Comprehensive Transaction Platform
Trade and Quotation

Application Programming Interface

Version：4.2

Publish Date: Nov. 6, 2009

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

I. History List of File Changes

version	Change date	memo
V4.2	2009-11-6	Create the English Version

Directory

Chapter 1. Introduction

Comprehensive Transaction Platform (CTP) is developed as a future trade and broker information management system, including trade server, risk management server and settlement information management subsystem and so on.

The API we will introduct is used to communicate with the CTP trade server. From the API, investor could receive quotation data from SHFE, DCE and CZCE, send trading directive to the three exchanges, receive corresponding response and trade status return.

1.1. Background

In 2006, after shanghai future information technology coporation completed the New Generation Exchange System (NGES) development for SHFE; we transferred the success experience to our CTP development.

April 2007, we obtained the first order of CTP from the future broker system field in China. For great effort of recent three years, investors trading via CTP have expanded all over the world and the broker quantity increased to thirty in China.

1.2. Introduction of API files

The API of CTP trade server is based on C++ library and carrys out

the communication between trade client and CTP trade server. Trade clients includes CTP standard trade client (such as Q7, pobo, weisoft etc. developed by third part) free used by all investor of CTP, and trade tools only used personally (developed by investors or their partners). By using the API, trade client could insert or cancel common order and condition order, contract status fire order, query order or trade record and get the current Account and position status. This library includes the following files:

File Name	File Description
ThostFtdcTraderApi.h	Trading interface c++ head file
ThostFtdcMdApi.h	Quotation interface c++ head file
ThostFtdcUserApiStruct.h	Defines all data type
ThostFtdcUserApiDataType.h	Defines all data structure
thosttraderapi.dll	The dynamic link library of trading interface.
thosttraderapi.lib	
thostmduserapi.dll	The dynamic link library of quotation interface.
thostmduserapi.lib	
error.dtd	The api error code and information in xml format.
error.xml	

Note: using compilers such as MS VC 6.0，MS VC.NET 2003 and so on, please turn on the multi-thread option in compile setting.

Chapter 2. Archetecture

The communication protocol between CTP API and CTP trade server is futures TradingData Exchange protocol (FTD), an information exchange protocol based on TCP.

2.1. Communication Mode

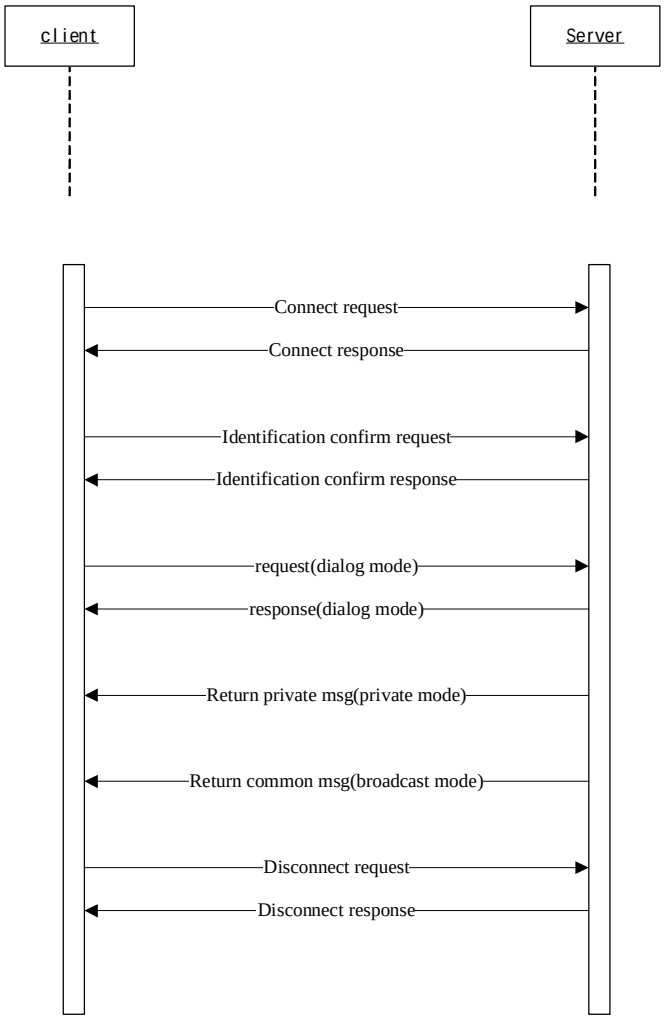
In FTD protocol, communication mode includes the following three modes:

- Dialog mode, client submits a request to CTP, and CTP will return corresponding results.
- Private mode, CTP sends private messages to specific client those messages are all private notify message such as order status or trade confirmation.
- Broadcast mode, CTP publishs common information to all clients registerd to ctp.

Each communication mode is not confined to one network connection. That means, with one network connection, the client can use all the three communication modes, or several different client connection can use the same communication mode. For example, the client can use broadcast mode to receive instrument status change message, and at the same time receive its own private message such as order confirmation

message.

The following diagram explains communication process of these three modes:



2.2. Data Stream

CTP support dialog, private and broadcast communication mode. With dialog communication mode, dialog data stream and query data stream could be transmitted.

Dialog and query data stream are both bi-direction data stream, the

client application submit request and CTP server return response. CTP server doesn’t maintain the status of dialog and query data stream. when something is wrong such as reconnect happens, the dialog and query data stream will be reset after the communication rebuilding and data on fly will lost .

With private communication mode, private data stream is transmitted. Private data stream is a unidirectional data stream, using it, the CTP server send private message to the corresponding client application. Private message includes risk notice, order status, order confirmation, trade confirmation. The private data stream is reliable, when the client application lost connection with CTP server, at any time in the same trading day, the client application can reconnect the CTP server with specified sequence number of it’s own private data flow and without any risk of lost those private trading data.

With the broadcast communication mode, public data stream is transmitted. It is a unidirectional and reliable data stream just like the private data stream, the only difference between them is the broad cast communication data will broadcast to all connecting client application. Its main useage is pulic instrument status or any public important message.

Chapter 3. Progamming Interface type

CTP trade API provides the two interfaces, CThostFtdcTraderApi and CThostFtdcTraderSpi. The CTP quotation API provides CThostFtdcMdApi and CThostFtdcMdSpi. The four interfaces implement FTD protocol; the client could submit requests by invoking functions of the CThostFtdcXXXApi and receive the CTP response with reloaded callback functions of their own object inherited from CThostFtdcXXXSpi.

3.1. Dialog mode programming interface

Communication functions of the interface with dialog mode usually like the following definition:

request： int CThostFtdcTraderApi::ReqXXX(
CThostFtdcXXXField *pReqXXX,
int nRequestID)
int CThostFtdcMDApi::ReqXXX(
CThostFtdcXXXField *pReqXXX,
int nRequestID)

response： void CThostFtdcTraderSpi::OnRspXXX(
CThostFtdcXXXField *pRspXXX,
CThostFtdcRspInfoField *pRspInfo,
int nRequestID,
bool bIsLast)

```
void CThostFtdcMDSpi::OnRspXXX(
        CThostFtdcXXXField *pRspXXX,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast)
```

The first parameter of request functions is request content and should not be empty. The second parameter is the request Id, which should be maintained by client trade application, and within one session the ID is strongly recommended be unique, when the client receive the response from the CTP server, the client could relate request and response with same request ID.

When the client receive any response from CTP server, the reloaded callback function of CThostFtdcXXXSpi will be invoked, if the response has more than one records, the reloaded callback function would be invoked repeatly until the whole message is received.

The first parameter of response functions is the data of the reponse, which usually includes the original request data. If something wrong happened or CTP can not find any record for the request, the parameter will be NULL. The second parameter is a flag used by CTP to show whether this response is one successful response. When the callback function is invoked more than one time, except the first time of the callback being invoked, this second parameter may be NULL in the following callback action. The third parameter is request ID which is

same as the corresponding request. The last parameter is the end marker of the response, the value “true” manifest the current response is the last one related with the same request.

3.2. Private mode programming interface

The following example shows the usual way of defining the private interface:

*void CThostFtdcTraderSpi::OnRtnXXX(CThostFtdcXXXField *pXXX)*

*void CThostFtdcTraderSpi::OnErrRtnXXX(CThostFtdcXXXField *pXXX,*
*CThostFtdcRspInfoField *pRspInfo)*

There is no function of the quotation API interface to communicate with CTP server in private mode.

When CTP server issue return data with private data stream, the reloaded callback function of the object inherited from CThostFtdcTradeSpi will be invoked.

The first parameter of all callback functions is the return content from CTP server, the second parameter of the OnErrRtn CThostFtdcTradeSpi functions is detail error information when something is wrong.

3.3. Boadcast mode programming interface

In CTP there is only two functions communicate with CTP server in broadcast mode, which is

void CThostFtdcTraderSpi::OnRtnInstrumentStatus (

*CThostFtdcInstrumentStatusField *pInstrumentStatus)*

void CThostFtdcTraderSpi::OnRtnDepthMarketData (

*CThostFtdcDepthMarketDataField *pDepthMarketData)*

When the trade status of some instruments changes the CTP server will callback the "OnRtnInstrumentStatus" function to notify all clients of CTP.

When the market quotation data is updated, and the CTP server received the updated data from exchanges, it will callback the "OnRtnDepthMarketData" function to notify all clients of CTP.

Chapter 4. Operation mode

4.1. Working thread

In the CTP trade client process, there be at least two thread, one is the application main thread and the othe is trade API working thread, if the client want to receive quotation data, another quotation API working thread is need. The communication between trade client and CTP server is activated by API working thread.

The trade and quotation API interface is thread-safe, two or more working threads wouldn't conflict, but one callback function chokes up, the corresponding working thread will be choken up and the communication between the working thread and CTP server will hold down. So, reloading callback function, please return as soon as possible, the data buffer or the windows message mechanism is recommended.

4.2. Files in location

CTP API dynamic link library running, some data will be save in location files, which extending name is ".con". When trade client creates API instances with "CreateFtdcTraderApi() or CreateFtdcMdApi()", the first parameter demonstrates the path of location files.

4.3. Business terminology and interface function contrast

Type	Business	Request interface	Response interface	Stream
login	login	CThostFtdcTraderApi:: ReqUserLogin	CThostFtdcTraderSpi::OnRspUserLogin	dialog
	logout	CThostFtdcTraderApi::ReqUserLogout	CThostFtdcTraderSpi::OnRspUserLogout	dialog
	Password modification	CThostFtdcTraderApi::ReqUserPasswordUpdate	CThostFtdcTraderSpi::OnRspUserPasswordUpdate	dialog
trade	Order insertion	CThostFtdcTraderApi::ReqOrderInsert	CThostFtdcTraderSpi::OnRspOrderInsert	dialog
	Order modification	CThostFtdcTraderApi::ReqOrderAction	CThostFtdcTraderSpi::OnRspOrderAction	dialog
Private return	Trade return	N/A	CThostFtdcTraderSpi::OnRtnTrade	private
	Order return	N/A	CThostFtdcTraderSpi::OnRtnOrder	private
	Order insertion error return	N/A	CThostFtdcTraderSpi::OnErrRtnOrderInsert	private
	Order monification error return	N/A	CThostFtdcTraderSpi::OnErrRtnOrderAction	private
query	Order query	CThostFtdcTraderApi::ReqQryOrder	CThostFtdcTraderSpi::OnRspQryOrder	query
	Trade query	CThostFtdcTraderApi::ReqQryTrade	CThostFtdcTraderSpi::OnRspQryTrade	query
	Investor query	CThostFtdcTraderApi::ReqQryInvestor	CThostFtdcTraderSpi::OnRspQryInvestor	query
	Investor position query	CThostFtdcTraderApi::ReqQryInvestor Position	CThostFtdcTraderSpi::OnRspQryInvestor Position	query
	Instrument query	CThostFtdcTraderApi::ReqQryInstrument	CThostFtdcTraderSpi::OnRspQryInstrument	query

Chapter 5. CTP API specification

5.1. General rules

The lifecycle of the communication between CTP client and server involves two phase, one is the initialization and the other is function invocation.

The trade API lifecycle involves the following detail steps:

1. Creating a “CThostFtdcTraderApi” instance.
2. Creating an event hangdle instance inherited from “CThostFtdcTraderSpi” interface, and registering this instance with the “RegisterSpi” function of the “CThostFtdcTraderApi”.
3. Subscribing private stream with the “SubscribePrivateTopic” function of the “CThostFtdcTraderApi”.
4. Subscribing public stream with the “SubscribePublicTopic” function of the “CThostFtdcTraderApi”.
5. Registering the trade front addresses of the CTP server with the “RegisterFront” function of the “CThostFtdcTraderApi”. The client could invoke the function more times, for more reliable communication, it is strongly recommended.
6. Applying connection with CTP server with the “Init” function of the “CThostFtdcTraderApi”.

