重启期间回报会丢失，或者您的客户端并不总是在线，需要收取不在线期间的回报，那么您可以阅读 5.1 订阅模式和 5.2 登陆补漏的内容。

2、如果您交易量较大，同时网络负载能力较弱，担心连续交易期间会丢失回报，那么可以阅读 5.3 定时补漏和 5.4 主推有序的内容。

3、如果您需要进行性能测试，可以阅读 5.5Timelog 章节的内容。

4、如果您想要提升低频交易下 ETF 申赎和可转债买卖等 T+0 业务的性能，可参考 5.6 Nano 业务接口。

## 5.1 订阅模式

调用 RegisterSubModel 接口设置回报订阅模式，调用该接口不会跟后台有任何交互，只是 API 本地缓存一下订阅模式参数，该参数在后续调用 ReqUserLogin 客户登录接口时生效。订阅模式有 5 种（参见 HSDataType.h 中的 SUB_TERT_TYPE 枚举），当不调用 RegisterSubModel 函数或订阅模式参数传错时，订阅模式将采用默认值 HS_TERT_QUICK（2），这 5 种订阅模式说明如下：

1、HS_TERT_RESTART（0）：从本交易日开始重传，即登录时，API 会把当天所有主推信息通过报单回报 OnRtnOrder 和成交回报 OnRtnTrade 接口回调。

2、HS_TERT_RESUME（1）：从上次收到的续传，即登录时，API 会把当天未收到过的主推信息通过报单回报 OnRtnOrder 和成交回报 OnRtnTrade 接口回调。

3、HS_TERT_QUICK（2）：从登录后最新的开始传，即登录时，API 不回调当天历史的主推信息，只回调登录之后新报单的主推信息。

4、HS_TERT_DESIGNATED（3）：从指定序号开始续传，即登录时，API 会从指定的序号开始回调主推信息。

5、HS_TERT_REFUSE（4）：不接收主推消息。

## 5.2 登陆补漏

登陆补漏开关是范例-HSSecuTradeApi.ini 中的 enable_login_lsqry 参数控制的。登陆补漏主要是将账号当天的历史回报进行一个补漏，它是将从本地 HSAPI 缓存文件读取和去服务端查询两种方式结合获取该客户端的历史回报。这些历史回报会存储在 API 缓存和本地 HSAPI 文件中（如果开启 save_mode=0），根据订阅模式选择是否将历史回报推送给周边。登陆补漏会在登录成功回调前完成对历史回报的获取，并开启另外的线程将回报推送给周边。

注意，并不是开启了登陆补漏开关就一定会触发登陆补漏，如果以下四个条件任一满足则不会发起登录补漏查询：

（1）登录应答返回的主推序号为 0，即还未对核心发起报单。

（2）本地的 HSAPI 文件中的最大主推序号和登录应答返回的主推序号一致。
（3）订阅模式为从指定序号开始续传（HS_TERT_DESIGNATED），且指定主推序号大于登录应答返回的主推序号。
（4）关闭登陆补漏开关（enable_login_lsqry=0）。
当触发登陆补漏时，system 日志会打印“开始登录补漏查询”关键信息。

## 5.3 定时补漏

定时补漏是用于盘中补回该账号在其他客户端的回报，还用于保证主推有序补回丢失的回报，他与登陆补漏不同的是，登陆补漏发生在登录时，是一次性的，而定时补漏是伴随 API 整个生命周期。

定时补漏的查询间隔是范例-HSSecuTradeApi.ini 中的 query_time 参数控制的。该参数控制定时补漏查询多久发起一次，但如果主推有序的情况下发生乱序会立即触发定时补漏查询。当触发定时补漏查询时，system 日志会打印“定时补漏查询”关键信息。

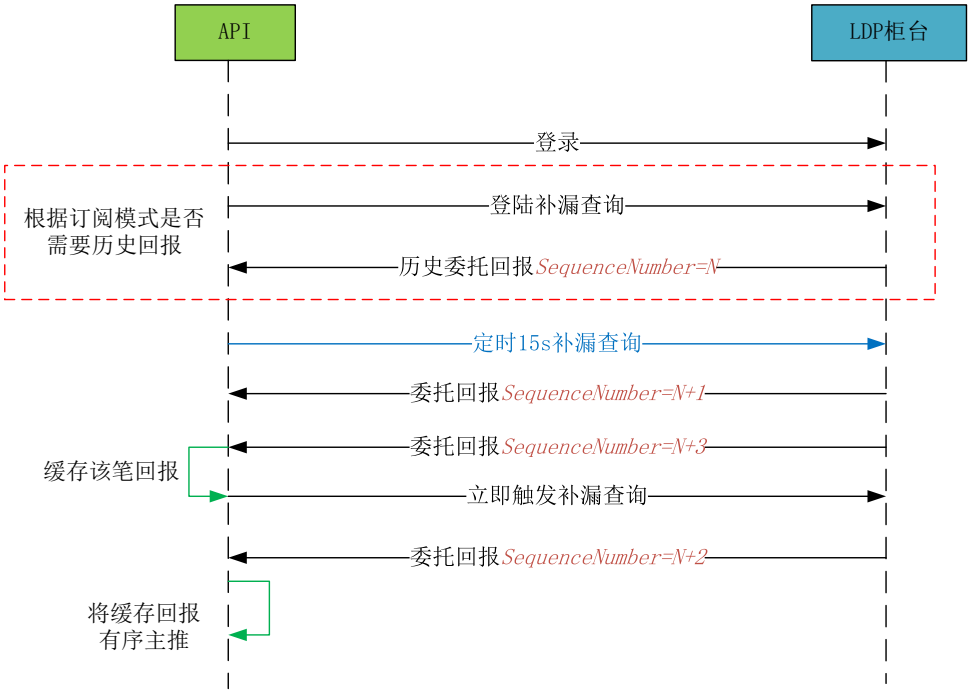
当存在大量的历史回报需要收取（如订阅模式 0-从本交易日开始重传、1-从上次收到的续传、3-从指定序号开始续传），如果关闭登录补漏或 HSAPI 文件损坏导致登录补漏失效时，拉取历史回报会完全依赖定时补漏，此时由于主推有序的原因，新下单的回报会回调的比较慢，需要等历史回报都收取完才会收到新下单的回报。如果没有新下单触发快速拉取历史回报，那么历史回报会按照 HSSecuTradeApi.ini 中设置的 query_time 时间间隔每次查询 500 条，如果本端登录之后有新下单，为了快速收到新下单的回报而又不至于查询频繁影响网络带宽，会触发 1 秒一次快速查询历史回报，直到查完后恢复按照 HSSecuTradeApi.ini 中设置的 query_time 时间间隔来定时查询。

例如有 1 万笔历史回报需要拉取，那么：
（1） 本端登录之后没有新下单，默认 query_time=15，每 15 秒查询 500 笔，需要 300 秒。
（2） 本端登录之后马上有新下单，1 秒一次查询 500 笔，需要 20 秒。

如果觉得拉取历史回报的时间太长，可以修改订阅模式为 2-从登录后最新的开始传，从而不拉取历史回报。也可以开启登录补漏，若是 HSAPI 文件损坏导致登录补漏失效的情况，可以先停止 API 进程，删除 HSAPI 文件，然后重启 API 进程，这样操作之后仍然报错“SQLite 损坏”，那么需要检查进程中一个账户的多个会话的缓存目录是否不同（NewSecuTradeApi 接口传入的缓存路径），需要进行调整保证账户的多个实例缓存目录不相同（参见 4.3 接口初始化）。

