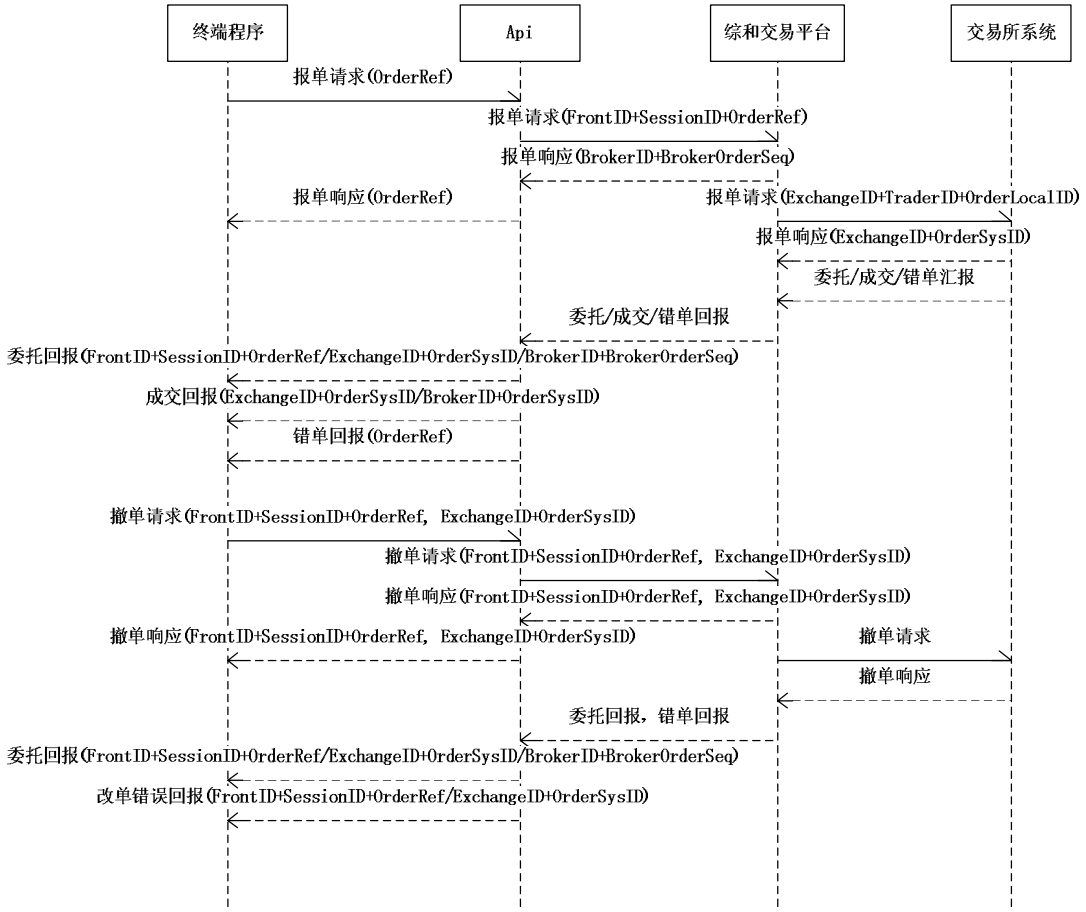
综合交易平台交易 API 特别说明

1 报单指令



因为 TraderApi 是异步处理的，所以交易指令都是分 2 阶段提交。

首先，Thost 收到交易指令后，客户端会收到报单响应，确认收到客户端的交易指令。

同时，Thost 把交易指令转发到交易所。

之后，Thost 收到来自交易所的响应和回报，通过 TraderApi 的回报事件通知给客户端。

报单指令是：ReqOrderInsert。

- 报单指令中如下字段需要如下设置：

```
///成交量类型：任何数量
fldOrder.VolumeCondition = THOST_FTDC_VC_AV;
/// 最小成交量：1
fldOrder.MinVolume = 1;
```

```
/// 强平原因：非强平
fldOrder.ForceCloseReason = THOST_FTDC_FCC_NotForceClose;
/// 自动挂起标志：否
fldOrder.IsAutoSuspend = 0;
///用户强评标志：否
fldOrder.UserForceClose = 0;
```

