交易托管系统

交易员应用程序接口

2008 年 7 月

1. 文件属性

文件属性	内容
文件名称	交易托管系统_ TradeAPI 接口
文件编号	
文件版本号	V0.1
文件状态	草稿
作　　者	上海期货信息技术有限公司
文档编写日期	2008-7-28
文档发布日期	

2. 文件变更历史清单

文件版本号	修正日期	修正人	备　　注
1.1	2008-1-11	托管平台开发组	1、 增加银期转账接口 2、 增加接收合约状态接口
1.1.1	2008-1-29	托管平台开发组	1、增加查询客户通知功能；
1.2	2008-3-4	托管平台开发组	1、增加查询历史结算单的功能（使用查询结算单接口）。 2、增加查询结算确认接口。 3、修改资金密码修改接口。

			4、删除行情接口中查询深度行情的接口，交易接口中的深度行情查询可以使用。 5、增加行情客户端开发示例。
1.3	2008-5-23	托管平台开发组	1、持仓记录中增加了三个字段，查询需要做处理。 2、持仓明细中增加了两个字段查询需要做处理。 3、根据OrderRef撤单的接口增加了合约代码字段。 4、允许客户报套利单，套利单的价格可以为0或者负数,这在单一报单是不允许的，需要客户端处理。 5、转帐报文头中增加RequestID字段，需要填写，内容与API中的RequestID相同，用于支持异步交易的处理。
2.1.2a	2008-7-28	托管平台开发组	在 CThostFtdcTraderApi

			接口中增加了 16 中方法：资金账户口令更新请求、查询最大报单数量、投资者结算结果确认、请求银行资金转期货、请求期货资金转银行、请求查询银行资金、请求查询银行交易明细、请求查询资金账户、请求查询交易编码、请求查询交易所、请求查询合约、请求查询行情、请求查询投资者结算结果、请求查询转帐银行、请求查询投资者持仓明细、请求查询客户、请求查询签约银行通知。 在 CThostFtdcTraderSpi 接口中增加了对以上 16 中方法的响应方法。 总共新增 32 个方法。

3. 本次修改变更说明

序号	变更内容简述
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	

目 录

1. 介绍

交易托管系统 API 是一个基于 C++的类库，通过使用和扩展类库提供的接口来实现交易、行情相关功能。

交易 API 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、投资者查询、投资者持仓查询、合约查询、交易日获取等。

行情 API 包括订阅行情、退订行情，处理深度行情回报。

文件名	版本	文件大小	文件描述
FtdcUserApiStruct.h	V1.0	45,500 字节	定义了 API 所需的一系列数据类型的头文件
FtdcUserApiDataType.h	V1.0	36,509 字节	定义了一系列业务相关的数据结构的头文件
ThostFtdcTraderApi.h	V1.0	6,600 字节	交易接口头文件
ThostTraderApi.dll	V1.0	331,776 字节	交易动态链接库二进制文件
ThostTraderApi.lib	V1.0	3,562 字节	交易导入库文件
ThostFtdcMdApi.h	V1.0	3,998 字节	行情接口头文件
ThostFtdcMdApi.dll	V1.0	376,832 字节	行情动态链接库二进制文件
ThostFtdcMdApi.lib	V1.0	1,792 字节	行情导入库文件