## 5.4 主推有序

主推有序开关是范例-HSSecuTradeApi.ini 中的 sort_mode 参数控制的。主推有序主要是将收到的回报按照 SequenceNumber 进行从小到大排序并主推给周边。当开启此开关时，客户不用担心回报乱序，如成交先于确认回来、网络问题导致回报丢失等情况。主推有序的实现逻辑是依赖于 SequenceNumber 有序，当发生 SequenceNumber 跳号等情况，会立即触发一次定时补漏查询。




## 5.5 TimeLog

TimeLog 是用于测试 API 的一个时延分析工具，需要联系经纪公司在后台管理端上开启，是根据对应的客户端 ID 进行控制的，点击对应客户端 ID 的接入控制信息，点击“修改”，勾选“是否启用全链路度量日志 TimeLog”即可开启。

开启之后，在 API 的 log 的目录下会生成.timelog 后缀的文件，里面会记录 API 的时延信息。

## 5.6 Nano 业务接口

Nano 业务接口是基于 udp 的通讯方式进行报单和撤单业务的，适用于低频交易场景的性能提升，核心穿透时延低于 2.5us，支持可转债买卖、ETF 基金跨市套利等 T+0 交易业务。如需开通此业务，请咨询经纪公司。

Nano 业务接口涉及 ReqNanoOrderInsert 和 ReqNanoOrderAction 两个接口，应答和回报仍通过原来的报单接口和主推接口返回，具体可参考接口手册关于这两个接口的说明。

此外，Nano 业务还有预热的功能，预热功能是为了使 Nano 报单通道始终处于一种“兴奋”状态，从而达到行情到来时能够极速报单的效果。

### 5.6.1 预热配置

取 API 发布包\03API\c++\config\范例-HSSecuTradeApi.ini，去掉“范例-”，然后放置于进程启动目录即可生效。

预热配置信息位于[preheat]节下，可配置需要预热的交易所、证券代码，可配置多个。

配置好需要预热的交易所类别和证券代码后，API 所在进程启动后会自动预热，无需投资者调用任何接口。

### 5.6.2 定时预热绑核 timer_cpuid

```
# 定时预热线程绑核 id，不配置则不绑定 CPU，配置错误也不绑定 CPU
timer_cpuid=1
```

定时预热线程的绑核 ID，如不配置，有操作系统调度。当配置的绑核 ID 无效时，也有操作系统调度。

### 5.6.3 定时预热证券代码信息 prefetch_info

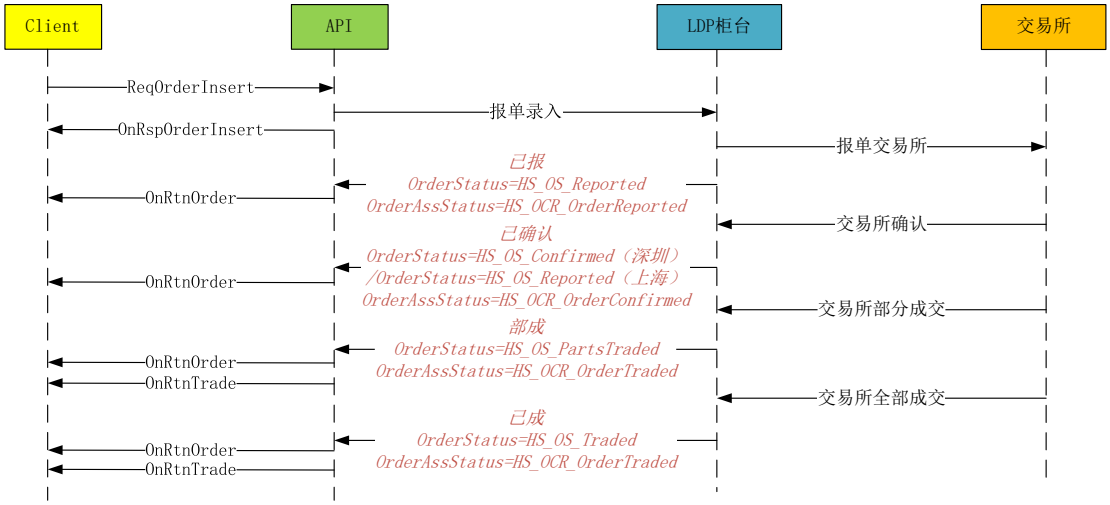
需要预热的证券代码信息长度总和不能超过 1024，数据格式为:

```
# 最大支持字符串长度为 1024，格式 exchange_id=1,stock=600570,000002|;exchange_id=2,stock=000001,000002|;
prefetch_info=exchange_id=1,stock=600001,600570|;exchange_id=2,stock=000001,000002|;
```

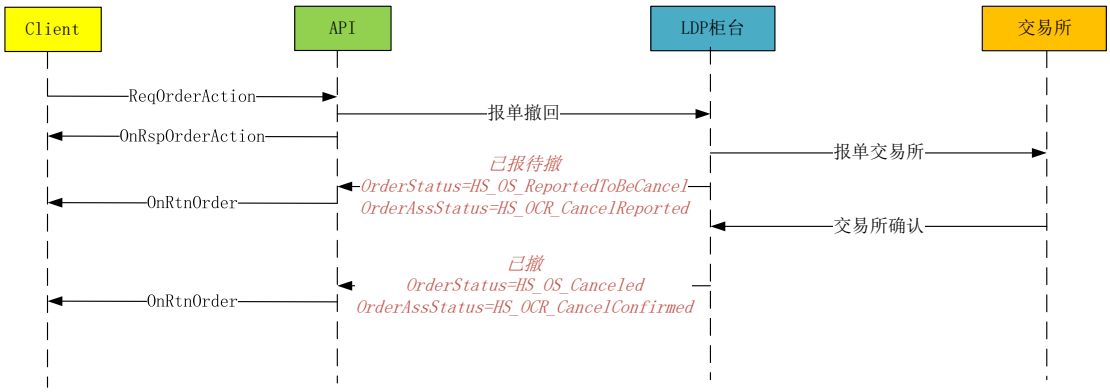
目前仅支持上海和深圳市场。当市场无效时，所有后续的合约均无效，当合约不符合正确的股票代码规则时，配置的预热信息记录也无效。