- 如果发送立即限价单：

```
/// 报单价格条件类型：限价
OrderPriceType = THOST_FTDC_OPT_LimitPrice;
/// 价格：用户设定
LimitPrice = ……;
/// 有效期类型类型：当日有效
TimeCondition = THOST_FTDC_TC_GFD;
```

- 如果发送立即市价单

```
/// 报单价格条件类型：任意价
fldOrder.OrderPriceType = THOST_FTDC_OPT_AnyPrice;
///价格：0
fldOrder.LimitPrice = 0;
///有效期类型类型：立即完成，否则撤销
fldOrder.TimeCondition = THOST_FTDC_TC_IOC;
```

- 如果发送触发单

```
///触发条件：用户设定
ContingentCondition = ……;
///止损价：用户设定
StopPrice = ……;
/// 报单价格条件类型：限价
OrderPriceType = THOST_FTDC_OPT_LimitPrice;
/// 价格：用户设定
LimitPrice = ……;
/// 有效期类型类型：当日有效
TimeCondition = THOST_FTDC_TC_GFD;
```

- 关于平仓

  上期所区分昨仓和今仓。

  平昨仓时，开平标志类型设置为平仓 THOST_FTDC_OF_Close

  平今仓时，开平标志类型设置为平今仓 THOST_FTDC_OF_CloseToday

  其他交易所不区分昨仓和今仓。

  开平标志类型统一设置为平仓 THOST_FTDC_OF_Close

- 在报单交易过程中，会产生如下几组交易序列号：

ü FrontID + SessionID + OrderRef

用户使用这组交易序列号可以按照自己的方式来唯一标示发出的任何一笔委托。

用户登入成功后，会收到前置机编号 FrontID，会话编号 SessionID 和最大报单引用 MaxOrderRef。

用户在报单时设定报单引用 OrderRef。 OrderRef 可以从 MaxOrderRef 开始递增。

如果用户没有设定 OrderRef，在报单响应中，Thost 会为用户设置一个的 OrderRef。使得每个报单的这组序列号保持唯一。

因为这组交易序列号是由用户设定的。所以在没有得到报单响应前，就可以使用这组交易序列号进行撤单操作。

ü BrokerID + BrokerOrderSeq

Thost 收到用户报单后，为每个经纪公司的报单生成 1 组交易序列号。

ü exchangeID + traderID + OrderLocalID

交易席位在向交易所报单时，产生这组交易序列号，标示每一笔发往交易所的报单。

ü exchangeID + OrderSysID

交易所接受了投资者报单，产生这组交易序列号，标示每一笔收到的报单。

用户撤单时也可以使用这组交易序列号。

l 报单响应和回报

Thost 收到报单指令，如果没有通过参数校验，拒绝接受报单指令。用户就会收到 OnRspOrderInsert 消息，其中包含了错误编码和错误消息。

如果 Thost 接受了报单指令，用户不会收到 OnRspOrderInser，而会收到 OnRtnOrder，用来更新委托状态。

交易所收到报单后，通过校验。用户会收到 OnRtnOrder、OnRtnTrade。

如果交易所认为报单错误，用户就会收到 OnErrRtnOrder。

2 撤单指令

撤单指令是：ReqOrderAction。

l 撤单输入参数：

/// 报单操作引用，

/// OrderRef 相似,有用户自己设定，保持递增。如果用户不设定的话，有 Thost 来设定。

OrderActionRef

/// 操作标志类型：撤单

ActionFlag = THOST_FTDC_AF_Delete

/// 交易序列号

FrontID +SessionID+OrderRef，

ExchangID+OrderSysID。

/// 其他参数

BrokerID，

UserID,
InvestorID，
InstrumentID，

如果报单还停留在 Thost，Thost 可以用 Front +SessionID+OrderRef 来定位
如果报单停留在交易所，Thost 可以用 ExchangID+OrderSysID 来定位，然后向交易所转发撤单指令。

