拓界科技

MdAPI说明书

2025年10月22日

文档标识

|  |  |
| --- | --- |
| 项目名称 |  |
| 文档名称 | Md API |
| 版本号 | 0.9 |
| 状况 |  |

目录

目录

[第1章 介绍 1](#_Toc446424684)

[第2章 体系结构 2](#_Toc446424685)

[2.1 通讯模式 2](#_Toc446424686)

[2.2 数据流 3](#_Toc446424687)

[第3章 运行模式 5](#_Toc446424688)

[3.1 工作线程 5](#_Toc446424689)

[3.2 本地文件 5](#_Toc446424690)

[第4章 开发接口 6](#_Toc446424691)

[4.1 CQdpFtdcMduserSpi 接口 6](#_Toc446424692)

[4.1.1 OnFrontConnected 方法 6](#_Toc446424693)

[4.1.2 OnFrontDisconnected 方法 6](#_Toc446424694)

[4.1.3 OnHeartBeatWarning方法 7](#_Toc446424695)

[4.1.4 OnPackageStart方法 7](#_Toc446424696)

[4.1.5 OnPackageEnd方法 7](#_Toc446424697)

[4.1.6 OnMultiHeartbeat方法 7](#_Toc446424698)

[4.1.7 OnRspError方法 8](#_Toc446424699)

[4.1.8 OnRspUserLogin方法 8](#_Toc446424700)

[4.1.9 OnRspUserLogout 方法 9](#_Toc446424701)

[4.1.10 OnRspSubscribeTopic方法 10](#_Toc446424702)

[4.1.11 OnRspQryTopic方法 11](#_Toc446424703)

[4.1.12 OnRtnDepthMarketData方法 12](#_Toc446424704)

[4.1.13 OnRtnMultiDepthMarketData方法 15](#_Toc446424705)

[4.1.14 OnRspSubMarketData方法 17](#_Toc446424706)

[4.1.15 OnRspUnSubMarketData方法 17](#_Toc446424707)

[4.2 CQdpFtdcMduserApi接口 18](#_Toc446424708)

[4.2.1 CreateFtdcMduserApi方法 18](#_Toc446424709)

[4.2.2 GetVersion方法 19](#_Toc446424710)

[4.2.3 Release 方法 19](#_Toc446424711)

[4.2.4 SetMultiCast方法 19](#_Toc446424712)

[4.2.5 RegTopicMultiAddr方法 19](#_Toc446424713)

[4.2.6 Init 方法 20](#_Toc446424714)

[4.2.7 Join 方法 20](#_Toc446424715)

[4.2.8 GetTradingDay方法 20](#_Toc446424716)

[4.2.9 RegisterFront 方法 20](#_Toc446424717)

[4.2.10 RegisterNameServer 方法 21](#_Toc446424718)

[4.2.11 RegisterSpi 方法 21](#_Toc446424719)

[4.2.12 RegisterCertificateFile方法 21](#_Toc446424720)

[4.2.13 SubscribeMarketDataTopic方法 22](#_Toc446424721)

[4.2.14 SubMarketData方法 22](#_Toc446424722)

[4.2.15 UnSubMarketData方法 23](#_Toc446424723)

[4.2.16 SetHeartbeatTimeout方法 23](#_Toc446424724)

[4.2.17 ReqUserLogin 方法 23](#_Toc446424725)

[4.2.18 ReqUserLogout 方法 24](#_Toc446424726)

[4.2.19 ReqSubscribeTopic方法 25](#_Toc446424727)

[4.2.20 ReqQryTopic方法 25](#_Toc446424728)

[4.2.21 ReqSubMarketData方法 26](#_Toc446424729)

[4.2.22 ReqUnSubMarketData方法 26](#_Toc446424730)

[第5章 开发示例 27](#_Toc446424731)

# 介绍

行情客户端系统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行情数据的接收。该类库包含以下5个文件：

|  |  |  |  |
| --- | --- | --- | --- |
| 文件名 | 版本 | 文件大小 | 文件描述 |
| QdpFtdcMdApi.h | V1.0 |  | 行情接口头文件 |
| QdpFtdcUserApiDataType.h | V1.0 |  | 定义了 API 所需的一系列数据类型的头文件 |
| QdpFtdcUserApiStruct.h | V1.0 |  | 定义了一系列业务相关的数据结构的头文件 |
| qdpmdapi.dll | V1.0 |  | 动态链接库二进制文件 |
| qdpmdapi.lib | V1.0 |  | 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。

# 

# 体系结构

行情API使用建立在TCP协议之上FTD协议与拓界的行情发布服务器进行通讯。行情发布服务器负责行情信息的产生与发布，但不参与交易过程。参与交易需要使用另外的“交易员API”。

## 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。

行情发布涉及的通讯模式共有二种：

* 对话通讯模式
* 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。该请求被拓界服务端接收和处理，并给予响应。例如登入与登出。这种通讯模式与普通的客户/服务器模式相同。

广播通讯模式是指拓界服务端主动，向市场中的相关会员发出相同的信息。例如行情等。

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：



1. 各通讯模式的工作过程

## 数据流

行情发布支持对话通讯模式、广播通讯模式：

对话通讯模式下支持对话数据流：

对话数据流是一个双向数据流，会员系统发送请求，行情发布系统反馈应答。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。

广播通讯模式下支持行情数据流：

行情数据流是一个单向数据流，由行情发布系统发向会员系统，用于发送行情信息；行情流是一个可靠的数据流，行情系统维护整个系统的行情流，在一个交易日内，会员系统断线恢复连接时，可以请求行情系统发送指定序号之后的行情流数据。

行情服务所提供的行情内容是按照主题组织的。每个主题包括一组合约的行情，还包括了行情发布内容和发布方式，包括行情深度、采样频率、延迟时间等。目前是拓界行情系统参考中金所公布的各行情主题，设定行情主题。每个行情主题对应着一个行情流。

要获得行情通知，客户端必需在连接行情服务器时，订阅一个或多个行情发布主题。

# 

# 运行模式

## 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。

QdpFtdcMduserApi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。

QdpFtdcMduserSpi提供的接口回调是由API工作线程驱动，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QdpFtdcTraderSpi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。

## 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CreateFtdcMduserApi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”。

# 开发接口

行情客户端系统API提供了二个接口，分别为CQdpFtdcMduserApi和CQdpFtdcMduserSpi。

## CQdpFtdcMduserSpi 接口

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

### OnFrontConnected 方法

当客户端与行情发布服务器建立起通信连接时（还未登录前），该方法被调用。

**函数原型：**

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。

### OnFrontDisconnected 方法

当客户端与交易后台通信连接断开时，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。

**函数原型：**

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

### 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。

**函数原型：**

void OnHeartBeatWarning(int nTimeLapse)；

**参数：**

nTimeLapse：距离上次接收报文的时间

### OnPackageStart方法

报文回调开始通知。当API收到一个报文后，首先调用本方法，然后是各数据域的回调，最后是报文回调结束通知。

**函数原型：**

void OnPackageStart(int nTopicID, int nSequenceNo)