支持 MS VC 6.0，MS VC.NET 2003 编译器。需要打开多线程编译选项/MT。

2. 体系结构

交易员 API 使用建立在 TCP 协议之上 FTD 协议与交易托管系统进行通讯，交易托管系统负责投资者的交易业务处理。

2.1. 通讯模式

FTD 协议中的所有通讯都基于某个通讯模式。通讯模式实际上就是通讯双方协同工作的方式。

FTD 涉及的通讯模式共有三种：

- 对话通讯模式
- 私有通讯模式
- 广播通讯模式

对话通讯模式是指由会员端主动发起的通讯请求。该请求被交易所端接收和处理，并给予响应。例如报单、查询等。这种通讯模式与普通的客户/服务器模式相同。

私有通讯模式是指交易所端主动，向某个特定的会员发出的信息。例如成交回报等。

广播通讯模式是指交易所端主动，向市场中的所有会员都发出相同的信息。例如公告、市场公共信息等。

通讯模式和网络的连接不一定存在简单的一对一的关系。也就是说，一个网络连接中可能传送多种不同通讯模式的报文，一种通讯模式的报文也可以在多个不同的连接中传送。

无论哪种通讯模式，其通讯过程都如图 1 所示：

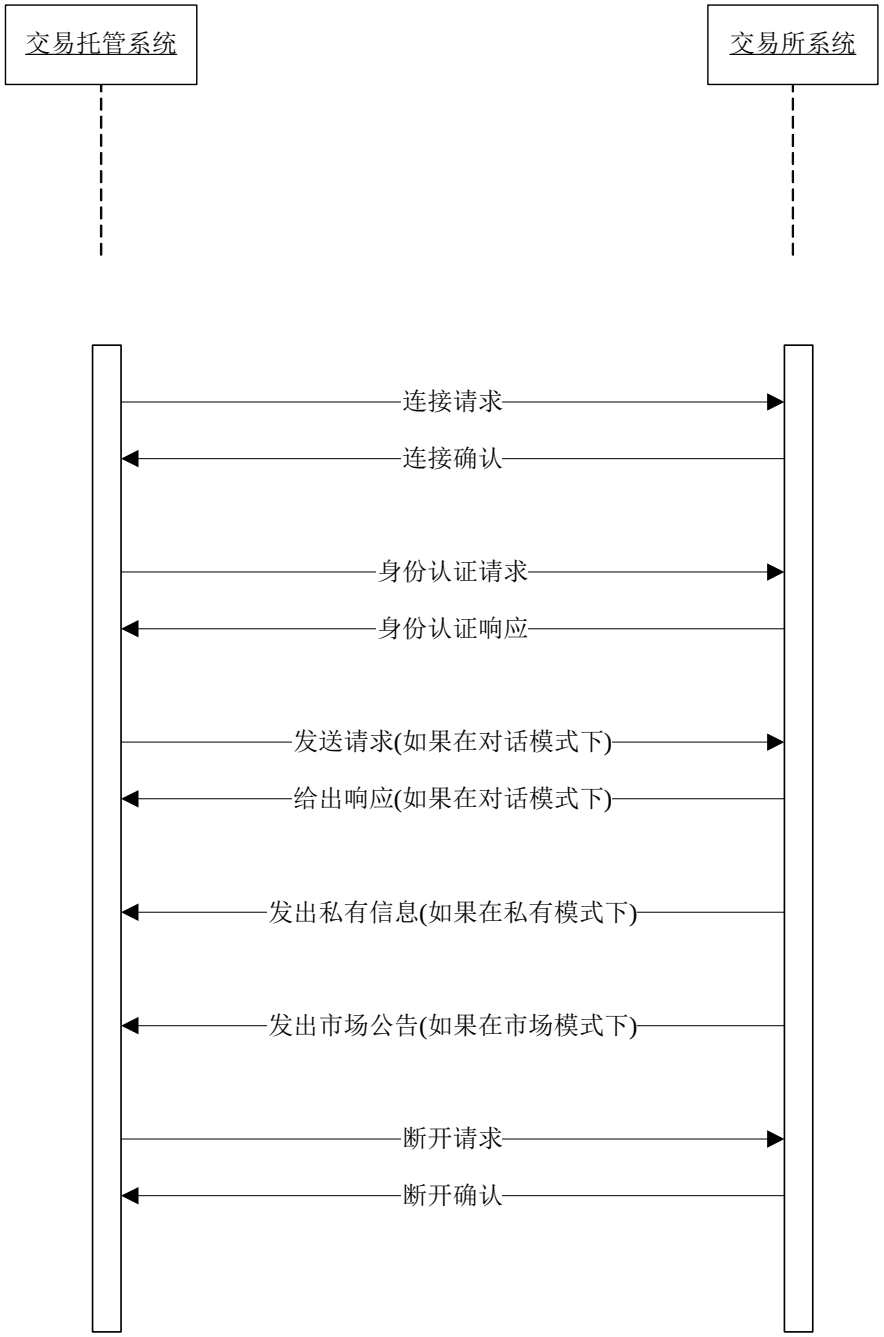


图1) 各通讯模式的工作过程

本接口暂时没有使用广播通信方式。

2.2. 数据流

交易托管系统支持对话通讯模式、私有通讯模式、广播通讯模式：

对话通讯模式下支持对话数据流和查询数据流：

对话数据流是一个双向数据流，交易托管系统发送交易请求，交易系统反馈

应答。交易系统不维护对话流的状态。系统故障时，对话数据流会重置，通讯途中的数据可能会丢失。

查询数据流是一个双向数据流，交易托管系统发送查询请求，交易系统反馈应答。交易系统不维护查询流的状态。系统故障时，查询数据流会重置，通讯途中的数据可能会丢失。

私有通讯模式下支持私有数据流：

私有流是一个单向数据流，由交易系统发向交易托管系统，用于传送交易员私有的通知和回报信息。私有流是一个可靠的数据流，交易系统维护每个交易托管系统的私有流，在一个交易日内，交易托管系统断线后恢复连接时，可以请求交易系统发送指定序号之后的私有流数据。私有数据流向交易托管系统提供报单状态报告、成交回报更等信息。

广播通讯模式下支持公共数据流：

公共数据流是一个单向数据流，由交易系统发向交易托管系统，用于发送市场公共信息；公共数据流也是一个可靠的数据流，交易系统维护整个系统的公共数据流，在一个交易日内，交易托管系统断线恢复连接时，可以请求交易系统发送指定序号之后的公共数据流数据。

3. 接口模式

