上海期货
信息技术有限公司
Shanghai Futures Information Technology Co.,Ltd.

CTP 客户端开发指南（国际版）

应用高新技术，改善人类生活

修改时间	版本	说明	修改人
2016-03-03	1.0	1. 创建 2. 修改行情篇章说明	乔煜
2016-09-05	2.0	1. 红色加粗表示外盘新增的功能 2. 斜体划线表示外盘删除的功能	陈靖宇

更新：2016-09-05

前言

这是一份由上期技术提供的旨在帮助开发者快速了解、学习与综合交易平台进行对接的开发接口的文档。这 份文档提供了综合交易平台接口的整体介绍，解释了接口的运行机制，简述了使用相应的接口开发客户端的 常规步骤。文档中还会列举出其他开发人员咨询的问题及我们给出的回复。

这份指南是在参考了已有的一些文档的基础上，对以前的文档内容进行了总结和归纳，并补充了最新版本接 口中新增的特性。

如果读者认为本文档中还有可以完善的地方或还有需要重点指出但没有覆盖到的地方，欢迎提出宝贵的意见 和建议。

联系方式

上期技术,金融事业部

apiSupport@sfit.shfe.com.cn

更多上期技术实时信息请关注上期技术微信公众平台

目录

1 CTP

1.1 介绍

综合交易平台（Comprehensive Transaction Platform，CTP）是专门为期货公司开发的一套期货经纪业务管理 系统，由交易、风险控制和结算三大系统组成。

其中，交易系统主要负责订单处理、行情转发及银期转账业务，结算系统负责交易管理、帐户管理、经纪人 管理、资金管理、费率设置、日终结算、信息查询以及报表管理等，风控系统则主要在盘中进行高速的实时 试算，以及时揭示并控制风险。系统能够同时连通国内四家期货交易所，支持国内商品期货和股指期货的交 易结算业务，并能自动生成、报送保证金监控文件和反洗钱监控文件。

综合交易平台借鉴代表了国际衍生品领域交易系统先进水平的上期所“新一代交易所系统”的核心技术，采 用创新的完全精确重演的分布式体系架构。

综合交易平台是基于全内存的交易系统，支持7x24小时连续交易，运维人员不必每日启停系统，可以做到“一 键运维”。该特性使得综合交易平台新增交易中心以扩展业务规模时不用增加运维人力的成本。

支持FENS机制的“一键切换”多活交易中心也是目前市场上只有CTP系统实现了的特性。该机制使得交易系统 可在某个交易中心宕机的情况下立即切换到另一个备用交易中心，得以实现真真正正的连续交易。

综合交易平台公开并对外开放交易系统接口，使用该接口可以接收交易所的行情数据和执行交易指令。该接 口采用开放接口（API）的方式接入，早已在期货界已经形成事实上的行业标准。

综合交易平台mini版（CTP mini），是一款速度更快，更轻量级的CTP系统。相对于CTP来说，它追求的是更 小型化的配置，更节约化的资源配备。而用CTP的API开发的客户端程序也可以完美兼容CTP mini系统。

CTP国际版是上期技术最新推出的支持境外交易所交易业务的柜台系统。国际版扎根于上期技

术研发多年，应用了更快速更稳定的底层架构的CTP二代系统，拥有超越CTP一代的速度和稳

定性，且国际版支持境外交易所的行情转发、交易路由，风险控制等境外规则下执行的业务；

CTP国际版目前已经通过了CME Group（芝加哥商品交易所）的认证，且已经经过了实盘检

验。

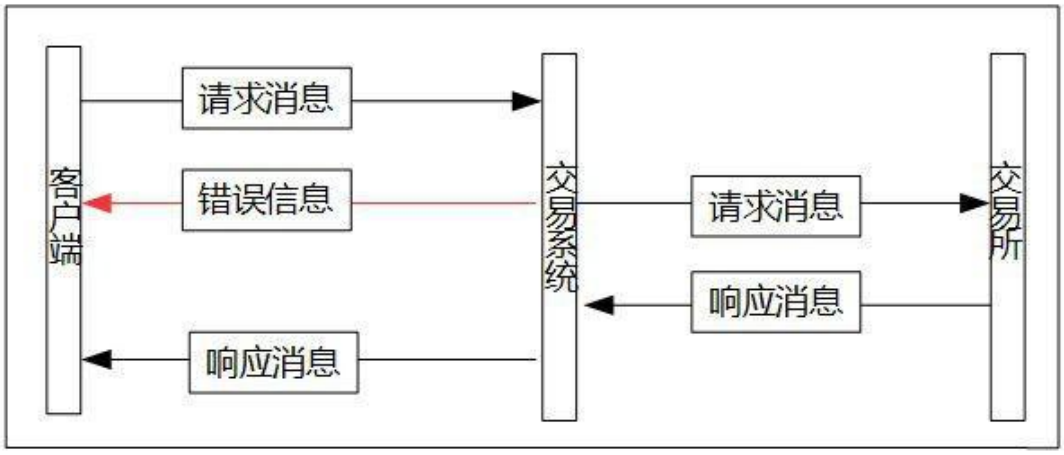
1.2 FTD 通讯协议

期货交易数据交换协议（Futures Trading Data Exchange Protocol， FTD），适用于期货交易系统与其下端交易客户端之间进行交易所需的数据交换和通讯。本章对 FTD 协议中的通讯模式和数据流进行相应的介绍。

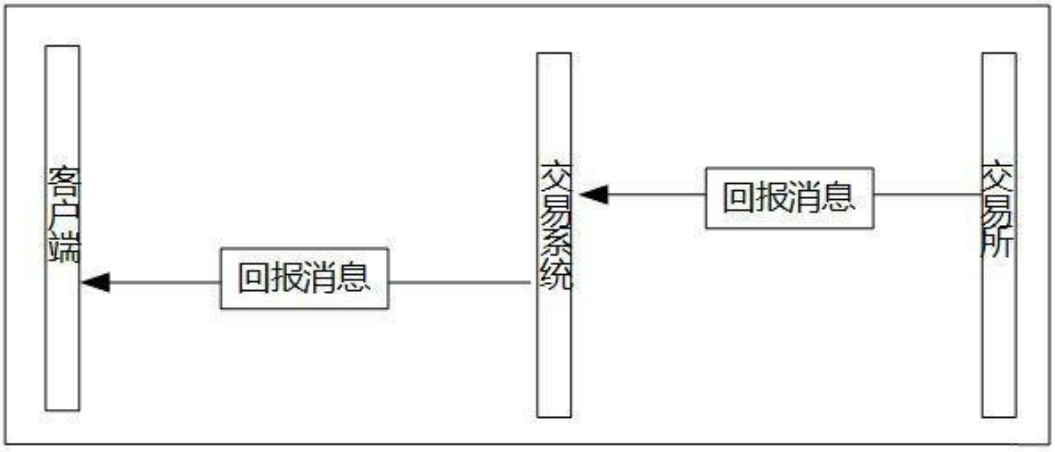
1.2.1 通讯模式

FTD 协议中的所有通讯都是基于某种通讯模式进行的。通讯模式用来说明通讯双方协同工作的方式。FTD 协议涉及到的通讯模式有三种：

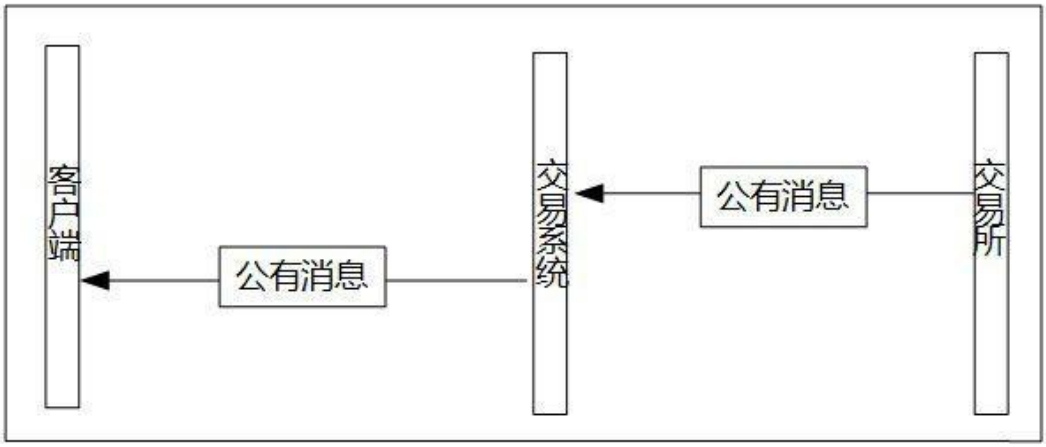
对话通讯模式是由客户端主动发起的通讯请求。该请求被交易系统端接收和处理，并向客户端返回响应。例如查询合约。这种通讯模式与普通的 Client/Server 模式相同。