**参数：**

param nTopicID 主题代码（如私有流、公共流、行情流等）

param nSequenceNo 报文序号

### OnPackageEnd方法

当API收到一个报文后，首先调用报文回调开始通知，然后是各数据域的回调，最后调用本方法。

**函数原型：**

void OnPackageEnd(int nTopicID, int nSequenceNo)

**参数：**

param nTopicID 主题代码（如私有流、公共流、行情流等）

param nSequenceNo 报文序号

### OnMultiHeartbeat方法

多播心跳接口

**函数原型：**

void OnMultiHeartbeat(char \*CurrTime,char \*MultiCastIP)

**参数：**

param CurrTime

param MultiCastIP

### OnRspError方法

错误应答

**函数原型：**

void OnRspError(CQdpFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

**参数：**

**pRspInfo**返回消息

响应信息结构：

struct CQdpFtdcRspInfoField

{

///错误代码

ErrorID;

///错误信息

ErrorMsg;

};

**nRequestID**对应错误的请求ID号

**bIsLast**是否为最后一条消息

### OnRspUserLogin方法

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。

**函数原型：**

void OnRspUserLogin(

CQdpFtdcRspUserLoginField \*pRspUserLogin,

CQdpFtdcRspInfoField \*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 CQdpFtdcRspUserLoginField

{

///交易日

TQdpFtdcDateType TradingDay;

///经纪公司编号

TQdpFtdcBrokerIDType BrokerID;

///交易用户代码

TQdpFtdcUserIDType UserID;

///登录成功时间

TQdpFtdcTimeType LoginTime;

///用户最大本地报单号

TQdpFtdcUserOrderLocalIDType MaxOrderLocalID;

///交易系统名称

TQdpFtdcTradingSystemNameType TradingSystemName;

///数据中心代码

TQdpFtdcDataCenterIDType DataCenterID;

///会员私有流当前长度

TQdpFtdcSequenceNoType PrivateFlowSize;

///交易员私有流当前长度

TQdpFtdcSequenceNoType UserFlowSize;

};

**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 CQdpFtdcRspInfoField

{

///错误代码

TQdpFtdcErrorIDType ErrorID;

///错误信息

TQdpFtdcErrorMsgType 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。

### OnRspUserLogout 方法

当客户端发出登出请求之后，该方法会被调用，通知客户端登出是否成功。

**函数原型：**

void OnRspUserLogout(

CQdpFtdcRspUserLogoutField \*pRspUserLogout,

CQdpFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast)；

**参数：**

**pRspUserLogout**：返回用户登出信息的地址。

用户登出信息结构：

struct CQdpFtdcRspUserLogoutField

{

///交易用户代码

TQdpFtdcUserIDType UserID;

///会员代码

TQdpFtdcParticipantIDType ParticipantID;

};

**pRspInfo**：返回用户响应信息的地址。

响应信息结构：

struct CQdpFtdcRspInfoField

{

///错误代码

TQdpFtdcErrorIDType ErrorID;

///错误信息

TQdpFtdcErrorMsgType 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。

### OnRspSubscribeTopic方法

订阅主题应答。

**函数原型：**

void OnRspSubscribleTopic(

CQdpFtdcDisseminationField \*pDissemination,

CQdpFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast)；

**参数：**

**pDissemination**：

///信息分发

struct CQdpFtdcDisseminationField

{

///序列系列号

TQdpFtdcSequenceSeriesType SequenceSeries;

///序列号

TQdpFtdcSequenceNoType SequenceNo;

};

**pRspInfo:**

struct CQdpFtdcRspInfoField

{

///错误代码

TQdpFtdcErrorIDType ErrorID;

///错误信息

TQdpFtdcErrorMsgType ErrorMsg;

};

**nRequestID:**

**bIsLast:**

### OnRspQryTopic方法

主题查询应答。

**函数原型：**

void OnRspQryTopic (

CQdpFtdcDisseminationField \*pDissemination,

CQdpFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast)；

**参数：**

**pDissemination**：

///信息分发

struct CQdpFtdcDisseminationField

{

///序列系列号

TQdpFtdcSequenceSeriesType SequenceSeries;

///序列号

TQdpFtdcSequenceNoType SequenceNo;

};

**pRspInfo:**

struct CQdpFtdcRspInfoField

{

///错误代码

TQdpFtdcErrorIDType ErrorID;

///错误信息

TQdpFtdcErrorMsgType ErrorMsg;

};

**nRequestID:**

**bIsLast:**

### OnRtnDepthMarketData方法

行情通知，行情服务器会主动通知客户端。

**函数原型：**

void OnRtnDepthMarketData(CQdpFtdcDepthMarketDataField \*pDepthMarketData)；

**参数：**

**pDepthMarketData**：返回市场行情信息的地址。

深度市场行情信息结构：

struct CQdpFtdcDepthMarketDataField

{

///交易日

TQdpFtdcDateType TradingDay;

///结算组代码

TQdpFtdcSettlementGroupIDType SettlementGroupID;

///结算编号

TQdpFtdcSettlementIDType SettlementID;

///昨结算

TQdpFtdcPriceType PreSettlementPrice;

///昨收盘

TQdpFtdcPriceType PreClosePrice;

///昨持仓量

TQdpFtdcLargeVolumeType PreOpenInterest;

///昨虚实度

TQdpFtdcRatioType PreDelta;

///今开盘

TQdpFtdcPriceType OpenPrice;

///最高价

TQdpFtdcPriceType HighestPrice;

///最低价

TQdpFtdcPriceType LowestPrice;

///今收盘

TQdpFtdcPriceType ClosePrice;

///涨停板价

TQdpFtdcPriceType UpperLimitPrice;

///跌停板价

TQdpFtdcPriceType LowerLimitPrice;

///今结算

TQdpFtdcPriceType SettlementPrice;

///今虚实度

TQdpFtdcRatioType CurrDelta;

///最新价

TQdpFtdcPriceType LastPrice;

///数量

TQdpFtdcVolumeType Volume;

///成交金额

TQdpFtdcMoneyType Turnover;

///持仓量

TQdpFtdcLargeVolumeType OpenInterest;

///申买价一

TQdpFtdcPriceType BidPrice1;

///申买量一

TQdpFtdcVolumeType BidVolume1;

///申卖价一

TQdpFtdcPriceType AskPrice1;

///申卖量一

TQdpFtdcVolumeType AskVolume1;