交易 API 提供了二个接口，分别为 CThostFtdcTraderApi 和 CThostFtdcTraderSpi。这两个接口对 FTD 协议进行了封装，方便客户端应用程序的开发。客户端应用程序可以通过 CThostFtdcTraderApi 发出操作请求，通过继承 CThostFtdcTraderSpi 并重载回调函数来处理后台服务的响应。

行情 API 的使用方式和交易 API 相类似，也提供 2 个接口 CThostFtdcMdApi 和 CThostFtdcMdSpi。客户端应用程序通过 CThostFtdcMdApi 发出操作请求，通过继承 CThostFtdcMdSpi 并重载回调函数来处理后台服务的响应。

3.1. 对话流和查询流编程接口

通过对话流进行通讯的编程接口通常如下：

请求：int CThostFtdcTraderApi::ReqXXX(

CThostFtdcXXXField *pReqXXX,

int nRequestID)

响应：void CThostFtdcTraderSpi::OnRspXXX(

CThostFtdcXXXField *pRspXXX,

CThostFtdcRspInfoField *pRspInfo,

int nRequestID,

bool bIsLast)

其中请求接口第一个参数为请求的内容，不能为空。

第二个参数为请求号。请求号由客户端应用程序负责维护，正常情况下每个请求的请求号不要重复。在接收交易托管系统的响应时，可以得到当时发出请求时填写的请求号，从而可以将响应与请求对应起来。

当收到后台服务应答时，CThostFtdcTraderSpi 的回调函数会被调用。如果响应数据不止一个，则回调函数会被多次调用。

回调函数的第一个参数为响应的具体数据，如果出错或没有结果有可能为 NULL。

第二个参数为处理结果，表明本次请求的处理结果是成功还是失败。在发生

多次回调时，除了第一次回调，其它的回调该参数都可能为 NULL。

第三个参数为请求号，即原来发出请求时填写的请求号。

第四个参数为响应结束标志，表明是否是本次响应的最后一次回调。

3.2. 私有流编程接口

私有流中的数据中会员的私有信息，包括报单回报、成交回报等。

通过私有流接收回报的编程接口通常如下：

void CThostFtdcTraderSpi::OnRtnXXX(CThostFtdcXXXField *pXXX) 或

void CThostFtdcTraderSpi::OnErrRtnXXX(CThostFtdcXXXField *pXXX,

CThostFtdcRspInfoField *pRspInfo)