私有通讯模式是指交易系统端主动向某个特定的客户端发送信息。例如报单回报。



广播通讯模式是指交易系统端主动向所有连接到系统上的客户端都发出相同的信息。如行情。



一般，CTP 系统中对话模式下被返回的消息成为响应。而私有模式和广播模式下被返回的消息被称为回报。

1.2.2 数据流

FTD 协议中需要区分的两个重要概念就是通讯模式和数据流。数据流表示的是一个单向或双向的，连续的，没有重复和遗漏的数据报文的序列。通讯模式则是一个数据流进行互动的工作模式。每个数据流应该对应一个通讯模式，但是一个通讯模式下可能有多个数据流。

一种实现方式可以为每种通讯模式构造一种数据流，产生了对话流，私有流和广播流。也可以为一个通讯模 式建立多种数据流，例如在对话通讯模式下建立两个流：查询流和交易流。广播模式下建立两个流：通知流 和行情流。FTD 只规定各个报文在哪个通讯模式下工作，但是不规定数据流的划分。

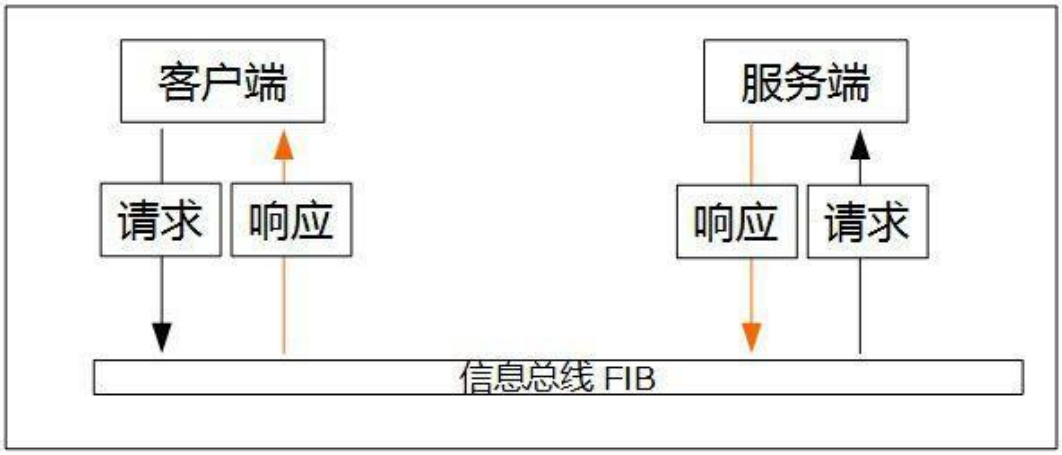
不同的通讯模式有着不同的数据流管理原则。在对话模式下，一个数据流是一个连接的过程，在这个连接内 将保障各个信息的完整性和有序性。但是，当连接断开后，重新连接将开始一个新的数据流，这个数据流和 原来的数据流没有直接的关系。如果客户端在提交了一个请求之后，未收到该请求的响应之前断开了连接， 则再次连接后，该请求的响应并不会被新的数据流接收。

而对于私有模式和广播模式，一个数据流对应一个交易日内的完成某项功能的所有连接。除非强制指定，否 则客户端会在重新连接之后，默认的从上次断开连接的地方继续接收下去，而不是从头开始。

1.3 两种数据交换模式

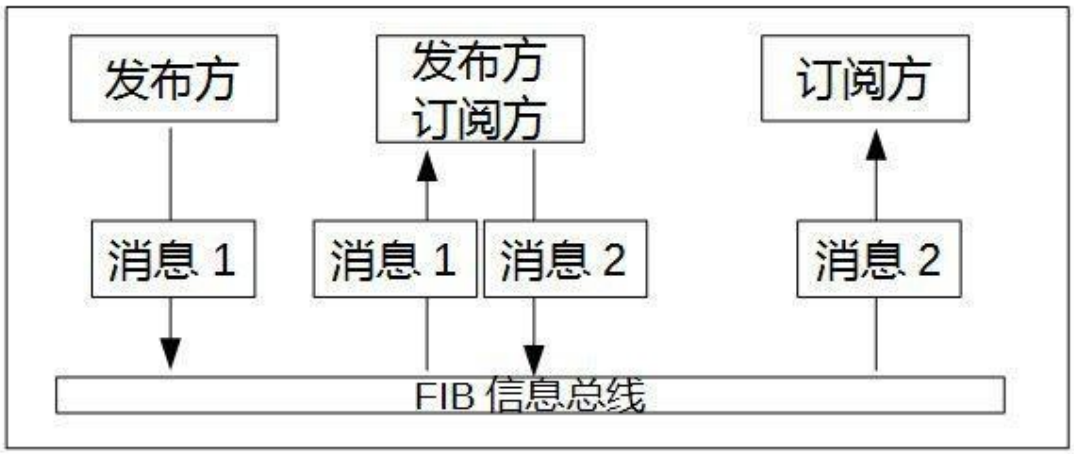
1.3.1 请求/应答模式

客户端程序（Client）产生一个请求，发向服务端程序（Server），服务端程序收到后进行处理，并把结果返 回给发出请求的客户端程序。



1.3.2 发布/订阅模式

发布/订阅模式是一种异步消息传输模式。发布者发布消息到主题，订阅者从主题订阅消息。发布者与订阅者 保持相对独立，不需要接触即可保证消息的传送。一个 FIB 应用即可作为发布者，也可作为订阅者。



1.4 接口文件

开发者可以通过其所在的期货公司向我司索要交易接口或直接到我司官网上下载已经发布的交易接口。两者的版本可能有差
异。 官网地址：

http://www.sfit.com.cn/

接口文件列表：

文件名	详情
ThostFtdcTraderApi.h	C++头文件 包含交易相关的指令，如报单。
ThostFtdcMdApi.h	C++头文件 包含获取行情相关的指令。
ThostFtdcUserApiStruct.h	包含了所有用到的数据结构。
ThostFtdcUserApiDataType.h	包含了所有用到的数据类型。
thosttraderapi.dll	交易部分的动态链接库和静态链接库。
thosttraderapi.lib	
thostmduserapi.dll	行情部分的动态链接库和静态链接库。
thostmduserapi.lib	
error.dtd	包含所有可能的错误信息。
error.xml	

1.5 通用规则

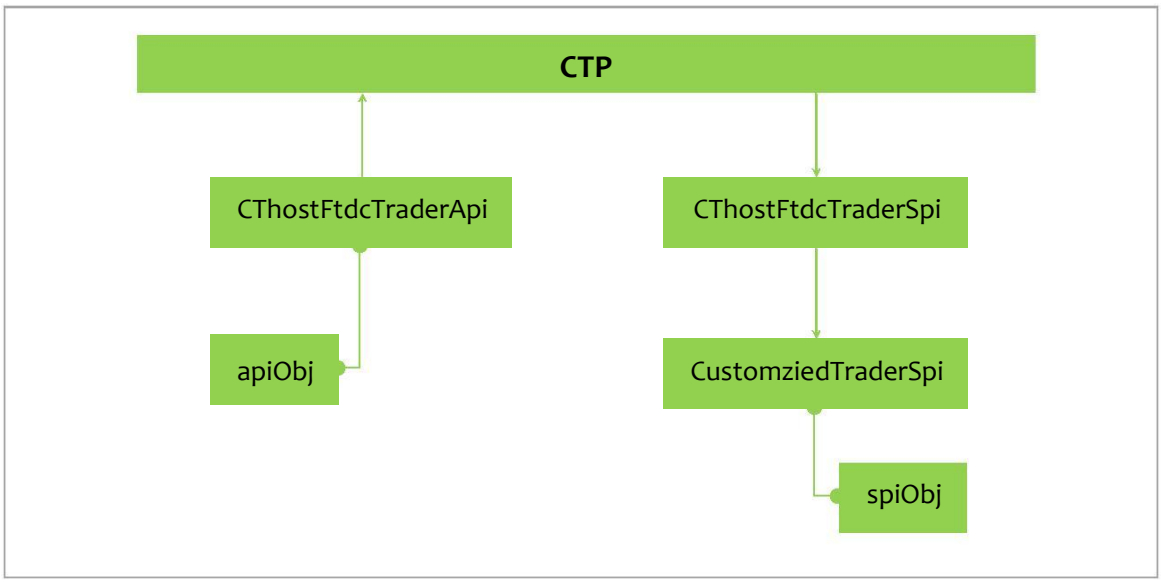
1.5.1 命名规则

消息	格式	示例
请求	Req------	ReqUserLogin
响应	OnRsp------	OnRspUserLogin
查询	ReqQry------	ReqQryInstrument
查询请求的响应	OnRspQry------	OnRspQryInstrument
回报	OnRtn------	OnRtnOrder
错误回报	OnErrRtn------	OnErrRtnOrderInsert

