飞马平台
MdUserAPI 说明书

2013 年 5 月 18 日

文档标识

项目名称	FEMAS
文档名称	MdUser API
版本号	0.9
状况	○ 草案 ○ 评审过的 ◉ 更新过的 ○ 定为基线的

文档修订历史

版本	日期	描述	修订者
V0.1	20121201	初稿	徐忠华
V0.6	20130605	修改为与交易 api 相同的结构体	徐忠华
V0.9	20130618	实测版本	徐忠华

此版本文档的正式核准

姓名	签字	日期

目录

第1章 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
UstpFtdcMduserApi.h	V1.0		行情接口头文件
UstpFtdcUserApiDataType.h	V1.0		定义了 API 所需的一系列数据类型的头文件
UstpFtdcUserApiStruct.h	V1.0		定义了一系列业务相关的数据结构的头文件
USTPmduserapi.dll	V1.0		动态链接库二进制文件
USTPmduserapi.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。

第2章 体系结构

行情 API 使用建立在 TCP 协议之上 FTD 协议与飞马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”。

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。

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。该请求被飞马服务端接收和处理，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。

广播通讯模式是指飞马服务端主动，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：

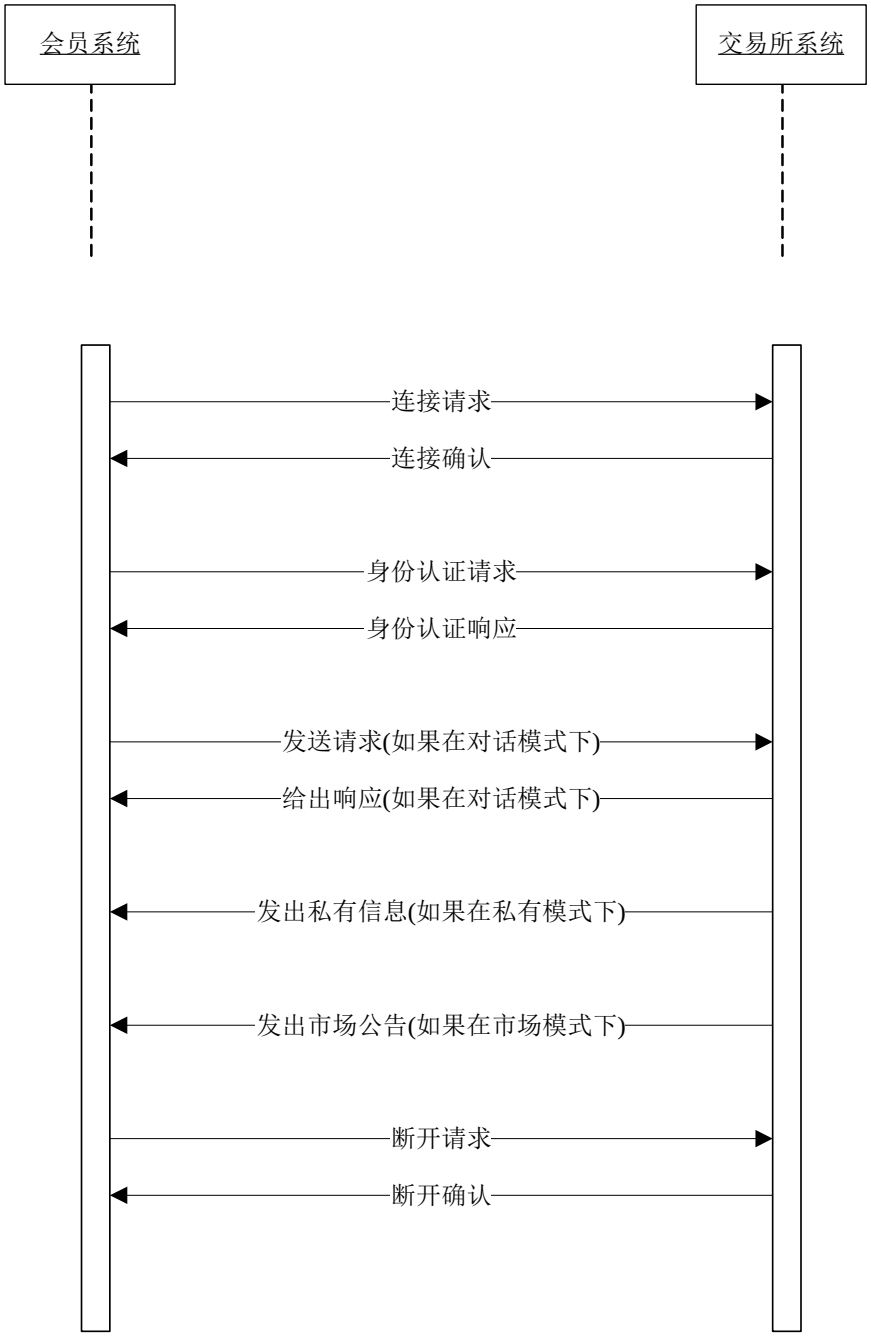

图1) 各通讯模式的工作过程

2.2 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。目前是飞马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。

第3章 运行模式

3.1 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。

UstpFtdcMduserApi 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。

UstpFtdcMduserSpi 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 UstpFtdcTraderSpi 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。

3.2 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 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”。