///申买价二

TQdpFtdcPriceType BidPrice2;

///申买量二

TQdpFtdcVolumeType BidVolume2;

///申买价三

TQdpFtdcPriceType BidPrice3;

///申买量三

TQdpFtdcVolumeType BidVolume3;

///申卖价二

TQdpFtdcPriceType AskPrice2;

///申卖量二

TQdpFtdcVolumeType AskVolume2;

///申卖价三

TQdpFtdcPriceType AskPrice3;

///申卖量三

TQdpFtdcVolumeType AskVolume3;

///申买价四

TQdpFtdcPriceType BidPrice4;

///申买量四

TQdpFtdcVolumeType BidVolume4;

///申买价五

TQdpFtdcPriceType BidPrice5;

///申买量五

TQdpFtdcVolumeType BidVolume5;

///申卖价四

TQdpFtdcPriceType AskPrice4;

///申卖量四

TQdpFtdcVolumeType AskVolume4;

///申卖价五

TQdpFtdcPriceType AskPrice5;

///申卖量五

TQdpFtdcVolumeType AskVolume5;

///合约代码

TQdpFtdcInstrumentIDType InstrumentID;

///最后修改时间

TQdpFtdcTimeType UpdateTime;

///最后修改毫秒

TQdpFtdcMillisecType UpdateMillisec;

///交易所代码

TQdpFtdcExchangeIDType ExchangeID;

};

### OnRtnMultiDepthMarketData方法

行情通知，行情服务器会主动通知客户端。

**函数原型：**

void OnRtnMultiDepthMarketData(CQdpFtdcDepthMarketDataField \*pDepthMarketData)；

**参数：**

**pDepthMarketData**：返回市场行情信息的地址。

深度市场行情信息结构：

struct CQdpFtdcDepthMarketDataField

{

///交易日

TQdpFtdcDateType TradingDay;

///结算组代码

TQdpFtdcSettlementGroupIDType SettlementGroupID;

///结算编号

TQdpFtdcSettlementIDType SettlementID;

///昨结算

TQdpFtdcPriceType PreSettlementPrice;

///昨收盘

TQdpFtdcPriceType PreClosePrice;

///昨持仓量

TQdpFtdcLargeVolumeType PreOpenInterest;

///昨虚实度

TQdpFtdcRatioType PreDelta;

///今开盘

TQdpFtdcPriceType OpenPrice;

///最高价

TQdpFtdcPriceType HighestPrice;

///最低价

TQdpFtdcPriceType LowestPrice;

///今收盘

TQdpFtdcPriceType ClosePrice;

///涨停板价

TQdpFtdcPriceType UpperLimitPrice;

///跌停板价

TQdpFtdcPriceType LowerLimitPrice;

///今结算

TQdpFtdcPriceType SettlementPrice;

///今虚实度

TQdpFtdcRatioType CurrDelta;

///最新价

TQdpFtdcPriceType LastPrice;

///数量

TQdpFtdcVolumeType Volume;

///成交金额

TQdpFtdcMoneyType Turnover;

///持仓量

TQdpFtdcLargeVolumeType OpenInterest;

///申买价一

TQdpFtdcPriceType BidPrice1;

///申买量一

TQdpFtdcVolumeType BidVolume1;

///申卖价一

TQdpFtdcPriceType AskPrice1;

///申卖量一

TQdpFtdcVolumeType AskVolume1;

///申买价二

TQdpFtdcPriceType BidPrice2;

///申买量二

TQdpFtdcVolumeType BidVolume2;

///申买价三

TQdpFtdcPriceType BidPrice3;

///申买量三

TQdpFtdcVolumeType BidVolume3;

///申卖价二

TQdpFtdcPriceType AskPrice2;

///申卖量二

TQdpFtdcVolumeType AskVolume2;

///申卖价三

TQdpFtdcPriceType AskPrice3;

///申卖量三

TQdpFtdcVolumeType AskVolume3;

///申买价四

TQdpFtdcPriceType BidPrice4;

///申买量四

TQdpFtdcVolumeType BidVolume4;

///申买价五

TQdpFtdcPriceType BidPrice5;

///申买量五

TQdpFtdcVolumeType BidVolume5;

///申卖价四

TQdpFtdcPriceType AskPrice4;

///申卖量四

TQdpFtdcVolumeType AskVolume4;

///申卖价五

TQdpFtdcPriceType AskPrice5;

///申卖量五

TQdpFtdcVolumeType AskVolume5;

///合约代码

TQdpFtdcInstrumentIDType InstrumentID;

///最后修改时间

TQdpFtdcTimeType UpdateTime;

///最后修改毫秒

TQdpFtdcMillisecType UpdateMillisec;

///交易所代码

TQdpFtdcExchangeIDType ExchangeID;

};

### OnRspSubMarketData方法

针对用户请求的订阅的合约返回已经订阅的合约结果。

**函数原型：**

void OnRspSubMarketData(CQdpFtdcSpecificInstrumentField \*pSpecificInstrument, CQdpFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

**参数：**

**pSpecificInstrument**：返回用户成功订阅的合约。

struct CQdpFtdcSpecificInstrumentField

{

///合约代码

TQdpFtdcInstrumentIDType InstrumentID;

};

**nRequestID**：返回用户登出请求的ID，该ID由用户在登出时指定。这里无效。

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

### OnRspUnSubMarketData方法

针对用户请求的订阅的合约返回已经订阅的合约结果。

**函数原型：**

void OnRspUnSubMarketData(

CQdpFtdcSpecificInstrumentField \*pSpecificInstrument,

CQdpFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast)

**参数：**

**pSpecificInstrument**：返回用户成功订阅的合约。

struct CQdpFtdcSpecificInstrumentField

{

///合约代码

TQdpFtdcInstrumentIDType InstrumentID;

};

**nRequestID**：返回用户登出请求的ID，该ID由用户在登出时指定。这里无效。

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

### OnMutiData方法

接收交易所原始组播行情（客户可以继承此接口接收上期所或者大商所的原始报文）。

**函数原型：**

void OnMutiData (char \*data, int len);

**参数：**

data 原始报文数据

len 原始报文长度

### OnRtnShfeMultiMarketData方法

API解析后的组播数据（包含上期所和大商所组播行情）。

**函数原型：**

void OnRtnShfeMultiMarketData(CQdFtdcDepthMarketDataField \*pMarketData);

**参数：**

**pMarketData**深度行情。

深度行情结构：

struct CqdFtdcDepthMarketDataField

{

///交易日

TradingDay;

///昨结算

PreSettlementPrice;

///昨收盘

PreClosePrice;

///昨持仓量

PreOpenInterest;

///昨虚实度

PreDelta;

///今开盘价

OpenPrice;