7. After the CTP server confirmed the connection, the “OnFrontConnected” function of the “CThostFtdcTraderSpi” interface will be callbacked. In the function implementation of the client, it is favorable to submit the “login” request with the “ReqUserLogin” function of the “CThostFtdcTraderApi”.
8. After the CTP server confirmed the login, the “OnRspUserLogin” function of the “CThostFtdcTraderSpi” interface will be callbacked.
9. Now, the communication between the client and CTP server is bound successfully, and the other functions of CTP API interface could be invoked and callback.

In the quotation API lifecycle, the detail steps just like above mentioned, except subscribing private and public stream.

Other needs attention:

1. The parameters of all request function must not be NULL.
2. The type of return value of all request functions is “int”, “0” means right, others mean something is wrong; their detail information is descripted in the “error.xml” file.

5.2. CThostFtdcTraderSpi

CThostFtdcTraderSpi is the event interface for CTP server to notify client. For receiving the notification from CTP server, the client must

inherit it and impletment functions which the client interests.

5.2.1. OnFrontConnected

When the connection between client and CTP server is created successfully, the CTP server will callback it, the client could submit "ReqUserLogin" request in it.

definition：

void OnFrontConnected()；

5.2.2. OnFrontDisconnected

When the connection between client and CTP server disconnects, the CTP server will callback it. The client can discard this message, and the API instance will select one front address in the initial registered front list to reconnect the CTP server.

definition：

void OnFrontDisconnected (int nReason)；

parameters：

nReason：the reason of disconnecion
0x1001 net reading failed
0x1002 net writing failed
0x2001 heartbea receivingt timeout
0x2002 heartbeat sendingt timeout
0x2003 receiving anerror message

5.2.3. OnHeartBeatWarning

When there is no active message transmitting on the connection for

a long time, the CTP server will callback it to notify the client that their connection is still working.

definition：

void OnHeartBeatWarning(int nTimeLapse)；

parameters：

nTimeLapse：This is the length of time since the last message received.

5.2.4. OnRspUserLogin

After the CTP server received the “ReqUserLogin” request from the client, the “OnRspUserLogin” will be callbacked to notify the client the operation whether successful or not.

definition：

```
void OnRspUserLogin(
        CThostFtdcRspUserLoginField *pRspUserLogin,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast)；
```

parameters：

pRspUserLogin：*This is the pointer pointing to the structure of user login response information. The structure is defined as:*

```
struct CThostFtdcRspUserLoginField
{
    ///交易日
    TThostFtdcDateType TradingDay;
    ///登录成功时间
    TThostFtdcTimeType LoginTime;
    ///经纪公司代码
    TThostFtdcBrokerIDType BrokerID;
    ///用户代码
    TThostFtdcUserIDType    UserID;
    ///交易系统名称
    TThostFtdcSystemNameType   SystemName;
};
```

pRspInfo：*This is the pointer pointing to the structure of system response information. The structure is defined as:*

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

5.2.5. OnRspUserLogout

After the CTP server received the "ReqUserLogout" request from the client, the "OnRspUserLogout" will be callbacked to notify the client the operation whether successful or not.

definition：

```
void OnRspUserLogout(
                        CThostFtdcUserLogoutField *pUserLogout,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast)；
```

parameters：

pRspUserLogout：*This is the pointer pointing to the structure of user logout response information.Tthe structure is defined as:*

```
struct CThostFtdcUserLogoutField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType BrokerID;
    ///用户代码
    TThostFtdcUserIDType    UserID;
};
```

5.2.6. OnRspUserPasswordUpdate

After the CTP server received the "ReqUserPasswordUpdate" request from the client, the "OnRspUserLogout" will be callbacked to notify the client the operation whether successful or not.

definition：

```
void OnRspUserPasswordUpdate(
                        CThostFtdcUserPasswordUpdateField
                        *pUserPasswordUpdate,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast)；
```

parameters：

pUserPasswordUpdate：*This is the pointer pointing to the structure of user password modification response information. Tthe structure is defined as,*

```
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;
};
```

5.2.7. OnRspTradingAccountPasswordUpdate

After the CTP server received the "ReqTradingAccountPasswordUpdate" request from the client, the "OnRspTradingAccountPasswordUpdate" will be callbacked to notify the client the operation whether successful or not.

definition：

void OnRspTradingAccountPasswordUpdate(
*CThostFtdcTradingAccountPasswordUpdateField *pTradingAccountPasswordUpdate,*
*CThostFtdcRspInfoField *pRspInfo,*
int nRequestID,
bool bIsLast)；

parameters：

pTradingAccountPasswordUpdate：*This is the pointer pointing to the structure of trading account password modification response information. Tthe structure is defined as,*

```
struct CThostFtdcTradingAccountPasswordUpdateField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType BrokerID;
    ///投资者帐号
    TThostFtdcAccountIDType    AccountID;
    ///原来的口令
    TThostFtdcPasswordType OldPassword;
    ///新的口令
    TThostFtdcPasswordType NewPassword;
};
```

5.2.8. OnRspError 方法

针对用户请求的出错通知。

函数原形：

```
void OnRspError(
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID,
                bool bIsLast)
```

参数：

pRspInfo：返回用户响应信息的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
```

```
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回用户操作请求的 ID，该 ID 由用户在操作请求时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.9. OnRspOrderInsert 方法

报单录入应答。当客户端发出过报单录入指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspOrderInsert(
                        CThostFtdcInputOrderField *pInputOrder,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast);
```

参数：

pInputOrder：指向报单录入结构的地址，包含了提交报单录入时的输入数据，和后台返回的报单编号。

输入报单结构：

```
struct CThostFtdcInputOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType    OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///报单价格条件
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TThostFtdcDirectionType    Direction;
    ///组合开平标志
```

```
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///组合投机套保标志
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量
    TThostFtdcVolumeType VolumeTotalOriginal;
    ///有效期类型
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD 日期
    TThostFtdcDateType     GTDDate;
    ///成交量类型
    TThostFtdcVolumeConditionType VolumeCondition;
    ///最小成交量
    TThostFtdcVolumeType MinVolume;
    ///触发条件
    TThostFtdcContingentConditionType    ContingentCondition;
    ///止损价
    TThostFtdcPriceType     StopPrice;
    ///强平原因
    TThostFtdcForceCloseReasonType       ForceCloseReason;
    ///自动挂起标志
    TThostFtdcBoolType     IsAutoSuspend;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///请求编号
    TThostFtdcRequestIDType    RequestID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType     ErrorMsg;
};
```

nRequestID：返回报单录入操作请求的 ID，该 ID 由用户在报单录入时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.10. OnRspOrderAction 方法

报单操作应答。报单操作包括报单的撤销、报单的挂起、报单的激活、报单的修改。当客户端发出过报单操作指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspOrderAction(
                        CThostFtdcOrderActionField *pOrderAction,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast);
```

参数：

pOrderAction：指向报单操作结构的地址，包含了提交报单操作的输入数据，和后台返回的报单编号。

报单操作结构：

```
struct CThostFtdcOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///报单操作引用
    TThostFtdcOrderActionRefType    OrderActionRef;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///请求编号
    TThostFtdcRequestIDType     RequestID;
    ///前置编号
    TThostFtdcFrontIDType FrontID;
    ///会话编号
    TThostFtdcSessionIDType     SessionID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
    ///操作标志
    TThostFtdcActionFlagType    ActionFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
```

```
    ///数量变化
    TThostFtdcVolumeType VolumeChange;
    ///操作日期
    TThostFtdcDateType    ActionDate;
    ///操作时间
    TThostFtdcTimeType    ActionTime;
    ///交易所交易员代码
    TThostFtdcTraderIDType    TraderID;
    ///安装编号
    TThostFtdcInstallIDType    InstallID;
    ///本地报单编号
    TThostFtdcOrderLocalIDType    OrderLocalID;
    ///操作本地编号
    TThostFtdcOrderLocalIDType    ActionLocalID;
    ///会员代码
    TThostFtdcParticipantIDType    ParticipantID;
    ///客户代码
    TThostFtdcClientIDType    ClientID;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///报单操作状态
    TThostFtdcOrderActionStatusType    OrderActionStatus;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///状态信息
    TThostFtdcErrorMsgType    StatusMsg;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回用户报单操作请求的 ID，该 ID 由用户在报单操作时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.11. OnRspQueryMaxOrderVolume 方法

查询最大报单数量应答。当客户端发出查询最大报单数量指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQueryMaxOrderVolume(
        CThostFtdcQueryMaxOrderVolumeField *pQueryMaxOrderVolume,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pQueryMaxOrderVolume：指向查询最大报单数量结构的地址，包含了最大允许报单数量。

最大报单数量结构：

```
struct CThostFtdcQueryMaxOrderVolumeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
    ///买卖方向
    TThostFtdcDirectionType    Direction;
    ///开平标志
    TThostFtdcOffsetFlagType    OffsetFlag;
    ///投机套保标志
    TThostFtdcHedgeFlagType    HedgeFlag;
    ///最大允许报单数量
    TThostFtdcVolumeType MaxVolume;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
```

```
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回用户报单操作请求的 ID，该 ID 由用户在报单操作时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.12. OnRspSettlementInfoConfirm 方法

投资者结算结果确认应答。当客户端发出投资者结算结果确认指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspSettlementInfoConfirm(
        CThostFtdcSettlementInfoConfirmField *pSettlementInfoConfirm,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pSettlementInfoConfirm：指向投资者结算结果确认信息结构的地址，包含了最大允许报单数量。

投资者结算结果确认信息结构：

```
struct CThostFtdcSettlementInfoConfirmField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///确认日期
    TThostFtdcDateType      ConfirmDate;
    ///确认时间
    TThostFtdcTimeType      ConfirmTime;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
```

```
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回用户报单操作请求的 ID，该 ID 由用户在报单操作时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.13. OnRspTransferBankToFuture 方法

请求银行资金转期货响应。当客户端发出请求银行资金转期货指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspTransferBankToFuture(
        CThostFtdcTransferBankToFutureRspField *pTransferBankToFutureRsp,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pTransferBankToFutureRsp：指向银行资金转期货请求响应结构的地址。

银行资金转期货请求响应结构：

```
struct CThostFtdcTransferBankToFutureRspField
{
    ///响应代码
    TThostFtdcRetCodeType       RetCode;
    ///响应信息
    TThostFtdcRetInfoType RetInfo;
    ///资金账户
    TThostFtdcAccountIDType    FutureAccount;
    ///转帐金额
    TThostFtdcMoneyType  TradeAmt;
    ///应收客户手续费
    TThostFtdcMoneyType  CustFee;
    ///币种
    TThostFtdcCurrencyCodeType      CurrencyCode;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
```

```
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回用户报单操作请求的 ID，该 ID 由用户在报单操作时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.14. OnRspTransferFutureToBank 方法

请求期货资金转银行响应。当客户端发出请求期货资金转银行指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspTransferFutureToBank(
        CThostFtdcTransferFutureToBankRspField *pTransferFutureToBankRsp,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pTransferFutureToBankRsp：指向期货资金转银行请求响应结构的地址。

期货资金转银行请求响应结构：

```
struct CThostFtdcTransferFutureToBankRspField
{
    ///响应代码
    TThostFtdcRetCodeType    RetCode;
    ///响应信息
    TThostFtdcRetInfoType RetInfo;
    ///资金账户
    TThostFtdcAccountIDType    FutureAccount;
    ///转帐金额
    TThostFtdcMoneyType  TradeAmt;
    ///应收客户手续费
    TThostFtdcMoneyType  CustFee;
    ///币种
    TThostFtdcCurrencyCodeType    CurrencyCode;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
```

```
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回用户报单操作请求的 ID，该 ID 由用户在报单操作时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.15. OnRspTransferQryBank 方法

请求查询银行资金响应。当客户端发出请求查询银行资金指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspTransferQryBank(
        CThostFtdcTransferQryBankRspField *pTransferQryBankRsp,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pTransferQryBankRsp：指向查询银行资金请求响应结构的地址。

查询银行资金请求响应结构：

```
struct CThostFtdcTransferQryBankRspField
{
    ///响应代码
    TThostFtdcRetCodeType    RetCode;
    ///响应信息
    TThostFtdcRetInfoType RetInfo;
    ///资金账户
    TThostFtdcAccountIDType    FutureAccount;
    ///银行余额
    TThostFtdcMoneyType  TradeAmt;
    ///银行可用余额
    TThostFtdcMoneyType  UseAmt;
    ///银行可取余额
    TThostFtdcMoneyType  FetchAmt;
    ///币种
    TThostFtdcCurrencyCodeType    CurrencyCode;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回用户报单操作请求的 ID，该 ID 由用户在报单操作时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.16. OnRspTransferQryDetail 方法

请求查询银行交易明细响应。当客户端发出请求查询银行交易明细指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspTransferQryDetail(
        CThostFtdcTransferQryDetailRspField *pTransferQryDetailRsp,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pTransferQryDetailRsp：指向查询银行交易明细请求响应结构的地址。