# 6 订单状态变化模型

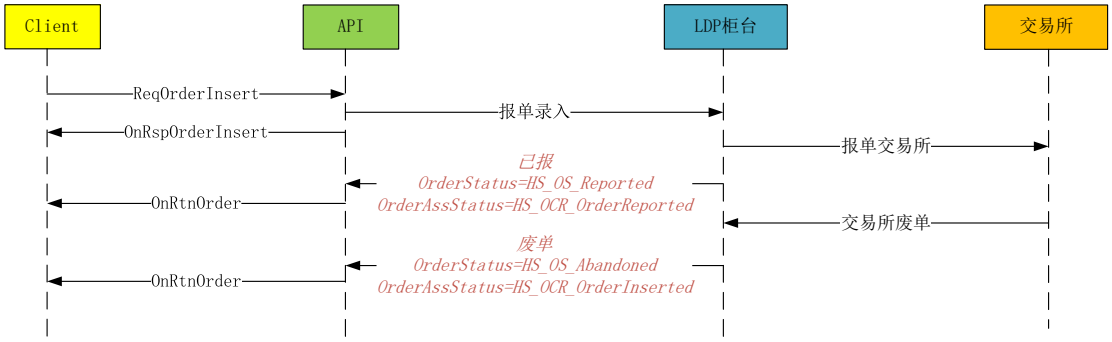
## 6.1 报单成交场景



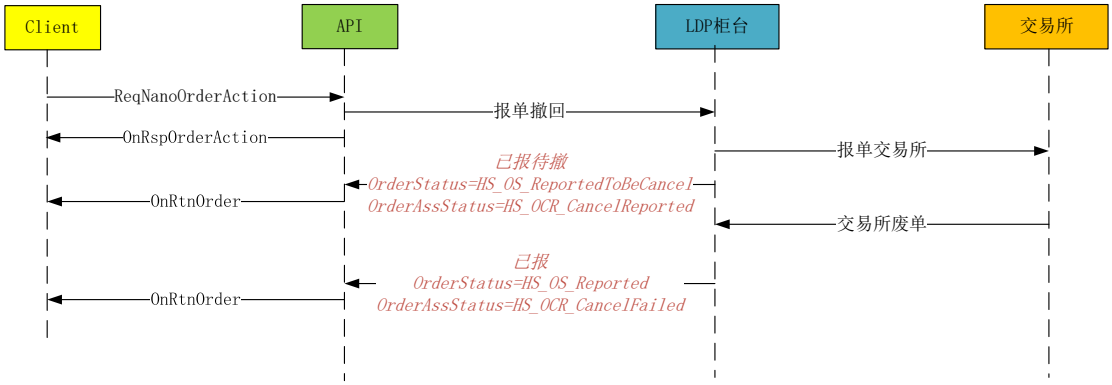

## 6.2 撤单成交场景



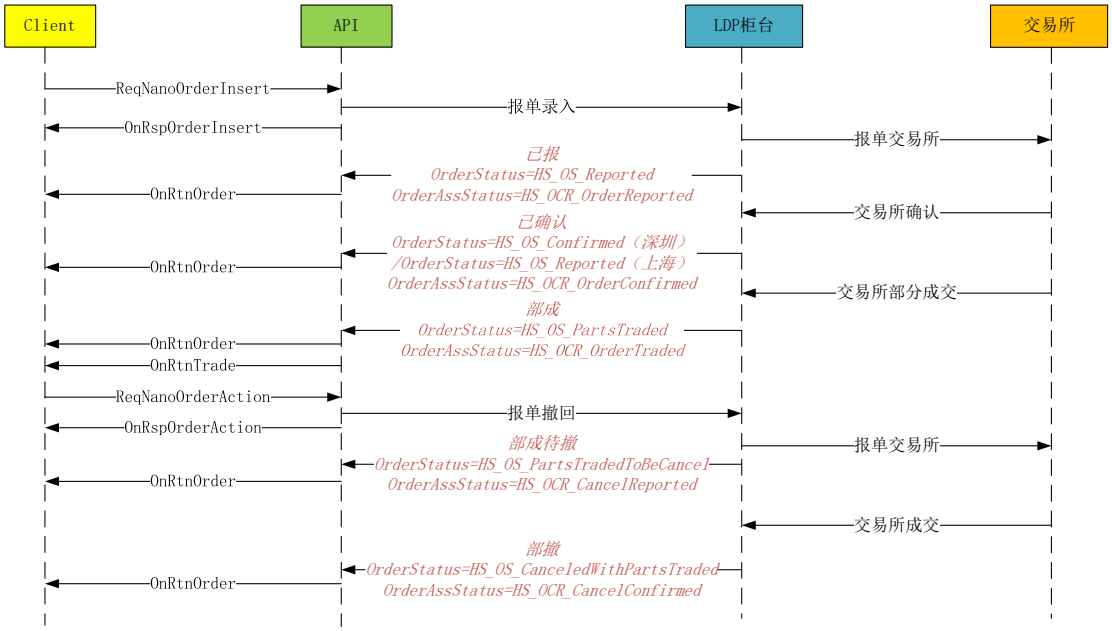
## 6.3 报单废单场景




## 6.4 撤单废单场景



## 6.5 部成部撤场景


Client
API
LDP柜台
交易所
ReqNanoOrderInsert
报单录入
OnRspOrderInsert
报单交易所
已报
OrderStatus=HS_OS_Reported
OrderAssStatus=HS_OCR_OrderReported
OnRtnOrder
交易所确认
已确认
OrderStatus=HS_OS_Confirmed（深圳）
/OrderStatus=HS_OS_Reported（上海）
OrderAssStatus=HS_OCR_OrderConfirmed
OnRtnOrder
交易所部分成交
部成
OrderStatus=HS_OS_PartsTraded
OrderAssStatus=HS_OCR_OrderTraded
OnRtnOrder
OnRtnTrade
ReqNanoOrderAction
报单撤回
OnRspOrderAction
部成待撤
OrderStatus=HS_OS_PartsTradedToBeCancel
OrderAssStatus=HS_OCR_CancelReported
OnRtnOrder
报单交易所
交易所成交
部撤
OrderStatus=HS_OS_CanceledWithPartsTraded
OrderAssStatus=HS_OCR_CancelConfirmed
OnRtnOrder

# 7 性能配置示例

在 03API\c++\lib\linux.x64 目录下 HSSecuTradeApi.ini 文件中配置绑核，需要将 HSSecuTradeApi.ini 文件放置在程序的启动目录下生效。生效后可以看见日期-system.log 中打印如下日志：

字符串日志: 配置信息 [main]:query_time=15,save_mode=2...api_bind_cpu=;...

tcprun 线程最好单独占用一个 CPU，不与其他线程共用。如果核数不足，日志线程可以都绑定在同一个 CPU 上。API 请求是使用的外部线程，需要自行绑核。

查询和交易可以分两个进程，相互不会干扰。

main 段

```
[main]
;日志记录绑核设置，默认 -1：不绑核，需要绑核需要填具体的核数，配置错误或没有对应的核会绑核失败
log_bind_cpuid=5
```

taskset 段

```
[taskset]
;配置 api 线程绑定 CPU，绑定多个 CPU 通过","分隔。如绑定 1 号和 2 号 CPU，则配置为"api_bind_cpuid=1,2"。不配置则不绑定 CPU，配置错误也不绑定 CPU
api_bind_cpuid=2,3
;单独配置 tcp 通讯线程绑定 CPU，绑定多个 CPU 通过","分隔。如绑定 3 号和 4 号 CPU，则配置为"tcp_run=3,4"。不配置则不绑定 CPU，配置错误也不绑定 CPU
tcp_run=4
;单独配置日志线程绑定 CPU，绑定多个 CPU 通过","分隔。如绑定 5 号和 6 号 CPU，则配置为"write_log=5,6"。不配置则不绑定 CPU，配置错误也不绑定 CPU
write_log=5
```

# 8 注意事项

## 8.1 构造请求结构体

恒生证券极速交易接口字符串类型均为 char 数组。以 HSExchangeID 为例，第 5 位须为\0，也就是可用字节数为 4 位。赋值前请先 memset 初始化，避免乱码导致不可预测的字段值。

## 8.2 断线重连

连接断开时，会回调 OnFrontDisconnected，API 内部会自动重连;