1.5.2 接口类

Spi（如 CThostFtdcTraderSpi），包含有所有的响应和回报函数，用于接收综合交易平台发送或交易所发送综合交易平台转发的信息。开发者需要继承该接口类，并实现其中相应的虚函数。

Api（如 CThostFtdcTraderApi），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。



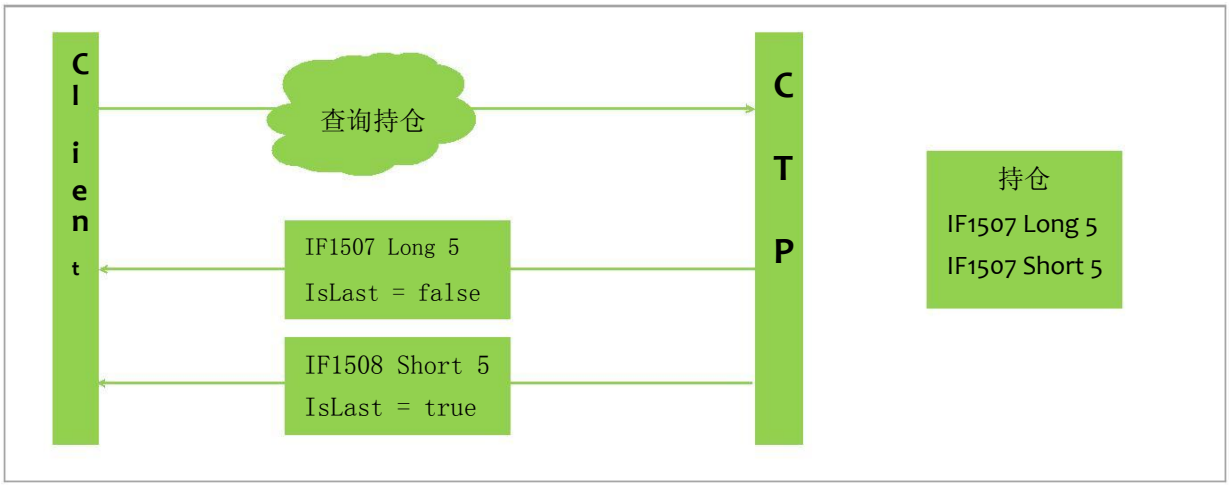
1.5.3 通用参数

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。

IsLast 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 **IsLast** 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。

举例：

查询持仓时，如果查询结果为多条记录，则会分多次回调返回，此时除了最后一次 IsLast 为 true 外，其余全 为 false。



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。

error.xml 文件中包含所有可能的错误信息。

1.5.4 接口的初始化步骤

以下步骤描述的是创建和初始化行情接口和交易接口的过程，通过以下步骤开启交易接口的工作线程。 风控接口和结算接口的使用暂时不包含在此文档中。

1. 创建继承自 SPI，并创建出实例，以及 API 实例。
   SPI 指 CThostFtdcTraderSpi 或 CThostFtdcMdSpi
   API 指 CThostFtdcMdApi 或 CThostFtdcTraderApi
2. 向 API 实例注册 SPI 实例。
3. 向 API 实例注册前置地址。交易接口需要注册交易前置地址，行情接口需要注册行情前置地址。
4. 订阅公有流（仅限交易接口，行情接口不需要）。用于接收公有数据，如合约在场上的交易状态。默认

模式是从上次断开连接处继续收取交易所发布数据（Resume 模式）开发者还可以指定全部重新获取（Restart），或从登陆后获取（Quick）。

5. 订阅私有流（仅限交易接口，行情接口不需要）。用于接收私有数据，如报单回报。默认模式是从上次断开连接处继续收取交易所发布数据（Resume 模式）开发者还可以指定全部重新获取（Restart），或从 登陆后获取（Quick）。
6. 初始化。（Init）
7. 等待线程退出。（Join）

示例代码和详细的解释见 DEMO 部分。

DEMO 开发

下面将会详细介绍使用行情和交易接口的步骤以及开发时要注意的要点，并会附上部分笔者自己在开发 DEMO 时编写的代码片段，仅供参考。

这份文档不会对编程细节做出解释，不包含可视化界面开发的指导。

2 行情 DEMO 开发

行情接口相对简单，容易上手，却可以帮助开发者快速掌握本接口的一些特点，对使用交易接口大有帮助。

2.1 准备工作

目前综合交易平台的行情和交易接口有 PC 上的 windows 版，linux 版，以及 Android 版和 iOS 版。
本文档中的代码片段均以 PC 上 windows 版为例，程序语言为 c/c++，开发工具为 visual studio 2010。

下面列出三个笔者推荐的 C/C++开发 IDE：

- **Visual Studio**
- **Code Blocks**
- **QT Creator**

导入接口文件

开发者需要将前面介绍过的所有的 API 文件复制到开发者创建的工程目录下。并将所有的头文件和静态、动态链接库导入工程中。

实现 **SPI** 接口类

开发者需要先继承行情接口类 **CThostFtdcMdSpi**，并实现需要实现的虚函数。

Response（查询流，对话流，会话内有效，不可恢复）	
OnFrontConnected	程序启动（Init）后，与后台网络连接建立成功后自动调用这个接口
OnFrontDisconnected	已经建立的网络连接因网络原因导致连接中断或登录失败被后台主动断开连接时调用这个接口
OnRspUserLogin	登录响应
OnRspUserLogout	登出响应
OnRspSubMarketData	订阅行情响应，指示订阅操作是否成功
OnRspUnSubMarketData	退订行情响应
OnRspError	其他错误回报，如后台无法识别消息协议则调用此接口

Return（私有流，日内有效，日内可恢复）	
OnRtnDepthMarketData	推送切片行情

行情接口工作原理简图

2.2 行情接口的初始化

```
1  CThostFtdcMdApi *api =
2            CThostFtdcMdApi::CreateFtdcMdApi(mdflowpath, true);
3  MarketDataCollector mdCollector(api);
4  api->RegisterSpi(&mdCollector);
5  api->RegisterFront(mdfront);
6  api->Init();
7  api->SetMarketDataTime(nTime);
8  api->Join();
```

行 1&2

使用函数 **CreateFtdcMdApi** 创建 **CThostFtdcMdApi** 的实例。其中第一个参数是本地流文件生成的目录。流文 件是行情接口或交易接口在本地生成的流文件，后缀名为.con。流文件中记录着客户端收到的所有的数据流 的数量。第二个参数描述是否使用 UDP 传输模式，true 表示使用 UDP 模式，false 表示使用 TCP 模式。

行 3

然后创建 SPI 实例。**MarketDataCollector** 是笔者自己创建的实体类，继承了 **CThostFtdcMdSpi**。

行 4

向 API 实例注册 SPI 实例。

行 5

向 API 实例注册前置地址。前置地址的格式为：tcp://127.0.0.1:17001。tcp 字段是开始字符串（不表示通讯模式），127.0.0.1 是托管服务器的行情前置地址，17001 是该行情前置的端口号。

行 6&8

初始化行情接口的工作线程。初始化之后，线程自动启动，并使用上一步中注册的地址向服务端请求建立连 接。

行 7

用户设置行情回调时间，nTime 行情回调时间间隔，单位 ms，最小为 ms，取消设置调用 UnSetMarketDataTime()。

综合交易平台接口都有独立的工作线程。如果开发者在进行可视化程序的开发，请务必注意线程冲突的问题。

2.3 登录

初始化之后，行情工作线程开启后将自动使用注册好的前置地址向服务端请求建立无身份验证的连接。连接建立后函数 **OnFrontConnected** 即会被调用。如果连接不能建立，或程序运行过程中该连接断开，则函数 **OnFrontDisconnected** 会被调用。

连接建立之后，即可使用函数 **ReqUserLogin** 请求登录系统，核心的数据结构是 **CThostFtdcReqUserLoginField**。

```
1  CThostFtdcReqUserLoginField req;
2  memset(&req, 0, sizeof(req));
3  strcpy(req.BrokerID, “2030”);
4  strcpy(req.UserID, “023526”);
5  strcpy(req.PassWord, “*******”);
6  int ret = pUserApi->ReqUserLogin(&req, ++requestId);
```

BrokerID 是期货公司的会员号。
UserID 是投资者在该期货公司的客户号。
Password 是该投资者密码。
上述三项为登录时必备的三个校验项，缺一不可，其余字段可根据情况自行添加。