///最高价

HighestPrice;

///最低价

LowestPrice;

///今收盘价

ClosePrice;

///涨停板价

UpperLimitPrice;

///跌停板价

LowerLimitPrice;

///今结算价

SettlementPrice;

///今虚实度

CurrDelta;

///最新价

LastPrice;

///成交量

Volume;

///成交金额

Turnover;

///持仓量

OpenInterest;

///申买价一

BidPrice1;

///申买量一

BidVolume1;

///申卖价一

AskPrice1;

///申卖量一

AskVolume1;

///申买价二

BidPrice2;

///申买量二

BidVolume2;

///申买价三

BidPrice3;

///申买量三

BidVolume3;

///申卖价二

AskPrice2;

///申卖量二

AskVolume2;

///申卖价三

AskPrice3;

///申卖量三

AskVolume3;

///申买价四

BidPrice4;

///申买量四

BidVolume4;

///申买价五

BidPrice5;

///申买量五

BidVolume5;

///申卖价四

AskPrice4;

///申卖量四

AskVolume4;

///申卖价五

AskPrice5;

///申卖量五

AskVolume5;

///合约代码

InstrumentID;

///最后修改时间

UpdateTime;

///最后修改毫秒

UpdateMillisec;

///交易所代码

ExchangeID;

///总卖出数量

VolumeAskLot;

///总买入数量

VolumeBidLot;

///合约状态

InstrumentStatus;

///当前日期

CalendarDate;

///组播增量序号

PacketNo;

///合约序号

InstrumentNo;

///数据中心代码

DataCenterID;

};

### OnRtnMBLMarketData方法

分价行情通知。

**函数原型：**

void OnRtnMBLMarketData(CQdFtdcMBLMarketDataField \*pMBLMarketData);

**参数：**

**pMBLMarketData**分价行情。

分价行情结构：

struct CQdFtdcMBLMarketDataField

{

///合约代码

InstrumentID;

///买卖方向

Direction；

///价格

Price;

///数量

Volume；

///最后修改时间

UpdateTime;

///最后修改毫秒

UpdateMillisec；

};

### OnRtnQmdInstrumentStatu方法

郑州合约状态通知。

**函数原型：**

void OnRtnQmdInstrumentStatu(CQdFtdcQmdInstrumentStateField \*pQmdInstrumentState);

**参数：**

**pQmdInstrumentState**合约状态。

合约状态结构：

struct CQdFtdcQmdInstrumentStateField

{

///交易所代码

ExchangeID;

///合约代码

InstrumentID；

///合约交易状态

InstrumentStatus;

};

### OnRspQryMarketData方法

合约行情查询应答。

**函数原型：**

void OnRspQryMarketData(

CQdFtdcRspMarketDataField \*pRspMarketData,

CQdFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast

);

**参数：**

**pRspMarketData**行情查询应答的地址。

深度市场行情信息结构：

struct CQdFtdcDepthMarketDataField

{

///交易所代码

ExchangeID;

///交易日

TradingDay；

///昨结算价

PreSettlementPrice;

///昨收盘价

PreClosePrice;

///昨持仓量

PreOpenInterest；

///昨虚实度

PreDelta;

///今开盘价

OpenPrice;

///最高价

HighestPrice；

///最低价

LowestPrice;

///今收盘价

ClosePrice;

///涨停板价

UpperLimitPrice；

///跌停板价

LowerLimitPrice;

///今结算价

SettlementPrice;

///今虚实度

CurrDelta；

///最新价

LastPrice;

///数量

Volume;

///成交金额

Turnover；

///持仓量

OpenInterest;

///申买价一

BidPrice1;

///申买量一

BidVolume1；

///申卖价一

AskPrice1;

///申卖量一

AskVolume1;

///合约代码

InstrumentID；

///最后修改时间

UpdateTime;

///最后修改毫秒

UpdateMillisec;

///合约交易状态

InstrumentStatus；

};

**pRspInfo** 返回用户响应信息的地址

响应信息CQdFtdcRspInfoField结构说明，详见OnRspError方法

struct CQdpFtdcRspInfoField

{

///错误代码

ErrorID;

///错误信息

ErrorMsg;

};

**nRequestID** 返回行情查询请求的ID，该ID由用户在查询行情时指定

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

### OnRspQryInstrumentList方法

市场合约列表查询应答。

**函数原型：**

void OnRspQryInstrumentList(

CQdFtdcSpecificInstrumentField \*pSpecificInstrument,

CQdFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast

);

**参数：**

**pSpecificInstrument**返回用户成功订阅的合约。

订阅合约结构：

struct CQdFtdcSpecificInstrumentField

{

///合约代码

InstrumentID;

};

**pRspInfo** 返回用户响应信息的地址

响应信息CQdFtdcRspInfoField结构说明，详见OnRspError方法

struct CQdpFtdcRspInfoField

{

///错误代码

ErrorID;

///错误信息

ErrorMsg;

};

**nRequestID** 返回行情查询请求的ID，该ID由用户在查询行情时指定

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

### OnRtnShfeMultiLevel方法

组播深度行情信息通知（包含上期所和大商所）。

**函数原型：**

void OnRtnShfeMultiLevel(CQdFtdcShfeMultiLevelField \*pShfeMultiLevel);

**参数：**

**pShfeMultiLevel**返回组播深度信息。

订阅合约结构：

struct CQdFtdcShfeMultiLevelField

{

///订阅号

TopicID;

///深度

Level;

};

### OnRtnShfeMultiParameters方法

组播参数通知（包含上期所和大商所）。

**函数原型：**

void OnRtnShfeMultiParameters(CQdFtdcShfeMultiParametersField \*pShfeMultiParameters);

**参数：**

**pShfeMultiParameters**返回组播解析参数。

订阅合约结构：

struct CQdFtdcShfeMultiParametersField

{

///合约代码

InstrumentID;

///交易日

TradingDay;

///昨结算

PreSettlementPrice;

///昨收盘

PreClosePrice;

///昨持仓量

PreOpenInterest;

///昨虚实度

PreDelta;

///今开盘

OpenPrice;

///最高价

HighestPrice;

///最低价

LowestPrice;

///今收盘

ClosePrice;

///涨停板价

UpperLimitPrice;

///跌停板价

LowerLimitPrice;

///今结算

SettlementPrice;

///今虚实度

CurrDelta;

///品种

ProductClass;

///执行价格

StrikePrice;

///合约乘数

VolumeMultiple;

///最小变动价格

PriceTick;

///编解码价格

CodecPrice;

///合约序号

InstrumentNo;

///交易所代码

ExchangeID;

};

### OnRspQryDepthMarketData方法

全量快照行情查询应答。

**函数原型：**

void OnRspQryDepthMarketData (

CQdFtdcDepthMarketDataField \*pDepthMarketData,

CQdFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast

);