当断线重连成功后，会回调 OnFrontConnected，之前已登录成功用户无需重复登录，API 自动重登成功后会回调 OnRspUserLogin。

## 8.3 函数返回值

一般来说，函数返回 0 表示成功，否则是一个错误码，通过 GetApiErrorMsg 函数可以获取到详细的错误信息。

## 8.4 释放 API 实例

释放 API 实例需要调用 ReleaseAPI，否则可能引起程序异常。

## 8.5 文件句柄限制

当登录 API 时，会创建 SQLite 文件，此时若文件句柄不够，则会导致创建失败崩溃，故要在环境中查看当前文件句柄限制（ulimit -c），如果不够，需要扩大（ulimit -n）以预留足够的文件句柄。

## 8.6 配置文件编码格式

配置文件编码格式需要为 UTF-8，如果为 UTF-8-BOM 格式，可能存在第一行标签栏无法读取问题。

## 8.7 使用 HSSecuTradeApi.ini 配置文件

1、为了更好地使用 API 程序以及 API 问题排查，建议周边终端系统加上 API 配置文件 HSSecuTradeApi.ini。该文件可以从系统发布包的 03API\c++\lib\linux.x64 目录获取。

2、如果在运行 API 程序时没有加载 HSSecuTradeApi.ini 配置文件，API 程序会按照 ini 文件的默认配置值进行加载。因为 API 的一些参数需要根据实际情况进行设置才能达到最佳效果，所以建议正确加载和配置 HSSecuTradeApi.ini 文件。

3、为了 API 程序能正确读取配置文件，HSSecuTradeApi.ini 文件需要放置在工程运行目录下，注意区分工程目录和 log 文件夹所在的目录。另外，也可以将 HSSecuTradeApi.ini 文件和 log 文件夹放置在相同的目录下，这样可以更方便地管理配置文件和日志文件，避免在不同目录下寻找文件时出现麻烦。

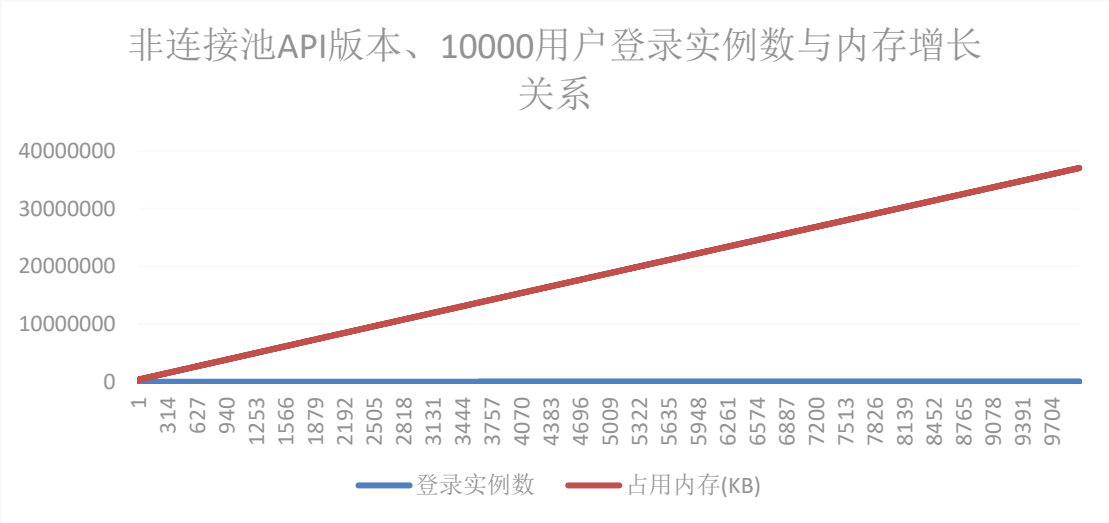
## 8.8 关于周边接口允许输入的字符串和密码长度

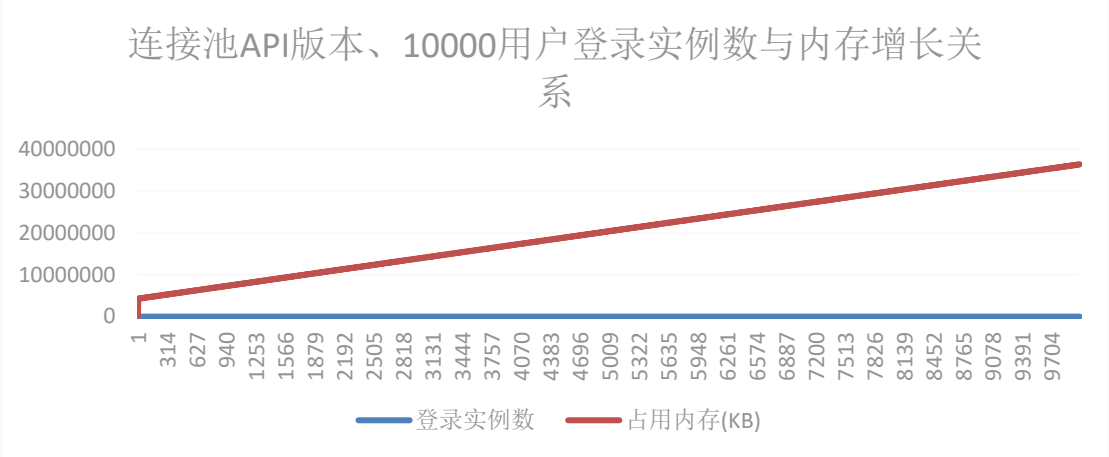
1、对于周边接口输入的字符串：必须保证\0 结尾，否则 API 会报错-1016，字符串必须以\0 结尾，请检查字符串长度是否合法。

2、对于周边接口输入的密码：长度不能超过 13 位，否则 API 会报错-1024，密码长度不能超过 13 位。

## 8.9 内存占用问题

一个 API 实例差不多占用 4M 内存左右，连接池版本会有一个初始内存占用，请注意机器得内存情况，避免内存不足。




## 8.10 回调线程

回调线程跟请求线程不是同一个线程，如果您在回调中处理请求，比如在连接成功回调函数中发起认证，需要保证数据的原子性，避免在请求函数和应答函数中同时对同一份数据进行操作。

## 8.11 拒单响应

柜台拒单只会给报这笔单的客户端返回相应的错误应答。交易所拒单会通过 OnRtnOrder 报单状态更新接口返回废单回报，如果是普通模式登录，会返回给该账户所有的客户端，如果是绑定会话模式登录，只会返回给报这笔单的客户端。

## 8.12 编码库

API 会对错误信息种的中文字符进行编码转换，使用的编码库版本为 GLIBC2.17。周边请

尽量使用该版本，否则可能出现编码转换失败的问题。已知的，当使用 GLIBC2.23 版本时会出现编码转换失败的问题。

## 8.13 升级前需要删除旧版本 sqlite 文件和日志

盘中升级新版本 api 前，需要备份 sqlite 文件和日志，再将当天的 sqlite 文件和日志删掉，否则升级新版本 API 加载当天旧版的 sqlite 文件，会出现补漏主推记录丢失的情况。删除时，需要删除初始化实例时传入 log 目录下当天的所有文件。

## 8.14 ReleaseApiStaticResource

该接口调用之后，不可再操作其他的 API 接口，建议在进程退出时调用，可释放所有 sdk 连接和静态资源。

## 8.15 RegisterSpi(NULL)

注意，在 ReleaseApi 之前需要调用 RegisterSpi(NULL)，并且在调用 RegisterSpi(NULL)之前外部不可释放 Spi 对象（在 Init 之前由 RegisterSpi 传入的指针）。不可以在 API 的回调线程中调用 RegisterSpi(NULL)，否则会产生死锁。