该函数将会返回一个整数值（代码片段中的 **ret**），标志该请求是否被成功发送出去，而不代表该请求是否会 被服务端处理。

返回值取值

- **0:** 发送成功
- **-1:** 因网络原因发送失败
- **-2:** 未处理请求队列总数量超限。
- **-3:** 每秒发送请求数量超限。

综合交易平台接口中的大部分请求操作的返回值都如上述描述一样。

```
1  void onRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin,
2         CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
```

服务器端成功接收该用户登录请求，而且对期货公司会员号，投资者客户号，投资者登录密码检查无误后， 通过函数 **OnRspUserLogin** 发送服务端对该登录请求操作作出的响应。

开发者可以通过第二个参数 **pRspInfo** 中的 **ErrorID** 判断登录是否已经成功。如果 ErrorID 为 0，则登录成功，否则登录失败。

上述三个必要校验项（经纪公司代码、投资者代码、密码）互不匹配时就会返回“不合法登录”的错误信息。

登录成功后，第一个参数 **pRspUserLogin** 中包含了服务器端返回的一些基础数据，如 SessionID，FrontID，MaxOrderRef 以及各交易所服务器上的时间。

2.4 订阅行情

```
1  int ret = pUserApi->SubscribeMarketData(arrayOfConracts, sizeOfArray);
```

登录成功后，才可以进行行情的订阅。

客户端使用函数 **SubscribeMarketData** 进行行情订阅。第一个参数是一个包含所有要订阅的合约的数组，第二个参数是该数组的长度。

订阅行情时需指定要订阅行情的合约代码及该合约代码所属交易所，使用结构体为：

```
struct CThostFtdcSpecificInstrumentField
{
    ///合约代码
    TThostFtdcInstrumentIDType   InstrumentID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
};
```

订阅行情可以指定订阅单个合约，或者进行批量订阅，示例代码如下：

单合约

```
unsigned int countofinst=1;
CThostFtdcSpecificInstrumentField instFld;
memset(&instFld, 0, sizeof(instFld));
strcpy_s(instFld.ExchangeID, "CME");
strcpy_s(instFld.InstrumentID, "ZW1605"); m_Api-
>SubscribeMarketData(&instFld,countofinst);
```

批量

```
unsigned int countofinst=2;
CThostFtdcSpecificInstrumentField *instFld = new
CThostFtdcSpecificInstrumentField[countofinst];
strcpy_s(instFld[0].ExchangeID, "CME");
strcpy_s(instFld[0].InstrumentID, "ZW1605");

strcpy_s(instFld[1].ExchangeID, "CME");
strcpy_s(instFld[1].InstrumentID, "6A1606");

pUserApi->SubscribeMarketData(instFld,countofinst);
```

```
1  void OnRspSubscribeMarketData(
2     CThostFtdcSpecificInstrumentField * pSpecificInstrument,
3     CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

4  void OnRtnDepthMarketData(
5     CThostFtdcDepthMarketDataField * pDepthMarketData)
```

客户端发送订阅行情的请求之后，函数 OnRspSubMarketData 和函数 OnRtnDepthMarketData 将会被调用。

OnRspSubMarketData

如果客户端订阅行情的请求是不合法的，该函数返回服务器端给出的错误信息（pRspInfo）。 如果客户端发送的订阅请求是合法的，该函数也会被调用，而返回的信息则是“CTP：No Error”。

OnRtnDepthMarketData

行情订阅请求是合法的，服务端直接返回某合约的市场行情。切片行情推送频率由后台维护人员自行配置， 最低是每秒两次，另外可以配置为每秒 **4** 次、**8** 次等。

函数 ReqUserLogout 和函数 UnSubscribeMarketData 分别是用来退出登录和退订行情数据，用法和上述的请求登录和订阅行情操作一致。

注意：目前，通过 **ReqUserLogout** 登出系统的话，会先将现有的连接断开。客户端重新登录后系统会再重新建立一个新的连接，而 **SessionID** 会重置，因此 **MaxOrderRef** 一般也会重新从 **0** 计数。

2.5Demo 实例

```
#include  <iostream>
#define WINDOWS
#include "ThostFtdcMdApi.h"
#include <cstring>
using namespace  std;
using namespace CTPGlobal;
#define CONNECT_LOCAL
#pragma warning(disable:4996)
#define MDFront_MdAddress1 "tcp://172.19.125.39:41301"
int nTime = 1000;

class CMdSpiService : public CThostFtdcMdSpi
{

public:
	FILE *fp;

	CMdSpiService(CThostFtdcMdApi *userApi)
	{
		m_pMdApi = userApi;
		fp = fopen("out", "w");
	}

	virtual void OnFrontConnected()
	{

		cout << "On Front Connected" << endl;

		CThostFtdcReqUserLoginField field;
		strcpy(field.BrokerID, "8000");
		strcpy(field.UserID, "8000_admin");
		strcpy(field.Password, "1");

		int rtn = m_pMdApi->ReqUserLogin(&field, 1);
		cout << "send login " << " " << rtn << endl;

	}
	virtual void OnFrontDisconnected(int nReason){};

	virtual void OnRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
	{
		if (pRspInfo && pRspInfo->ErrorID != 0)
		{
			printf("Login ErrorID[%d] ErrMsg[%s]\n", pRspInfo->ErrorID, pRspInfo-
>ErrorMsg);
			return;
		}

		cout << "OnRspUserLogin " << endl;
```

```
                CThostFtdcSpecificInstrumentField field;

                strcpy(field.ExchangeID, "CME");
                strcpy(field.InstrumentID, "ES1609");

                int i = m_pMdApi->SubscribeMarketData(&field,1);
                cout << "Send SubMarketData " << i << endl;


        };

        virtual void OnRspSubMarketData(CThostFtdcSpecificInstrumentField *pSpecificInstrument,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
                //printf("ExchangeID[%s],InstrumentID[%s]\n", pSpecificInstrument->ExchangeID,
pSpecificInstrument->InstrumentID);

        }

        virtual void OnRspUnSubMarketData(CThostFtdcSpecificInstrumentField *pSpecificInstrument,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
                printf("Received OnRspUnSubMarketData\n");
        }



        ///深度行情通知
        virtual void OnRtnDepthMarketData(CThostFtdcDepthMarketDataField *pDepthMarketData)
        {
                //cout << "OnRtn" << endl;
                if (pDepthMarketData)
                {
                        printf("InstrumentID[%s]\n", pDepthMarketData->InstrumentID);
                        printf("AskVol1[%d], AskPri1[%f], BidVol1[%d], BidPri1[%f]\n",
                                pDepthMarketData->AskVolume1,
                                pDepthMarketData->AskPrice1,
                                pDepthMarketData->BidVolume1,
                                pDepthMarketData->BidPrice1);
                        printf("AskVol2[%d], AskPri2[%f], BidVol2[%d], BidPri2[%f]\n",
                                pDepthMarketData->AskVolume2,
                                pDepthMarketData->AskPrice2,
                                pDepthMarketData->BidVolume2,
                                pDepthMarketData->BidPrice2);
                        printf("AskVol3[%d], AskPri3[%f], BidVol3[%d], BidPri3[%f]\n",
                                pDepthMarketData->AskVolume3,
                                pDepthMarketData->AskPrice3,
                                pDepthMarketData->BidVolume3,
                                pDepthMarketData->BidPrice3);
                        printf("AskVol4[%d], AskPri4[%f], BidVol4[%d], BidPri4[%f]\n",
                                pDepthMarketData->AskVolume4,
                                pDepthMarketData->AskPrice4,
                                pDepthMarketData->BidVolume4,
                                pDepthMarketData->BidPrice4);
```

```
                        printf("AskVol5[%d], AskPri5[%f], BidVol5[%d], BidPri5[%f]\n",
                                pDepthMarketData->AskVolume5,
                                pDepthMarketData->AskPrice5,
                                pDepthMarketData->BidVolume5,
                                pDepthMarketData->BidPrice5);
                        printf("AskVol6[%d], AskPri6[%f], BidVol6[%d], BidPri6[%f]\n",
                                pDepthMarketData->AskVolume6,
                                pDepthMarketData->AskPrice6,
                                pDepthMarketData->BidVolume6,
                                pDepthMarketData->BidPrice6);
                        printf("AskVol7[%d], AskPri7[%f], BidVol7[%d], BidPri7[%f]\n",
                                pDepthMarketData->AskVolume7,
                                pDepthMarketData->AskPrice7,
                                pDepthMarketData->BidVolume7,
                                pDepthMarketData->BidPrice7);
                        printf("AskVol8[%d], AskPri8[%f], BidVol8[%d], BidPri8[%f]\n",
                                pDepthMarketData->AskVolume8,
                                pDepthMarketData->AskPrice8,
                                pDepthMarketData->BidVolume8,
                                pDepthMarketData->BidPrice8);
                        printf("AskVol9[%d], AskPri9[%f], BidVol9[%d], BidPri9[%f]\n",
                                pDepthMarketData->AskVolume9,
                                pDepthMarketData->AskPrice9,
                                pDepthMarketData->BidVolume9,
                                pDepthMarketData->BidPrice9);
                        printf("AskVol10[%d], AskPri10[%f], BidVol10[%d], BidPri10[%f]\n\n",
                                pDepthMarketData->AskVolume10,
                                pDepthMarketData->AskPrice10,
                                pDepthMarketData->BidVolume10,
                                pDepthMarketData->BidPrice10);
                }
        }


private:
        CThostFtdcMdApi *m_pMdApi;

};



int main()
{
        CThostFtdcMdApi *pMdApi;

        pMdApi = CThostFtdcMdApi::CreateFtdcMdApi();

        CThostFtdcMdSpi *pMdSpi;

        pMdSpi = new CMdSpiService(pMdApi);

        pMdApi->RegisterSpi(pMdSpi);

        int nFrontAddressPos = 0;
```

```
        pMdApi->RegisterFront(MDFront_MdAddress1);


        pMdApi->Init();

        pMdApi->SetMarketDataTime(nTime);

        pMdApi->Join();
        return 0;
}
```