- 撤单响应和回报：

和报单响应和回报相似。
Thost 收到撤单指令，如果没有通过参数校验，拒绝接受撤单指令。用户就会收到 OnRspOrderAction 消息，其中包含了错误编码和错误消息。
如果 Thost 接受了撤单指令，用户不会收到 OnRspOrderAction，而会收到 OnRtnOrder，用来更新委托状态。
交易所收到撤单后，通过校验，执行了撤单操作。用户会收到 OnRtnOrder。
如果交易所认为报单错误，用户就会收到 OnErrRtnOrderAction。

3 委托回报

委托回报的事件处理方法是：OnRtnOrder()。

委托回报描述了报单的当前状态，其中包括：

- 原始的报单指令

- 上述几组交易序列号：

FrontID + SessionID + OrderRef，
BrokerID + BrokerOrderSeq，
ExchangeID + TraderID + LocalOrderID
ExchangeID + OrderSysID，

- 报单委托状态

```
///全部成交
#define THOST_FTDC_OST_AllTraded '0'
///部分成交还在队列中
#define THOST_FTDC_OST_PartTradedQueueing '1'
///部分成交不在队列中
#define THOST_FTDC_OST_PartTradedNotQueueing '2'
///未成交还在队列中
#define THOST_FTDC_OST_NoTradeQueueing '3'
///未成交不在队列中
#define THOST_FTDC_OST_NoTradeNotQueueing '4'
///撤单
#define THOST_FTDC_OST_Canceled '5'
```

```
///未知，表示Thost已经接受用户的委托指令，还没有触发到交易所
#define THOST_FTDC_OST_Unknown 'a'
///尚未触发
#define THOST_FTDC_OST_NotTouched 'b'
```

- 报单数量状态

剩余数量 VolumeTotal：表示这笔委托还没有成交的数量。如果 1 笔委托有 10 手，部分成交后剩余 5 手。撤单成功后，剩余数量还是 5 手。

- 单个投资者帐号在多点登入

Thost 支持同一个投资者帐号创建多个 TradeApi 对象同时登入系统进行交易。为区分各自的委托回报和成交回报，可以采用如下方式：

登入成功后，记下当前会话的 FrontID + SessionID.