# 附录 1

## MFCHSSecuTradeApiDemo.exe

该工具目的在于协助开发者在开发过程中比对接口输入输出参数、测试经纪公司后台测试环境是否就绪等。位于发布包\03API\c++\demo\bin\win64（64 位）和\03API\c++\demo\bin\win64（32 位）目录下，是 windows 环境下的测试工具。

如果您的 windows 环境上没有安装 VS，那么需要取发布包\03API\c++\Environment\Windows 目录下的 vc_redist.x64.exe 搭建运行环境，双击安装即可。如果安装提示设置失败，可能是由于环境上已经安装了其他版本的 VS，则可以直接跳过该步骤。


以 64 位 windows 环境为例，将\03API\c++\lib\win64 目录下的所有程序项拷贝至目录\03API\c++\demo\bin\win64 下，双击 MFCHSSecuTradeApiDemo.exe 即可运行，运行界面如图 2-1 所示。

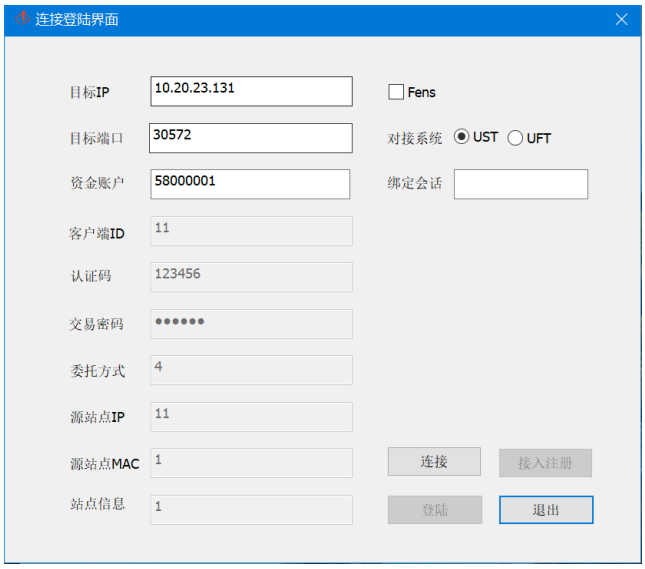



图 2-1

如果拥有经纪公司对外开放测试的地址和账号信息，则可以尝试进行接入测试。如果是 fens 地址，需要勾选 Fens，如果是前置地址，则直接填入目标 IP 和目标端口，点击“连接”，当弹出“连接成功”提示框，则关闭弹框，填入资金账户、客户端 ID、认证码、交易密码、