3 交易 DEMO 开发

交易接口相对于行情接口来说，功能更多，更复杂。交易接口提供了投资者进行交易需要执行的操作的接口： 报单，撤单，银期转账，信息查询。

3.1 交易接口的初始化

与行情接口一样，开发者需要先继承接口类 **CThostFtdcTraderSpi**，并根据需要实现相应的虚函数。

```
1  CThostFtdcTraderApi *api =
2            CThostFtdcTraderApi::CreateFtdcTraderApi();
3  TradeChannel tdChnl(api);
4  api->RegisterSpi(&tdChnl);
5  api->RegisterFront(tdfront);
6  api->SubscribePrivateTopic(THOST_TERT_QUICK);
7  api->SubscribePublicTopic(THOST_TERT_QUICK);
8  api->Init();
9  api->Join();
```

初始化的步骤和代码基本上与行情接口（3.2 节）的初始化一致。

不同之处

1. 创建 API 实例时不能指定数据的传输协议。即第二行中的函数 **CreateFtdcTraderApi** 中只接受一个参数（即 流文件目录）。
2. 需要订阅公有流和私有流。 公有流：交易所向所有连接着的客户端发布的信息。
   比如说：合约场上交易状态。 私有流：交易所向特定客户端发送的信息。如报单回报，成交回报。

订阅模式

- Restart：接收所有交易所当日曾发送过的以及之后可能会发送的所有该类消息。
- Resume：接收客户端上次断开连接后交易所曾发送过的以及之后可能会发送的所有该类消息。
- Quick：接收客户端登录之后交易所可能会发送的所有该类消息。

3. 注册的前置地址是交易前置机的地址。

3.2 登陆系统

~~*身份认证*~~

~~*在登陆之前，服务端要求对客户端进行身份认证，客户端通过认证之后才能请求登录。身份认证功能是否启用在期货公司的业务人员使用的结算平台上是可以进行配置的。期货公司可以选择关闭身份认证功能，则客户端可不必进行身份认证。否则期货公司需要在结算平台上维护该客户端程序的认证码（AuthCode）。*~~

~~*请求进行身份认证使用的函数接口为ReqAuthenticate（请求身份认证）和OnRspAuthenticate（服务端返回的身份认证的响应）。*~~

登录

交易接口中请求登录的过程与行情接口（3.3 节）一致。

登录成功之后，响应函数 OnRspUserLogin 中的参数 pRspUserLogin 中包含了前置编号（FrontID），会话编号（SessionID），最大报单编号（MaxOrderRef）。

- 前置编号：客户端连接到的前置机的编号。
- 会话编号：客户端连接到前置机的连接会话编号。
- 最大报单编号：每一笔报单都有一个唯一的不重复的编号（OrderRef）。客户端若不赋值，服务端自动赋值；客户端若赋值，可从 MaxOrderRef 向上 逐一递增， 防止与其他的报单重复。

注意：目前，通过 **ReqUserLogout** 登出系统的话，会先将现有的连接断开，再重新建立一个新的连接，重新登录后 **SessionID** 会重置，因此 **MaxOrderRef** 一般也会重新从 **0** 计数。

3.3 修改密码

用户可通过 API 申请修改登录密码和资金账户密码。

修改登录密码

使用接口 ReqUserPasswordUpdate 修改登录密码，此密码仅适用于用户登录时身份校验；接口 OnRspUserPasswordUpdate 作为申请修改登录密码的响应被调用。

结构体为：

```
struct CThostFtdcUserPasswordUpdateField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType BrokerID;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///原来的口令
    TThostFtdcPasswordType OldPassword;
    ///新的口令
```

```
    TThostFtdcPasswordType NewPassword;
};
```

修改资金账户密码

使用接口 ReqTradingAccountPasswordUpdate 修改资金账户密码，此密码仅适用于用户进行银期转账时身份 校验；接口 OnRspTradingAccountPasswordUpdate 作为申请修改资金账户密码的响应被调用。 资金账户密码与上述的登录密码是两个独立维护的密码。

结构体为：

```
struct CThostFtdcTradingAccountPasswordUpdateField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType BrokerID;
    ///资金账号
    TThostFtdcAccountIDType  AccountID;
    ///原来的口令
    TThostFtdcPasswordType OldPassword;
    ///新的口令
    TThostFtdcPasswordType NewPassword;
    ///币种代码
    TThostFtdcCurrencyIDType  CurrencyID;
};
```

3.4 结算单确认

为了使投资者及时准确的了解自己的交易状况，如可用资金，持仓，保证金占用等，从而及时了解自己的风 险状况，综合交易平台要求投资者在每一个交易日进行交易前都必须对前一交易日的结算结果进行确认，多 次结算确认不影响交易。

结算确认相关的函数

请求函数	响应函数	描述
ReqQrySettlementInfo	OnRspQrySettlementInfo	请求查询结算单及响应；查询结算单时可查询指定交易日（格式为 20160303）的结算单；也可查询指定的月结单（格式为 201603），交易日填空表示查询最近一次结算单
ReqSettlementInfoConfirm	OnRspSettlementInfoConfirm	请求确认结算单及响应

ReqQrySettlementInfoConfirm	OnRspQrySettlementInfoConfirm	查询结算单确认的日期

3.5 查询接口

3.5.1 合约

查询合约：查询后台系统支持的合约的合约代码及合约相关信息。 请求函数：ReqQryInstrument

查询合约请求域：

```
struct CThostFtdcQryInstrumentField
{
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///合约在交易所代码
    TThostFtdcExchangeInstIDType ExchangeInstID;
    ///品种代码
    TThostFtdcProductIDType  ProductID;
    ///交易所子市场代码
    TThostFtdcSubExchangeIDType SubExchangeID;
};
```

查询合约时，如果请求域中不指定查询条件，即不对上述请求域中的任何字段赋值，则后台将通过响应函数 返回所有合约信息。相反，可以通过指定查询条件，来缩小查询结果的数量；如指定品种代码则查询得到该 品种下所有的合约，如果指定合约代码则只会得到一个该合约代码对应的合约信息。

响应函数：OnRspQryInstrument

查询合约响应域：CThostFtdcInstrumentField

响应域对应一个合约的相关信息，如果查询结果包含多条数据，则该响应函数被调用多次，每次对应一个合 约代码的信息。

另外，API 中提供查询交易所信息（ReqQryExchange / OnRspQryExchange）、查询产品信息(ReqQryProduct / OnRspQryProduct)、查询投资者信息(ReqQryInvestor / OnRspQryInvestor)函数可用于查询相关信息。其用 法与上述查询合约的用法类似，可参考使用。

3.5.2 资金账户

CTP 国际版中的资金账户的设计与内盘差异甚大，以下进行详细解释。

1. 国际版系统目前的设计方案是一个投资者代码（InvestorID）仅对应一个资金账户（AccountID），即不存在一对多或多对一的情况。

2. 每个资金账户（AccountID）下挂靠多个币种（CurrencyID），每个币种对应一份该币种下的资金信息。

3. 每个资金账户下默认挂靠一个基币（BaseCurrency），该基币币种可能是美元可能是人民币（根据系统设定），而基币下各个资金属性（如权益，可用资金等）等于其他所有币种按照系统内现行汇率兑换并加总所得。