第4章 开发接口

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

4.1 CUstpFtdcMduserSpi 接口

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

4.1.1 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。

4.1.2 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

4.1.3 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。

函数原型：

void OnHeartBeatWarning(int nTimeLapse)；

参数：

nTimeLapse：距离上次接收报文的时间

4.1.4 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(
CUstpFtdcRspUserLoginField *pRspUserLogin,
CUstpFtdcRspInfoField *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 CUstpFtdcRspUserLoginField
{
    ///交易日
    TUstpFtdcDateType       TradingDay;
    ///登录成功时间
    TUstpFtdcTimeType       LoginTime;
    ///最大本地报单号
    TUstpFtdcOrderLocalIDTypeMaxOrderLocalID;
    ///交易用户代码
    TUstpFtdcUserIDType    UserID;
    ///会员代码
    TUstpFtdcParticipantIDType  ParticipantID;
};
```

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 CUstpFtdcRspInfoField
{
    ///错误代码
    TUstpFtdcErrorIDType   ErrorID;
    ///错误信息
    TUstpFtdcErrorMsgType      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。

4.1.5 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(
CUstpFtdcRspUserLogoutField *pRspUserLogout,
CUstpFtdcRspInfoField *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 CUstpFtdcRspUserLogoutField
{
    ///交易用户代码
    TUstpFtdcUserIDType   UserID;
    ///会员代码
    TUstpFtdcParticipantIDType  ParticipantID;
};
```

