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

应用高新技术，改善人类生活

更新：2014-11-7

版本：2.0

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。

笔者联系方式
乔煜
金融事业部,上期技术
qiao.yu@sfit.shfe.com.cn
+86 021 20617728

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。

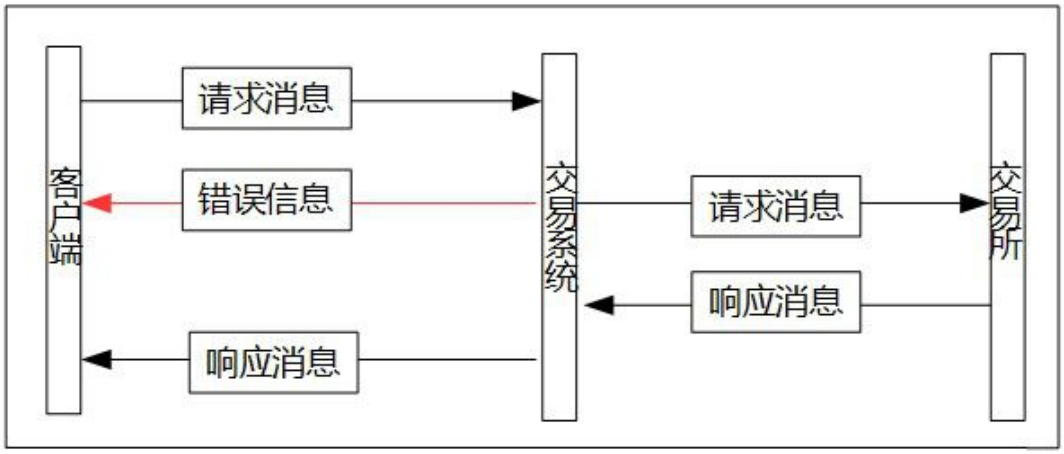
1.2 FTD 通讯协议

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。

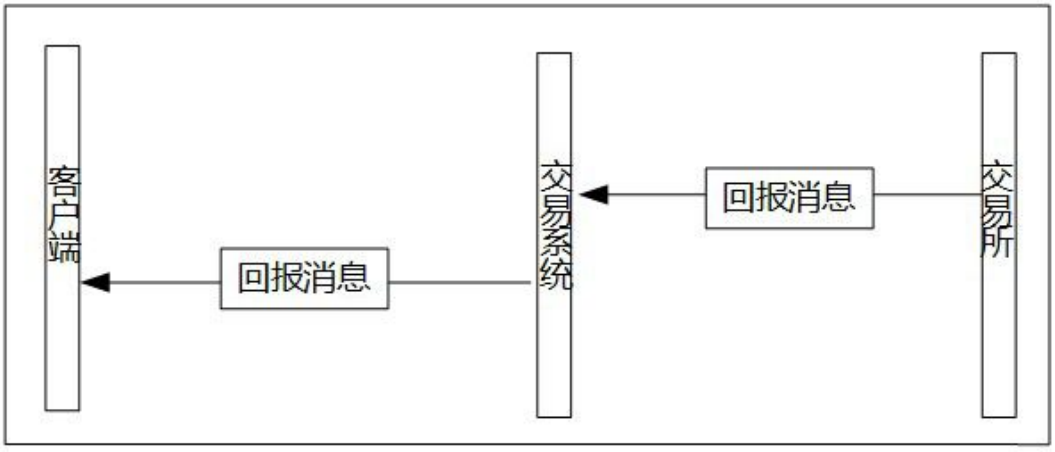
1.2.1 通讯模式

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：

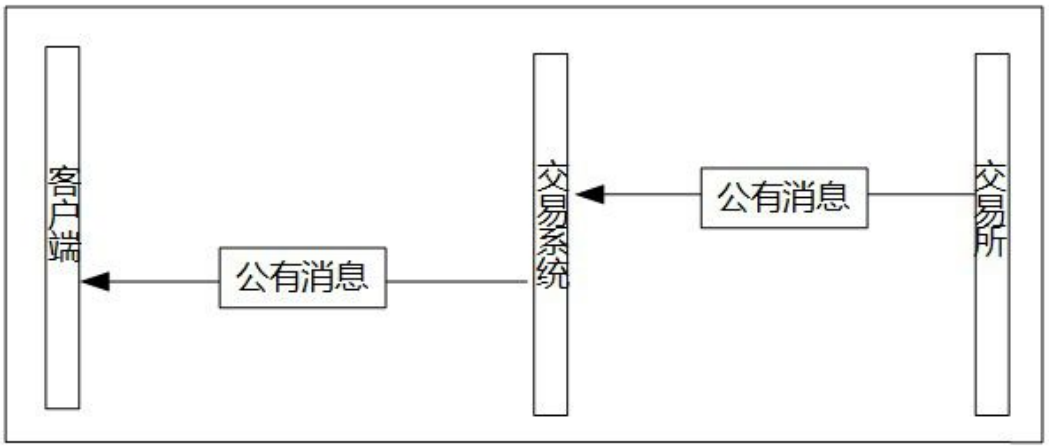
对话通讯模式是由客户端主动发起的通讯请求。该请求被交易系统端接收和处理，并向客户端返回响应。例如查询合约。这种通讯模式与普通的 Client/Server 模式相同。