4. 交易前风控仅根据基币上的可用资金额度去判断该笔交易是否可报单。如投资者针对某人民币计价的合约申报开仓，而该投资者人民币账户下资金不足，但是美元账户下资金充足，汇总到基币账户上资金是充足的， 因此系统允许该投资者开仓。

查询资金账户：查询该投资者代码在该经纪公司下的资金账户信息。 查询请求函数：ReqQryTradingAccount

查询请求域：

```
struct CThostFtdcQryTradingAccountField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType     BrokerID;
    ///资金账号
    TThostFtdcAccountIDType  AccountID;
    ///币种代码
    TThostFtdcCurrencyIDType CurrencyID;
};
```

查询资金账户时，如果不指定查询条件，则后台将返回当前登录账户下属的所有币种的资金账户信息。相反， 登录用户可以通过指定币种代码来查询某一币种当前的资金信息。 登录账户只能查询该用户自己的资金账户信息，因此经纪公司代码和资金账户如果要填，只能填自己的相应 信息。

查询响应函数：OnRspQryTradingAccount

查询响应域：CThostFtdcTradingAccountField

响应域对应一个币种下的所有资金相关信息，如权益，可用资金，手续费，盈亏等。当查询结果包含多个数 据条目时，响应函数将被调用多次，每次调用对应一个币种的资金信息。

国际币种将遵照国际惯例定义其币种代码，如美元为 USD，日元为 JPY，人民币为 CNY；而非惯例币种的币种代码可能是非国际惯例的定义格式。

基币为这些币种中的一个，但是在本系统中的币种代码固定用 ooo 表示。这么做的理由是基币是作为其他所 有币种的资金汇总信息而存在，而非一种实际的币种。举个例子，加入系统内只有人民币和美元两个币种， 同时指定基币为美元，而人民币对美元的汇率假设为 1：6. 则基币上的权益为：

权益（ooo） = 权益（CNY）/6 + 权益（USD）。

查询基币币种（ReqQryBaseCurrencyAccount / OnRspQryBaseCurrencyAccount）接口用于查询本系统内定义的基币对应的币种代码，返回币种代码为其他币种中的一种（如美元为 USD）。

查询汇率（ReqQryExchangeRate / OnRspQryExchangeRate）接口用于查询系统内存储的两个币种之间的汇率。 应用查询时可通过指定币种来精确查找要查询两个币种间的汇率，也可以不赋值进行模糊查询，此时后台将 返回所有的币种间汇率。

3.5.3 费率

本系统内保证金率及手续费率需分别通过单独的接口函数查询得到。 保证金率有两种，交易所层级保证金率以及投资者层级保证金率；一般后者在数额上会比前者高，这可能出 于经纪公司控制客户风险的一种手段。对投资者来说，只需要关心投资者层级的保证金率以及手续费率即可。

查询投资者保证金率（ReqQryInstrumentMarginRate / OnRspQryInstrumentMarginRate）

应用查询操作时，可通过指定合约代码及投资者代码来精确查找保证金率信息，也可通过模糊查找（即不赋 值条件）来获得当前持仓不为 0 的所有合约的保证金率。 保证金率按照境外惯例分为维持保证金率和初始保证金率，按照持仓方向分为多空两种，按照收取方式分为 按比例收取和按持仓数量收取。

查询投资者手续费率（ReqQryInstrumentCommissionRate / OnRspQryInstrumentCommissionRate）

应用查询操作时，可通过指定合约代码及投资者代码精确查找合约手续费率信息，也可通过模糊查找方式来 获得所有合约的手续费率。

模糊查询下，查询结果分为两种，一种是只细化到产品级的费率，这种情况下该产品下所有合约的费率都按照该费率收取，另一种是细化到合约级的费率，这种情况下仅该合约的费率按该费率收取。

手续费率按持仓方向分为多空两种，按收取方式分为按比例收取和按持仓数量收取。

3.6 持仓计算

查询持仓相关接口如下：

请求函数	响应函数	描述
ReqQryInvestorPosition	OnRspQryInvestorPosition	查询持仓（汇总）
ReqQryInvestorPositionDetail	OnRspQryInvestorPositionDetail	查询持仓明细
ReqQryInvestorPositionCombineDetail	OnRspQryInvestorPositionCombineDetail	查询组合持仓明细

【持仓明细】

CTP 系统根据来自交易所的成交记录生成的持仓明细记录，一笔成交记录对应一条持仓明细记录。

【持仓汇总】

CTP 系统将持仓明细记录按合约，持仓方向，开仓日期（仅针对上期所，区分昨仓、今仓）进行汇总。

持仓汇总记录中：

YdPosition 表示昨日收盘时持仓数量（≠ 当前的昨仓数量，静态，日间不随着开平仓而变化）

Position 表示当前持仓数量

TodayPosition 表示今新开仓 当前的昨仓数量 =

∑Position -∑TodayPosition

【组合持仓明细】

仅针对组合合约的持仓明细记录。

3.7 报单函数简介

ReqOrderInsert
请求报单

OnRspOrderInsert
综合交易平台交易核心返回的包含错误信息的报单响应

OnRtnOrder

交易系统返回的报单状态，每次报单状态发生变化时被调用。一次报单过程中会被调用数次：交易系统将报 单向交易所提交时，交易所撤销或接受该报单时，该报单成交时。

新增处理方式：报单在交易所端校验失败，被交易所拒绝后，**CTP** 系统同样通过此接口返回报错信息；其中的 **OrderStatusMsg** 字段包含报错信息。

OnErrRtnOrderInsert
此接口仅在报单被 CTP 端拒绝时被调用用来进行报错。

OnRtnTrade

如果该报单由交易所进行了撮合成交，交易所再次返回该报单的状态（已成交）。并通过此函数返回该笔成交。 报单成交之后，一个报单回报（OnRtnOrder）和一个成交回报（OnRtnTrade）会被发送到客户端，报单回报 中报单的状态为“已成交”。

建议客户端将成交回报作为报单成交的标志，因为 CTP 的交易核心在收到 OnRtnTrade 之后才会更新该报单的状态。如果客户端通过报单回报来判断报单成交与否并立即平仓，有极小的概率会出现在平仓指令到达 CTP 交易核心时该报单的状态仍未更新，就会导致无法平仓。

3.8 报单

报单使用的函数是 ReqOrderInsert，其中的核心数据结构是 CThostFtdcInputOrderField。
【通用的赋值代码】

```
CThostFtdcInputOrderField ord;
memset(&ord, 0, sizeof(ord));
strcpy(ord.BrokerID, “2030”);
strcpy(ord.InvestorID, “023526”);
strcpy(ord.InstrumentID, “rb1601”);
strcpy(ord.OrderRef, “”);
ord.Direction = THOST_FTDC_D_Buy;
ord.CombOffsetFlag[0] = THOST_FTDC_OF_Open;
ord.CombHedgeFlag[0] = THOST_FTDC_HF_Speculation;
ord.VolumeTotalOriginal = 1;
ord.ContingentCondition =
THOST_FTDC_CC_Immediately; ord.VolumeCondition =
THOST_FTDC_VC_AV; ord.MinVolume = 1;
ord.ForceCloseReason = THOST_FTDC_FCC_NotForceClose;
ord.IsAutoSuspend = 0;
ord.UserForceClose = 0;
```

不同的报单类型，可通过各个字段不同取值的组合来实现。
指定报单类型时，需要特别注意的两个字段为触发条件（**ContingentCondition** ）和报单价格类型（**OrderPriceType**）。

ContingentCondition	**OrderPriceType**
///立即 **#define THOST_FTDC_CC_Immediately '1'** ///止损 **#define THOST_FTDC_CC_Touch '2'** ///止赢 **#define THOST_FTDC_CC_TouchProfit '3'**	///任意价 **#define THOST_FTDC_OPT_AnyPrice '1'** ///限价 **#define THOST_FTDC_OPT_LimitPrice '2'** ///最优价 **#define THOST_FTDC_OPT_BestPrice '3'**

当报单类型为非止损型的限价单或市价单时，触发条件取值为立即执行（**THOST_FTDC_CC_Immediately**），限价时价格类型取 THOST_FTDC_OPT_LimitPrice，市价取 THOST_FTDC_OPT_AnyPrice。注意，限价单需指定价格，否则会报错。

当报单类型为止损单时，触发条件取值为止损（THOST_FTDC_CC_Touch）。 当报单类型为市价转限价时，则分别取值立即执行和最优价（THOST_FTDC_OPT_BestPrice）。

下表是 CTP 国际版目前已支持的报单类型列表

报单类型	触发条件	价格类型
限价单	立即执行	限价类型
市价单	立即执行	市价类型
限价止损单	止损	限价类型
市价止损单	止损	市价类型
市价转限价	立即执行	最优价类型