**参数：**

**pDepthMarketData**返回深度行情。

深度行情CQdFtdcDepthMarketDataField结构说明，详见

OnRtnShfeMultiMarketData方法

**pRspInfo** 返回用户响应信息的地址

响应信息CQdFtdcRspInfoField结构说明，详见OnRspError方法

**nRequestID** 返回用户查询行情快照请求的ID，该ID由用户在查询行情快照时指定。

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

## CQdpFtdcMduserApi接口

CQdpFtdcMduserApi接口提供给客户登陆、登出行情查询服务器，进行行情查询等功能。

### CreateFtdcMduserApi方法

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

**函数原型：**

static CQdpFtdcMduserApi \*CreateFtdcMduserApi(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行情服务发布消息的状态。 默认值代表当前目录。

**返回值：**

返回一个指向CQdpFtdcMduserApi实例的指针。

### GetVersion方法

获取系统版本号。

**函数原型：**

static const char \*GetVersion(int &nMajorVersion, int &nMinorVersion);

**参数：**

param nMajorVersion 主版本号

param nMinorVersion 子版本号

**返回值：**

系统标识字符串返回值。

### Release 方法

释放一个CQdpFtdcMduserApi实例。

**函数原型：**

void Release()；

### SetMultiCast方法

多播设置开关。

**函数原型：**

void SetMultiCast(bool bMultiCast=false)

**参数：**

bMultiCast：开关控制。

### RegTopicMultiAddr方法

注册多播地址。

**函数原型：**

void RegTopicMultiAddr(char \*pMultiAddr)；

**参数：**

pMultiAddr：

### Init 方法

使客户端开始与行情发布服务器建立连接，连接成功后可以进行登陆。

**函数原型：**

void Init()；

### Join 方法

客户端等待一个接口实例线程的结束。

**函数原型：**

void Join()；

### GetTradingDay方法

获得当前交易日。只有当与服务器连接建立后才会取到正确的值。

**函数原型：**

const char \*GetTradingDay()；

**返回值：**

返回一个指向日期信息字符串的常量指针。

### RegisterFront 方法

注册前置机网络地址

**函数原型：**

void RegisterFront(char \*pszFrontAddress) = 0;

**参数：**

param pszFrontAddress：前置机网络地址。

网络地址的格式为：“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。

### RegisterNameServer 方法

注册名字服务器网络地址

**函数原型：**

void RegisterNameServer(char \* pszNsAddress) = 0;

**参数：**

param pszNsAddress：名字服务器网络地址。

网络地址的格式为：“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。

### RegisterSpi 方法

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

**函数原型：**

void RegisterSpi(CQdpFtdcMduserSpi \*pSpi) ;

**参数：**

pSpi：实现了CQdpFtdcMduserSpi接口的实例指针。

### RegisterCertificateFile方法

加载证书

**函数原型：**

int RegisterCertificateFile(

const char \*pszCertFileName,

const char \*pszKeyFileName,

const char \*pszCaFileName,

const char \*pszKeyFilePassword) = 0;

**参数：**

pSpi：实现了CQdpFtdcMduserSpi接口的实例指针。

param pszCertFileName 用户证书文件名

param pszKeyFileName 用户私钥文件名

param pszCaFileName 可信任CA证书文件名

param pszKeyFilePassword 用户私钥文件密码

**返回值：**

return 0 操作成功

return -1 可信任CA证书载入失败

return -2 用户证书载入失败

return -3 用户私钥载入失败

return -4 用户证书校验失败

### SubscribeMarketDataTopic方法

客户端订阅自己需要的行情。订阅后行情服务器会自动发出行情通知给客户端。

**函数原型：**

void SubscribeMarketDataTopic (int nTopicID, TE\_RESUME\_TYPE nResumeType)；

**参数：**

**nTopicID**：代表深度行情的主题，由中金所公布，拓界采用其主题。

**nResumeType**：市场行情重传方式

TERT\_RESTART:从本交易日开始重传

TERT\_RESUME:从上次收到的续传(非订阅全部合约时，不支持续传模式)

TERT\_QUICK:先传送当前行情快照,再传送登录后市场行情的内容

### SubMarketData方法

订阅合约行情。

**函数原型：**

int SubMarketData(char \*ppInstrumentID[], int nCount)=0;

**参数：**

param ppInstrumentID 合约ID

param nCount 要订阅/退订行情的合约个数

**返回值：**

0，代表成功。其它值代表失败。

### UnSubMarketData方法

退订合约行情。

**函数原型：**

int UnSubMarketData(char \*ppInstrumentID[], int nCount)=0;

**参数：**

param ppInstrumentID 合约ID

param nCount 要订阅/退订行情的合约个数

**返回值：**

0，代表成功。其它值代表失败。

### SetHeartbeatTimeout方法

设置心跳超时时间。

**函数原型：**

void SetHeartbeatTimeout(unsigned int timeout) = 0;

**参数：**

param timeout心跳超时时间(秒)

### ReqUserLogin 方法

用户发出登陆请求。

**函数原型：**

int ReqUserLogin(

CQdpFtdcReqUserLoginField \*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 CQdpFtdcReqUserLoginField

{

///交易日

TQdpFtdcDateType TradingDay;

///交易用户代码

TQdpFtdcUserIDType UserID;

///会员代码

TQdpFtdcParticipantIDType ParticipantID;

///密码

TQdpFtdcPasswordType Password;

};

**nRequestID**：用户登录请求的ID，该ID由用户指定，管理。

**返回值：**

0，代表成功。其它值代表失败。

### ReqUserLogout 方法

用户发出登出请求。

**函数原型：**

int ReqUserLogout(

CQdpFtdcReqUserLogoutField \*pReqUserLogout,

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 CQdpFtdcReqUserLogoutField

{

///交易用户代码

TQdpFtdcUserIDType UserID;

///会员代码

TQdpFtdcParticipantIDType ParticipantID;

};

**nRequestID**：用户登出请求的ID，该ID由用户指定，管理。

**返回值：**

1. 代表成功。其它值代表失败。

### ReqSubscribeTopic方法

订阅主题请求。

**函数原型：**

int ReqSubscribeTopic(

CQdpFtdcDisseminationField \*pDissemination,

int nRequestID) = 0;

**参数：**

Param pDissemination：

信息分发

struct CQdpFtdcDisseminationField

{

///序列系列号

TQdpFtdcSequenceSeriesType SequenceSeries;

///序列号

TQdpFtdcSequenceNoType SequenceNo;

};

Param nRequestID：

**返回值：**

0，代表成功。其它值代表失败。

### ReqQryTopic方法

主题查询请求。

**函数原型：**

int ReqQryTopic(CQdpFtdcDisseminationField \*pDissemination, int nRequestID) = 0;