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。

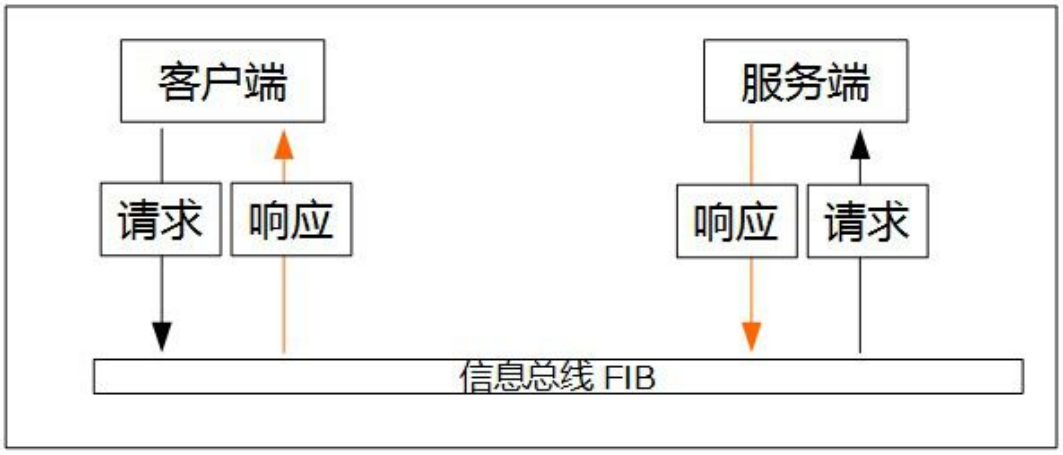
不同的通讯模式有着不同的数据流管理原则。在对话模式下，一个数据流是一个连接的过程，在这个连接内将保障各个信息的完整性和有序性。但是，当连接断开后，重新连接将开始一个新的数据流，这个数据流和原来的数据流没有直接的关系。如果客户端在提交了一个请求之后，未收到该请求的响应之前断开了连接，则再次连接后，该请求的响应并不会被新的数据流接收。