3.8.1 FOK & FAk

FOK（Fill or Kill）：是一种特殊的报单类型：该报单被交易所接收后，交易所会扫描市场行情，如果在当时的市场行情下该报单可以立即全部成交，则该报单会参与撮合成交，否则立即全部撤销。

FAK（Fill and Kill）：是一种特殊的报单类型：该报单被交易所接收后，交易所会扫描市场行情，在当时的行 情下能立即成交多少手即参与撮合成交多少手，剩余的则立即全部撤销。

综合交易平台接口是通过字段的组合来实现 FAK 和 FOK 指令的。

	FAK	FOK
TThostFtdcOrderPriceTypeType	THOST_FTDC_OPT_LimitPrice	THOST_FTDC_OPT_LimitPrice
TThostFtdcTimeConditionType	THOST_FTDC_TC_IOC	THOST_FTDC_TC_IOC
TThostFtdcVolumeConditionType	THOST_FTDC_VC_AV / THOST_FTDC_VC_MV	THOST_FTDC_VC_CV

字段 **MinVolume**

针对 FAK 指令，上表中 THOST_FTDC_VC_AV 代表任意数量，而 THOST_FTDC_VC_MV 代表最小数量。若为后者，投资者需要指定能成交手数的最小值。该字段表示立即能成交的手数如果小于该数量，则不会参与撮合成交，全部立即撤销。

3.8.2 报单序列号

在综合交易平台和交易所中，每笔报单都有 3 组唯一序列号，保证其与其他报单是不重复的。

FrontID + SessionID + OrderRef

登陆之后，交易核心会返回对应此次连接的前置机编号 FrontID 和会话编号 SessionID。这两个编号在此次连接中是不变的。

OrderRef 是报单操作的核心数据结构 CThostFtdcInputOrderField 中的一个字段。开发者可以让 OrderRef 在一次登录期间从 MaxOrderRef 起逐一递增，以保证报单的唯一性。开发者也可以选择不对它赋值，则交易核 心会自动赋予一个唯一性的值。

这组报单序列号可以由客户端自行维护，客户端可以通过该序列号随时进行撤单操作。

ExchangeID + TraderID + OrderLocalID

交易核心将报单提交到报盘管理之后由交易核心生成 OrderLocalID 并返回给客户端的。ExchangeID 合约所在 交易所的代码，TraderID 由交易核心选定返回。 与第一组序列号不同的是：该序列号是由综合交易平台的交易核心维护。

ExchangeID + OrderSysID

交易所在接收了报单之后，会为该报单生成报单在交易所的编号 OrderSysID。再经由综合交易平台转发给客户端。ExchangeID 是固定的。
客户端也可以通过这组序列号进行撤单操作。

这组序列号由交易所维护。

3.8.3 报单回报

使用的函数是 OnRtnOrder。核心数据结构为 CThostFtdcOrderField。

报单回报主要作用是通知客户端该报单的最新状态，如已提交，已撤销，未触发，已成交等。 每次报单状态有变化，该函数都会被调用一次。

VolumeTotalOriginal & VolumeTraded & VolumeTotal

上述三个字段分别对应该报单的原始报单数量，已成交数量和剩余数量。 如果报单是分笔成交，则每次成交都会有一次 OnRtnOrder 返回。

条件单触发时，交易核心会对该报单的合法性进行校验，如果校验失败，通过函数 OnRtnErrorConditionalOrder 返回校验失败的错误信息。

OrderStatus

- 0 全部成交
- 1 部分成交，订单还在交易所撮合队列中
- 3 未成交，订单还在交易所撮合队列中
- 5 已撤销
- a 未知-订单已提交交易所，未从交易所收到确认信息

3.8.4 成交回报

使用的函数是 OnRtnTrade。

函数返回报单成交回报，每笔成交都会调用一次成交回报。成交回报中只包含合约，成交数量，价格等信息。 成交回报只包含该笔成交相关的信息，并不包含该笔成交之后投资者的持仓，资金等信息。函数 ReqQryTradingAccount 用于查询投资者最新的资金状况。如保证金，手续费，持仓盈利，可用资金等。

建议客户端以成交通知为准判断报单是否成交，以及成交数量和价格；若以报单回报（状态为“部分成交或 全不成交”）为准，由于报单回报与成交回报之间存在理论上的时间差（微乎其微），而 CTP 后台是以成交

回报为准更新报单状态的，因此有可能导致平仓不成功.

3.9 撤单及改单

撤单使用的函数是 ReqOrderAction。核心的数据结构是 CThostFtdcInputOrderActionField。
撤单操作与报单操作很类似，但需要注意两个地方。

1. 字段 ActionFlag
   取值 THOST_FTDC_AF_Delete 表示撤单
   取值 THOST_FTDC_AF_Modify 表示改单
2. 序列号
   撤单或改单操作需要对应可以定位该报单的序列号。上一节最后介绍的三组报单序列号中的第一组和第二组都可以用来撤单。

目前，CTP 国际版支持对报单的数量及价格的修改。其中字段 **VolumeChange** 中存放要报单要修改成的数量， 而不是数量的变化。而字段 LimitPrice 中存放报单要修改成的价格。当然，如果只修改数量或只修改价格， 只需要其中之一即可。

多嘴说一句，目前 CTP 国际版对接的 CME 对改单有很复杂的逻辑，修改数量时不会改变该报单在报单簿中的 优先级，但是如果改单时原报单发生部分成交，且新指定的数量小于等于成交数量，那么剩余未成交报单将 被交易所撤销，即被撤单。而修改价格则必将导致报单失去报单簿中的优先级，而重新排队。

撤单响应和回报

OnRspOrderAction：撤单响应。交易核心返回的含有错误信息的撤单响应。 OnRtnOrder：交易核心确认了撤单指令的合法性后，将该撤单指令提交给交易所，同时返回对应报单的新状 态。 OnErrRtnOrderAction：交易所会再次验证撤单指令的合法性，如果交易所认为该指令不合法，交易核心通过 此函数转发交易所给出的错误。如果交易所认为该指令合法，同样会返回对应报单的新状态（OnRtnOrder）。

3.10 流文件

综合交易平台接口在初始化时会在本地生成一些流文件。这些流文件用来保存当日客户端程序接收到的公有 流，对话流，私有流等报文的数量。

流文件主要用来实现 Resume 模式下，重新收取交易所数据的功能，以及在使用综合交易平台风控接口时用来批量查询数据。

行情交易接口中，开发者对流文件只需要注意两点：

1. 客户端程序会对流文件进行大量的读写操作，如果客户端不对系统中的句柄数量进行管理的话，很可能出 现句柄被用光的情况。

2. 请注意在进行多账户开发时不能将多个账户收取的流文件放在同一个目录下，不然会造成一个账户能收到 回报，而其他的账户无法收取回报。

行情接口生成的流文件

Quotation API	
DialogRsp.con	Received dialog response data flow
QueryRsp.con	Received query response data flow
TradingDay.con	Trading Day

交易接口生成的流文件

Trade API	
DialogRsp.con	Received dialog response data flow
QueryRsp.con	Received query response data flow
TradingDay.con	Trading Day
Public.con	Received public return data flow
Private.con	Received private return data flow

3.11 流量控制

3.11.1 查询流量限制

交易接口中的查询操作的限制为：

- 每秒钟最多只能进行一次查询操作。
- 在途的查询操作最多只能有一个。

在途：查询操作从发送请求，到接收到响应为一个完整的过程。如果请求已经发送，但是未收到响应，则称 该查询操作在途。

上述限制只针对交易接口中的数据查询操作（ReqQryXXX），对报单，撤单，报价，询价等操作没有影响。

3.12 断线重连

客户端与服务端断开连接时，函数 OnFrontDisconnected 被调用，其中的参数 nReason 描述了断线的原因。

可能的原因

Decimal System(10D)	Hexadecimal(16H)	Explanation
4097	0x1001	网络读失败
4098	0x1002	网络写失败
8193	0x2001	读心跳超时
8194	0x2002	发送心跳超时
8195	0x2003	收到不能识别的错误消息