报单时，自己设定 OrderRef。

收到委托回报时，使用 FrontID+SessionID 过滤出自己的委托回报。同时记下关联的 ExchangeID+OrderSysID。

收到成交回报时，使用以上得到的 ExchangeID + OrderSysID 过滤出自己的成交回报。

注意：上述方法只在当前会话中的报单有效。

4 成交回报

成交回报的事件处理方法是：OnRtnTrade()。

成交回报描述了报单的成交事件，包括分笔成交。其中包括：

- 下列几组交易序列号：

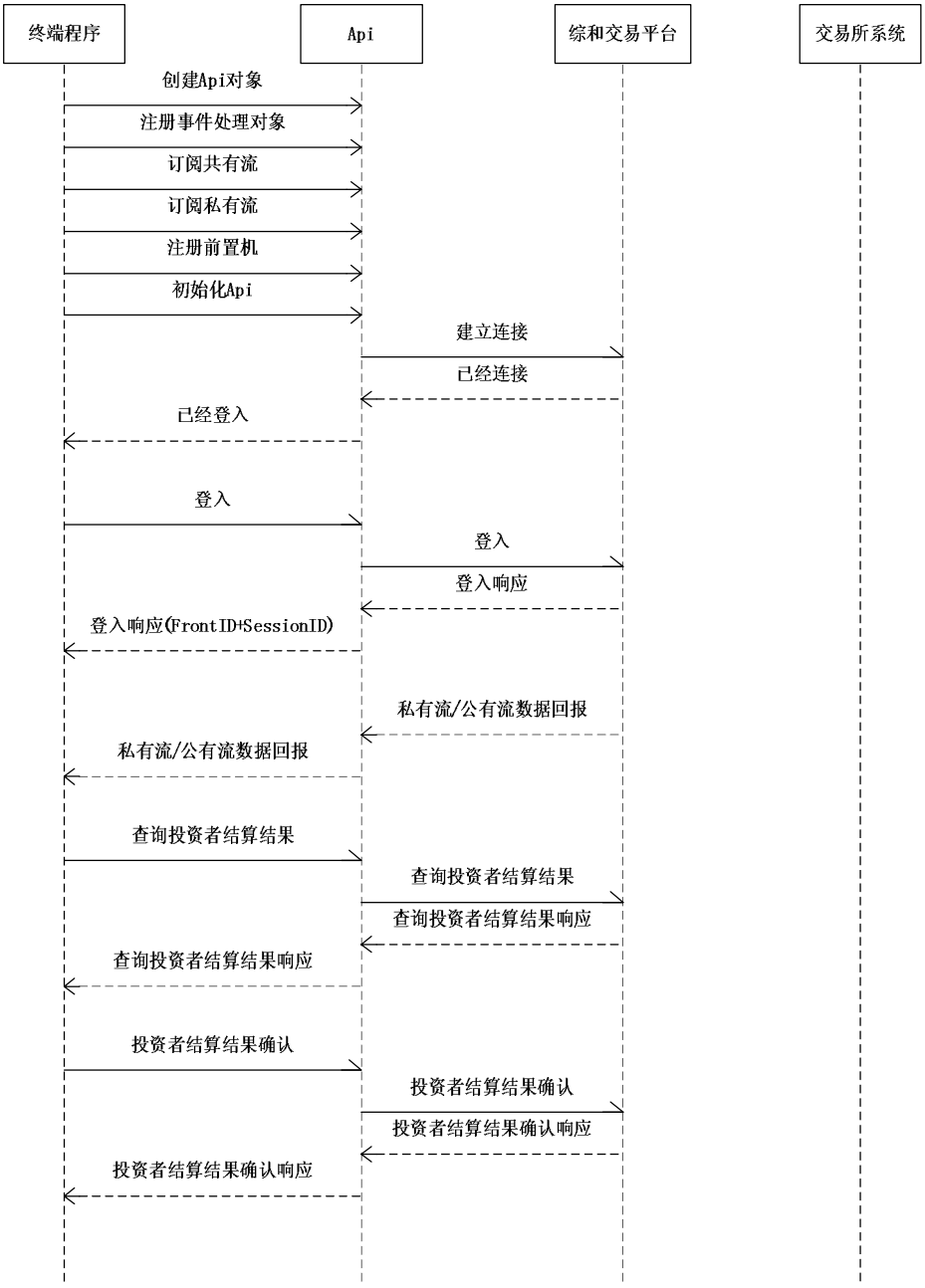
BrokerID + BrokerOrderSeq，

ExchangeID + TraderID + LocalOrderID

ExchangeID + OrderSysID.

已知一笔委托的 FrontID + SessionID + OrderRef，要在成交回报中找到相关的成交记录。可以在委托回报中，从 FrontID + SessionID + OrderRef 印射到相关的 ExchangeID + OrderSysID。然后在成交回报中，用 ExchangeID + OrderSysID 找出这笔委托的相关成交记录。

5 初始化过程



首先，终端程序创建 TraderApi 对象(CreateFtdcTraderApi)，注册用户定义的事件处理对象(RegisterSpi)，订阅公用流(SubscribePublicTopic)、私有流(SubscribePrivateTopic)，注册前置机(RegisterFront)（可以注册多个前置机，TraderApi 自动选择 1 个可用前置机登入）。完成上述 TraderApi 设置后，初始化 TraderApi(Init)，连接 Thost。

Thost 接受 TraderApi 的连接后，终端程序就会收到连线通知(OnFrontConnected)。

终端程序收到连线通知后，发出登入请求(ReqUserLogin)。

用户登入时需要输入：

```
///经纪公司代码
TThostFtdcBrokerIDType	BrokerID;
///用户代码，就是投资者代码
TThostFtdcUserIDType	UserID;
///密码
TThostFtdcPasswordType	Password;
///用户端产品信息，就是终端程序的名称
TThostFtdcProductInfoType	UserProductInfo;
```