查询银行交易明细请求响应结构：

```
struct CThostFtdcTransferQryDetailRspField
{
    ///交易日期
    TThostFtdcDateType    TradeDate;
    ///交易时间
    TThostFtdcTradeTimeType    TradeTime;
    ///交易代码
    TThostFtdcTradeCodeType    TradeCode;
    ///期货流水号
    TThostFtdcTradeSerialNoType    FutureSerial;
    ///期货公司代码
    TThostFtdcFutureIDType    FutureID;
    ///资金帐号
```

```
    TThostFtdcFutureAccountType     FutureAccount;
    ///银行流水号
    TThostFtdcTradeSerialNoType     BankSerial;
    ///银行代码
    TThostFtdcBankIDType BankID;
    ///银行分中心代码
    TThostFtdcBankBrchIDType BankBrchID;
    ///银行账号
    TThostFtdcBankAccountType       BankAccount;
    ///证件号码
    TThostFtdcCertCodeType      CertCode;
    ///货币代码
    TThostFtdcCurrencyCodeType      CurrencyCode;
    ///发生金额
    TThostFtdcMoneyType  TxAmount;
    ///有效标志
    TThostFtdcTransferValidFlagType Flag;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回用户报单操作请求的 ID，该 ID 由用户在报单操作时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.17. OnRspQryOrder 方法

报单查询请求。当客户端发出报单查询指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryOrder(
                    CThostFtdcOrderField *pOrder,
                    CThostFtdcRspInfoField *pRspInfo,
```

```
int nRequestID,
bool bIsLast);
```

参数：

pOrder：指向报单信息结构的地址。

报单信息结构：

```
struct CThostFtdcOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///报单价格条件
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TThostFtdcDirectionType     Direction;
    ///组合开平标志
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///组合投机套保标志
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量
    TThostFtdcVolumeType VolumeTotalOriginal;
    ///有效期类型
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD 日期
    TThostFtdcDateType      GTDDate;
    ///成交量类型
    TThostFtdcVolumeConditionType VolumeCondition;
    ///最小成交量
    TThostFtdcVolumeType MinVolume;
    ///触发条件
    TThostFtdcContingentConditionType    ContingentCondition;
    ///止损价
    TThostFtdcPriceType     StopPrice;
    ///强平原因
```

```
TThostFtdcForceCloseReasonType	ForceCloseReason;
///自动挂起标志
TThostFtdcBoolType	IsAutoSuspend;
///业务单元
TThostFtdcBusinessUnitTypeBusinessUnit;
///请求编号
TThostFtdcRequestIDType	RequestID;
///本地报单编号
TThostFtdcOrderLocalIDType	OrderLocalID;
///交易所代码
TThostFtdcExchangeIDType ExchangeID;
///会员代码
TThostFtdcParticipantIDType	ParticipantID;
///客户代码
TThostFtdcClientIDType	ClientID;
///合约在交易所的代码
TThostFtdcExchangeInstIDType	ExchangeInstID;
///交易所交易员代码
TThostFtdcTraderIDType	TraderID;
///安装编号
TThostFtdcInstallIDType	InstallID;
///报单提交状态
TThostFtdcOrderSubmitStatusType	OrderSubmitStatus;
///报单提示序号
TThostFtdcSequenceNoType NotifySequence;
///交易日
TThostFtdcDateType	TradingDay;
///结算编号
TThostFtdcSettlementIDTypeSettlementID;
///报单编号
TThostFtdcOrderSysIDType	OrderSysID;
///报单来源
TThostFtdcOrderSourceType OrderSource;
///报单状态
TThostFtdcOrderStatusType	OrderStatus;
///报单类型
TThostFtdcOrderTypeType	OrderType;
///今成交数量
TThostFtdcVolumeType VolumeTraded;
///剩余数量
TThostFtdcVolumeType VolumeTotal;
///报单日期
TThostFtdcDateType	InsertDate;
///插入时间
```

```
    TThostFtdcTimeType    InsertTime;
    ///激活时间
    TThostFtdcTimeType    ActiveTime;
    ///挂起时间
    TThostFtdcTimeType    SuspendTime;
    ///最后修改时间
    TThostFtdcTimeType    UpdateTime;
    ///撤销时间
    TThostFtdcTimeType    CancelTime;
    ///最后修改交易所交易员代码
    TThostFtdcTraderIDType    ActiveTraderID;
    ///结算会员编号
    TThostFtdcParticipantIDType    ClearingPartID;
    ///序号
    TThostFtdcSequenceNoType SequenceNo;
    ///前置编号
    TThostFtdcFrontIDType FrontID;
    ///会话编号
    TThostFtdcSessionIDType    SessionID;
    ///用户端产品信息
    TThostFtdcProductInfoType  UserProductInfo;
    ///状态信息
    TThostFtdcErrorMsgType    StatusMsg;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回用户报单查询请求的 ID，该 ID 由用户在报单查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.18. OnRspQryTrade 方法

成交单查询应答。当客户端发出成交单查询指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryTrade(
                    CThostFtdcTradeField *pTrade,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast)；
```

参数：

pTrade：指向成交信息结构的地址。

成交信息结构：

```
struct CThostFtdcTradeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///成交编号
    TThostFtdcTradeIDType       TradeID;
    ///买卖方向
    TThostFtdcDirectionType     Direction;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
    ///会员代码
    TThostFtdcParticipantIDType     ParticipantID;
    ///客户代码
    TThostFtdcClientIDType      ClientID;
    ///交易角色
    TThostFtdcTradingRoleType TradingRole;
    ///合约在交易所的代码
    TThostFtdcExchangeInstIDType    ExchangeInstID;
    ///开平标志
    TThostFtdcOffsetFlagType    OffsetFlag;
    ///投机套保标志
    TThostFtdcHedgeFlagType    HedgeFlag;
```

```
    ///价格
    TThostFtdcPriceType     Price;
    ///数量
    TThostFtdcVolumeType Volume;
    ///成交时期
    TThostFtdcDateType      TradeDate;
    ///成交时间
    TThostFtdcTimeType      TradeTime;
    ///成交类型
    TThostFtdcTradeTypeType     TradeType;
    ///成交价来源
    TThostFtdcPriceSourceType  PriceSource;
    ///交易所交易员代码
    TThostFtdcTraderIDType      TraderID;
    ///本地报单编号
    TThostFtdcOrderLocalIDType      OrderLocalID;
    ///结算会员编号
    TThostFtdcParticipantIDType      ClearingPartID;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///序号
    TThostFtdcSequenceNoType SequenceNo;
    ///交易日
    TThostFtdcDateType      TradingDay;
    ///结算编号
    TThostFtdcSettlementIDTypeSettlementID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回用户成交单请求的 ID，该 ID 由用户在成交单查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.19. OnRspQryInvestor 方法

会员客户查询应答。当客户端发出会员客户查询指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQry Investor (
                        CThostFtdcInvestorField *pInvestor,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast)；
```

参数：

p Investor：指向投资者信息结构的地址。

投资者信息结构：

```
struct CThostFtdcInvestorField
{
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者分组代码
    TThostFtdcInvestorIDType    InvestorGroupID;
    ///投资者名称
    TThostFtdcPartyNameType     InvestorName;
    ///证件类型
    TThostFtdcIdCardTypeType    IdentifiedCardType;
    ///证件号码
    TThostFtdcIdentifiedCardNoType IdentifiedCardNo;
    ///是否活跃
    TThostFtdcBoolType      IsActive;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.20. OnRspQryInvestorPosition 方法

投资者持仓查询应答。当客户端发出投资者持仓查询指令后，后交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQry InvestorPosition(
                        CThostFtdcInvestorPositionField *pInvestorPosition,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast);
```

参数：

pInvestorPosition：指向投资者持仓应答结构的地址。

投资者持仓应答结构：

```
struct CThostFtdcInvestorPositionField
{
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///持仓多空方向
    TThostFtdcPosiDirectionType    PosiDirection;
    ///投机套保标志
    TThostFtdcHedgeFlagType    HedgeFlag;
    ///持仓日期
```

```
TThostFtdcPositionDateType PositionDate;
///上日持仓
TThostFtdcVolumeType YdPosition;
///今日持仓
TThostFtdcVolumeType Position;
///多头冻结
TThostFtdcVolumeType LongFrozen;
///空头冻结
TThostFtdcVolumeType ShortFrozen;
///开仓冻结金额
TThostFtdcMoneyType  LongFrozenAmount;
///开仓冻结金额
TThostFtdcMoneyType  ShortFrozenAmount;
///开仓量
TThostFtdcVolumeType OpenVolume;
///平仓量
TThostFtdcVolumeType CloseVolume;
///开仓金额
TThostFtdcMoneyType  OpenAmount;
///平仓金额
TThostFtdcMoneyType  CloseAmount;
///持仓成本
TThostFtdcMoneyType  PositionCost;
///上次占用的保证金
TThostFtdcMoneyType  PreMargin;
///占用的保证金
TThostFtdcMoneyType  UseMargin;
///冻结的保证金
TThostFtdcMoneyType  FrozenMargin;
///冻结的资金
TThostFtdcMoneyType  FrozenCash;
///冻结的手续费
TThostFtdcMoneyType  FrozenCommission;
///资金差额
TThostFtdcMoneyType  CashIn;
///手续费
TThostFtdcMoneyType  Commission;
///平仓盈亏
TThostFtdcMoneyType  CloseProfit;
///持仓盈亏
TThostFtdcMoneyType  PositionProfit;
///上次结算价
TThostFtdcPriceType     PreSettlementPrice;
///本次结算价
```

```
    TThostFtdcPriceType     SettlementPrice;
    ///交易日
    TThostFtdcDateType      TradingDay;
    ///结算编号
    TThostFtdcSettlementIDTypeSettlementID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回会员持仓查询请求的 ID，该 ID 由用户在会员持仓查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.21. OnRspQryTradingAccount 方法

请求查询资金账户响应。当客户端发出请求查询资金账户指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryTradingAccount(
                    CThostFtdcTradingAccountField *pTradingAccount,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast);
```

参数：

pTradingAccount：指向资金账户结构的地址。

资金账户结构：

```
struct CThostFtdcTradingAccountField
{
    ///经纪公司代码
```

TThostFtdcBrokerIDType	BrokerID;

///投资者帐号

TThostFtdcAccountIDType	AccountID;

///上次质押金额

TThostFtdcMoneyType	PreMortgage;

///上次信用额度

TThostFtdcMoneyType	PreCredit;

///上次存款额

TThostFtdcMoneyType	PreDeposit;

///上次结算准备金

TThostFtdcMoneyType	PreBalance;

///上次占用的保证金

TThostFtdcMoneyType	PreMargin;

///利息基数

TThostFtdcMoneyType	InterestBase;

///利息收入

TThostFtdcMoneyType	Interest;

///入金金额

TThostFtdcMoneyType	Deposit;

///出金金额

TThostFtdcMoneyType	Withdraw;

///冻结的保证金

TThostFtdcMoneyType	FrozenMargin;

///冻结的资金

TThostFtdcMoneyType	FrozenCash;

///冻结的手续费

TThostFtdcMoneyType	FrozenCommission;

///当前保证金总额

TThostFtdcMoneyType	CurrMargin;

///资金差额

```
    TThostFtdcMoneyType  CashIn;
    ///手续费
    TThostFtdcMoneyType  Commission;
    ///平仓盈亏
    TThostFtdcMoneyType  CloseProfit;
    ///持仓盈亏
    TThostFtdcMoneyType  PositionProfit;
    ///期货结算准备金
    TThostFtdcMoneyType  Balance;
    ///可用资金
    TThostFtdcMoneyType  Available;
    ///可取资金
    TThostFtdcMoneyType  WithdrawQuota;
    ///基本准备金
    TThostFtdcMoneyType  Reserve;
    ///交易日
    TThostFtdcDateType     TradingDay;
    ///结算编号
    TThostFtdcSettlementIDTypeSettlementID;
    ///信用额度
    TThostFtdcMoneyType  Credit;
    ///质押金额
    TThostFtdcMoneyType  Mortgage;
    ///交易所保证金
    TThostFtdcMoneyType  ExchangeMargin;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
```

```
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.22. OnRspQryTradingCode 方法

请求查询交易编码响应。当客户端发出请求查询交易编码指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryTradingCode(
                CThostFtdcTradingCodeField *pTradingCode,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID,
                bool bIsLast) ；
```

参数：

pTradingCode：指向交易编码结构的地址。

交易编码结构：

```
struct CThostFtdcTradingCodeField
{
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///交易编码
    TThostFtdcClientIDType    ClientID;
    ///是否活跃
```

```
    TThostFtdcBoolType    IsActive;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.23. OnRspQryExchange 方法

请求查询交易所响应。当客户端发出请求查询交易所指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryExchange(
                    CThostFtdcExchangeField *pExchange,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast) ;
```