客户端与服务端的连接断开有两种情况：

- 网络原因导致连接断开
- 服务端主动断开连接 服

务器主动断开连接有两种可能：

- 客户端长时间没有从服务端接收报文，时间超时
- 客户端建立的连接数超过限制

心跳机制

综合交易平台采用心跳机制来确认客户端与服务端的连接是否正常。如果客户端没有从服务端接收报文，服 务端会发送心跳。

心跳机制目前仅在接口内部实现，并不会表现到客户端层面上来，即函数 OnHeartBeatWarning 并不会被调用。

客户端与服务端连接断开后，交易接口会自动尝试重新连接。

3.13Demo 实例

```
#include  <iostream>
#include "ThostFtdcTraderApi.h"
#include <string>
using namespace  std;
using namespace CTPGlobal;
#define CONNECT_LOCAL

#define Fctp1_TradeAddress "tcp://172.19.125.39:67809"

#define  ReqQry(fieldname) \
        CThostFtdc##fieldname##Field field;\
        memset(&field, 0, sizeof(field));

#define SendQry(fieldname)\
        m_pUserApi->Req##fieldname##(&field, 1); \
```

```
class CTraderBaseSpiService : public CThostFtdcTraderSpi
{
public:
	FILE *fp;

	CTraderBaseSpiService(CThostFtdcTraderApi *userApi)
	{
		m_pUserApi = userApi;
		fopen_s(&fp,"out", "w");
	}

	virtual void OnFrontConnected()
	{

		cout << "On Front Connected" << endl;

		CThostFtdcReqUserLoginField field;
		strcpy_s(field.BrokerID, "8000");
		strcpy_s(field.UserID, "8000_admin");
		strcpy_s(field.Password, "1");

		int rtn=m_pUserApi->ReqUserLogin(&field, 1);
		cout << "send login "<<" "<<rtn<< endl;

	}

	virtual void OnFrontDisconnected(int nReason){};

	virtual void OnRspError(CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast){};

	virtual void OnRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
	{
		if (pRspInfo && pRspInfo->ErrorID != 0)
		{
			printf("Login ErrorID[%d] ErrMsg[%s]\n", pRspInfo->ErrorID, pRspInfo-
>ErrorMsg);

			return;
		}
		//printf("%s\n", m_pUserApi->GetTradingDay());
		cout << "OnRspUserLogin "<< endl;

//		CThostFtdcInputOrderField order;
//		memset(&order, 0, sizeof(order));
//		strcpy_s(order.InstrumentID, "ES1609");
//		strcpy_s(order.InvestorID, "00008");
//		strcpy_s(order.InvestUnitID, "00008");
//		strcpy_s(order.BrokerID, "8000");
//		strcpy_s(order.ExchangeID, "CME");
//		order.Direction = THOST_FTDC_D_Sell;
//		order.LimitPrice = 2001.5;
//		order.VolumeTotalOriginal = 3;
//		order.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
//		order.ContingentCondition = THOST_FTDC_CC_Immediately;
```

```
//              order.TimeCondition = THOST_FTDC_TC_GFD;
//              order.VolumeCondition = THOST_FTDC_VC_AV;
//              strcpy_s(order.CombHedgeFlag,"1");
                //m_pUserApi->ReqOrderInsert(&order, 1);
                CThostFtdcQryTradeField field;
                memset(&field, 0, sizeof(field));
                strcpy_s(field.BrokerID, "8000");
                strcpy_s(field.InvestorID, "00008");
                //strcpy_s(field.InstrumentID, "ES1709");
                //strcpy_s(field.TradeID, "0000005");
                strcpy_s(field.ExchangeID, "CME");
                //strcpy_s(field.TradeTimeStart, "15:00:00");
                //strcpy_s(field.TradeTimeEnd, "17:00:00");
                strcpy_s(field.InvestUnitID, "00008");
                m_pUserApi->ReqQryTrade(&field, 1);
                //m_pUserApi->ReqQryTrade(&field, 1);
                //m_pUserApi->ReqQryTrade(&field, 1);
                //m_pUserApi->ReqQryTrade(&field, 1);
                printf("Send ReqQryTrade Ok\n");
        };

        virtual void OnRspQryTrade(CThostFtdcTradeField *pTrade, CThostFtdcRspInfoField *pRspInfo,
int nRequestID, bool bIsLast)
        {
                printf("OnRspQryTrade\n");
                if (pRspInfo && pRspInfo->ErrorID != 0)
                {
                        printf("OnRspQryTrade, ErrMsg=[%s]\n", pRspInfo->ErrorMsg);

                }
                if (pTrade)
                {
                        printf("volumn[%d],price[%f]\n", pTrade->Volume, pTrade->Price);
                }
        }
        virtual void OnRspOrderAction(CThostFtdcInputOrderActionField *pInputOrderAction,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
                if (pRspInfo && pRspInfo->ErrorID != 0)
                {
                        printf("OnRspOrderAction, ErrMsg=[%s]\n", pRspInfo->ErrorMsg);

                }
        }



        virtual void OnRspQryInstrument(CThostFtdcInstrumentField *pInstrument,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast){};

        virtual void OnRspOrderInsert(CThostFtdcInputOrderField *pInputOrder,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
                if (pRspInfo && pRspInfo->ErrorID != 0)
                {
```

```
                printf("OnRspOrderInsert, ErrMsg=[%s]\n", pRspInfo->ErrorMsg);
        }
        else
        {
                printf("Insert Order Ok!\n");
        }
    }

    virtual void OnRtnTrade(CThostFtdcTradeField *pTrade)
    {
        printf("OnRtnTrade Volume[%d]Price[%f]\n", pTrade->Volume,pTrade->Price);
    }


    virtual void OnRtnInstrumentStatus(CThostFtdcInstrumentStatusField *pTarget){};

    ///报单回报
    virtual void OnRtnOrder(CThostFtdcOrderField *pOrder)
    {
        printf("OnRtnOrder OrderStatus[%c]\n", pOrder->OrderStatus);
    }

    ///报单录入错误回报
    virtual void OnErrRtnOrderInsert(CThostFtdcInputOrderField *pInputOrder,
CThostFtdcRspInfoField *pRspInfo){};

    ///报单操作错误回报
    virtual void OnErrRtnOrderAction(CThostFtdcOrderActionField *pOrderAction,
CThostFtdcRspInfoField *pRspInfo){};

    ///查询基币
    virtual void OnRspQryBaseCurrencyAccount(CThostFtdcBaseCurrencyAccountField
*pBaseCurrencyAccount, CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
    {
        if (pRspInfo && pRspInfo->ErrorID != 0)
        {
                printf("OnRspQryBaseCurrencyAccount, ErrMsg=[%s]\n", pRspInfo->ErrorMsg);

        }

        if (pBaseCurrencyAccount)
        {
                cout <<"AccountID["<<pBaseCurrencyAccount->AccountID << "] CurrencyID[" <<
pBaseCurrencyAccount->CurrencyID << "]"<<endl;
        }

        if (bIsLast)
        {
                cout << "Query BaseCurrencyAccount end" << endl;
        }
    }

    virtual void OnRspQryInstrumentMarginRate(CThostFtdcInstrumentMarginRateField
*pInstrumentMarginRate,
```

```
                CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
        if (pRspInfo && pRspInfo->ErrorID != 0)
                {
                        printf("OnRspQryInstrumentMarginRate, ErrMsg=[%s]\n", pRspInfo->ErrorMsg);

                }

        if (pInstrumentMarginRate)
                {
                cout << "InvestorID[" << pInstrumentMarginRate->InvestorID << "] Maginrate[" <<
pInstrumentMarginRate->LMaintenMarginRatioByVolume << "]" << endl;
                fprintf(fp, "InvestorID[%s] Maginrate[%f]\n", pInstrumentMarginRate->InvestorID,
pInstrumentMarginRate->LMaintenMarginRatioByVolume);
                }

                if (bIsLast)
                {
                        cout << "Query nstrumentMarginRate end" << endl;
                        fclose(fp);
                }
        }



        ///查询持仓
        virtual void OnRspQryInvestorPosition(CThostFtdcInvestorPositionField *pInvestorPosition,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
                if (pRspInfo && pRspInfo->ErrorID != 0)
                {
                        printf("OnRspQryBaseCurrencyAccount, ErrMsg=[%s]\n", pRspInfo->ErrorMsg);

                }

                if (pInvestorPosition)
                {
                //      cout << "AccountID[" << pInvestorPosition->InvestorID << "] CurrencyID[" <<
pInvestorPosition->LongMarginRateByVolume << "]" << endl;
                }

                if (bIsLast)
                {
                        cout << "Query BaseCurrencyAccount end" << endl;
                }
        }

private:
        CThostFtdcTraderApi *m_pUserApi;

};
int main()
{
```

```
        CThostFtdcTraderApi *pUserApi;

        pUserApi = CThostFtdcTraderApi::CreateFtdcTraderApi();

        CThostFtdcTraderSpi *pUserSpi;

        pUserSpi = new CTraderBaseSpiService(pUserApi);

        pUserApi->RegisterSpi(pUserSpi);

        int nFrontAddressPos = 0;

        pUserApi->RegisterFront(Fctp1_TradeAddress);

        pUserApi->SubscribePrivateTopic(THOST_TERT_RESTART);

        pUserApi->SubscribePublicTopic(THOST_TERT_RESTART);

        pUserApi->Init();

        pUserApi->Join();
        return 0;
}
```