登入成功后，会收到登入响应(OnRspUserLogin)

- 当前会话的参数。用户可以用这些参数定义自己的交易序列号 FrontID+SessionID+OrderRef

```
///前置编号
TThostFtdcFrontIDType	FrontID;
///会话编号
TThostFtdcSessionIDType	SessionID;
///最大报单引用
TThostFtdcOrderRefType	MaxOrderRef;
```

- 当前各个交易所的时间。终端程序依此可以估计未来各个交易所的时间。

```
///上期所时间
TThostFtdcTimeType	SHFETime;
///大商所时间
TThostFtdcTimeType	DCETime;
///郑商所时间
TThostFtdcTimeType	CZCETime;
///中金所时间
TThostFtdcTimeType	FFEXTime;
```

登入成功后，终端程序还会收到私有流数据，如委托回报、成交回报等。

为了让投资者了解当前的交易风险。每天，终端程序第一次登入 Thost 成功后，必须查询投资者结算结果(ReqQrySettlementInfo)和确认投资者结算结果(ReqSettlementInfoConfirm)，才能正常发送交易指令，包括报单、撤单、服务器预埋单等指令。

在一天中，如果投资者中已经确认了结算结果，以后登入 Thost，就不再必须确认结算结果，就可以直接发送交易指令了。

在正式开始交易之前，客户端可能还需要查询如下基础数据，包括：

查询合约：ReqQryInstrument。

查询资金：ReqQryTradingAccount

查询持仓：ReqQryInvestorPosition

查询报单：ReqQryOrder，支持分时段查询。

查询成交：ReqQryTrade，支持分时段查询。

6 样例代码

```
// tradeapitest.cpp :
// 一个简单的例子，介绍CThostFtdcTraderApi和CThostFtdcTraderSpi接口的使用。
// 本例将演示一个报单录入操作的过程

#include <stdio.h>
#include <windows.h>
#include "FtdcTraderApi.h"

// 报单录入操作是否完成的标志
// Create a manual reset event with no signal
HANDLE g_hEvent = CreateEvent(NULL, true, false, NULL);

// 会员代码
TThostFtdcBrokerIDType g_chBrokerID;
// 交易用户代码
TThostFtdcUserIDType g_chUserID;

class CSimpleHandler : public CThostFtdcTraderSpi
{
public:
    // 构造函数，需要一个有效的指向CThostFtdcMduserApi实例的指针
    CSimpleHandler(CThostFtdcTraderApi *pUserApi) : m_pUserApi(pUserApi) {}

    ~CSimpleHandler() {}

    // 当客户端与交易托管系统建立起通信连接，客户端需要进行登录
    virtual void OnFrontConnected()
    {
        CThostFtdcReqUserLoginField reqUserLogin;
        // get BrokerID
        printf("BrokerID:");
        scanf("%s", &g_chBrokerID);
        strcpy(reqUserLogin.BrokerID, g_chBrokerID);
        // get userid
        printf("userid:");
        scanf("%s", &g_chUserID);
        strcpy(reqUserLogin.UserID, g_chUserID);
        // get password
        printf("password:");
```

```
            scanf("%s", &reqUserLogin.Password);
            // 发出登陆请求
            m_pUserApi->ReqUserLogin(&reqUserLogin, 0);
        }

        // 当客户端与交易托管系统通信连接断开时，该方法被调用
        virtual void OnFrontDisconnected(int nReason)
        {
            // 当发生这个情况后，API会自动重新连接，客户端可不做处理
            printf("OnFrontDisconnected.\n");
        }

        // 当客户端发出登录请求之后，该方法会被调用，通知客户端登录是否成功
        virtual void OnRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
            printf("OnRspUserLogin:\n");
            printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
            printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

            if (pRspInfo->ErrorID != 0) {
                // 端登失败，客户端需进行错误处理
                printf("Failed to login, errorcode=%d errormsg=%s requestid=%d
chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
                exit(-1);
            }

            // 端登成功,发出报单录入请求
            CThostFtdcInputOrderField ord;
            memset(&ord, 0, sizeof(ord));

            //经纪公司代码
            strcpy(ord.BrokerID, g_chBrokerID);
            //投资者代码
            strcpy(ord.InvestorID, "12345");
            // 合约代码