参数：

pExchange：指向交易所结构的地址。

交易所结构：

```
struct CThostFtdcExchangeField
{
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///交易所名称
```

```
    TThostFtdcExchangeNameType    ExchangeName;
    ///交易所属性
    TThostFtdcExchangePropertyType       ExchangeProperty;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType     ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.24. OnRspQryInstrument 方法

请求查询合约响应。当客户端发出请求查询合约指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryInstrument(
                    CThostFtdcInstrumentField *pInstrument,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast) ;
```

参数：

pInstrument：指向合约结构的地址。

合约结构：

```
struct CThostFtdcInstrumentField
{
    ///合约代码
```

TThostFtdcInstrumentIDType InstrumentID;

///交易所代码

TThostFtdcExchangeIDType ExchangeID;

///合约名称

TThostFtdcInstrumentNameType InstrumentName;

///合约在交易所的代码

TThostFtdcExchangeInstIDType ExchangeInstID;

///产品代码

TThostFtdcInstrumentIDType ProductID;

///产品类型

TThostFtdcProductClassTypeProductClass;

///交割年份

TThostFtdcYearType DeliveryYear;

///交割月

TThostFtdcMonthType DeliveryMonth;

///市价单最大下单量

TThostFtdcVolumeType MaxMarketOrderVolume;

///市价单最小下单量

TThostFtdcVolumeType MinMarketOrderVolume;

///限价单最大下单量

TThostFtdcVolumeType MaxLimitOrderVolume;

///限价单最小下单量

TThostFtdcVolumeType MinLimitOrderVolume;

///合约数量乘数

TThostFtdcVolumeMultipleType VolumeMultiple;

///最小变动价位

TThostFtdcPriceType PriceTick;

///创建日

TThostFtdcDateType CreateDate;

///上市日

```
    TThostFtdcDateType      OpenDate;
    ///到期日
    TThostFtdcDateType      ExpireDate;
    ///开始交割日
    TThostFtdcDateType      StartDelivDate;
    ///结束交割日
    TThostFtdcDateType      EndDelivDate;
    ///合约生命周期状态
    TThostFtdcInstLifePhaseType     InstLifePhase;
    ///当前是否交易
    TThostFtdcBoolType      IsTrading;
    ///持仓类型
    TThostFtdcPositionTypeType PositionType;
    ///持仓日期类型
    TThostFtdcPositionDateTypeType PositionDateType;
    ///多头保证金率
    TThostFtdcRatioType     LongMarginRatio;
    ///空头保证金率
    TThostFtdcRatioType     ShortMarginRatio;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.25. OnRspQryDepthMarketData 方法

请求查询行情响应。当客户端发出请求查询行情指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryDepthMarketData(
                    CThostFtdcDepthMarketDataField *pDepthMarketData,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast) ;
```

参数：

pDepthMarketData：指向深度行情结构的地址。

深度行情结构：

```
struct CThostFtdcDepthMarketDataField
{
    ///交易日
    TThostFtdcDateType    TradingDay;
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///合约在交易所的代码
    TThostFtdcExchangeInstIDType    ExchangeInstID;
    ///最新价
    TThostFtdcPriceType    LastPrice;
    ///上次结算价
    TThostFtdcPriceType    PreSettlementPrice;
    ///昨收盘
    TThostFtdcPriceType    PreClosePrice;
    ///昨持仓量
    TThostFtdcLargeVolumeType    PreOpenInterest;
```

```
///今开盘
TThostFtdcPriceType	OpenPrice;
///最高价
TThostFtdcPriceType	HighestPrice;
///最低价
TThostFtdcPriceType	LowestPrice;
///数量
TThostFtdcVolumeType	Volume;
///成交金额
TThostFtdcMoneyType	Turnover;
///持仓量
TThostFtdcLargeVolumeType	OpenInterest;
///今收盘
TThostFtdcPriceType	ClosePrice;
///本次结算价
TThostFtdcPriceType	SettlementPrice;
///涨停板价
TThostFtdcPriceType	UpperLimitPrice;
///跌停板价
TThostFtdcPriceType	LowerLimitPrice;
///昨虚实度
TThostFtdcRatioType	PreDelta;
///今虚实度
TThostFtdcRatioType	CurrDelta;
///最后修改时间
TThostFtdcTimeType	UpdateTime;
///最后修改毫秒
TThostFtdcMillisecType	UpdateMillisec;
///申买价一
TThostFtdcPriceType	BidPrice1;
```

///申买量一

TThostFtdcVolumeType BidVolume1;

///申卖价一

TThostFtdcPriceType AskPrice1;

///申卖量一

TThostFtdcVolumeType AskVolume1;

///申买价二

TThostFtdcPriceType BidPrice2;

///申买量二

TThostFtdcVolumeType BidVolume2;

///申卖价二

TThostFtdcPriceType AskPrice2;

///申卖量二

TThostFtdcVolumeType AskVolume2;

///申买价三

TThostFtdcPriceType BidPrice3;

///申买量三

TThostFtdcVolumeType BidVolume3;

///申卖价三

TThostFtdcPriceType AskPrice3;

///申卖量三

TThostFtdcVolumeType AskVolume3;

///申买价四

TThostFtdcPriceType BidPrice4;

///申买量四

TThostFtdcVolumeType BidVolume4;

///申卖价四

TThostFtdcPriceType AskPrice4;

///申卖量四

TThostFtdcVolumeType AskVolume4;

```
    ///申买价五
    TThostFtdcPriceType     BidPrice5;
    ///申买量五
    TThostFtdcVolumeType BidVolume5;
    ///申卖价五
    TThostFtdcPriceType     AskPrice5;
    ///申卖量五
    TThostFtdcVolumeType AskVolume5;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.26. OnRspQrySettlementInfo 方法

请求查询投资者结算结果响应。当客户端发出请求查询投资者结算结果指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQrySettlementInfo(
                    CThostFtdcSettlementInfoField *pSettlementInfo,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast) ；
```

参数：

pSettlementInfo：指向投资者结算结果结构的地址。

投资者结算结果结构：

```
struct CThostFtdcSettlementInfoField
{
    ///交易日
    TThostFtdcDateType      TradingDay;
    ///结算编号
    TThostFtdcSettlementIDTypeSettlementID;
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///序号
    TThostFtdcSequenceNoType SequenceNo;
    ///消息正文
    TThostFtdcContentType Content;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.27. OnRspQryTransferBank 方法

请求查询转帐银行响应。当客户端发出请求查询转帐银行指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryTransferBank(
                    CThostFtdcTransferBankField *pTransferBank,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast)；
```

参数：

pTransferBank：指向转帐银行结构的地址。

转帐银行结构：

```
struct CThostFtdcTransferBankField
{
    ///银行代码
    TThostFtdcBankIDType BankID;
    ///银行分中心代码
    TThostFtdcBankBrchIDType BankBrchID;
    ///银行名称
    TThostFtdcBankNameType    BankName;
    ///是否活跃
    TThostFtdcBoolType    IsActive;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.28. OnRspQryInvestorPositionDetail 方法

请求查询投资者持仓明细响应。当客户端发出请求请求查询投资者持仓明细指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryInvestorPositionDetail(
                    CThostFtdcInvestorPositionDetailField *pInvestorPositionDetail,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast) ；
```

参数：

pInvestorPositionDetail：指向投资者持仓明细结构的地址。

投资者持仓明细结构：

```
struct CThostFtdcInvestorPositionDetailField
{
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///投机套保标志
    TThostFtdcHedgeFlagType    HedgeFlag;
    ///买卖方向
    TThostFtdcDirectionType      Direction;
    ///开仓日期
    TThostFtdcDateType      OpenDate;
```

```
    ///成交编号
    TThostFtdcTradeIDType       TradeID;
    ///数量
    TThostFtdcVolumeType Volume;
    ///开仓价
    TThostFtdcPriceType     OpenPrice;
    ///交易日
    TThostFtdcDateType      TradingDay;
    ///结算编号
    TThostFtdcSettlementIDTypeSettlementID;
    ///成交类型
    TThostFtdcTradeTypeType     TradeType;
    ///组合合约代码
    TThostFtdcInstrumentIDType      CombInstrumentID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.29. OnRspQryNotice 方法

请求查询客户通知响应。当客户端发出请求查询客户通知指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryNotice(
                        CThostFtdcNoticeField *pNotice,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast)；
```

参数：

pNotice：指向客户通知结构的地址。

客户通知结构：

```
///客户通知
struct CThostFtdcNoticeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///消息正文
    TThostFtdcContentType Content;
    ///经纪公司通知内容序列号
    TThostFtdcSequenceLabelType     SequenceLabel;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回会员客户查询请求的 ID，该 ID 由用户在会员客户查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.30. OnRspQryInstrument 方法

合约查询应答。当客户端发出合约查询指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryInstrument(
                        CThostFtdcInstrumentField *pInstrument,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast);
```

参数：

pRspInstrument：指向合约结构的地址。

合约结构：

```
struct CThostFtdcInstrumentField
{
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///合约名称
    TThostFtdcInstrumentNameType  InstrumentName;
    ///合约在交易所的代码
    TThostFtdcExchangeInstIDType   ExchangeInstID;
    ///产品代码
    TThostFtdcInstrumentIDType      ProductID;
    ///产品类型
    TThostFtdcProductClassTypeProductClass;
    ///交割年份
    TThostFtdcYearType     DeliveryYear;
    ///交割月
    TThostFtdcMonthType  DeliveryMonth;
    ///市价单最大下单量
    TThostFtdcVolumeType MaxMarketOrderVolume;
    ///市价单最小下单量
    TThostFtdcVolumeType MinMarketOrderVolume;
    ///限价单最大下单量
    TThostFtdcVolumeType MaxLimitOrderVolume;
    ///限价单最小下单量
    TThostFtdcVolumeType MinLimitOrderVolume;
```

```
    ///合约数量乘数
    TThostFtdcVolumeMultipleType    VolumeMultiple;
    ///最小变动价位
    TThostFtdcPriceType     PriceTick;
    ///创建日
    TThostFtdcDateType      CreateDate;
    ///上市日
    TThostFtdcDateType      OpenDate;
    ///到期日
    TThostFtdcDateType      ExpireDate;
    ///开始交割日
    TThostFtdcDateType      StartDelivDate;
    ///结束交割日
    TThostFtdcDateType      EndDelivDate;
    ///合约生命周期状态
    TThostFtdcInstLifePhaseType     InstLifePhase;
    ///当前是否交易
    TThostFtdcBoolType      IsTrading;
    ///持仓类型
    TThostFtdcPositionTypeType PositionType;
    ///持仓日期类型
    TThostFtdcPositionDateTypeType PositionDateType;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

nRequestID：返回合约查询请求的 ID，该 ID 由用户在合约查询时指定。

bIsLast：指示该次返回是否为针对 nRequestID 的最后一次返回。

5.2.31. OnRtnTrade 方法

成交回报。当发生成交时交易托管系统会通知客户端，该方法会被调用。

函数原形：

```
void OnRtnTrade(CThostFtdcTradeField *pTrade);
```

参数：

pTrade：指向成交信息结构的地址。

成交信息结构：

```
struct CThostFtdcTradeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType    OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
    ///成交编号
    TThostFtdcTradeIDType    TradeID;
    ///买卖方向
    TThostFtdcDirectionType    Direction;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
    ///会员代码
    TThostFtdcParticipantIDType    ParticipantID;
    ///客户代码
    TThostFtdcClientIDType    ClientID;
    ///交易角色
    TThostFtdcTradingRoleType TradingRole;
    ///合约在交易所的代码
    TThostFtdcExchangeInstIDType    ExchangeInstID;
    ///开平标志
    TThostFtdcOffsetFlagType    OffsetFlag;
    ///投机套保标志
    TThostFtdcHedgeFlagType    HedgeFlag;
    ///价格
    TThostFtdcPriceType    Price;
    ///数量
    TThostFtdcVolumeType Volume;
    ///成交时期
    TThostFtdcDateType    TradeDate;
    ///成交时间
```

```
    TThostFtdcTimeType     TradeTime;
    ///成交类型
    TThostFtdcTradeTypeType    TradeType;
    ///成交价来源
    TThostFtdcPriceSourceType  PriceSource;
    ///交易所交易员代码
    TThostFtdcTraderIDType     TraderID;
    ///本地报单编号
    TThostFtdcOrderLocalIDType     OrderLocalID;
    ///结算会员编号
    TThostFtdcParticipantIDType     ClearingPartID;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///序号
    TThostFtdcSequenceNoType SequenceNo;
    ///交易日
    TThostFtdcDateType     TradingDay;
    ///结算编号
    TThostFtdcSettlementIDTypeSettlementID;
};
```

5.2.32. OnRtnOrder 方法

报单回报。当客户端进行报单录入、报单操作及其它原因（如部分成交）导致报单状态发生变化时，交易托管系统会主动通知客户端，该方法会被调用。

函数原形：