**参数：**

pDissemination：

nRequestID：

### ReqSubMarketData方法

订阅合约的相关信息。

**函数原型：**

int ReqSubMarketData(

CQdpFtdcSpecificInstrumentField \*pSpecificInstrument,

int nRequestID) = 0;

**参数：**

pSpecificInstrument：

struct CQdpFtdcSpecificInstrumentField

{

///合约代码

TQdpFtdcInstrumentIDType InstrumentID;

};

nRequestID：

### ReqUnSubMarketData方法

订阅合约的相关信息。

**函数原型：**

int ReqUnSubMarketData(

CQdpFtdcSpecificInstrumentField \*pSpecificInstrument,

int nRequestID) = 0;

**参数：**

pSpecificInstrument：

nRequestID：

### ReqQryMarketData方法

合约行情查询请求。

**函数原型：**

int ReqQryMarketData(

CQdFtdcQryMarketDataField \*pQryMarketData,

int nRequestID);

**参数：**

**pQryMarketData**：

struct CQdFtdcQryMarketDataField

{

///交易所代码

ExchangeID;

///合约代码

InstrumentID;

///订阅号

TopicID;

};

nRequestID 用户行情查询请求的ID，该ID由用户指定，管理。

**返回值：**

0代表成功，其它值代表失败。

### ReqQryInstrumentList方法

市场合约列表查询请求。

**函数原型：**

int ReqQryInstrumentList (

CQdFtdcMarketDataExchangeIDField \*pMarketDataExchangeID,

int nRequestID);

**参数：**

**pMarketDataExchangeID**：

struct CQdFtdcMarketDataExchangeIDField

{

///交易所代码

ExchangeID;

};

nRequestID 用户合约列表查询请求的ID，该ID由用户指定，管理。

**返回值：**

0代表成功，其它值代表失败。

### ReqQrySHFEMultiInfo方法

组播信息查询请求。

**函数原型：**

int ReqQrySHFEMultiInfo (

CQdFtdcQryShfeMultiInfoField \*pQryShfeMultiInfo,

int nRequestID);

**参数：**

**pQryShfeMultiInfo**：

struct CQdFtdcQryShfeMultiInfoField

{

///交易所代码

ExchangeID;

///合约代码

InstrumentID;

};

**nRequestID** 用户组播信息查询请求的ID，该ID由用户指定，管理。

**返回值：**

0代表成功，其它值代表失败。

### ReqQryDepthMarketData方法

快照行情查询请求。

**函数原型：**

int ReqQryDepthMarketData (

CQdFtdcQryMarketDataField \*pQryMarketData,

int nRequestID);

**参数：**

**pQryMarketData**：

struct CQdFtdcQryMarketDataField

{

///交易所代码

ExchangeID;

///合约代码

InstrumentID;

///订阅号

TopicID;

};

nRequestID 用户行情查询请求的ID，该ID由用户指定，管理。

**返回值：**

0代表成功，其它值代表失败。

### ActiveMultiMarketData方法

启动多播。

注意：在无法登录qmarket服务器的情况下，调用此接口可以接收组播行情，在可以登录的情况下无需调用此接口

**函数原型：**

void ActiveMultiMarketData(char \*TradingDay)

**参数：**

TradingDay组播地址

交易日，在没有登录的情况下，可以是自然日，但是要注意交易所的交易日

# 开发示例

## *普通Tcp方式接收行情Demo*

// mddemo.cpp :

//一个简单的例子，介绍QdpFtdcMduserApi和QdpFtdcMduserSpi接口的使用。

#include <stdio.h>

#include <string.h>

#include <float.h>

#include "QdpFtdcMdApi.h"

class CSimpleHandler : public CQdpFtdcMduserSpi

{

public:

// 构造函数，需要一个有效的指向CQdpFtdcMduserApi实例的指针

CSimpleHandler(CQdpFtdcMduserApi \*pUserApi) : m\_pUserApi(pUserApi) {}

~CSimpleHandler() {}

// 当客户端与行情发布服务器建立起通信连接，客户端需要进行登录

void OnFrontConnected()

{

CQdpFtdcReqUserLoginField reqUserLogin;

strcpy(reqUserLogin.TradingDay, m\_pUserApi->GetTradingDay());

strcpy(reqUserLogin.BrokerID, "0001");

strcpy(reqUserLogin.UserID, "t002");

strcpy(reqUserLogin.Password, "111111");

m\_pUserApi->ReqUserLogin(&reqUserLogin, 0);

}

// 当客户端与行情发布服务器通信连接断开时，该方法被调用

void OnFrontDisconnected()

{

// 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

void OnRspUserLogin(CQdpFtdcRspUserLoginField \*pRspUserLogin, CQdpFtdcRspInfoField \*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);

return;

}

char \* contracts[3]={"","",""};

contracts[0]="IF1503";

contracts[1]="IF1504";

contracts[2]="IF1506";

m\_pUserApi->SubMarketData(contracts, 3);

char \* uncontracts[2]={"",""};

uncontracts[0]="IF1509";

uncontracts[1]="IH1503";

m\_pUserApi->UnSubMarketData(uncontracts, 2);

}

// 深度行情通知，行情服务器会主动通知客户端

void OnRtnDepthMarketData(CQdpFtdcDepthMarketDataField \*pMarketData)

{

// 客户端按需处理返回的数据

printf("%s,%s,%d,",pMarketData->InstrumentID,pMarketData->UpdateTime,pMarketData->UpdateMillisec);

if (pMarketData->AskPrice1==DBL\_MAX)

printf("%s,","");

else

printf("%f,",pMarketData->AskPrice1);

if (pMarketData->BidPrice1==DBL\_MAX)

printf("%s \n","");

else

printf("%f \n",pMarketData->BidPrice1);

}

// 针对用户请求的出错通知

void OnRspError(CQdpFtdcRspInfoField \*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

}

///订阅合约的相关信息

void OnRspSubMarketData(CQdpFtdcSpecificInstrumentField \*pSpecificInstrument, CQdpFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

{

printf("Sub 返回订阅合约：%s \n",pSpecificInstrument->InstrumentID);

}

///订阅合约的相关信息

void OnRspUnSubMarketData(CQdpFtdcSpecificInstrumentField \*pSpecificInstrument, CQdpFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

{

printf("UnSub 返回订阅合约：%s \n",pSpecificInstrument->InstrumentID);

}

private:

// 指向CQdpFtdcMduserApi实例的指针

CQdpFtdcMduserApi \*m\_pUserApi;

};

int main()

{

// 产生一个CQdpFtdcMduserApi实例

CQdpFtdcMduserApi \*pUserApi = CQdpFtdcMduserApi::CreateFtdcMduserApi();