委托方式、源站点 IP、源站点 MAC、站点信息即可点击认证登录。登陆成功后可看到如图 2-2 所示界面：

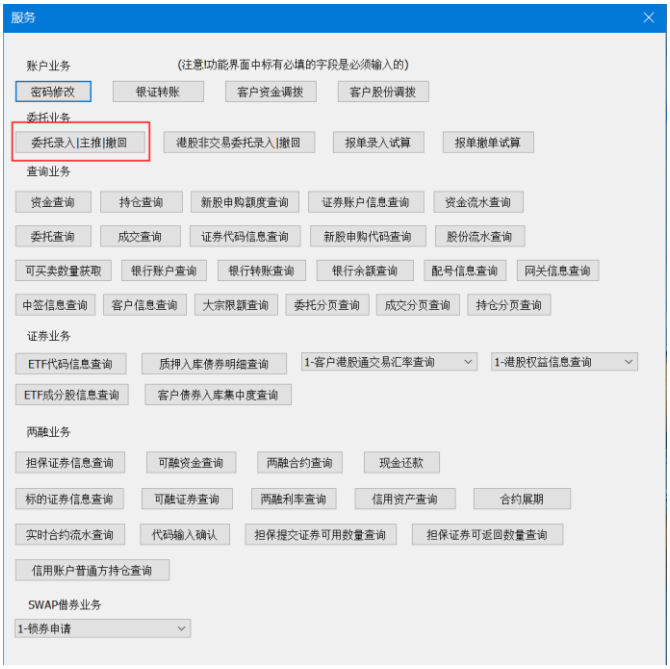



图 2-2

点击委托业务下的“委托录入|主推|撤回”可进入到如图 2-3 所示报单界面进行报单业务。

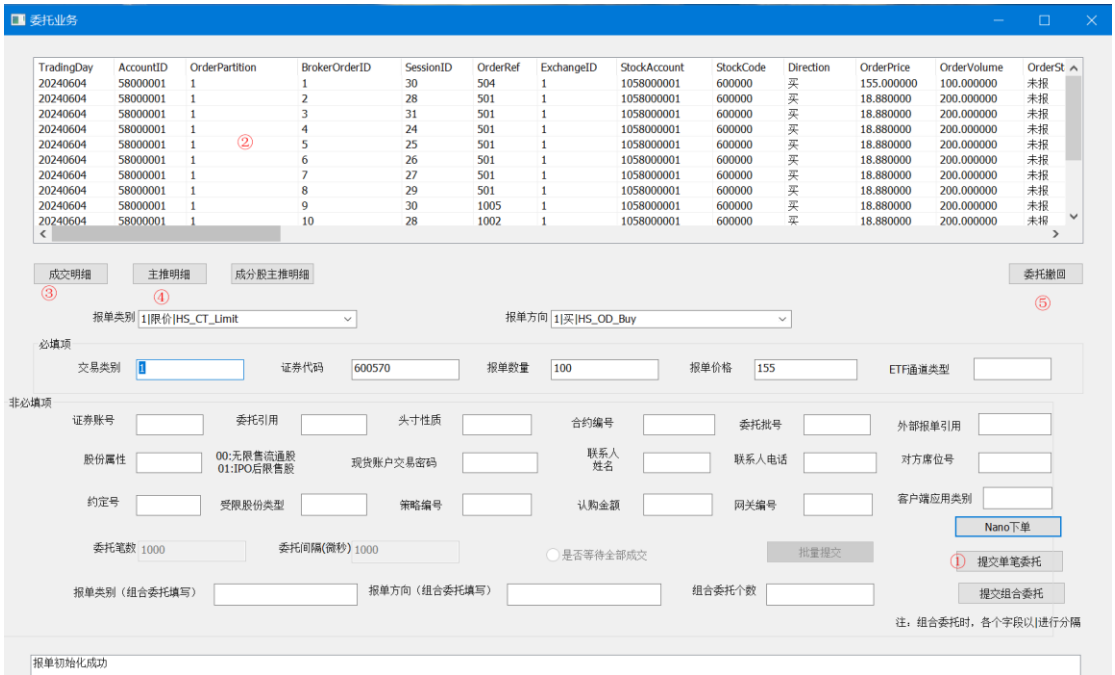



图 2-3

① 填好报单必填参数项后，点击此处进行报单。

② 显示该笔报单的最新的回报参数（包括报单状态、成交金额等）。

③ 选中界面②上的一条委托，点击此处显示该笔委托的所有成交回报。

④ 选中界面②上的一条委托，点击此处显示该笔委托的所有委托回报。

⑤ 选中界面②上的一条委托，点击此处对该笔委托进行撤单。
其余接口功能可自行体验。

# 附录 2

Demo 源代码如下：

```
#if (defined _WIN32) || (defined _WIN64)
#include <windows.h>
#else
#include <unistd.h>
#define    Sleep(x) usleep((x)*1000)
#endif

#include <atomic>
#include <string.h>
#include <iostream>
#include <stdio.h>
#include "HSSecuTradeApi.h"



HSAccountID g_FundAccount = "38000200";              // 资金账号
std::string g_FrontAddress = "tcp://10.20.23.131:30572";   // 前置地址
std::string g_FensAddress = "fens://10.20.23.131:33336";   // Fens 地址
bool bIsFens = false; // 是否通过 fens 接入
HSAppID g_AppID = "11";   // 客户端编码
HSAuthCode g_AuthCode = "123456";   // 认证码
HSPassword g_Password = "111111";   // 密码
HSUserApplicationType g_UserApplicationType = '7'; //投资者端应用类别
HSUserApplicationInfo g_UserApplicationInfo = "demo";   // 投资者端应用信息
HSUserStationInfo g_UserStationInfo = "127.0.0.1";   // 站点信息

// 回调类，继承自 CHSSecuTradeSpi，可以只实现部分回调函数
class DemoSpi : public CHSSecuTradeSpi
{
public:
        DemoSpi()
        {
                m_bConnectedFlag.store(false);
                m_bAuthedFlag.store(false);
                m_bLoginedFlag.store(false);
                m_bFirstLoginedFlag.store(false);
```

```
        m_bPasswordWrong.store(false);
    };
    ~DemoSpi() {};
public:

    // 是否允许交易
    bool IsTradeAllowed()
    {
        return m_bLoginedFlag.load();
    };

    // 是否认证成功
    bool IsAuthed()
    {
        return m_bAuthedFlag.load();
    };

    // 是否密码错误
    bool IsWrongPassword()
    {
        return m_bPasswordWrong.load();
    };

    // 是否连接成功
    bool IsConnected()
    {
        return m_bConnectedFlag.load();
    };

    // 密码错误或还未成功登录过，需要周边发起重登
    bool bNeedReLogin()
    {
        return !m_bFirstLoginedFlag.load() || m_bPasswordWrong.load();
    };

    // 连接成功
    void OnFrontConnected()
    {
        std::cout << "----------连接成功  OnFrontConnected ----------" << std::endl;
        m_bConnectedFlag.store(true);
    };

    // 连接失败
```

```
	void OnFrontDisConnected(int nResult)
	{
		std::cout << "----------连接失败  OnFrontDisConnected ErrorID: " << nResult << " ----------" << std::endl;
		m_bConnectedFlag.store(false);
		m_bLoginedFlag.store(false);
	};

	// 认证应答
	void OnRspAuthenticate(CHSSecuRspAuthenticateField *pRspAuthenticate, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
	{
		if (pRspInfo->ErrorID != 0)
		{
			std::cout << "----------认证失败  OnRspAuthenticate Error ----------" << std::endl
				<< "\t【错误代码】ErrorID = " << pRspInfo->ErrorID << std::endl
				<< "\t【错误提示】ErrorMsg = " << pRspInfo->ErrorMsg << std::endl;
		}
		else
		{
			if (pRspAuthenticate)
			{
				std::cout << "----------认证成功  OnRspAuthenticate Success ----------" << std::endl
					<< "\t【资产账号】AccountID = " << pRspAuthenticate->AccountID << std::endl
					<< "\t【客户端 id】AppID = " << pRspAuthenticate->AppID << std::endl
					<< "\t【认证码】AuthCode = " << pRspAuthenticate->AuthCode << std::endl;
				m_bAuthedFlag.store(true);
			}
		}
	};

	// 登录应答
	void OnRspUserLogin(CHSSecuRspUserLoginField *pRspUserLogin, CHSSecuRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
	{
		if (pRspInfo->ErrorID != 0)
		{
			if (pRspInfo->ErrorID == 1000 || pRspInfo->ErrorID == 369046)
			{
				m_bPasswordWrong.store(true);
			}
			std::cout << "----------登陆失败  OnRspUserLogin Error ----------" << std::endl
				<< "\t【错误代码】ErrorID =    " << pRspInfo->ErrorID << std::endl
				<< "\t【错误提示】ErrorMsg = " << pRspInfo->ErrorMsg << std::endl;
```

```
            }
            else
            {
                if (pRspUserLogin)
                {
                    std::cout << "----------登陆成功 OnRspUserLogin Success ----------" << std::endl
                        << "\t【营业部号】BranchID = " << pRspUserLogin->BranchID << std::endl
                        << "\t【资产账号】AccountID = " << pRspUserLogin->AccountID << std::endl
                        << "\t【资产属性】AssetProp = " << pRspUserLogin->AssetProp << std::endl
                        << "\t【客户姓名】UserName = " << pRspUserLogin->UserName << std::endl
                        << "\t【交易日】TradingDay = " << pRspUserLogin->TradingDay << std::endl
                        << "\t【报单引用】OrderRef = " << pRspUserLogin->OrderRef << std::endl
                        << "\t【会话编号】SessionID = " << pRspUserLogin->SessionID << std::endl
                        << "\t【客户编号】UserID = " << pRspUserLogin->UserID << std::endl
                        << "\t【客户风险等级】CorpRiskLevel = " << pRspUserLogin->CorpRiskLevel << std::endl
                        << "\t【客户姓名长】UserNameLong = " << pRspUserLogin->UserNameLong << std::endl;
                    m_bFirstLoginedFlag.store(true);
                    m_bLoginedFlag.store(true);
                }
            }
        };

        void OnRspOrderInsert(CHSSecuRspOrderInsertField *pRspOrderInsert, CHSSecuRspInfoField *pRspInfo, int
nRequestID, bool bIsLast)
        {
            if (pRspInfo->ErrorID != 0)
            {
                std::cout << "---------- 报单录入应答 OnRspOrderInsert Error ----------" << std::endl
                    << "\t【错误代码】ErrorID =  " << pRspInfo->ErrorID << std::endl
                    << "\t【错误提示】ErrorMsg = " << pRspInfo->ErrorMsg << std::endl;
            }
            else
            {
                if (pRspOrderInsert)
                {
                    std::cout << "---------- 报单录入应答 OnRspOrderInsert Success ----------" << std::endl
                        << "\t【报单分区】OrderPartition = " << pRspOrderInsert->OrderPartition << std::endl
                        << "\t【经纪公司报单编码】BrokerOrderID = " << pRspOrderInsert->BrokerOrderID <<
std::endl
                        << "\t【会话编号】SessionID = " << pRspOrderInsert->SessionID << std::endl
                        << "\t【报单引用】OrderRef = " << pRspOrderInsert->OrderRef << std::endl
                        << "\t【报单批次号】BatchNo = " << pRspOrderInsert->BatchNo << std::endl
                        << "\t【客户端报单编号】ClientOrderID = " << pRspOrderInsert->ClientOrderID << std::endl
```

```
                        << "\t【交易所申报编号】OrderID = " << pRspOrderInsert->OrderID << std::endl