```
void OnRtnOrder(CThostFtdcOrderField *pOrder);
```

参数：

pOrder：指向报单信息结构的地址。

报单信息结构：

```
struct CThostFtdcOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType     BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType     InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType     OrderRef;
```

```
///用户代码
TThostFtdcUserIDType UserID;
///报单价格条件
TThostFtdcOrderPriceTypeType OrderPriceType;
///买卖方向
TThostFtdcDirectionType Direction;
///组合开平标志
TThostFtdcCombOffsetFlagType CombOffsetFlag;
///组合投机套保标志
TThostFtdcCombHedgeFlagType CombHedgeFlag;
///价格
TThostFtdcPriceType LimitPrice;
///数量
TThostFtdcVolumeType VolumeTotalOriginal;
///有效期类型
TThostFtdcTimeConditionType TimeCondition;
///GTD 日期
TThostFtdcDateType GTDDate;
///成交量类型
TThostFtdcVolumeConditionType VolumeCondition;
///最小成交量
TThostFtdcVolumeType MinVolume;
///触发条件
TThostFtdcContingentConditionType ContingentCondition;
///止损价
TThostFtdcPriceType StopPrice;
///强平原因
TThostFtdcForceCloseReasonType ForceCloseReason;
///自动挂起标志
TThostFtdcBoolType IsAutoSuspend;
///业务单元
TThostFtdcBusinessUnitTypeBusinessUnit;
///请求编号
TThostFtdcRequestIDType RequestID;
///本地报单编号
TThostFtdcOrderLocalIDType OrderLocalID;
///交易所代码
TThostFtdcExchangeIDType ExchangeID;
///会员代码
TThostFtdcParticipantIDType ParticipantID;
///客户代码
TThostFtdcClientIDType ClientID;
///合约在交易所的代码
TThostFtdcExchangeInstIDType ExchangeInstID;
```

```
///交易所交易员代码
TThostFtdcTraderIDType	TraderID;
///安装编号
TThostFtdcInstallIDType	InstallID;
///报单提交状态
TThostFtdcOrderSubmitStatusType	OrderSubmitStatus;
///报单提示序号
TThostFtdcSequenceNoType NotifySequence;
///交易日
TThostFtdcDateType	TradingDay;
///结算编号
TThostFtdcSettlementIDTypeSettlementID;
///报单编号
TThostFtdcOrderSysIDType	OrderSysID;
///报单来源
TThostFtdcOrderSourceType OrderSource;
///报单状态
TThostFtdcOrderStatusType	OrderStatus;
///报单类型
TThostFtdcOrderTypeType	OrderType;
///今成交数量
TThostFtdcVolumeType VolumeTraded;
///剩余数量
TThostFtdcVolumeType VolumeTotal;
///报单日期
TThostFtdcDateType	InsertDate;
///插入时间
TThostFtdcTimeType	InsertTime;
///激活时间
TThostFtdcTimeType	ActiveTime;
///挂起时间
TThostFtdcTimeType	SuspendTime;
///最后修改时间
TThostFtdcTimeType	UpdateTime;
///撤销时间
TThostFtdcTimeType	CancelTime;
///最后修改交易所交易员代码
TThostFtdcTraderIDType	ActiveTraderID;
///结算会员编号
TThostFtdcParticipantIDType	ClearingPartID;
///序号
TThostFtdcSequenceNoType SequenceNo;
///前置编号
TThostFtdcFrontIDType FrontID;
```

```
    ///会话编号
    TThostFtdcSessionIDType     SessionID;
    ///用户端产品信息
    TThostFtdcProductInfoType   UserProductInfo;
    ///状态信息
    TThostFtdcErrorMsgType      StatusMsg;
};
```

5.2.33. OnErrRtnOrderInsert 方法

报单录入错误回报。由交易托管系统主动通知客户端，该方法会被调用。

函数原形：

```
void OnErrRtnOrderInsert(
                        CThostFtdcInputOrderField *pInputOrder,
                        CThostFtdcRspInfoField *pRspInfo)；
```

参数：

pInputOrder：指向报单录入结构的地址，包含了提交报单录入时的输入数据，和后台返回的报单编号。

输入报单结构：

```
struct CThostFtdcInputOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///报单价格条件
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TThostFtdcDirectionType     Direction;
    ///组合开平标志
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///组合投机套保标志
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///价格
```

```
    TThostFtdcPriceType     LimitPrice;
    ///数量
    TThostFtdcVolumeType VolumeTotalOriginal;
    ///有效期类型
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD 日期
    TThostFtdcDateType     GTDDate;
    ///成交量类型
    TThostFtdcVolumeConditionType VolumeCondition;
    ///最小成交量
    TThostFtdcVolumeType MinVolume;
    ///触发条件
    TThostFtdcContingentConditionType    ContingentCondition;
    ///止损价
    TThostFtdcPriceType     StopPrice;
    ///强平原因
    TThostFtdcForceCloseReasonType       ForceCloseReason;
    ///自动挂起标志
    TThostFtdcBoolType     IsAutoSuspend;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///请求编号
    TThostFtdcRequestIDType    RequestID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType     ErrorMsg;
};
```

5.2.34. OnErrRtnOrderAction 方法

报价操作错误回报。由交易托管系统主动通知客户端，该方法会被调用。

函数原形：

```
void OnErrRtnOrderAction (
                                CThostFtdcOrderActionField *pOrderAction,
                                CThostFtdcRspInfoField *pRspInfo);
```

参数：

pOrderAction：指向报价操作结构的地址，包含了报价操作请求的输入数据，和后台返回的报价编号。

报价操作结构：

```
struct CThostFtdcOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///报单操作引用
    TThostFtdcOrderActionRefType    OrderActionRef;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///请求编号
    TThostFtdcRequestIDType     RequestID;
    ///前置编号
    TThostFtdcFrontIDType FrontID;
    ///会话编号
    TThostFtdcSessionIDType     SessionID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
    ///操作标志
    TThostFtdcActionFlagType    ActionFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量变化
    TThostFtdcVolumeType VolumeChange;
    ///操作日期
    TThostFtdcDateType      ActionDate;
    ///操作时间
    TThostFtdcTimeType      ActionTime;
    ///交易所交易员代码
    TThostFtdcTraderIDType      TraderID;
    ///安装编号
    TThostFtdcInstallIDType     InstallID;
    ///本地报单编号
    TThostFtdcOrderLocalIDType      OrderLocalID;
    ///操作本地编号
    TThostFtdcOrderLocalIDType      ActionLocalID;
```

```
    ///会员代码
    TThostFtdcParticipantIDType    ParticipantID;
    ///客户代码
    TThostFtdcClientIDType    ClientID;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///报单操作状态
    TThostFtdcOrderActionStatusType    OrderActionStatus;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///状态信息
    TThostFtdcErrorMsgType    StatusMsg;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

5.2.35. OnRspQrySettlementInfoConfirm 方法

查询结算确认响应。由交易托管系统主动通知客户端，该方法会被调用。

函数原形：

```
void OnRspQrySettlementInfoConfirm(
        CThostFtdcSettlementInfoConfirmField *pSettlementInfoConfirm,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pSettlementInfoConfirm：指向返回的结算确认信息结构。

结算确认结构：

```
///投资者结算结果确认信息
struct CThostFtdcSettlementInfoConfirmField
```

```
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///确认日期
    TThostFtdcDateType    ConfirmDate;
    ///确认时间
    TThostFtdcTimeType    ConfirmTime;
};;
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

5.2.36. OnRspQryContractBank 方法

请求查询签约银行响应。

函数原形：

```
void OnRspQryContractBank(
        CThostFtdcContractBankField *pContractBank,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pContractBank：指向查询签约银行响应结构。

查询签约银行响应结构：

```
struct CThostFtdcContractBankField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///银行代码
    TThostFtdcBankIDType BankID;
    ///银行分中心代码
```

```
    TThostFtdcBankBrchIDType BankBrchID;
    ///银行名称
    TThostFtdcBankNameType    BankName;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

5.2.37. RspQryParkedOrder 方法

请求查询预埋单响应。

函数原形：

```
void OnRspQryParkedOrder(CThostFtdcParkedOrderField *pParkedOrder,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID,
                bool bIsLast);
```

参数：

pParkedOrder：指向请求查询预埋单响应结构。

请求查询预埋单响应结构：

```
///预埋单
struct CThostFtdcParkedOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///报单价格条件
```

```
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TThostFtdcDirectionType     Direction;
    ///组合开平标志
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///组合投机套保标志
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量
    TThostFtdcVolumeType VolumeTotalOriginal;
    ///有效期类型
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD 日期
    TThostFtdcDateType     GTDDate;
    ///成交量类型
    TThostFtdcVolumeConditionType VolumeCondition;
    ///最小成交量
    TThostFtdcVolumeType MinVolume;
    ///触发条件
    TThostFtdcContingentConditionType    ContingentCondition;
    ///止损价
    TThostFtdcPriceType     StopPrice;
    ///强平原因
    TThostFtdcForceCloseReasonType      ForceCloseReason;
    ///自动挂起标志
    TThostFtdcBoolType     IsAutoSuspend;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///请求编号
    TThostFtdcRequestIDType    RequestID;
    ///用户强平标志
    TThostFtdcBoolType     UserForceClose;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///预埋报单编号
    TThostFtdcParkedOrderIDType     ParkedOrderID;
    ///用户类型
    TThostFtdcUserTypeType     UserType;
    ///预埋单状态
    TThostFtdcParkedOrderStatusType      Status;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

5.2.38. RspQryParkedOrderAction 方法

请求查询预埋撤单响应。

函数原形：

```
void OnRspQryParkedOrderAction(
            CThostFtdcParkedOrderActionField *pParkedOrderAction,
            CThostFtdcRspInfoField *pRspInfo,
              int nRequestID, bool bIsLast);
```

参数：

pParkedOrderAction：指向请求查询预埋撤单响应结构。

请求查询预埋撤单响应结构：

```
struct CThostFtdcParkedOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///报单操作引用
    TThostFtdcOrderActionRefType    OrderActionRef;
    ///报单引用
    TThostFtdcOrderRefType    OrderRef;
    ///请求编号
    TThostFtdcRequestIDType    RequestID;
    ///前置编号
    TThostFtdcFrontIDType FrontID;
    ///会话编号
    TThostFtdcSessionIDType    SessionID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
```

```
    ///操作标志
    TThostFtdcActionFlagType    ActionFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量变化
    TThostFtdcVolumeType VolumeChange;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///预埋撤单单编号
    TThostFtdcParkedOrderActionIDType  ParkedOrderActionID;
    ///用户类型
    TThostFtdcUserTypeType      UserType;
    ///预埋撤单状态
    TThostFtdcParkedOrderStatusType      Status;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

5.2.39. RspQryInvestorPositionCombineDetail 方法

请求查询投资者持仓明细响应。

函数原形：

```
void OnRspQryInvestorPositionCombineDetail(
        CThostFtdcInvestorPositionCombineDetailField *pInvestorPositionCombineDetail,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID, bool bIsLast);
```

参数：

pInvestorPositionCombineDetail：指向请求查询投资者持仓明细响应结构。

请求查询投资者持仓明细响应结构：

```
struct CThostFtdcInvestorPositionCombineDetailField
```

```
{
    ///交易日
    TThostFtdcDateType      TradingDay;
    ///开仓日期
    TThostFtdcDateType      OpenDate;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///结算编号
    TThostFtdcSettlementIDTypeSettlementID;
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///组合编号
    TThostFtdcTradeIDType       ComTradeID;
    ///撮合编号
    TThostFtdcTradeIDType       TradeID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///投机套保标志
    TThostFtdcHedgeFlagType    HedgeFlag;
    ///买卖
    TThostFtdcDirectionType     Direction;
    ///持仓量
    TThostFtdcVolumeType TotalAmt;
    ///投资者保证金
    TThostFtdcMoneyType  Margin;
    ///交易所保证金
    TThostFtdcMoneyType  ExchMargin;
    ///保证金率
    TThostFtdcRatioType    MarginRateByMoney;
    ///保证金率(按手数)
    TThostFtdcRatioType    MarginRateByVolume;
    ///组合持仓合约编码
    TThostFtdcInstrumentIDType      CombInstrumentID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
```

```
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

5.2.40. RspParkedOrderInsert 方法

预埋单录入请求响应。

函数原形：

```
void OnRspParkedOrderInsert(CThostFtdcParkedOrderField *pParkedOrder,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID, bool bIsLast);
```

参数：

pContractBank：指向预埋单录入请求响应结构。

预埋单录入请求响应结构：

```
struct CThostFtdcParkedOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///报单价格条件
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TThostFtdcDirectionType     Direction;
    ///组合开平标志
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///组合投机套保标志
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量
    TThostFtdcVolumeType VolumeTotalOriginal;
    ///有效期类型
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD 日期
```