当收到交易托管系统通过私有流发布的回报数据时，CThostFtdcTraderSpi 的回调函数会被调用。回调函数的参数为回报的具体内容。

4. 运行模式

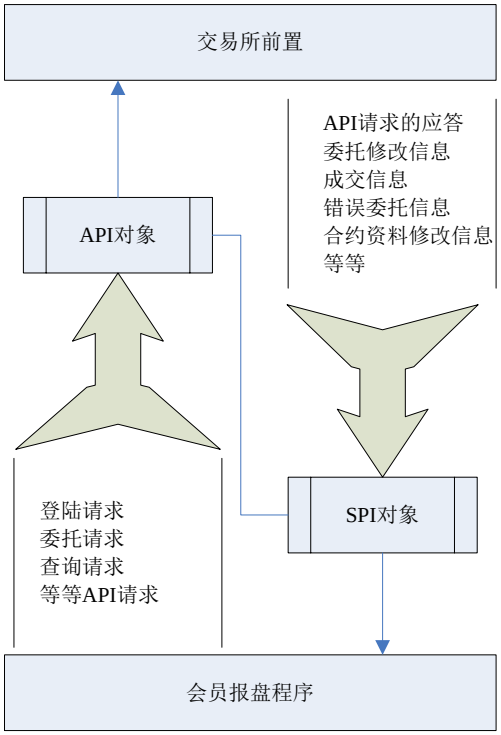
4.1. 工作线程

交易员客户端应用程序至少由两个线程组成，一个是应用程序主线程，一个是交易员 API 工作线程。应用程序与交易系统的通讯是由 API 工作线程驱动的。

CThostFtdcTraderApi 提供的接口是线程安全的，可以有多个应用程序线程同时发出请求。

CThostFtdcTraderSpi 提供的接口回调是由 API 工作线程驱动，通过实现 SPI 中的接口方法，可以从交易托管系统收取所需数据。

如果重载的某个回调函数阻塞，则等于阻塞了 API 工作线程，API 与交易系统的通讯会停止。因此，在 CThostFtdcTraderSpi 派生类的回调函数中，通常应迅速返回，可以利用将数据放入缓冲区或通过 Windows 的消息机制来实现。



4.2. 本地文件

交易员 API 在运行过程中，会将一些数据写入本地文件中。调用 CreateFtdcTraderApi 函数，可以传递一个参数，指明存贮本地文件的路径。该路径必须在运行前已创建好。本地文件的扩展名都是”.con”。

5. 业务与接口对照

业务类型	业务	请求接口	响应接口	数据流
登录	登录	CThostFtdcTraderApi:: ReqUserLogin	CThostFtdcTraderSpi::OnRspUserLogin	对话流
	登出	CThostFtdcTraderApi::ReqUserLogout	CThostFtdcTraderSpi::OnRspUserLogout	对话流
	修改用户口令	CThostFtdcTraderApi::ReqUserPasswordUpdate	CThostFtdcTraderSpi::OnRspUserPasswordUpdate	对话流
交易	报单录入	CThostFtdcTraderApi::ReqOrderInsert	CThostFtdcTraderSpi::OnRspOrderInsert	对话流
	报单操作	CThostFtdcTraderApi::ReqOrderAction	CThostFtdcTraderSpi::OnRspOrderAction	对话流
	报价录入	CThostFtdcTraderApi::ReqQuoteInsert	CThostFtdcTraderSpi::OnRspQuoteInsert	对话流
	报价操作	CThostFtdcTraderApi::ReqQuoteAction	CThostFtdcTraderSpi::OnRspQuoteAction	对话流
私有回报	成交回报	N/A	CThostFtdcTraderSpi::OnRtnTrade	私有流
	报单回报	N/A	CThostFtdcTraderSpi::OnRtnOrder	私有流
	报单录入错误回报	N/A	CThostFtdcTraderSpi::OnErrRtnOrderInsert	私有流
	报单操作错误回报	N/A	CThostFtdcTraderSpi::OnErrRtnOrderAction	私有流
查询	报单查询	CThostFtdcTraderApi::ReqQryOrder	CThostFtdcTraderSpi::OnRspQryOrder	查询流
	成交查询	CThostFtdcTraderApi::ReqQryTrade	CThostFtdcTraderSpi::OnRspQryTrade	查询流
	投资者查询	CThostFtdcTraderApi::ReqQry Investor	CThostFtdcTraderSpi::OnRspQry Investor	查询流
	投资者持仓查询	CThostFtdcTraderApi::ReqQry Investor Position	CThostFtdcTraderSpi::OnRspQry Investor Position	查询流
	合约查询	CThostFtdcTraderApi::ReqQryInstrument	CThostFtdcTraderSpi::OnRspQryInstrument	查询流