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

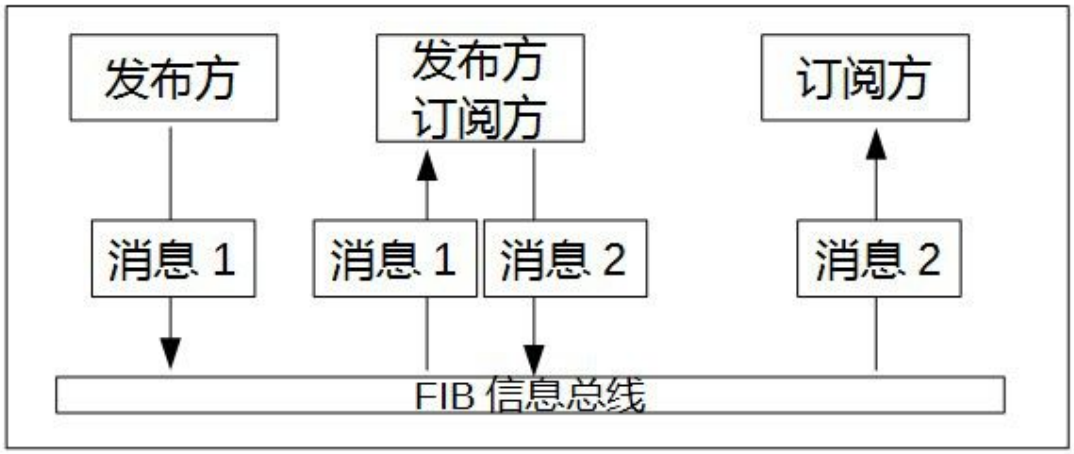
1.3.1 请求/应答模式

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。

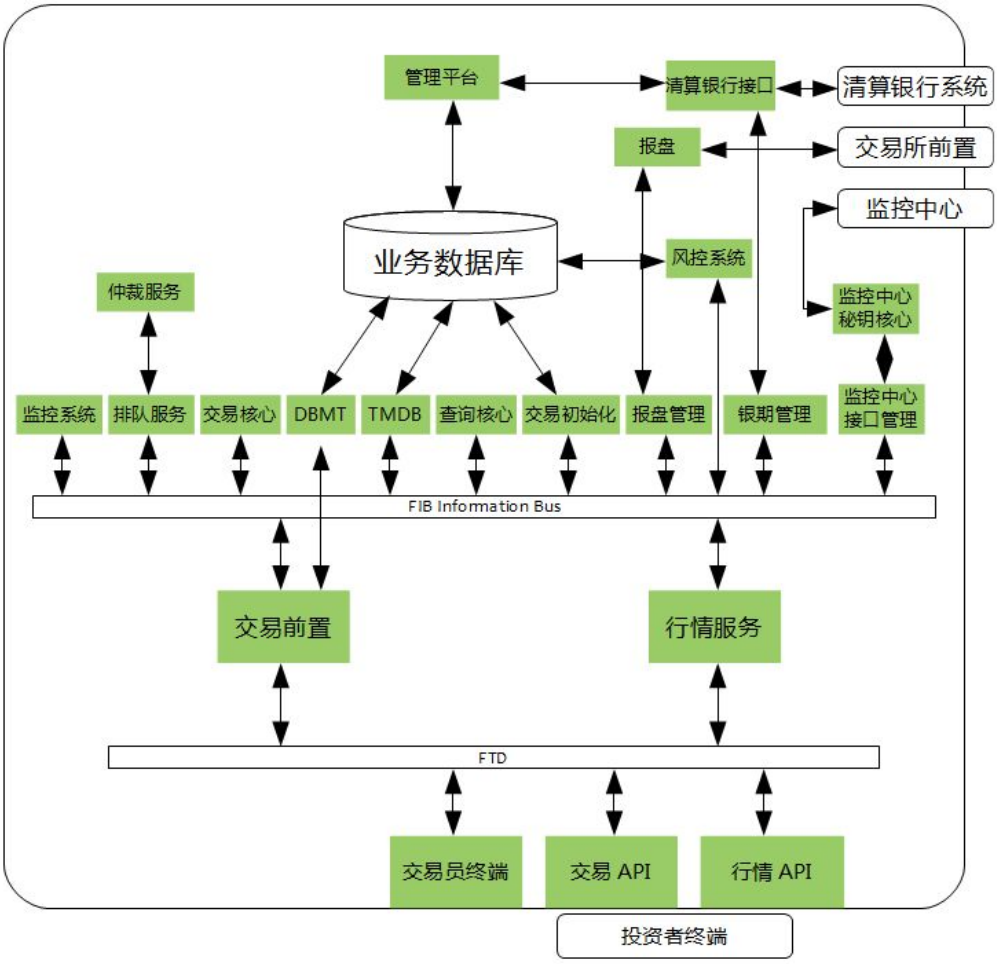


1.3.2 发布/订阅模式

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 CTP 系统架构



投资者终端：实现了综合交易平台的交易接口和行情接口的交易客户端，提供接收行情，交易，银期转账等功能。

交易员终端：实现了综合交易平台 UserAPI 接口，为期货公司交易员提供报单，银期转账，交易数据查询等功能。UserAPI 接口是面向期货公司层面的接口，可一次性控制的数据范围比交易 API 接口大得多，考虑到数据安全和权限划分的问题，暂时不会对期货公司和投资者开放。

FTD 通讯协议：期货交易数据交换协议。

交易前置：交易前置服务一方面通过 TCP 协议与交易终端连接，另一方面通过 FIB 总线与其他后台连接。交易前置主要负责与业务无关的通讯工作，可以分散交易系统的压力，降低交易系统的复杂度，提高安全性。交易前置的功能有三个：链路管理，协议转换和数据路由。

行情前置：行情前置一方面通过 FIB 从报盘管理订阅所有行情数据，另一方面通过 TCP 连接把该行情数据转发给订阅了某合约行情数据的交易终端。

FIB 信息总线：期货交易信息总线，是交易系统的通讯底层构件，为上层应用提供了数据包的封装，请求/应答通讯模式，以及发布/订阅通讯模式。

监控系统：旁路了交易系统部分交易数据用以应用数据监控，同时兼顾了交易系统物理部件的容量性能等检测。

仲裁服务：指导排队服务的状态切换。

排队服务：排队服务负责将交易请求序列化，发布交易序列，作为交易核心（tKernel）处理数据的来源。
交易核心：交易核心负责基于投资者的持仓，报单，成交以及出入金情况进行实时的资金和仓位计算的工作，做到事前风控。同时对所有的报单进行校验，驱动交易所报盘接口，及发布实时交易结果到 FIB 总线上。
交易查询：内置了与交易核心完全相同的内存数据库机构以及业务规则的实现，基于对投资者实时结算的结果更新内存数据库，通过 FIB 总线为交易客户端提供相关交易数据查询的服务。
DBMT：与业务数据库进行实时交互，将需要上下场的业务数据通过交易前置送到交易核心进行处理。
TMDB：向 FIB 总线订阅交易核心的处理结果，将相关的业务数据实时回写到物理数据库中供结算使用。
交易初始化：交易初始化主要承担两大职能：1，根据数据库的数据生成交易核心所需的初始化数据；2，向系统发出交易准备指令，使交易系统开始新一轮的交易（trading session）。
报盘管理：用于管理交易和行情报盘，使交易核心避免了直接处理报盘接口与交易所前置之间复杂的通讯情况，简化了交易核心的处理逻辑。
报盘：实现了交易所的行情和交易 API 接口，通过交易所提供的远程交易席位报单，收取报单，成交回报，以及获取行情等。
风控系统：旁路了交易系统排队机发布的交易序列以及交易核心发布的交易结果，对交易数据进行实时监控，同时提供风险试算和强平功能。
银期管理：用于管理银期转账接口。
清算银行接口（银期接口）：实现了各银行的银期转账接口，为交易系统与银行系统提供数据交互通道。
监控中心秘钥接口：用于从保证金监控中心查询期货公司秘钥。
监控中心接口管理：用于管理监控中心秘钥接口。
业务数据库：为整个系统提供物理数据存储数据来源。
管理平台：提供期货公司各种业务操作入口。

2 API

2.1 介绍

我司发布的 API 接口包含交易接口（Trader API），风控接口（Risk API），以及结算接口（CSV）。使用这三个接口都可以实现与上期技术综合交易平台系统的对接，从而进行交易，风控以及每日结算数据的保存。

交易接口主要用于获取交易所行情和下达交易指令，如订阅行情，下单，撤单，预埋单，银期转账，信息查询等。交易接口是三个接口中应用最广泛的接口，它的受众主要是终端软件开发商如快期，对交易终端有特殊需求的个人、机构或自营单位投资者。

风控接口可用于获取场上的实时数据如资金，持仓，保证金占用等，对投资者风险度进行试算，并在必要时采取强平手段以降低风险度，从而为期货公司的风控人员提供交易过程的风险揭示和风险管理服务。
上期技术为期货公司风控人员提供的 RcWin 操作平台就是使用风控接口开发而成。由于风控接口所管理的数据是面向所有投资者的，所以风控接口的受众主要是对风控系统有特殊需求的期货公司，而不对个人投资者开放。