```
    TThostFtdcDateType      GTDDate;
    ///成交量类型
    TThostFtdcVolumeConditionType VolumeCondition;
    ///最小成交量
    TThostFtdcVolumeType MinVolume;
    ///触发条件
    TThostFtdcContingentConditionType    ContingentCondition;
    ///止损价
    TThostFtdcPriceType     StopPrice;
    ///强平原因
    TThostFtdcForceCloseReasonType       ForceCloseReason;
    ///自动挂起标志
    TThostFtdcBoolType      IsAutoSuspend;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///请求编号
    TThostFtdcRequestIDType    RequestID;
    ///用户强平标志
    TThostFtdcBoolType      UserForceClose;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
    ///预埋报单编号
    TThostFtdcParkedOrderIDType     ParkedOrderID;
    ///用户类型
    TThostFtdcUserTypeType      UserType;
    ///预埋单状态
    TThostFtdcParkedOrderStatusType       Status;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

5.2.41. RspParkedOrderAction 方法

预埋撤单录入请求响应。

函数原形：

```
void OnRspParkedOrderAction(
                CThostFtdcParkedOrderActionField *pParkedOrderAction,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID, bool bIsLast);
```

参数：

pParkedOrderAction：指向预埋撤单录入请求响应结构。

预埋撤单录入请求响应结构：

```
struct CThostFtdcParkedOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///报单操作引用
    TThostFtdcOrderActionRefType    OrderActionRef;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///请求编号
    TThostFtdcRequestIDType     RequestID;
    ///前置编号
    TThostFtdcFrontIDType FrontID;
    ///会话编号
    TThostFtdcSessionIDType     SessionID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
    ///操作标志
    TThostFtdcActionFlagType    ActionFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量变化
    TThostFtdcVolumeType VolumeChange;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///预埋撤单单编号
    TThostFtdcParkedOrderActionIDType  ParkedOrderActionID;
    ///用户类型
```

```
    TThostFtdcUserTypeType      UserType;
    ///预埋撤单状态
    TThostFtdcParkedOrderStatusType     Status;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType      ErrorMsg;
};
```

5.2.42. RspRemoveParkedOrder 方法

删除预埋单响应。

函数原形：

```
void OnRspRemoveParkedOrder(
                CThostFtdcRemoveParkedOrderField *pRemoveParkedOrder,
                CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast);
```

参数：

pRemoveParkedOrder：指向删除预埋单响应结构。

删除预埋单响应结构：

```
struct CThostFtdcRemoveParkedOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///预埋报单编号
    TThostFtdcParkedOrderIDType     ParkedOrderID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
```

```
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

5.2.43. RspRemoveParkedOrderAction 方法

删除预埋撤单响应。

函数原形：

```
void OnRspRemoveParkedOrderAction(
        CThostFtdcRemoveParkedOrderActionField *pRemoveParkedOrderAction,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID, bool bIsLast);
```

参数：

pRemoveParkedOrderAction：指向删除预埋撤单响应结构。

删除预埋撤单响应结构：

```
struct CThostFtdcRemoveParkedOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///预埋撤单编号
    TThostFtdcParkedOrderActionIDType  ParkedOrderActionID;
};
```

pRspInfo：指向响应信息结构的地址。

响应信息结构：

```
struct CThostFtdcRspInfoField
{
    ///错误代码
    TThostFtdcErrorIDType ErrorID;
    ///错误信息
    TThostFtdcErrorMsgType    ErrorMsg;
};
```

5.3. CThostFtdcTraderApi 接口

CThostFtdcTraderApi 接口提供给用户的功能包括，报单与报价的录入、报单与报价的撤销、报单与报价的挂起、报单与报价的激活、报单与报价的修改、报单与报价的查询、成交单查询、会员客户查询、会员持仓查询、客户持仓查询、合约查询、合约交易状态查询、交易所公告查询等功能。

5.3.1. CreateFtdcTraderApi 方法

产生一个 CThostFtdcTradeApi 的一个实例，不能通过 new 来产生。

函数原形：

static CThostFtdcTradeApi *CreateFtdcTradeApi(const char *pszFlowPath = "");

参数：

pszFlowPath：常量字符指针，用于指定一个文件目录来存贮交易托管系统发布消息的状态。 默认值代表当前目录。

返回值：

返回一个指向 CThostFtdcTradeApi 实例的指针。

5.3.2. Release 方法

释放一个 CThostFtdcTradeApi 实例。不能使用 delete 方法

函数原形：

void Release();

5.3.3. Init 方法

使客户端开始与交易托管系统建立连接，连接成功后可以进行登陆。

函数原形：

void Init();

5.3.4. Join 方法

客户端等待一个接口实例线程的结束。

函数原形：

void Join()；

5.3.5. GetTradingDay 方法

获得当前交易日。只有当与交易托管系统连接建立后才会取到正确的值。

函数原形：

const char *GetTradingDay()；

返回值：

返回一个指向日期信息字符串的常量指针。

5.3.6. RegisterSpi 方法

注册一个派生自 CThostFtdcTraderSpi 接口类的实例，该实例将完成事件处理。

函数原形：

void RegisterSpi(CThostFtdcTraderSpi *pSpi) ;

参数：

pSpi：实现了 CThostFtdcTraderSpi 接口的实例指针。

5.3.7. RegisterFront 方法

设置交易托管系统的网络通讯地址，交易托管系统拥有多个通信地址，但用户只需要选择一个通信地址。

函数原形：

void RegisterFront(char *pszFrontAddress);

参数：

pszFrontAddress：指向后台服务器地址的指针。服务器地址的格式为：“protocol://ipaddress:port”，如：”tcp://127.0.0.1:17001”。 “tcp”代表传输协议，“127.0.0.1”代表服务器地址。”17001”代表服务器端口号。

5.3.8. SubscribePrivateTopic 方法

订阅私有流。该方法要在 Init 方法前调用。若不调用则不会收到私有流的数据。

函数原形：

void SubscribePrivateTopic(TE_RESUME_TYPE nResumeType);

参数：

nResumeType： 私有流重传方式

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

5.3.9. SubscribePublicTopic 方法

订阅公共流。该方法要在 Init 方法前调用。若不调用则不会收到公共流的数据。

函数原形：

void SubscribePublicTopic(TE_RESUME_TYPE nResumeType);

参数：

nResumeType： 公共流重传方式

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

5.3.10. ReqUserLogin 方法

用户发出登陆请求。

函数原形：

```
int ReqUserLogin(
                CThostFtdcReqUserLoginField *pReqUserLoginField,
                int nRequestID);
```

参数：

pReqUserLoginField：指向用户登录请求结构的地址。

用户登录请求结构：

```
struct CThostFtdcReqUserLoginField
{
    ///交易日
    TThostFtdcDateType    TradingDay;
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///密码
    TThostFtdcPasswordType    Password;
    ///用户端产品信息
    TThostFtdcProductInfoType  UserProductInfo;
    ///接口端产品信息
    TThostFtdcProductInfoType  InterfaceProductInfo;
    ///协议信息
    TThostFtdcProtocolInfoType ProtocolInfo;
};
```

nRequestID：用户登录请求的 ID，该 ID 由用户指定，管理。

用户需要填写 UserProductInfo 字段，即客户端的产品信息，如软件开发商、版本号等，例如：SFITTraderV100。

InterfaceProductInfo 和 ProtocolInfo 只须占位，不必有效赋值。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.11. ReqUserLogout 方法

用户发出登出请求。

函数原形：

```
int ReqUserLogout(
                    CThostFtdcUserLogoutField *pUserLogout,
                    int nRequestID);
```

参数：

pReqUserLogout：指向用户登出请求结构的地址。

用户登出请求结构：

```
struct CThostFtdcUserLogoutField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///用户代码
    TThostFtdcUserIDType  UserID;
};
```

nRequestID：用户登出请求的 ID，该 ID 由用户指定，管理。

返回值：

0,代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.12. ReqUserPasswordUpdate 方法

用户密码修改请求。

函数原形：

```
int ReqUserPasswordUpdate(
                                CThostFtdcUserPasswordUpdateField *pUserPasswordUpdate,
```

int nRequestID)；

参数：

pUserPasswordUpdate：指向用户口令修改结构的地址。

用户口令修改结构：

```
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;
};
```

nRequestID：用户操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.13. ReqTradingAccountPasswordUpdate 方法

资金账户口令更新请求。

函数原形：

```
int ReqTradingAccountPasswordUpdate(
                        CThostFtdcTradingAccountPasswordUpdateField
                        *pTradingAccountPasswordUpdate,
                        int nRequestID) ;
```

参数：

pUserPasswordUpdate：指向资金账户口令修改结构的地址。

资金账户口令修改结构：

```
struct CThostFtdcTradingAccountPasswordUpdateField
{
```

```
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者帐号
    TThostFtdcAccountIDType     AccountID;
    ///原来的口令
    TThostFtdcPasswordType      OldPassword;
    ///新的口令
    TThostFtdcPasswordType      NewPassword;
};
```

nRequestID：用户操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.14. ReqOrderInsert 方法

客户端发出报单录入请求。

函数原形：

```
int ReqOrderInsert(
                    CThostFtdcInputOrderField *pInputOrder,
                    int nRequestID);
```

参数：

pInputOrder：指向输入报单结构的地址。

输入报单结构：

```
struct CThostFtdcInputOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///报单引用
```

```
    TThostFtdcOrderRefType      OrderRef;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///报单价格条件
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TThostFtdcDirectionType      Direction;
    ///组合开平标志
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///组合投机套保标志
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量
    TThostFtdcVolumeType VolumeTotalOriginal;
    ///有效期类型
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD 日期
    TThostFtdcDateType      GTDDate;
    ///成交量类型
    TThostFtdcVolumeConditionType VolumeCondition;
    ///最小成交量
    TThostFtdcVolumeType MinVolume;
    ///触发条件
    TThostFtdcContingentConditionType    ContingentCondition;
    ///止损价
    TThostFtdcPriceType     StopPrice;
    ///强平原因
    TThostFtdcForceCloseReasonType       ForceCloseReason;
    ///自动挂起标志
    TThostFtdcBoolType      IsAutoSuspend;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///请求编号
    TThostFtdcRequestIDType    RequestID;
};
```

nRequestID：用户报单请求的 ID，该 ID 由用户指定，管理。在一次会话中，该 ID 不能重复。

OrderRef：报单引用，只能单调递增。每次登入成功后，可以从 OnRspUserLogin 的输出参数 CThostFtdcRspUserLoginField 中获得上次登入用过的最大 OrderRef, MaxOrderRef。

因为交易后台按照字符串比较 OrdeRef 的大小，所以在设置 OrderRef 时要填满 TThostFtdcOrderRefType 的全部空间。

返回值：

0，代表成功；

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.15. ReqOrderAction 方法

客户端发出报单操作请求，包括报单的撤销、报单的挂起、报单的激活、报单的修改。

函数原形：

```
int ReqOrderAction(
                    CThostFtdcOrderActionField *pOrderAction,
                    int nRequestID);
```

参数：

pOrderAction：指向报单操作结构的地址。

报单操作结构：

```
struct CThostFtdcOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///报单操作引用
    TThostFtdcOrderActionRefType    OrderActionRef;
    ///报单引用
    TThostFtdcOrderRefType      OrderRef;
    ///请求编号
    TThostFtdcRequestIDType     RequestID;
    ///前置编号
    TThostFtdcFrontIDType FrontID;
```

```
    ///会话编号
    TThostFtdcSessionIDType     SessionID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
    ///操作标志
    TThostFtdcActionFlagType    ActionFlag;
    ///价格
    TThostFtdcPriceType     LimitPrice;
    ///数量变化
    TThostFtdcVolumeType VolumeChange;
    ///操作日期
    TThostFtdcDateType      ActionDate;
    ///操作时间
    TThostFtdcTimeType      ActionTime;
    ///交易所交易员代码
    TThostFtdcTraderIDType      TraderID;
    ///安装编号
    TThostFtdcInstallIDType      InstallID;
    ///本地报单编号
    TThostFtdcOrderLocalIDType      OrderLocalID;
    ///操作本地编号
    TThostFtdcOrderLocalIDType      ActionLocalID;
    ///会员代码
    TThostFtdcParticipantIDType      ParticipantID;
    ///客户代码
    TThostFtdcClientIDType      ClientID;
    ///业务单元
    TThostFtdcBusinessUnitTypeBusinessUnit;
    ///报单操作状态
    TThostFtdcOrderActionStatusType      OrderActionStatus;
    ///用户代码
    TThostFtdcUserIDType  UserID;
    ///状态信息
    TThostFtdcErrorMsgType      StatusMsg;
};
```

nRequestID：用户报单操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.16. ReqQueryMaxOrderVolume 方法

查询最大报单数量请求。

函数原形：