                        << "\t【外部报单引用】ExtOrderRef = " << pRspOrderInsert->ExtOrderRef << std::endl;
                }
        }
};

void OnRtnOrder(CHSSecuOrderField *pRtnOrder)
{
        if (pRtnOrder)
        {
                std::cout << "----------  报单更新通知  OnRtnOrder Success ----------" << std::endl
                        << "\t【交易日期】TradingDay = " << pRtnOrder->TradingDay << std::endl
                        << "\t【资产账户】AccountID = " << pRtnOrder->AccountID << std::endl
                        << "\t【报单分区】OrderPartition = " << pRtnOrder->OrderPartition << std::endl
                        << "\t【经纪公司报单编码】BrokerOrderID = " << pRtnOrder->BrokerOrderID << std::endl
                        << "\t【会话编号】SessionID = " << pRtnOrder->SessionID << std::endl
                        << "\t【报单引用】OrderRef = " << pRtnOrder->OrderRef << std::endl
                        << "\t【交易所代码】ExchangeID = " << pRtnOrder->ExchangeID << std::endl
                        << "\t【证券账号】StockAccount = " << pRtnOrder->StockAccount << std::endl
                        << "\t【证券代码】StockCode = " << pRtnOrder->StockCode << std::endl
                        << "\t【报单方向】Direction = " << pRtnOrder->Direction << std::endl
                        << "\t【报单价格】OrderPrice = " << pRtnOrder->OrderPrice << std::endl
                        << "\t【报单数量】OrderVolume = " << pRtnOrder->OrderVolume << std::endl
                        << "\t【报单状态】OrderStatus = " << pRtnOrder->OrderStatus << std::endl
                        << "\t【报单指令】OrderCommand = " << pRtnOrder->OrderCommand << std::endl
                        << "\t【申报时间】ReportTime = " << pRtnOrder->ReportTime << std::endl
                        << "\t【废单原因】ErrorMsg = " << pRtnOrder->ErrorMsg << std::endl
                        << "\t【成交数量】TradeVolume = " << pRtnOrder->TradeVolume << std::endl
                        << "\t【报单批次号】BatchNo = " << pRtnOrder->BatchNo << std::endl
                        << "\t【撤单数量】WithdrawVolume = " << pRtnOrder->WithdrawVolume << std::endl
                        << "\t【报单时间】OrderTime = " << pRtnOrder->OrderTime << std::endl
                        << "\t【经纪公司撤单编码】BrokerWithdrawOrderID = " << pRtnOrder->BrokerWithdrawOrderID
<< std::endl
                        << "\t【委托冻结金额】FrozenBalance = " << pRtnOrder->FrozenBalance << std::endl
                        << "\t【委托冻结费用】FrozenFare = " << pRtnOrder->FrozenFare << std::endl
                        << "\t【营业部号】BranchID = " << pRtnOrder->BranchID << std::endl
                        << "\t【报单辅助状态】OrderAssStatus = " << pRtnOrder->OrderAssStatus << std::endl
                        << "\t【撤单订单状态】WithdrawOrderStatus = " << pRtnOrder->WithdrawOrderStatus << std::endl
                        << "\t【发生时间】OccurTime = " << pRtnOrder->OccurTime << std::endl
                        << "\t【客户端报单编号】ClientOrderID = " << pRtnOrder->ClientOrderID << std::endl
                        << "\t【证券名称】StockName = " << pRtnOrder->StockName << std::endl
                        << "\t【成交价格】TradePrice = " << pRtnOrder->TradePrice << std::endl
                        << "\t【成交金额】BusinessBalance = " << pRtnOrder->BusinessBalance << std::endl
```

```
                        << "\t【证券类别】StockType = " << pRtnOrder->StockType << std::endl
                        << "\t【交易所申报编号】OrderID = " << pRtnOrder->OrderID << std::endl
                        << "\t【头寸性质】CashgroupProp = " << pRtnOrder->CashgroupProp << std::endl
                        << "\t【外部报单引用】ExtOrderRef = " << pRtnOrder->ExtOrderRef << std::endl
                        << "\t【总成交费用】TotalBusinessFare = " << pRtnOrder->TotalBusinessFare << std::endl
                        << "\t【主推序号】SequenceNumber = " << pRtnOrder->SequenceNumber << std::endl
                        << "\t【投资者端应用类别】UserApplicationType = " << pRtnOrder->UserApplicationType <<
std::endl
                        << "\t【补单标志】RenewFlag = " << pRtnOrder->RenewFlag << std::endl
                        << "\t【策略编号】StrategyId = " << pRtnOrder->StrategyId << std::endl
                        << "\t【总资产回报买入金额】TotalRealBuyBalance = " << pRtnOrder->TotalRealBuyBalance <<
std::endl
                        << "\t【总资产回报卖出金额】TotalRealSellBalance = " << pRtnOrder->TotalRealSellBalance <<
std::endl
                        << "\t【系统号】SystemNo = " << pRtnOrder->SystemNo << std::endl
                        << "\t【废单错误代码】ErrorNo = " << pRtnOrder->ErrorNo << std::endl
                        << "\t【认购金额】SubscribeBalance = " << pRtnOrder->SubscribeBalance << std::endl
                        << "\t【网关编号】DTgwIndex = " << pRtnOrder->DTgwIndex << std::endl
                        << "\t【实际网关编号】RealDTgwIndex = " << pRtnOrder->RealDTgwIndex << std::endl;
                }
        };

        void OnRtnTrade(CHSSecuTradeField *pRtnTrade)
        {
                if (pRtnTrade)
                {
                        std::cout << "----------  报单成交通知  OnRtnTrade Success ----------" << std::endl
                                << "\t【交易日期】TradingDay = " << pRtnTrade->TradingDay << std::endl
                                << "\t【资产账户】AccountID = " << pRtnTrade->AccountID << std::endl
                                << "\t【报单分区】OrderPartition = " << pRtnTrade->OrderPartition << std::endl
                                << "\t【经纪公司报单编码】BrokerOrderID = " << pRtnTrade->BrokerOrderID << std::endl
                                << "\t【会话编号】SessionID = " << pRtnTrade->SessionID << std::endl
                                << "\t【报单引用】OrderRef = " << pRtnTrade->OrderRef << std::endl
                                << "\t【交易所代码】ExchangeID = " << pRtnTrade->ExchangeID << std::endl
                                << "\t【证券账号】StockAccount = " << pRtnTrade->StockAccount << std::endl
                                << "\t【证券代码】StockCode = " << pRtnTrade->StockCode << std::endl
                                << "\t【买卖方向】Direction = " << pRtnTrade->Direction << std::endl
                                << "\t【报单指令】OrderCommand = " << pRtnTrade->OrderCommand << std::endl
                                << "\t【成交状态】TradeStatus = " << pRtnTrade->TradeStatus << std::endl
                                << "\t【成交编号】TradeID = " << pRtnTrade->TradeID << std::endl
                                << "\t【成交数量】TradeVolume = " << pRtnTrade->TradeVolume << std::endl