结算接口是部署在能够访问到综合交易平台物理数据库的电脑上的一系列数据库指令，通过执行相应的指令，操作员可以获取到数据库中的相应数据信息，如投资者资料，成交信息，资金出入信息，持仓信息等。结算接口的主要受众也是期货公司，不针对个人投资者开放。

2.1.1 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	

2.2 通用规则

2.2.1 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

2.2.2 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。

2.2.3 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。例如：如果只有一个数据包，该响应的 IsLast 值为 true。如果有两个，则第一个响应的 IsLast 值为 false，第二个为 true。

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。

2.2.4 交易接口的初始化步骤

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 SPI 和 API 实例。
   这里的 SPI 是指开发者创建的自己的类，该类已经继承了接口中的 SPI 接口类（CThostFtdcTraderSpi 或 CThostFtdcMdSpi）。而 API 即接口中提供的 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。

3 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。

3.1 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。这份文档中的代码片段均以 PC 上 windows 版为例，程序语言为 c/c++。

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。

OnFrontConnected	客户端到行情前置的无身份验证连接建立之后，这个函数会被调用，用于说明连接已经建立。连接建立之后，才能请求登录。
OnFrontDisconnected	如果客户端到行情前置的无身份验证连接建立失败，这个函数被调用。其中的参数说明连接失败的原因。
OnRspUserLogin	交易系统对客户端的请求登录消息作出的响应。
OnRspSubMarketData	交易核心认为客户端请求订阅行情的消息不合法时通过这个函数返回错误信息。

OnRspUnSubMarketData	交易核心认为客户端请求退订行情的消息不合法时通过这个函数返回错误信息。
OnRtnDepthMarketData	通过这个函数返回行情信息。频率是每秒两次。
OnRspError	如果交易系统无法识别客户端发送的请求消息，就通过这个函数返回错误信息。
OnHeartBeatWarning	如果超过一定时间在客户端和系统之间没有任何消息交换发生，这个函数会发送心跳用来说明客户端到系统服务器之间的连接是活跃的。

3.2 行情接口的初始化

```
1    CThostFtdcMdApi* pUserApi =
2          CThostFtdcMdApi::CreateFtdcMdApi(pathOfLocalFiles, true);
3    QCTPMdSpi* pUserSpi=new QCTPMdSpi(pUserApi);
4    pUserApi->RegisterSpi(pUserSpi);
5    pUserApi->RegisterFront(ipAddressOfmdFront);
6    pUserApi->Init();
7    pUserApi->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。

如果开发者要使用组播行情，则需要使用该函数的第三个参数 **bIsMulticast**。在第二个参数和第三个参数赋值均为 true 时，行情将以组播的形式传输。

注意：组播行情只能在内网中使用。

流文件的详细介绍可参见“客户端开发注意事项”章节。

行 3

然后创建 SPI 实例。QCTPMdSpi 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（不是通讯模式，不可更改），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。

注意： 综合交易平台的接口可以通过三种方式传输行情：TCP，UDP 和组播（Multicast）。 同一个前置地址可以使用三种传输模式中的任一种，而不需要为每个前置地址指定一个传输地址。

行 6&7

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。

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。

3.3 登录系统

上一节中，初始化之后，行情工作线程就会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**。

```
CThostFtdcReqUserLoginField req;
memset(&req, 0, sizeof(req));
strcpy(req.BrokerID, BROKERID);
strcpy(req.UserID, USERID);
strcpy(req.Password, PASSWORD);
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。

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。

```
void QCTPMdSpi::OnRspUserLogin(
        CThostFtdcRspUserLoginField *pRspUserLogin,
        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。

一般密码错误时会返回“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。

3.4 订阅行情

```
int ret=pUserApi->SubscribeMarketData(arrayOfContracts, 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。

```
void QCTPMdSpi::OnRspSubMarketData(
        CThostFtdcSpecificInstrumentField *pSpecificInstrument,
        CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
```

```
void QCTPMdSpi::OnRtnDepthMarketData(
          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）。即使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。频率是**每秒两次**。

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。

3.5 订阅和接收询价

综合交易平台的接口中，做市商可以从行情中订阅来自投资者的询价。
目前不同的交易所暂定的对投资者询价和做市商接收询价的处理方式不同，而且各家交易所对询价，报价等功能的实现程度各有不同，出于开发效率的考量，综合交易平台接口目前暂时把所有的投资者询价和做市商接收询价的功能都在行情接口中实现。

四家交易所已公开的实现询价的方式

上期所&中金所
询价流通过交易接口传输。交易接口中提供有投资者询价的接口和做市商接收询价的接口。做市商接收询价不需要订阅，因为服务端在做市商登录交易接口时对其账号进行身份验证，并根据其在交易所的权限，发送对应合约的询价。

大商所&郑商所
询价流通过行情接口传输。投资者仍是通过交易接口进行询价。而做市商则必须先订阅投资者询价，服务端才会转发询价流。

综合交易平台接口坚持与交易所的实现方式保持一致的原则。随着交易所对询价部分的实现程度逐渐完善，综合交易平台接口中询价部分的实现也会随之进行调整。

询价、报价相关的接口函数

行情接口中

请求函数	响应函数	描述
SubscribeForQuoteRsp	OnRspSubForQuoteRsp	做市商订阅投资者询价
UnSubscribeForQuoteRsp	OnRspUnSubForQuoteRsp	做市商取消订阅投资者询价
	OnRtnForQuoteRsp	做市商接收投资者询价

交易接口中

函数	描述
OnRtnForQuoteRsp	做市商接收投资者询价
ReqForQuoteInsert	投资者发送询价请求

行情接口中订阅询价的用法与订阅行情一样。交易接口不需要订阅。

投资者主动询价将在“交易 DEMO 开发”中详细介绍。

4 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。

但是交易接口与行情接口在用法上并没有太大差别。

4.1 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。