交易接口和私有流接口会有相互关联，如用户报单录入 ReqOrderInsert，马上会收到报单响应 OnRspOrderInsert，说明交易系统已经收到报单。报单进入交易系统后，如果报单的交易状态发生变化，就会收到报单回报 OnRtnOrder。如果报单被撮合(部分)成交，就会收到成交回报 OnRtnTrade。其中，一个用户的报单回报和成交回报也会被所属会员下其他的用户接受到。

业务类型	业务	请求接口	响应接口	数据流
登录	登录	CThostFtdcMdApi:: ReqUserLogin	CThostFtdcMdSpi::OnRspUserLogin	对话流
	登出	CThostFtdcMdApi::ReqUserLogout	CThostFtdcMdSpi::OnRspUserLogout	对话流
交易	订阅行情	CThostFtdcMdApi:: SubscribeMarketData	CThostFtdcMdApi::OnRspSubMarketData	对话流
	退订行情	CThostFtdcMdApi:: UnSubscribeMarketData	CThostFtdcMdApi::OnRspUnSubMarketData	对话流
私有回报	深度行情通知	N/A	CThostFtdcMdApi::OnRtnDepthMarketData	私有流

6. 开发接口

6.1. 通用规则

客户端和交易托管系统的通讯过程分为 2 个阶段：初始化阶段和功能调用阶段。

在初始化阶段，程序必须完成如下步骤（具体代码请参考开发实例）：

1, 产生一个 CThostFtdcTraderApi 实例
2, 产生一个事件处理的实例
3, 注册一个事件处理的实例
4, 订阅私有流
5, 订阅公共流
6, 设置交易托管服务的地址

在功能调用阶段，程序可以任意调用交易接口中的请求方法，如 ReqOrderInsert 等。同时按照需要响应回调接口中的。

其他注意事项：

1，API 请求的输入参数不能为 NULL。
2，API 请求的返回参数，0 表示正确，其他表示错误，详细错误编码请查表。

6.2. 托管服务地址设置要求

1. 交易系统现在提供多个接入地址。详细地址参见《托管交易主机地址列表.txt》
2. 客户端需要注册地址列表中的所有地址，ＡＰＩ会根据具体情况自动选择一个合适的主机进行连接。

6.3. 经纪公司代码设置要求

１．托管系统提供《经纪公司代码列表．xls》文件，客户端需要使用最新的地址列表文件，显示经纪公司简称，由客户选择。
２．客户端需要保存客户选择的经纪公司代码。

6.4. CThostFtdcTraderSpi 接口

CThostFtdcTraderSpi 实现了事件通知接口。用户必需派生 CThostFtdcTraderSpi 接口，编写事件处理方法来处理感兴趣的事件。

6.4.1. OnFrontConnected 方法

当客户端与交易托管系统建立起通信连接时（还未登录前），该方法被调用。

函数原形：

void OnFrontConnected()；
本方法在完成初始化后调用，可以在其中完成用户登录任务。

6.4.2. OnFrontDisconnected 方法

当客户端与交易托管系统通信连接断开时，该方法被调用。当发生这个情况后，API 会自动重新连接，客户端可不做处理。自动重连地址，可能是原来注册的地址，也可能是系统支持的其它可用的通信地址，它由程序自动选择。