```
int ReqQueryMaxOrderVolume(
    CThostFtdcQueryMaxOrderVolumeField *pQueryMaxOrderVolume,
    int nRequestID);
```

参数：

pQueryMaxOrderVolume：指向查询最大报单数量结构的地址。

查询最大报单数量结构：

```
struct CThostFtdcQueryMaxOrderVolumeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
    ///买卖方向
    TThostFtdcDirectionType    Direction;
    ///开平标志
    TThostFtdcOffsetFlagType    OffsetFlag;
    ///投机套保标志
    TThostFtdcHedgeFlagType    HedgeFlag;
    ///最大允许报单数量
    TThostFtdcVolumeType MaxVolume;
};
```

nRequestID：用户报单操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.17. ReqSettlementInfoConfirm 方法

投资者结算结果确认。

函数原形：

```
int ReqSettlementInfoConfirm(
        CThostFtdcSettlementInfoConfirmField *pSettlementInfoConfirm,
        int nRequestID);
```

参数：

pSettlementInfoConfirm：指向投资者结算结果确认结构的地址。

投资者结算结果确认结构：

```
struct CThostFtdcSettlementInfoConfirmField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///确认日期
    TThostFtdcDateType    ConfirmDate;
    ///确认时间
    TThostFtdcTimeType    ConfirmTime;
};
```

nRequestID：用户报单操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.18. ReqTransferBankToFuture 方法

请求银行资金转期货。

函数原形：

```
int ReqTransferBankToFuture(
        CThostFtdcTransferHeaderField *pTransferHeader,
```

CThostFtdcTransferBankToFutureReqField *pTransferBankToFutureReq,
int nRequestID);

参数：

pTransferHeader：指向银期转帐报文头结构的地址。

pTransferBankToFutureReq：指向银行资金转期货请求结构的地址。

银期转帐报文头结构：

```
struct CThostFtdcTransferHeaderField
{
    ///版本号，常量，1.0
    TThostFtdcVersionType Version;
    ///交易代码，必填
    TThostFtdcTradeCodeType    TradeCode;
    ///交易日期，必填，格式：yyyymmdd
    TThostFtdcTradeDateType    TradeDate;
    ///交易时间，必填，格式：hhmmss
    TThostFtdcTradeTimeType    TradeTime;
    ///发起方流水号，N/A
    TThostFtdcTradeSerialType  TradeSerial;
    ///期货公司代码，必填
    TThostFtdcFutureIDType     FutureID;
    ///银行代码，根据查询银行得到，必填
    TThostFtdcBankIDType BankID;
    ///银行分中心代码，根据查询银行得到，必填
    TThostFtdcBankBrchIDType BankBrchID;
    ///操作员，N/A
    TThostFtdcOperNoType OperNo;
    ///交易设备类型，N/A
    TThostFtdcDeviceIDType     DeviceID;
    ///记录数，N/A
    TThostFtdcRecordNumType  RecordNum;
    ///会话编号，N/A
    TThostFtdcSessionIDType    SessionID;
    ///请求编号，N/A
    TThostFtdcRequestIDType    RequestID;
};
```

银行资金转期货请求结构：

```
struct CThostFtdcTransferBankToFutureReqField
{
    ///期货资金账户
    TThostFtdcAccountIDType    FutureAccount;
    ///密码标志
```

```
    TThostFtdcFuturePwdFlagType    FuturePwdFlag;
    ///密码
    TThostFtdcFutureAccPwdType    FutureAccPwd;
    ///转账金额
    TThostFtdcMoneyType  TradeAmt;
    ///客户手续费
    TThostFtdcMoneyType  CustFee;
    ///币种：RMB-人民币 USD-美圆 HKD-港元
    TThostFtdcCurrencyCodeType    CurrencyCode;
};
```

nRequestID：用户报单操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.19. ReqTransferFutureToBank 方法

请求期货资金转银行。

函数原形：

```
int ReqTransferFutureToBank(
        CThostFtdcTransferHeaderField *pTransferHeader,
        CThostFtdcTransferFutureToBankReqField *pTransferFutureToBankReq,
        int nRequestID);
```

参数：

pTransferHeader：指向银期转帐报文头结构的地址。

pTransferFutureToBankReq：指向期货资金转银行请求结构的地址。

银期转帐报文头结构：

```
struct CThostFtdcTransferHeaderField
{
    ///版本号，常量，1.0
    TThostFtdcVersionType Version;
    ///交易代码，必填
    TThostFtdcTradeCodeType    TradeCode;
```

```
    ///交易日期，必填，格式：yyyymmdd
    TThostFtdcTradeDateType    TradeDate;
    ///交易时间，必填，格式：hhmmss
    TThostFtdcTradeTimeType    TradeTime;
    ///发起方流水号，N/A
    TThostFtdcTradeSerialType  TradeSerial;
    ///期货公司代码，必填
    TThostFtdcFutureIDType     FutureID;
    ///银行代码，根据查询银行得到，必填
    TThostFtdcBankIDType BankID;
    ///银行分中心代码，根据查询银行得到，必填
    TThostFtdcBankBrchIDType BankBrchID;
    ///操作员，N/A
    TThostFtdcOperNoType OperNo;
    ///交易设备类型，N/A
    TThostFtdcDeviceIDType     DeviceID;
    ///记录数，N/A
    TThostFtdcRecordNumType  RecordNum;
    ///会话编号，N/A
    TThostFtdcSessionIDType    SessionID;
    ///请求编号，N/A
    TThostFtdcRequestIDType    RequestID;
};
```

期货资金转银行请求(TradeCode=202002)结构：

```
struct CThostFtdcTransferFutureToBankReqField
{
    ///期货资金账户
    TThostFtdcAccountIDType    FutureAccount;
    ///密码标志
    TThostFtdcFuturePwdFlagType    FuturePwdFlag;
    ///密码
    TThostFtdcFutureAccPwdType     FutureAccPwd;
    ///转账金额
    TThostFtdcMoneyType  TradeAmt;
    ///客户手续费
    TThostFtdcMoneyType  CustFee;
    ///币种：RMB-人民币 USD-美圆 HKD-港元
    TThostFtdcCurrencyCodeType     CurrencyCode;
};
```

nRequestID：用户报单操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.20. ReqTransferQryBank 方法

请求查询银行资金。

函数原形：

```
int ReqTransferQryBank(
        CThostFtdcTransferHeaderField *pTransferHeader,
        CThostFtdcTransferQryBankReqField *pTransferQryBankReq,
        int nRequestID);
```

参数：

pTransferHeader：指向银期转帐报文头结构的地址。

pTransferQryBankReq：指向查询银行资金请求结构的地址。

银期转帐报文头结构：

```
struct CThostFtdcTransferHeaderField
{
    ///版本号，常量，1.0
    TThostFtdcVersionType Version;
    ///交易代码，必填
    TThostFtdcTradeCodeType    TradeCode;
    ///交易日期，必填，格式：yyyymmdd
    TThostFtdcTradeDateType    TradeDate;
    ///交易时间，必填，格式：hhmmss
    TThostFtdcTradeTimeType    TradeTime;
    ///发起方流水号，N/A
    TThostFtdcTradeSerialType  TradeSerial;
    ///期货公司代码，必填
    TThostFtdcFutureIDType     FutureID;
    ///银行代码，根据查询银行得到，必填
    TThostFtdcBankIDType BankID;
    ///银行分中心代码，根据查询银行得到，必填
    TThostFtdcBankBrchIDType BankBrchID;
    ///操作员，N/A
    TThostFtdcOperNoType OperNo;
    ///交易设备类型，N/A
```

```
    TThostFtdcDeviceIDType     DeviceID;
    ///记录数，N/A
    TThostFtdcRecordNumType  RecordNum;
    ///会话编号，N/A
    TThostFtdcSessionIDType     SessionID;
    ///请求编号，N/A
    TThostFtdcRequestIDType     RequestID;
};
```

查询银行资金请求（TradeCode=204002）结构：

```
struct CThostFtdcTransferQryBankReqField
{
    ///期货资金账户
    TThostFtdcAccountIDType    FutureAccount;
    ///密码标志
    TThostFtdcFuturePwdFlagType     FuturePwdFlag;
    ///密码
    TThostFtdcFutureAccPwdType     FutureAccPwd;
    ///币种：RMB-人民币 USD-美圆 HKD-港元
    TThostFtdcCurrencyCodeType     CurrencyCode;
};
```

nRequestID：用户报单操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.21. ReqTransferQryDetail 方法

请求查询银行交易明细。

函数原形：

```
int ReqTransferQryDetail(
                CThostFtdcTransferHeaderField *pTransferHeader,
                CThostFtdcTransferQryDetailReqField *pTransferQryDetailReq,
                int nRequestID);
```

参数：

pTransferHeader：指向银期转帐报文头结构的地址。

pTransferQryDetailReq：指向查询银行交易明细请求结构的地址。

银期转帐报文头结构：

```
struct CThostFtdcTransferHeaderField
{
    ///版本号，常量，1.0
    TThostFtdcVersionType Version;
    ///交易代码，必填
    TThostFtdcTradeCodeType    TradeCode;
    ///交易日期，必填，格式：yyyymmdd
    TThostFtdcTradeDateType    TradeDate;
    ///交易时间，必填，格式：hhmmss
    TThostFtdcTradeTimeType    TradeTime;
    ///发起方流水号，N/A
    TThostFtdcTradeSerialType  TradeSerial;
    ///期货公司代码，必填
    TThostFtdcFutureIDType     FutureID;
    ///银行代码，根据查询银行得到，必填
    TThostFtdcBankIDType BankID;
    ///银行分中心代码，根据查询银行得到，必填
    TThostFtdcBankBrchIDType BankBrchID;
    ///操作员，N/A
    TThostFtdcOperNoType OperNo;
    ///交易设备类型，N/A
    TThostFtdcDeviceIDType     DeviceID;
    ///记录数，N/A
    TThostFtdcRecordNumType  RecordNum;
    ///会话编号，N/A
    TThostFtdcSessionIDType    SessionID;
    ///请求编号，N/A
    TThostFtdcRequestIDType    RequestID;
};
```

查询银行交易明细请求(TradeCode=204999)结构：

```
struct CThostFtdcTransferQryDetailReqField
{
    ///期货资金账户
    TThostFtdcAccountIDType    FutureAccount;
};
```

nRequestID：用户报单操作请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.22. ReqQryOrder 方法

报单查询请求。

函数原形：

```
int ReqQryOrder(
                CThostFtdcQryOrderField *pQryOrder,
                int nRequestID);
```

参数：

pQryOrder：指向报单查询结构的地址。

报单查询结构：

```
struct CThostFtdcQryOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType     BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType   InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///报单编号
    TThostFtdcOrderSysIDType  OrderSysID;
};
```

nRequestID：用户报单查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

注：不写 BrokerID 可以收全所有报单。

5.3.23. ReqQryTrade 方法

成交单查询请求。

函数原形：

```
int ReqQryTrade(
                CThostFtdcQryTradeField *pQryTrade,
                int nRequestID);
```

参数：

pQryTrade：指向成交查询结构的地址。

成交查询结构：

```
struct CThostFtdcQryTradeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///成交编号
    TThostFtdcTradeIDType       TradeID;
};
```

nRequestID：用户成交单查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.24. ReqQry Investor 方法

会员客户查询请求。

函数原形：

```
int ReqQry Investor (
                CThostFtdcQryInvestorField *pQryInvestor,
                int nRequestID);
```

参数：

pQry Investor：指向客户查询结构的地址。

客户查询结构：

```
struct CThostFtdcQryInvestorField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
};
```

nRequestID：用户客户查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.25. ReqQryInvestorPosition 方法

会员持仓查询请求。

函数原形：

```
int ReqQryInvestorPosition(
                CThostFtdcQryInvestorPositionField *pQryInvestorPosition,
                int nRequestID);
```

参数：

pQryInvestorPosition：指向会员持仓查询结构的地址。

会员持仓查询结构：

```
struct CThostFtdcQryInvestorPositionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
};
```

nRequestID：会员持仓查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.26. ReqQryTradingAccount 方法

请求查询资金账户。

函数原形：

```
int ReqQryTradingAccount(
                CThostFtdcQryTradingAccountField *pQryTradingAccount,
                int nRequestID);
```

参数：

pQryTradingAccount：指向查询资金账户结构的地址。

查询资金账户结构：

```
struct CThostFtdcQryTradingAccountField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
};
```

nRequestID：会员持仓查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.27. ReqQryTradingCode 方法

请求查询交易编码。

函数原形：

```
int ReqQryTradingCode(
                    CThostFtdcQryTradingCodeField *pQryTradingCode,
                    int nRequestID);
```

参数：

pQryTradingCode：指向查询交易编码结构的地址。

查询交易编码结构：

```
struct CThostFtdcQryTradingCodeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///交易编码
    TThostFtdcClientIDType      ClientID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.28. ReqQryExchange 方法