```
CThostFtdcTraderApi* pUserTraderApi =
            CThostFtdcTraderApi::CreateFtdcTraderApi(pathOfLocalFiles);
QCTPTradingSpi* pUserTraderSpi = new QCTPTradingSpi(pUserTraderApi);
pUserTraderApi->RegisterSpi(pUserTraderSpi);
pUserTraderApi->RegisterFront(ipAddressOfTradeFront);
pUserTraderApi->SubscribePublicTopic(THOST_TERT_RESTART);
pUserTraderApi->SubscribePrivateTopic(THOST_TERT_RESUME);
pUserTraderApi->Init();
pUserTraderApi->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。

4.2 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。

4.3 结算单确认

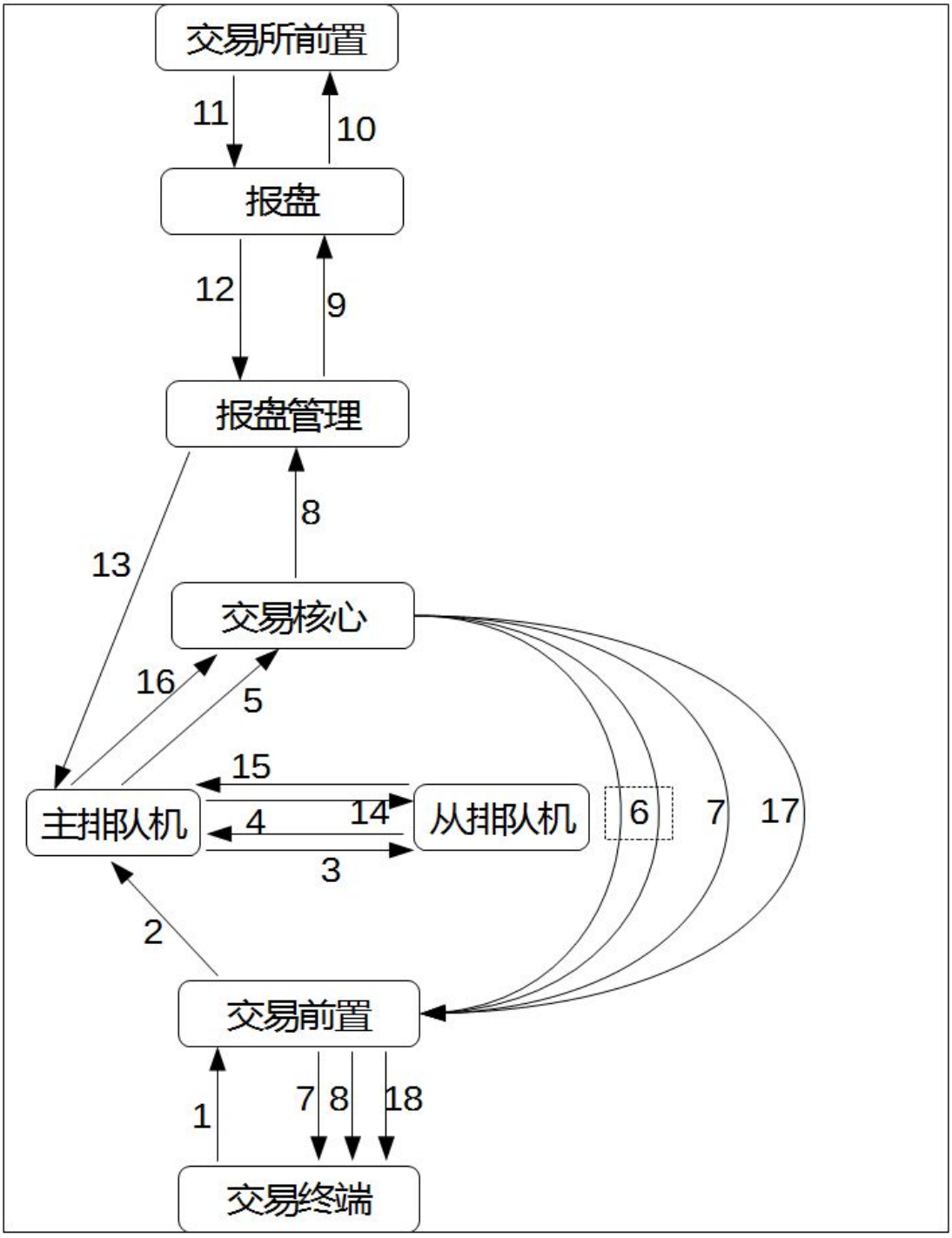
为了使投资者及时准确的了解自己的交易状况，如可用资金，持仓，保证金占用等，从而及时了解自己的风险状况，综合交易平台要求投资者在每一个交易日进行交易前都必须对前一交易日的结算结果进行确认。

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
ReqSettlementInfoConfirm	OnRspSettlementInfoConfirm	请求确认结算单及响应