                                << "\t【成交价格】TradePrice = " << pRtnTrade->TradePrice << std::endl
                                << "\t【成交时间】TradeTime = " << pRtnTrade->TradeTime << std::endl
```

```
                        << "\t【废单原因】ErrorMsg = " << pRtnTrade->ErrorMsg << std::endl
                        << "\t【报单批次号】BatchNo = " << pRtnTrade->BatchNo << std::endl
                        << "\t【经纪公司撤单编码】BrokerWithdrawOrderID = " << pRtnTrade->BrokerWithdrawOrderID
<< std::endl
                        << "\t【成交金额】BusinessBalance = " << pRtnTrade->BusinessBalance << std::endl
                        << "\t【营业部号】BranchID = " << pRtnTrade->BranchID << std::endl
                        << "\t【报单状态】OrderStatus = " << pRtnTrade->OrderStatus << std::endl
                        << "\t【发生时间】OccurTime = " << pRtnTrade->OccurTime << std::endl
                        << "\t【客户端报单编号】ClientOrderID = " << pRtnTrade->ClientOrderID << std::endl
                        << "\t【证券名称】StockName = " << pRtnTrade->StockName << std::endl
                        << "\t【报单价格】OrderPrice = " << pRtnTrade->OrderPrice << std::endl
                        << "\t【交易所申报编号】OrderID = " << pRtnTrade->OrderID << std::endl
                        << "\t【外部报单引用】ExtOrderRef = " << pRtnTrade->ExtOrderRef << std::endl
                        << "\t【总成交费用】TotalBusinessFare = " << pRtnTrade->TotalBusinessFare << std::endl
                        << "\t【主推序号】SequenceNumber = " << pRtnTrade->SequenceNumber << std::endl
                        << "\t【回报买入数量】RealBuyAmount = " << pRtnTrade->RealBuyAmount << std::endl
                        << "\t【回报买入金额】RealBuyBalance = " << pRtnTrade->RealBuyBalance << std::endl
                        << "\t【回报卖出数量】RealSellAmount = " << pRtnTrade->RealSellAmount << std::endl
                        << "\t【回报卖出金额】RealSellBalance = " << pRtnTrade->RealSellBalance << std::endl
                        << "\t【投资者端应用类别】UserApplicationType = " << pRtnTrade->UserApplicationType <<
std::endl
                        << "\t【补单标志】RenewFlag = " << pRtnTrade->RenewFlag << std::endl
                        << "\t【策略编号】StrategyId = " << pRtnTrade->StrategyId << std::endl
                        << "\t【系统号】SystemNo = " << pRtnTrade->SystemNo << std::endl
                        << "\t【预估现金】SumCashBalance = " << pRtnTrade->SumCashBalance << std::endl;
            }
        };

private:
        std::atomic<bool> m_bConnectedFlag;        // 连接成功标识
        std::atomic<bool> m_bAuthedFlag;           // 认证成功标识
        std::atomic<bool> m_bFirstLoginedFlag;   // 首次登录成功标识
        std::atomic<bool> m_bLoginedFlag;          // 登录成功标识
        std::atomic<bool> m_bPasswordWrong;        // 登录密码错误标识
};

int main()
{
        DemoSpi* lpDemoSpi = new(std::nothrow) DemoSpi();
        CHSSecuTradeApi* lpTradeApi = NewSecuTradeApi("./log"); // 日志和主推缓存存储在 log 目录下
        lpTradeApi->RegisterSpi(lpDemoSpi);
        int iRet = 0;
        if (bIsFens)
```

```
	{
		iRet = lpTradeApi->RegisterFensServer(g_FensAddress.c_str(), g_FundAccount);
	}
	else
	{
		iRet = lpTradeApi->RegisterFront(g_FrontAddress.c_str());
	}
	if (0 != iRet)
	{
		std::cout << "==========注册地址失败  ErrorID: " << iRet << " ErrorMsg: " << lpTradeApi->GetApiErrorMsg(iRet)
<< std::endl;
		return iRet;
	}
	std::cout << "==========连接开始==========" << std::endl;
	iRet = lpTradeApi->Init("");
	if (iRet == -1005 || iRet == -1018)
	{
		std::cout << "==========创建连接对象失败，请检查日志中错误信息，并进行相应调整" << std::endl;
		return -1;
	}
	while (true)
	{
		Sleep(1000);
		if (lpDemoSpi->IsConnected())
		{
			CHSSecuReqAuthenticateField stReqAuthenticate;
			memset(&stReqAuthenticate, 0, sizeof(CHSSecuReqAuthenticateField));
			memcpy(stReqAuthenticate.AccountID, g_FundAccount, sizeof(HSAccountID));
			memcpy(stReqAuthenticate.AppID, g_AppID, sizeof(HSAppID));
			memcpy(stReqAuthenticate.AuthCode, g_AuthCode, sizeof(HSAuthCode));
			lpTradeApi->ReqAuthenticate(&stReqAuthenticate, 1);
			Sleep(1000);
			if (lpDemoSpi->IsAuthed())
			{
				CHSSecuReqUserLoginField stReqUserLogin;
				memset(&stReqUserLogin, 0, sizeof(CHSSecuReqUserLoginField));
				memcpy(stReqUserLogin.AccountID, g_FundAccount, sizeof(HSAccountID));
				memcpy(stReqUserLogin.Password, g_Password, sizeof(HSPassword));
				stReqUserLogin.UserApplicationType = g_UserApplicationType;
				memcpy(stReqUserLogin.UserStationInfo, g_UserStationInfo, sizeof(HSUserStationInfo));
				lpTradeApi->ReqUserLogin(&stReqUserLogin, 2);;
				Sleep(1000);
				if (lpDemoSpi->IsTradeAllowed() || lpDemoSpi->IsWrongPassword())
```

```
                        {
                                break;
                        }
                }
        }

        if (lpDemoSpi->IsWrongPassword())
        {
                std::cout << "==========密码错误，请修改密码==========" << std::endl;
                return -2;
        }
        std::cout << "==========交易开始==========" << std::endl;
        CHSSecuReqOrderInsertField stReqOrderInsert;
        memset(&stReqOrderInsert, 0, sizeof(CHSSecuReqOrderInsertField));
        memcpy(stReqOrderInsert.ExchangeID, "1", sizeof(HSAccountID));
        memcpy(stReqOrderInsert.StockCode, "600000", sizeof(HSPassword));
        stReqOrderInsert.Direction = HS_OD_Buy;
        stReqOrderInsert.OrderCommand = HS_CT_Limit;
        stReqOrderInsert.OrderVolume = 200;
        stReqOrderInsert.OrderPrice = 18.88;
        stReqOrderInsert.UserApplicationType = g_UserApplicationType;
        memcpy(stReqOrderInsert.UserStationInfo, g_UserStationInfo, sizeof(HSUserStationInfo));
        iRet = lpTradeApi->ReqOrderInsert(&stReqOrderInsert, 3);
        if (0 != iRet)
        {
                std::cout << "==========ReqOrderInsert Failed ErrorID:" << iRet << " ErrorMsg: " <<
lpTradeApi->GetApiErrorMsg(iRet) << std::endl;
        }
        getchar();
        lpTradeApi->ReleaseApi();
        return iRet;
}
```

Demo.x64.gcc 内容如下：

```
ARM = aarch64
arch = $(shell uname -m)

ifeq ($(arch), $(ARM))
        OUT_DIR         = ../lib/linux.arm
        LIB_DIR     = -L../lib/linux.arm
else
        OUT_DIR         = ../lib/linux.x64
```

```
    LIB_DIR     =  -L../lib/linux.x64
endif

all:  Demo.o
    g++  -o  $(OUT_DIR)/Demo.out  -g  -O0  -Wl,--no-undefined  -Wl,$(LIB_DIR)  Demo.o
-lt2sdk  -lldptcpsdk  -lHSSecuTradeApi  -lpthread
Demo.o:  Demo.cpp
    g++  -std=c++11  -c  Demo.cpp  -g  -o  Demo.o  -I../include
.PHONY:clean
clean:
    rm -f *.o
```