函数原形：

void OnFrontDisconnected (int nReason)；

参数：

nReason：连接断开原因

0x1001 网络读失败

0x1002 网络写失败

0x2001 接收心跳超时

0x2002 发送心跳失败

0x2003 收到错误报文

6.4.3. OnHeartBeatWarning 方法

心跳超时警告。当长时间未收到报文时，该方法被调用。

函数原形：

void OnHeartBeatWarning(int nTimeLapse)；

参数：

nTimeLapse：距离上次接收报文的时间

6.4.4. OnRspUserLogin 方法

当客户端发出登录请求之后，交易托管系统返回响应时，该方法会被调用，通知客户端登录是否成功。

函数原形：

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

参数：

pRspUserLogin：返回用户登录信息的地址。

用户登录信息结构：

```
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
    TThostFtdcSystemNameTypeSystemName;
};
```

pRspInfo：返回用户响应信息的地址。特别注意在有连续的成功的响应数据时，中间有可能返回 NULL，但第一次不会，以下同。错误代码为 0 时，表示操作成功，以下同。

响应信息结构：

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



```
};
```

nRequestID：返回用户登录请求的 ID，该 ID 由用户在登录时指定。

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

6.4.5. OnRspUserLogout 方法

当客户端发出退出请求之后，交易托管系统返回响应时，该方法会被调用，通知客户端退出是否成功。

函数原形：

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

参数：

pRspUserLogout：返回用户退出信息的地址。

用户登出信息结构：

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

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：返回用户登出请求的 ID，该 ID 由用户在登出时指定。

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

6.4.6. OnRspUserPasswordUpdate 方法

用户密码修改应答。当客户端发出用户密码修改指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

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

参数：

pUserPasswordUpdate：指向用户密码修改结构的地址，包含了用户密码修改请求的输入数据。

用户密码修改结构：

```
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;
};
```

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：返回用户密码修改请求的 ID，该 ID 由用户在密码修改时指定。

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

6.4.7. OnRspTradingAccountPasswordUpdate 方法

资金账户口令更新应答。当客户端发出资金账户口令更新指令后，交易托管系统返回响应时，该方法会被调用。

函数原形：

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

参数：

pTradingAccountPasswordUpdate：指向资金账户口令变更域结构的地址，包含了用户密码修改请求的输入数据。

资金账户口令变更域结构：

```
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;
};
```

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：返回用户密码修改请求的 ID，该 ID 由用户在密码修改时指定。

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

6.4.8. 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。

6.4.9. 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。

6.4.10. 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
TThostFtdcBusinessUnitType	BusinessUnit;
///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。

6.4.11. 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。

6.4.12. 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。

6.4.13. 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。

6.4.14. 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。

6.4.15. 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。

6.4.16. 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。

6.4.17. 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
    TThostFtdcFrontIDTypeFrontID;
    ///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。

6.4.18. 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。

6.4.19. 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。

6.4.20. 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。

6.4.21. 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。

6.4.22. 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。

6.4.23. 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。

6.4.24. 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
    TThostFtdcProductClassType ProductClass;
    ///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
    TThostFtdcPositionTypeTypePositionType;
    ///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。

6.4.25. 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
TThostFtdcMillisecTypeUpdateMillisec;
///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。

6.4.26. 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。

6.4.27. 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。

6.4.28. 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。

6.4.29. 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：

```
struct CThostFtdcNoticeField
{
    ///经纪公司代码
    TThostFtdcBrokerIDType    BrokerID;
    ///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。

6.4.30. 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
    TThostFtdcProductClassType ProductClass;
    ///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。

6.4.31. 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;
};
```