ReqQrySettlementInfoConfirm	OnRspQrySettlementInfoConfirm	查询结算单确认的日期

4.4 报单流程



1. 交易终端通过交易接口向交易前置提交报单申请。
2. 主排队机从前置机订阅交易申请报文。
3. 主排队机将发布的交易序列报文发送给从排队机要求确认。
4. 从排队机收到待确认报文后将相关报文写入流水文件，并立即返回报文确认信息。
5. 交易核心从主排队机订阅交易序列报文。
6. 交易核心对收到的交易序列报文做合法性检查，检查出错误的交易申请报文后就会返回给交易前置一个包含错误信息的报单响应报文，交易前置立即将该报文信息转发给交易终端。如果检出为合法的交易申请报文，交易核心也会返回一个报单响应报文到交易前置，但是该报文不会被交易前置返回给交易终端。
7. 两种情况：1，交易前置从交易核心订阅到错误的报单响应报文，以对话模式将该报文转发给交易终端。2，交易核心返回给交易前置的响应报文是正确的，交易前置立即以私有模式返回对应报单的报单回报到交易终端（图中前置机到交易终端 8）。
8. 两个过程：1，交易前置在订阅到交易核心的报单回报后，以私有模式将该报单回报发送到交易终端。2，交易核心向交易前置发送了第一个报单回报后，立即产生向交易所申请该报单插入的申请报文，该报文被报盘管理订阅。
9. 报盘管理订阅到交易所报盘插入申请报文后，将该报文转发到对应报盘接口。
10. 报盘收到报盘管理的报单申请报文后，通过交易所提供的交易接口将该笔报单发送到交易所。
11. 报盘通过交易接口从交易所前置接收报单回报以及成交回报或出错的报单响应报文。
12. 报盘将从交易所接收到的报单回报及成交回报或报单响应汇总到报盘管理。
13. 主排队机从报盘管理订阅从交易所返回的报单信息。
14. 主排队机将报文信息序列化后发送给从排队机进行确认。
15. 从排队机收到待确认报文后将该报文写入流水，并立即返回报文确认信息。
16. 交易核心从主排队机订阅所有的报单以及成交回报信息。
17. 交易前置从交易核心订阅所有交易核心发布的交易结果数据。
18. 交易前置将订阅到的交易结果数据分发到各交易终端。

4.5 处理报单的函数

ReqOrderInsert
请求报单，对应于上一节中的第 1 步。

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，对应于上一节中的第 7 步的第 1 种情况。

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（上述流程 8 中的第 2 个过程），交易所撤销或接受该报单时，该报单成交时。

OnErrRtnOrderInsert
报盘将通过交易核心检查的报单发送到交易所前置，交易所会再次校验该报单。如果交易所认为该报单不合法，交易所会将该报单撤销，将错误信息返回给报盘，并返回更新后的该报单的状态。当客户端接收到该错误信息后，就会调用 OnErrRtnOrderInsert 函数， 而更新后的报单状态会通过调用函数 OnRtnOrder 发送到客户端。 如果交易所认为该报单合法，则只返回该报单状态（此时的状态应为：“尚未触发”）。

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”。但是仍然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。

4.6 报单

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

```
CThostFtdcInputOrderField orderInsert;
memset(&orderInsert, 0, sizeof(orderInsert));
strcpy(orderInsert.BrokerID, BROKERID);
strcpy(orderInsert.InvestorID, INVESTORID);
strcpy(orderInsert.InstrumentID, INSTRUMENTID);
strcpy(orderInsert.OrderRef, orderRef);

orderInsert.Direction = THOST_FTDC_D_Buy;
orderInsert.CombOffsetFlag[0] = THOST_FTDC_OF_Open;
orderInsert.CombHedgeFlag[0] = THOST_FTDC_HF_Speculation;
// volume of the instrument
orderInsert.VolumeTotalOriginal = 1;
// type of volume condition
// THOST_FTDC_VC_AV : any volume
// THOST_FTDC_VC_MV : minimum volume
// THOST_FTDC_VC_CV : all the volume
orderInsert.VolumeCondition = THOST_FTDC_VC_AV;
// minimum volume
orderInsert.MinVolume = 1;
// reason of forcible close
// see TThostFtdcForceCloseReasonType in head file for details
orderInsert.ForceCloseReason =
        THOST_FTDC_FCC_NotForceClose;
// is auto suspend
// 0:no,1:yes
orderInsert.IsAutoSuspend = 0;
// is forcible close
// 0:no, 1:yes
orderInsert.UserForceClose = 0;
```

对不同类型的报单，需要对相应的字段进行相应的赋值。

限价单

```
// price type of order
// see TThostFtdcOrderPriceTypeType in head file for details
//THOST_FTDC_OPT_LastPrice;
//THOST_FTDC_OPT_AnyPrice;
orderInsert.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
// price
orderInsert.LimitPrice = YourPrice;
// valid type
// see TThostFtdcTimeConditionType in head file for details
//THOST_FTDC_TC_IOC;
orderInsert.TimeCondition = THOST_FTDC_TC_GFD;
```

市价单

```
// price type of order
// see TThostFtdcOrderPriceTypeType in head file for details
//THOST_FTDC_OPT_LastPrice;
//THOST_FTDC_OPT_LimitPrice;
orderInsert.OrderPriceType = THOST_FTDC_OPT_AnyPrice;
// price
orderInsert.LimitPrice = 0;
// valid type
// see TThostFtdcTimeConditionType in head file for details
//THOST_FTDC_TC_GFD;
orderInsert.TimeCondition = THOST_FTDC_TC_IOC;
```

条件单

```
// type of condition
// THOST_FTDC_CC_Immediately : immediately match with current
market price
// see TThostFtdcContingentConditionType in head files for details
orderInsert.ContingentCondition = THOST_FTDC_CC_Immediately;
// stop price
// triggered when price falls or rises to this price
orderInsert.StopPrice = 2150;
// price type of order
// see TThostFtdcOrderPriceTypeType in head file for details
//THOST_FTDC_OPT_LastPrice;
//THOST_FTDC_OPT_AnyPrice;
orderInsert.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
// price
orderInsert.LimitPrice = YourPrice;
// valid type
// see TThostFtdcTimeConditionType in head file for details
//THOST_FTDC_TC_IOC;
orderInsert.TimeCondition = THOST_FTDC_TC_GFD;
```