// 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);

// 注册需要的深度行情主题

/// TERT\_RESTART:从本交易日开始重传

/// TERT\_RESUME:从上次收到的续传

/// TERT\_QUICK:先传送当前行情快照,再传送登录后市场行情的内容 //pUserApi-> SubscribeMarketDataTopic (101, TERT\_RESUME);

//pUserApi-> SubscribeMarketDataTopic (110, QDP\_TERT\_RESTART);

// 设置行情发布服务器的地址

pUserApi->RegisterFront("tcp://192.168.1.100:7220");

// 使客户端开始与行情发布服务器建立连接

pUserApi->Init();

// 释放useapi实例

pUserApi->Release();

return 0;

}

## *上期能源或者大商所组播行情Demo*

#include <stdio.h>

#include <string.h>

#include <float.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <iostream>

#include "QdFtdcMdApi.h"

using namespace std;

char \*g\_mdFile = "marketdata.log";

char \*g\_apiMdFile = "apimarketdata.log";

void write\_txt(std::string strMkData, char \*fileName)

{

ofstream outfile;

outfile.open(fileName, ios::app);

outfile << strMkData << endl;

outfile.close();

}

template <typename T>

string MyTostring(T num)

{

stringstream ss;

string str;

ss << num;

ss >> str;

return str;

}

class CSimpleHandler : public CQdFtdcMduserSpi

{

public:

CSimpleHandler(CQdFtdcMduserApi \*pUserApi) : m\_pUserApi(pUserApi) {}

~CSimpleHandler() {}

// Udp行情接受

void UdpMarketData(CQdFtdcDepthMarketDataField \*qmdata)

{

printf("UdpMarketData: InstrumentID %s,%s,%d\n", qmdata->InstrumentID, qmdata->UpdateTime, qmdata->UpdateMillisec);

};

// 当客户端与行情发布服务器建立起通信连接，客户端需要进行登录

void OnFrontConnected()

{

CQdFtdcReqUserLoginField reqUserLogin;

memset(&reqUserLogin, 0, sizeof(CQdFtdcReqUserLoginField));

strcpy(reqUserLogin.BrokerID, "0001");

strcpy(reqUserLogin.UserID, "t002");

strcpy(reqUserLogin.Password, "111111");

m\_pUserApi->ReqUserLogin(&reqUserLogin, 0);

}

// 当客户端与行情发布服务器通信连接断开时，该方法被调用

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

void OnRspUserLogin(CQdFtdcRspUserLoginField \*pRspUserLogin, CQdFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

{

printf("OnRspUserLogin:\n");

if (NULL == pRspInfo)

{

return;

}

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

return;

}

char \*sContracts[2] = {0};

sContracts[0] = "pg2104";

sContracts[1] = "p2104";

// 注册需要的合约

m\_pUserApi->SubMarketData(sContracts, 2);

// 取消不需要的合约

// m\_pUserApi->UnSubMarketData(contracts,1);

CQdFtdcQryShfeMultiInfoField fld;

m\_pUserApi->ReqQrySHFEMultiInfo(&fld,0);

}

// 深度行情通知，行情服务器会主动通知客户端

void OnRtnDepthMarketData(CQdFtdcDepthMarketDataField \*pMarketData)

{

if (NULL == pMarketData)

{

printf("NULL == pMarketData\n");

return;

}

printf("InstrumentID %s--vm:%d------%d------%d------%d------%d------%d------%d------%d------%d------%d------%d------%d\n",

pMarketData->InstrumentID,

pMarketData->Volume,

pMarketData->AskPrice1, pMarketData->AskVolume2, pMarketData->AskVolume3, pMarketData->AskVolume4, pMarketData->AskVolume5,

pMarketData->BidPrice1, pMarketData->BidVolume2, pMarketData->BidVolume3, pMarketData->BidVolume4, pMarketData->BidVolume5, pMarketData->PacketNo);

string strMarketData = "";

strMarketData +=

"ExchangeID=[" + string(pMarketData->ExchangeID) + "],

InstrumentID=[" + string(pMarketData->InstrumentID) + "],

InstrumentStatus=[" + string(1, pMarketData->InstrumentStatus) + "],

LastPrice=[" + MyTostring(pMarketData->LastPrice) + "],

AskPrice1=[" + MyTostring(pMarketData->AskPrice1) +"],

AskVolume1=[" + MyTostring(pMarketData->AskVolume1) + "],

BidPrice1=[" + MyTostring(pMarketData->BidPrice1) +"],

BidVolume1=[" + MyTostring(pMarketData->BidVolume1) + "],

AskPrice2=[" + MyTostring(pMarketData->AskPrice2) +"],

AskVolume2=[" + MyTostring(pMarketData->AskVolume2) + "],

BidPrice2=[" + MyTostring(pMarketData->BidPrice2) +"],

BidVolume2=[" + MyTostring(pMarketData->BidVolume2) + "],

AskPrice3=[" + MyTostring(pMarketData->AskPrice3) +"],

AskVolume3=[" + MyTostring(pMarketData->AskVolume3) + "],

BidPrice3=[" + MyTostring(pMarketData->BidPrice3) +"],

BidVolume3=[" + MyTostring(pMarketData->BidVolume3) + "],

AskPrice4=[" + MyTostring(pMarketData->AskPrice4) + "],

AskVolume4=[" + MyTostring(pMarketData->AskVolume4) + "],

BidPrice4=[" + MyTostring(pMarketData->BidPrice4) +"],

BidVolume4=[" + MyTostring(pMarketData->BidVolume4) + "],

AskPrice5=[" + MyTostring(pMarketData->AskPrice5) +"],

AskVolume5=[" + MyTostring(pMarketData->AskVolume5) + "],

BidPrice5=[" + MyTostring(pMarketData->BidPrice5) +"],

BidVolume5=[" + MyTostring(pMarketData->BidVolume5) + "],

UpdateTime=[" + MyTostring(pMarketData->UpdateTime) +"],

UpdateMillisec=[" + MyTostring(pMarketData->UpdateMillisec) + "],

PacketNo=[" + MyTostring(pMarketData->PacketNo) + "]";

write\_txt(strMarketData, g\_mdFile);

}

// 针对用户请求的出错通知

void OnRspError(CQdFtdcRspInfoField \*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

}

/// 订阅合约的相关信息

void OnRspSubMarketData(CQdFtdcSpecificInstrumentField \*pSpecificInstrument, CQdFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

{

printf("Sub 返回订阅合约：%s \n", pSpecificInstrument->InstrumentID);

}

/// 订阅合约的相关信息

void OnRspUnSubMarketData(CQdFtdcSpecificInstrumentField \*pSpecificInstrument, CQdFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