请求查询交易所。

函数原形：

```
int ReqQryExchange(
        CThostFtdcQryExchangeField *pQryExchange,
        int nRequestID);
```

参数：

pQryExchange：指向查询交易编码结构的地址。

查询交易所编码结构：

```
struct CThostFtdcQryExchangeField
{
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.29. ReqQryInstrument 方法

请求查询合约。

函数原形：

```
int ReqQryInstrument(
```

CThostFtdcQryInstrumentField *pQryInstrument,
int nRequestID);

参数：

pQryInstrument：指向查询查询合约结构的地址。

查询合约结构：

```
struct CThostFtdcQryInstrumentField
{
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
    ///交易所代码
    TThostFtdcExchangeIDType ExchangeID;
    ///合约在交易所的代码
    TThostFtdcExchangeInstIDType   ExchangeInstID;
    ///产品代码
    TThostFtdcInstrumentIDType      ProductID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.30. ReqQryDepthMarketData 方法

请求查询行情。

函数原形：

```
int ReqQryDepthMarketData(
                CThostFtdcQryDepthMarketDataField *pQryDepthMarketData,
                int nRequestID);
```

参数：

pQryDepthMarketData：指向查询行情结构的地址。

查询行情结构：

```
struct CThostFtdcQryDepthMarketDataField
```

```
{
    ///合约代码
    TThostFtdcInstrumentIDType      InstrumentID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.31. ReqQrySettlementInfo 方法

请求查询投资者结算结果。

函数原形：

```
int ReqQrySettlementInfo(
        CThostFtdcQrySettlementInfoField *pQrySettlementInfo,
        int nRequestID);
```

参数：

pQrySettlementInfo：指向查询投资者结算结果的地址。

查询投资者结算结果结构：

```
struct CThostFtdcQrySettlementInfoField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///交易日
    TThostFtdcDateType      TradingDay;
```

};

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.32. ReqQryTransferBank 方法

请求查询转帐银行。

函数原形：

```
int ReqQryTransferBank(
                    CThostFtdcQryTransferBankField *pQryTransferBank,
                    int nRequestID);
```

参数：

pQryTransferBank：指向查询转帐银行结构的地址。

查询转帐银行结构：

```
struct CThostFtdcQryTransferBankField
{
    ///银行代码
    TThostFtdcBankIDType        BankID;
    ///银行分中心代码
    TThostFtdcBankBrchIDType    BankBrchID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.33. ReqQryInvestorPositionDetail 方法

请求查询投资者持仓明细。

函数原形：

```
int ReqQryInvestorPositionDetail(
        CThostFtdcQryInvestorPositionDetailField *pQryInvestorPositionDetail,
        int nRequestID);
```

参数：

pQryInvestorPositionDetail：指向查询投资者持仓明细结构的地址。

查询投资者持仓明细结构：

```
struct CThostFtdcQryInvestorPositionDetailField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType    InstrumentID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.34. ReqQryNotice 方法

请求查询客户通知。

函数原形：

```
int ReqQryNotice(
                CThostFtdcQryNoticeField *pQryNotice,
                int nRequestID);
```

参数：

pQryNotice：指向查询客户通知结构的地址。

查询客户通知结构：

```
struct CThostFtdcQryNoticeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.35. ReqQrySettlementInfoConfirm 方法

查询结算信息确认。

函数原形：

```
int ReqQrySettlementInfoConfirm(
                    CThostFtdcQrySettlementInfoConfirmField
                    *pQrySettlementInfoConfirm,
                    int nRequestID);
```

参数：

pQrySettlementInfoConfirm：指向查询结算信息确认结构的地址。

查询结算信息确认：

```
struct CThostFtdcQrySettlementInfoConfirmField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.36. ReqQryContractBank 方法

请求查询签约银行。

函数原形：

```
int ReqQryContractBank(
                    CThostFtdcQryContractBankField *pQryContractBank,
                    int nRequestID);
```

参数：

pQryContractBank：指向查询签约银行请求结构的地址。其中 BrokerID 不能为空，BankID 和 BankBrchID 可以为空。

查询签约银行请求结构：

```
struct CThostFtdcQryContractBankField
{
```

```
    ///经纪公司代码
    TThostFtdcBrokerIDType      BrokerID;
    ///银行代码
    TThostFtdcBankIDType        BankID;
    ///银行分中心代码
    TThostFtdcBankBrchIDType  BankBrchID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.37. ReqQryParkedOrder 方法

请求查询预埋单。

函数原形：

```
int ReqQryParkedOrder(CThostFtdcQryParkedOrderField *pQryParkedOrder,
                      int nRequestID);
```

参数：

pQryParkedOrder：指向查询预埋单请求结构的地址。

查询预埋单请求结构：

```
struct CThostFtdcQryParkedOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDTypeBrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
```

TThostFtdcInstrumentIDType InstrumentID;

///交易所代码

TThostFtdcExchangeIDType ExchangeID;

};

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.38. ReqQryParkedOrderAction 方法

请求查询预埋撤单。

函数原形：

```
int ReqQryParkedOrderAction(
                CThostFtdcQryParkedOrderActionField *pQryParkedOrderAction,
                 int nRequestID);
```

参数：

pQryParkedOrderAction：指向请求查询预埋撤单结构的地址。

请求查询预埋撤单结构：

```
struct CThostFtdcQryParkedOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDTypeBrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///交易所代码
    TThostFtdcExchangeIDType  ExchangeID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.39. ReqQryInvestorPositionCombineDetail 方法

请求查询投资者组合持仓明细。

函数原形：

```
int ReqQryInvestorPositionCombineDetail(
    CThostFtdcQryInvestorPositionCombineDetailField *pQryInvestorPositionCombineDetail,
     int nRequestID);;
```

参数：

pQryInvestorPositionCombineDetail：指向请求查询投资者组合持仓明细请求结构的地址。其中 BrokerID 不能为空，BankID 和 BankBrchID 可以为空。

查询组合持仓明细请求结构：

```
struct CThostFtdcQryInvestorPositionCombineDetailField
{
    ///经纪公司代码
    TThostFtdcBrokerIDTypeBrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///组合持仓合约编码
    TThostFtdcInstrumentIDType CombInstrumentID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.40. ReqParkedOrderInsert 方法

预埋单录入请求。

函数原形：

```
int ReqParkedOrderInsert (CThostFtdcParkedOrderField *pParkedOrder,
                          int nRequestID);
```

参数：

pParkedOrder：指向预埋单录入请求结构的地址。

预埋单结构：

```
struct CThostFtdcParkedOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDTypeBrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///合约代码
    TThostFtdcInstrumentIDType InstrumentID;
    ///报单引用
    TThostFtdcOrderRefTypeOrderRef;
    ///用户代码
    TThostFtdcUserIDType    UserID;
    ///报单价格条件
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///买卖方向
    TThostFtdcDirectionTypeDirection;
    ///组合开平标志
```

```
TThostFtdcCombOffsetFlagType   CombOffsetFlag;
///组合投机套保标志
TThostFtdcCombHedgeFlagType  CombHedgeFlag;
///价格
TThostFtdcPriceTypeLimitPrice;
///数量
TThostFtdcVolumeType   VolumeTotalOriginal;
///有效期类型
TThostFtdcTimeConditionType     TimeCondition;
///GTD 日期
TThostFtdcDateType GTDDate;
///成交量类型
TThostFtdcVolumeConditionType VolumeCondition;
///最小成交量
TThostFtdcVolumeType   MinVolume;
///触发条件
TThostFtdcContingentConditionType   ContingentCondition;
///止损价
TThostFtdcPriceTypeStopPrice;
///强平原因
TThostFtdcForceCloseReasonType      ForceCloseReason;
///自动挂起标志
TThostFtdcBoolType IsAutoSuspend;
///业务单元
TThostFtdcBusinessUnitType BusinessUnit;
///请求编号
TThostFtdcRequestIDType     RequestID;
///用户强平标志
TThostFtdcBoolType UserForceClose;
```

///交易所代码

TThostFtdcExchangeIDType ExchangeID;

///预埋报单编号

TThostFtdcParkedOrderIDType ParkedOrderID;

///用户类型

TThostFtdcUserTypeType UserType;

///预埋单状态

TThostFtdcParkedOrderStatusType Status;

};

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.41. ReqParkedOrderAction 方法

预埋撤单录入请求。

函数原形：

```
int ReqParkedOrderAction(CThostFtdcParkedOrderActionField *pParkedOrderAction,
                         int nRequestID);
```

参数：

pParkedOrderAction：指向预埋撤单录入请求结构的地址。

预埋撤单录入请求结构：

struct CThostFtdcParkedOrderActionField

{

///经纪公司代码

TThostFtdcBrokerIDTypeBrokerID;

///投资者代码

```
TThostFtdcInvestorIDType     InvestorID;
///报单操作引用
TThostFtdcOrderActionRefType    OrderActionRef;
///报单引用
TThostFtdcOrderRefTypeOrderRef;
///请求编号
TThostFtdcRequestIDType     RequestID;
///前置编号
TThostFtdcFrontIDType   FrontID;
///会话编号
TThostFtdcSessionIDType     SessionID;
///交易所代码
TThostFtdcExchangeIDType   ExchangeID;
///报单编号
TThostFtdcOrderSysIDType   OrderSysID;
///操作标志
TThostFtdcActionFlagType     ActionFlag;
///价格
TThostFtdcPriceTypeLimitPrice;
///数量变化
TThostFtdcVolumeType   VolumeChange;
///用户代码
TThostFtdcUserIDType    UserID;
///合约代码
TThostFtdcInstrumentIDType InstrumentID;
///预埋撤单单编号
TThostFtdcParkedOrderActionIDType ParkedOrderActionID;
///用户类型
TThostFtdcUserTypeType      UserType;
```

```
    ///预埋撤单状态
    TThostFtdcParkedOrderStatusType      Status;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.42. ReqRemoveParkedOrder 方法

请求删除预埋单。

函数原形：

```
int ReqRemoveParkedOrder(CThostFtdcRemoveParkedOrderField *pRemoveParkedOrder,
                         int nRequestID);
```

参数：

pRemoveParkedOrder：指向请求删除预埋单结构的地址。

请求删除预埋单结构：

```
struct CThostFtdcRemoveParkedOrderField
{
    ///经纪公司代码
    TThostFtdcBrokerIDTypeBrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType     InvestorID;
    ///预埋报单编号
    TThostFtdcParkedOrderIDType     ParkedOrderID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

5.3.43. ReqRemoveParkedOrderAction 方法

请求删除预埋撤单。

函数原形：

```
Int ReqRemoveParkedOrderAction(
        CThostFtdcRemoveParkedOrderActionField *pRemoveParkedOrderAction,
        int nRequestID);;
```

参数：

pRemoveParkedOrderAction：指向请求删除预埋撤单结构的地址。

请求删除预埋撤单结构：

```
struct CThostFtdcRemoveParkedOrderActionField
{
    ///经纪公司代码
    TThostFtdcBrokerIDTypeBrokerID;
    ///投资者代码
    TThostFtdcInvestorIDType    InvestorID;
    ///预埋撤单编号
    TThostFtdcParkedOrderActionIDType ParkedOrderActionID;
};
```

nRequestID：合约查询请求的 ID，该 ID 由用户指定，管理。

返回值：

0，代表成功。

-1，表示网络连接失败；

-2，表示未处理请求超过许可数；

-3，表示每秒发送请求数超过许可数。

Chapter 6. 开发示例

7.1 交易 API 开发示例

```
    // 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;
}
```

7.2 行情 API 开发示例

```
// tradeapitest.cpp :
// 一个简单的例子，介绍CThostFtdcMdApi和CThostFtdcMdSpi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 "ThostFtdcMdApi.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 CThostFtdcMdSpi
{
public:
    // 构造函数，需要一个有效的指向CThostFtdcMdApi实例的指针
    CSimpleHandler(CThostFtdcMdApi *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
            // 订阅行情
           char  * Instrumnet[]={ “IF0809” ,” IF0812” };
            pUserApi->SubscribeMarketData (Instrumnet,2);
           //或退订行情
           pUserApi->UnSubscribeMarketData (Instrumnet,2);
        }
        // 行情应答
        virtual    void    OnRtnDepthMarketData(CThostFtdcDepthMarketDataField
*pDepthMarketData)
        {
            // 输出报单录入结果
            printf("ErrorCode=[%d],     ErrorMsg=[%s]\n",     pRspInfo->ErrorID,
pRspInfo->ErrorMsg);

            //收到深度行情
            SetEvent(g_hEvent);
        };


        // 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CThostFtdcMdApi实例的指针
        CThostFtdcMdApi *m_pUserApi;
    };
    int main()
    {
        // 产生一个CThostFtdcMdApi实例
        CThostFtdcMdApi *pUserApi = CThostFtdcMdApi::CreateFtdcMdApi();
        // 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);



        // 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;
}
```