条件单简介

条件单是一个带有触发条件的指令。该触发条件可以以市场上的最新行情为基准，也可以以时间为基准。比如：一个投资者有 1 手 IF1410 的空头持仓，并希望在市场价低于 2200 时买入平仓，他就可以使用条件单。这样当行情波动到满足该条件时，该报单就会被自动触发报出，而不需要他本人时刻盯着电脑屏幕去监视市场行情。

有效的使用条件单，可以做出限价止损指令（stop-and-limit order）和触及市价指令（market-if-touched order）。

4.6.1 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。

字段 CombOffsetFlag

该字段用来指定该报单是开仓，平仓还是平今仓。

```
req.CombOffsetFlag[0] = THOST_FTDC_OF_OPEN
```

该字段是一个长度为 5 的字符数组，可以同时用来描述单腿合约和组合合约的报单属性。单腿合约只需要为数组的第 1 个元素赋值，组合合约需要为数组的第 1&2 个元素赋值。字符取值为枚举值，在头文件“ThostFtdcUserApiStruct.h”中可以查到。

```
// open position
#define THOST_FTDC_OF_Open '0'
// close position
#define THOST_FTDC_OF_Close '1'
// force close position
#define THOST_FTDC_OF_ForceClose '2'
// close position which was opened within current trading day
#define THOST_FTDC_OF_CloseToday '3'
// close position which was opened before current trading day
#define THOST_FTDC_OF_CloseYesterday '4'
// Force off
#define THOST_FTDC_OF_ForceOff '5'
// Local Force close
#define THOST_FTDC_OF_LocalForceClose '6'

typedef char TThostFtdcOffsetFlagType;
```

上期所的持仓分今仓（当日开仓）和昨仓（历史持仓），平仓时需要指定是平今仓还是昨仓。

上述字段中，若对上期所的持仓直接使用 THOST_FTDC_OF_Close，则效果同使用 THOST_FTDC_OF_CloseYesterday。若对其他交易所的持仓使用了 THOST_FTDC_OF_CloseToday 或 THOST_FTDC_OF_CloseYesterday，则效果同使用 THOST_FTDC_OF_Close。

注意：现在四家交易所平仓顺序统一为先开先平，大商所在此基础上还有先平单腿仓再平组合仓的规则。

4.6.2 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。客户端也可以通过这组序列号进行撤单操作。

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。

4.6.3 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。

4.6.4 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。

查询合约保证金率

使用的函数是 ReqQryInstrumentMarginRate。

该函数只能查询单腿合约的保证金率。组合合约的保证金率可以通过查询两条单腿合约的保证金率，然后通过交易所的规则去计算。

4.7 预埋单

预埋单是一种能且仅能在非交易时段（集合竞价前或交易节之间的休息时间）报入并在新的交易时段开始时被触发并执行一定指令的一种指令。它包含预埋报单和预埋撤单。

预埋报单（Parked Order Insert）被触发时，一个新的报单被报入交易所。

预埋撤单（Parked Order Action）被触发时，一个撤单指令被报入交易所，请求撤销某笔已经存在的报单。

预埋报单使用的函数是 ReqParkedOrderInsert。核心数据结构为 CThostFtdcParkedOrderField。

预埋保单的用法与报单类似。对应的响应函数为 OnRspParkedOrderInsert，该函数用于返回交易核心给出的响应。

预埋撤单使用的函数是 ReqParkedOrderAction。对应的响应函数为 OnRspParkedOrderAction。

删除预埋报单使用的函数是 ReqRemoveParkedOrder，该函数用于删除已经报入但未触发的某笔预埋报单。

删除预埋报单使用的函数是 ReqRemoveParkedOrderAction，该函数用于删除已经报入但未触发的某笔预埋报单。

函数 ReqQryParkedOrder 和函数 ReqQryParkedOrderAction 分别用于查询预埋报单和预埋撤单情况。对应的响应函数分别是 OnRspQryParkedOrder 和 OnRspQryParkedOrderAction。

预埋报单或撤单在被触发后即转化为一个普通的报单或撤单指令，之后的处理过程与报单或撤单过程完全一样。

目前，预埋单被触发后报入交易所的报单的报单引用是由综合交易平台生成，暂时不受客户端控制。

4.8 撤单

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
   由于国内的交易所目前只支持撤单，不支持改单操作，因此函数 ReqOrderAction 只支持撤单操作，字段 ActionFlag 的赋值目前只能是 THOST_FTDC_AF_Delete。
2. 序列号
   撤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都可以用来撤单。

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）。

4.9 询价和报价

投资者询价

综合交易平台中投资者询价使用的函数是 ReqForQuoteInsert。核心数据结构是 CThostFtdcInputForQuoteField。只需要传入要询价的合约以及该询价的引用即可。
函数 OnRspForQuoteInsert 是在交易核心认为该询价指令不合法时被调用，返回错误信息。如果询价指令合法，则不会有信息返回。