{

printf("UnSub 返回订阅合约：%s \n",pSpecificInstrument->InstrumentID);

}

// 以下是组播行情回调函数

// 用户可继承此接口解析上期所组播原始报文-用户自己解析时候

void OnMutiData(char \*data, int len)

{

printf("[qdp multicast md] multidata: %d\n", len);

}

/// 组播参数查询结束通知

void OnRtnShfeMultiLevel(CQdFtdcShfeMultiLevelField \*pShfeMultiLevel)

{

printf("[qdp multicast md] Level: %d\n", pShfeMultiLevel->Level);

CQdFtdcQryMarketDataField reqFld;

reqFld.TopicID = 141;

m\_pUserApi->ReqQryDepthMarketData(&reqFld, 0);

}

/// 组播参数通知

void OnRtnShfeMultiParameters(

CQdFtdcShfeMultiParametersField \*pShfeMultiParameters)

{

printf("[qdp multicast md] OnRtnShfeMultiParameters: %s, %d, %f, %f\n", pShfeMultiParameters->InstrumentID,

pShfeMultiParameters->InstrumentNo, pShfeMultiParameters->CodecPrice, pShfeMultiParameters->PriceTick);

}

void OnRspQryDepthMarketData(

CQdFtdcDepthMarketDataField \*pDepthMarketData,

CQdFtdcRspInfoField \*pRspInfo,

int nRequestID,

bool bIsLast)

{

if (pDepthMarketData == NULL)

return;

printf("OnRspQryDepthMarketData InstrumentID=%s PacketNo = %d InstrumentNo = %d\n",pDepthMarketData->InstrumentID, pDepthMarketData->PacketNo, pDepthMarketData->InstrumentNo);

};

// 解析后的上期组播数据,api内部解析只回调这个函数

void OnRtnShfeMultiMarketData(CQdFtdcDepthMarketDataField \*pMarketData)

{

if (NULL == pMarketData)

{

printf("OnRtnShfeMultiMarketData: NULL == pMarketData\n");

return;

}

printf("OnRtnShfeMultiMarketData InstrumentID=%s,UpdateTime=%s,UpdateMillisec=%d,LastPrice=%.3f,Volume=%d,AskPrice1=%.3f,BidPrice1=%.3f,ExchangeID=%s,PacketNo=%d\n",

pMarketData->InstrumentID,

pMarketData->UpdateTime,

pMarketData->UpdateMillisec,

pMarketData->LastPrice,

pMarketData->Volume,

pMarketData->AskPrice1,

pMarketData->BidPrice1,

pMarketData->ExchangeID,

pMarketData->PacketNo);

string strMarketData = "";

strMarketData +=

"ExchangeID=[" + string(pMarketData->ExchangeID) + "],

InstrumentID=[" +string(pMarketData->InstrumentID) + "],

InstrumentStatus=[" + string(1, pMarketData->InstrumentStatus) +"],

LastPrice=[" + MyTostring(pMarketData->LastPrice) + "],

AskPrice1=[" + MyTostring(pMarketData->AskPrice1) +"],

AskVolume1=[" + MyTostring(pMarketData->AskVolume1) + "],

BidPrice1=[" + MyTostring(pMarketData->BidPrice1) +"],

BidVolume1=[" + MyTostring(pMarketData->BidVolume1) + "],

AskPrice2=[" + MyTostring(pMarketData->AskPrice2) +"],

AskVolume2=[" + MyTostring(pMarketData->AskVolume2) + "],

BidPrice2=[" + MyTostring(pMarketData->BidPrice2)+"],

BidVolume2=[" + MyTostring(pMarketData->BidVolume2) + "],

AskPrice3=[" + MyTostring(pMarketData->AskPrice3) +"],

AskVolume3=[" + MyTostring(pMarketData->AskVolume3) + "],

BidPrice3=[" + MyTostring(pMarketData->BidPrice3) +"],

BidVolume3=[" + MyTostring(pMarketData->BidVolume3) + "],

AskPrice4=[" + MyTostring(pMarketData->AskPrice4) +"],

AskVolume4=[" + MyTostring(pMarketData->AskVolume4) + "],

BidPrice4=[" + MyTostring(pMarketData->BidPrice4) +"],

BidVolume4=[" + MyTostring(pMarketData->BidVolume4) + "],

AskPrice5=[" + MyTostring(pMarketData->AskPrice5) +"],

AskVolume5=[" + MyTostring(pMarketData->AskVolume5) + "],

BidPrice5=[" + MyTostring(pMarketData->BidPrice5) +"],

BidVolume5=[" + MyTostring(pMarketData->BidVolume5) + "],

UpdateTime=[" + MyTostring(pMarketData->UpdateTime) +"],

UpdateMillisec=[" + MyTostring(pMarketData->UpdateMillisec) + "],

PacketNo=[" + MyTostring(pMarketData->PacketNo) + "]";

write\_txt(strMarketData, g\_apiMdFile);

}

private:

// 指向CQdFtdcMduserApi实例的指针

CQdFtdcMduserApi \*m\_pUserApi;

};

int main()

{

CQdFtdcMduserApi \*pUserApi = CQdFtdcMduserApi::CreateFtdcMduserApi();

// 产生一个事件处理的实例

CSimpleHandler \*sh = new CSimpleHandler(pUserApi);

// 注册一事件处理的实例

pUserApi->RegisterSpi(sh);

// 1、传统的tcp api行情

//pUserApi->RegisterFront("tcp://192.168.101.123:30007");

//下面大商和上期通道只能启动一个

// 2、大商所组播行情

// 大商所组播地址(双通道)

//char DMultiAddr1[256] = "141,dcemulti://10.252.3.106@239.3.42.71:21000#192.168.48.71";

//char DMultiAddr2[256] = "141,dcemulti://10.252.3.106@239.3.42.71:21000#192.168.48.71";

//pUserApi->SetMultiCast(true);

//pUserApi->RegTopicMultiAddr(DMultiAddr1);

//pUserApi->RegTopicMultiAddr(DMultiAddr2);

/\*// 3、上期主播行情

// char sFrontAaddr[256] = "tcp://1.1.1.1:30007";

//上期组播地址

char SMultiAddr[256] = "1000,multi://10.252.3.106@239.3.42.71:21000#192.168.48.71";

pUserApi->SetMultiCast(true);

pUserApi->SetUserFreedom(false);//这里由api解析

// pUserApi->SetUserFreedom(true);//这里由用户自己解析 (客户可以通过OnMutiData接口接收原始组播报文)

pUserApi->RegTopicMultiAddr(SMultiAddr);

// API初始化

pUserApi->Init();

pUserApi->Join();

pUserApi->Release();

return 0;

}