            strcpy(ord.InstrumentID, "cn0601");
           ///报单引用
           strcpy(ord.OrderRef, "000000000001");
            // 用户代码
            strcpy(ord.UserID, g_chUserID);
            // 报单价格条件
            ord.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
```

```
            // 买卖方向
            ord.Direction = THOST_FTDC_D_Buy;
            // 组合开平标志
            strcpy(ord.CombOffsetFlag, "0");
            // 组合投机套保标志
            strcpy(ord.CombHedgeFlag, "1");
            // 价格
            ord.LimitPrice = 50000;
            // 数量
            ord.VolumeTotalOriginal = 10;
            // 有效期类型
            ord.TimeCondition = THOST_FTDC_TC_GFD;
            // GTD日期
            strcpy(ord.GTDDate, "");
            // 成交量类型
            ord.VolumeCondition = THOST_FTDC_VC_AV;
            // 最小成交量
            ord.MinVolume = 0;
            // 触发条件
            ord.ContingentCondition = THOST_FTDC_CC_Immediately;
            // 止损价
            ord.StopPrice = 0;
            // 强平原因
            ord.ForceCloseReason = THOST_FTDC_FCC_NotForceClose;
            // 自动挂起标志
            ord.IsAutoSuspend = 0;

            m_pUserApi->ReqOrderInsert(&ord, 1);
        }

        // 报单录入应答
        virtual void OnRspOrderInsert(CThostFtdcInputOrderField *pInputOrder,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
            // 输出报单录入结果
            printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);

            // 通知报单录入完成
            SetEvent(g_hEvent);
        };

        ///报单回报
        virtual void OnRtnOrder(CThostFtdcOrderField *pOrder)
```

```
        {
            printf("OnRtnOrder:\n");
            printf("OrderSysID=[%s]\n", pOrder->OrderSysID);
        }

        // 针对用户请求的出错通知
        virtual void OnRspError(CThostFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast) {
            printf("OnRspError:\n");
            printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
            printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
            // 客户端需进行错误处理
            {客户端的错误处理}
        }

    private:
        // 指向CThostFtdcMduserApi实例的指针
        CThostFtdcTraderApi *m_pUserApi;
    };


    int main()
    {
        // 产生一个CThostFtdcTraderApi实例
        CThostFtdcTraderApi *pUserApi =
CThostFtdcTraderApi::CreateFtdcTraderApi();
        // 产生一个事件处理的实例
        CSimpleHandler sh(pUserApi);
        // 注册一事件处理的实例
        pUserApi->RegisterSpi(&sh);

        // 订阅私有流
        //         TERT_RESTART:从本交易日开始重传
        //         TERT_RESUME:从上次收到的续传
        //         TERT_QUICK:只传送登录后私有流的内容
        pUserApi->SubscribePrivateTopic(TERT_RESUME);

        // 订阅公共流
        //         TERT_RESTART:从本交易日开始重传
        //         TERT_RESUME:从上次收到的续传
        //         TERT_QUICK:只传送登录后公共流的内容
        pUserApi->SubscribePublicTopic(TERT_RESUME);
```

```
    // 设置交易托管系统服务的地址，可以注册多个地址备用
    pUserApi->RegisterFront("tcp://172.16.0.31:57205");
    // 使客户端开始与后台服务建立连接
    pUserApi->Init();

    // 客户端等待报单操作完成
    WaitForSingleObject(g_hEvent, INFINITE);

    // 释放API实例
    pUserApi->Release();

    return 0;
}
```