做市商接收询价

交易接口中有做市商权限的账号在登录系统之后会自动接收到投资者的询价，不需要请求或订阅。接收询价的函数是 OnRtnForQuoteRsp。

做市商报价

报价使用的函数是 ReqQuoteInsert。核心数据机构是 CThostFtdcInputQuoteField。

```
CThostFtdcInputQuoteField quote;
memset(&quote, 0, sizeof(quote));
strcpy(quote.BrokerID, IDofBrokerageFirm);
strcpy(quote.InvestorID, IDofInvestor);
strcpy(quote.InstrumentID, IDofInstrument);
strcpy(quote.QuoteRef, RefNumberOfQuote);
strcpy(quote.AskOrderRef, RefNumberOfAskOrder);
strcpy(quote.BidOrderRef, RefNumberOfBidOrder);
quote.AskPrice=PriceofAskOrder;
quote.BidPrice=PriceofBidOrder;
quote.AskVolume=VolumeofAskOrder;
quote.BidVolume=VolumeofBidOrder;

// offsetflag of ask order
quote.AskOffsetFlag=THOST_FTDC_OF_Open;

// offsetflag of bid order
quote.BidOffsetFlag=THOST_FTDC_OF_Open;

// hedgeflag of ask order
quote.AskHedgeFlag=THOST_FTDC_HF_Speculation;

//hedgeflag of bid order
quote.BidHedgeFlag=THOST_FTDC_HF_Speculation;
```

函数 OnRspQuoteInsert 是当交易核心认为该报价指令不合法，返回错误信息时被调用。
函数 OnRtnQuote 是当交易核心认为该报价指令合法，并向交易所提交时调用。同时综合交易平台会为该报价衍生出两笔报单，一并提交到交易所，此时还会调用函数 OnRtnOrder。
自 v6.3.0 版本开始，报价的接口更改为：新增了两个字段 AskOrderRef 和 BidOrdreRef。
上述两个报价中的新增字段会分别赋值给该报价衍生的两笔买卖报单的报单引用（OrderRef）。如果客户端没有赋值，交易核心会自行赋值。

撤销报价

做市商撤销报价使用函数 ReqQuoteAction。其用法与撤销报单类似。
撤销报价使用的序列号为 QuoteRef+SessionID+FrontID。

该函数在撤销报价时，会把其对应的未成交的衍生报单一起撤销。比如，某报价衍生的两笔报单中的一边已经部分成交，这种情况下，撤销报价的结果就是：该笔报价以及两笔报单中未成交的部分全都会被撤销。

做市商也可以单独撤销某笔报单。方法与上一节中描述的撤单方法一样。

4.10 行权

行权的函数是 ReqExecOrderInsert。核心数据结构是 CThostFtdcInputExecOrderField。

```
CthostFtdcInputExecOrderField execOrderReq;
memset(&execOrderReq, 0, sizeof(execOrderReq));
strcpy(execOrderReq.BrokerID, IDofBrokerageFirm);
strcpy(execOrderReq.InvestorID, IDofInvestor);
strcpy(execOrderReq.InstrumentID, IDofInstrument);
strcpy(execOrderReq.ExecOrderRef, RefNumberofExecOrder);
execOrderReq.volume = volumeofInstrument;

// for SHFE, it should be THOST_FTDC_OF_CloseToday or
// THOST_FTDC_OF_CloseYesterday
execOrderReq.OffsetFlag = THOST_FTDC_OF_Close;
execOrderReq.HedgeFlag = THOST_FTDC_HF_Speculation;

// to exercise or to abandon
execOrderReq.ActionType = THOST_FTDC_ACTP_Exec;

// long or short position to hold after exercising
execOrderReq.PosiDirection = THOST_FTDC_PD_Long;

// whether to hold future positions after exercising
// CFFEX: use THOST_FTDC_EOPF_UnReserve
// DCE/CZCE: use THOST_FTDC_EOPF_Reserve
execOrderReq.ReservePositionFlag =
                              THOST_FTDC_EOPF_UnReserve;

// whether to close future positions automatically
// CFFEX: use THOST_FTDC_EOCF_AutoClose
// DCE/CZCE: use THOST_FTDC_EOCF_NotToClose
execOrderReq.CloseFlag = THOST_FTDC_EOCF_AutoClose;
```

如果交易核心认为该行权指令不合法，函数 OnRspExecOrderInsert 会被调用。如果交易核心认为该行权指令合法，函数 OnRtnExecOrder 会被调用。

行权之后，如果收到返回信息“未执行”，则说明该行权指令已经被交易所接收。由于交易所都是在盘后结算时进行行权配对，所以盘中行权返回“未执行”的信息。

四家交易所对期权行权的处理

中金所

实值期权自动行权，虚值期权自动放弃。虚值期权不允许强制执行。期权行权之后立即平仓（因为期权行权日与对应标的期货交割日为同一天）。

上期所

实值期权自动行权，虚值期权自动放弃。投资者可以主动申请执行虚值期权。期权行权之后可以选择保留或不保留期货持仓。

大商所

期货公司在会服（大商所官方网站）设置了自动行权时会自动行权，否则需要投资者主动申请行权或放弃。

郑商所

实值期权自动行权，虚值期权自动放弃。投资者可以主动申请执行或放弃。

5 补充说明

5.1 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

5.2 流量控制

5.2.1 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。

5.2.2 报单流量限制

报单流量限制是由期货公司通过在系统中配置相关参数实现限制的。
不进行配置的情况下，默认流量限制为：

- 在一个连接会话（Session）中，每个客户端每秒钟最多只能发送 6 笔交易相关的指令（报单，撤单等）。
- 同一个账户同时最多只能建立 6 个会话（Session）。

注意：报单操作超过限制时，不会有错误返回，只是报单会处于排队状态，等待处理。该限制包含包括报单，撤单，查询等所有请求操作。

5.3 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。

超时

网络超时会导致服务端主动与客户端断开连接。

默认的读超时为：120 秒，写超时为：60 秒，心跳超时为：80 秒。

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，频率是每 5 秒一次。