6.4.32. 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;
};
```

6.4.33. 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;
};
```


6.4.34. 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;
};
```

6.4.35. 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;
};
```

6.4.36. 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;
};
```

6.5. 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。

6.5.1. 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。

6.5.2. Release 方法

释放一个 CThostFtdcTradeApi 实例。不能使用 delete 方法

函数原形：

void Release();

6.5.3. 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();

6.5.4. 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();

6.5.5. 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。

6.5.6. 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。

6.5.7. 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。

6.5.8. 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

6.5.9. 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

6.5.10. 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。

6.5.11. 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。

6.5.12. 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。

6.5.13. 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。

6.5.14. 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。

6.5.15. 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。

6.5.16. 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。

6.5.17. 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。

6.5.18. 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。

6.5.19. 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。

6.5.20. 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。

6.5.21. 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=214008)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。

6.5.22. 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。

6.5.23. 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。

6.5.24. 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。

6.5.25. 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。

6.5.26. 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。

6.5.27. 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。

6.5.28. 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。

6.5.29. 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。

6.5.30. 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。

6.5.31. 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。

6.5.32. 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。

6.5.33. 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。

6.5.34. 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。

6.5.35. 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。

6.5.36. 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。

6.6. CThostFtdcMdSpi 接口

CThostFtdcMdSpi 和 CThostFtdcTraderApi 类似，实现了事件通知接口。用户

必需派生 CThostFtdcMdSpi 接口，编写事件处理方法来处理感兴趣的事件。

CThostFtdcMdSpi 中的不少事件通知和 CThostFtdcTraderApi 一样的，包括：OnFrontConnected，OnFrontDisconnected，OnHeartBeatWarning，OnRspUserLogin，OnRspUserLogout，OnRspError。

6.6.1. OnRspSubMarketData 方法

请求订阅行情的响应。如果订阅失败，客户端就会收到包含失败信息的响应。如果成功，就不会收到这个响应。

函数原形：

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

参数：

pSpecificInstrument：

指定的合约

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

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：**nRequestID**：返回请求的 ID，该 ID 由用户在发送指令时指定。

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

6.6.2. OnRspUnSubMarketData 方法

请求退订行情的响应。如果退订失败，客户端就会收到包含失败信息的响应。如果成功，就不会收到这个响应。

函数原形：

```
void OnRspUnSubMarketData(
    CThostFtdcSpecificInstrumentField *pSpecificInstrument,
    CThostFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast)
```

参数：

pSpecificInstrument：

指定的合约

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

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：返回请求的 ID，该 ID 由用户在发送指令时指定。

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

6.6.3. OnRtnDepthMarketData 方法

深度行情通知。 当用户订阅的行情发生变化时，交易托管系统会通知客户端，该方法会被调用。

函数原形：

void OnRtnDepthMarketData(CThostFtdcDepthMarketDataField *pDepthMarketData)

参数：

pDepthMarketData：指向深度行情结构的地址。

深度行情

```
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
TThostFtdcMillisecTypeUpdateMillisec;
///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;
    ///当日均价
    TThostFtdcPriceType     AveragePrice;
};
```

6.7. CThostFtdcMdApi 接口

CThostFtdcMdApi 接口提供给用户的功能包括，订阅行情，退订行情等功能。

其他方法和 CThostFtdcTraderApi 接口的方法一致。

6.7.1. SubscribeMarketData 方法

订阅行情请求。

函数原形：

int SubscribeMarketData(char *ppInstrumentID[], int nCount)

参数：

ppInstrumentID：指向需要订阅的行情列表。

nCount：需要订阅的行情列表的长度。

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。

6.7.2. UnSubscribeMarketData 方法

退订行情请求。

函数原形：

int UnSubscribeMarketData(char *ppInstrumentID[], int nCount)

参数：

ppInstrumentID：指向需要订阅的行情列表。

nCount：需要订阅的行情列表的长度。

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。

7. 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;
}
```