pRspInfo：返回用户响应信息的地址。

响应信息结构：

```
struct CUstpFtdcRspInfoField
{
    ///错误代码
    TUstpFtdcErrorIDType  ErrorID;
    ///错误信息
    TUstpFtdcErrorMsgType    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。

4.1.6 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(CUstpFtdcDepthMarketDataField *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 CUstpFtdcDepthMarketDataField
{
///交易日
TUstpFtdcDateType    TradingDay;
///结算组代码
TUstpFtdcSettlementGroupIDType    SettlementGroupID;
///结算编号
TUstpFtdcSettlementIDType  SettlementID;
///最新价
TUstpFtdcPriceType    LastPrice;
///昨结算
TUstpFtdcPriceType    PreSettlementPrice;
///昨收盘
TUstpFtdcPriceType    PreClosePrice;
///昨持仓量
TUstpFtdcLargeVolumeType PreOpenInterest;
///今开盘
TUstpFtdcPriceType    OpenPrice;
///最高价
```

```
TUstpFtdcPriceType	HighestPrice;
///最低价
TUstpFtdcPriceType	LowestPrice;
///数量
TUstpFtdcVolumeType	Volume;
///成交金额
TUstpFtdcMoneyType	Turnover;
///持仓量
TUstpFtdcLargeVolumeType OpenInterest;
///今收盘
TUstpFtdcPriceType	ClosePrice;
///今结算
TUstpFtdcPriceType	SettlementPrice;
///涨停板价
TUstpFtdcPriceType	UpperLimitPrice;
///跌停板价
TUstpFtdcPriceType	LowerLimitPrice;
///昨虚实度
TUstpFtdcRatioType	PreDelta;
///今虚实度
TUstpFtdcRatioType	CurrDelta;
///最后修改时间
TUstpFtdcTimeType	UpdateTime;
///最后修改毫秒
TUstpFtdcMillisecType	UpdateMillisec;
///合约代码
TUstpFtdcInstrumentIDType InstrumentID;
///申买价一
TUstpFtdcPriceType	BidPrice1;
///申买量一
TUstpFtdcVolumeType	BidVolume1;
///申卖价一
TUstpFtdcPriceType	AskPrice1;
///申卖量一
TUstpFtdcVolumeType	AskVolume1;
///申买价二
TUstpFtdcPriceType	BidPrice2;
///申买量二
TUstpFtdcVolumeType	BidVolume2;
///申卖价二
TUstpFtdcPriceType	AskPrice2;
///申卖量二
TUstpFtdcVolumeType	AskVolume2;
///申买价三
```

```
TUstpFtdcPriceType      BidPrice3;
///申买量三
TUstpFtdcVolumeType     BidVolume3;
///申卖价三
TUstpFtdcPriceType      AskPrice3;
///申卖量三
TUstpFtdcVolumeType     AskVolume3;
///申买价四
TUstpFtdcPriceType      BidPrice4;
///申买量四
TUstpFtdcVolumeType     BidVolume4;
///申卖价四
TUstpFtdcPriceType      AskPrice4;
///申卖量四
TUstpFtdcVolumeType     AskVolume4;
///申买价五
TUstpFtdcPriceType      BidPrice5;
///申买量五
TUstpFtdcVolumeType     BidVolume5;
///申卖价五
TUstpFtdcPriceType      AskPrice5;
///申卖量五
TUstpFtdcVolumeType     AskVolume5;
};
```

4.1.7 OnRspError 方法

针对用户请求的出错通知。

函数原型：

```
void OnRspError(
CUstpFtdcRspInfoField *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 CUstpFtdcRspInfoField
{
    ///错误代码
    TUstpFtdcErrorIDType   ErrorID;
```

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

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

4.1.8 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(CUstpFtdcSpecificInstrumentField *pSpecificInstrument, CUstpFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pSpecificInstrument：返回用户成功订阅的合约。

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。

4.1.9 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(CUstpFtdcSpecificInstrumentField *pSpecificInstrument, CUstpFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)

参数：

pSpecificInstrument：返回用户成功订阅的合约。

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。

4.2 CUstpFtdcMduserApi 接口

CUstpFtdcMduserApi 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。

4.2.1 CreateFtdcMduserApi 方法

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

函数原型：

static CUstpFtdcMduserApi *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 CUstpFtdcMduserApi 实例的指针。

4.2.2 Release 方法

释放一个 CUstpFtdcMduserApi 实例。

函数原型：

void Release();

4.2.3 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();

4.2.4 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();

4.2.5 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。

4.2.6 RegisterSpi 方法

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

函数原型：

void RegisterSpi(CUstpFtdcMduserSpi *pSpi) ;

参数：

pSpi：实现了 CUstpFtdcMduserSpi 接口的实例指针。

4.2.7 RegisterFront 方法

设置行情发布服务器的地址。

函数原型：

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。

4.2.8 RegisterNameServer 方法

设置飞马 NameServer 的网络通讯地址，用于获取行情服务列表。交易系统拥有多个 NameServer，用户可以同时注册多个 NameServer 的网络通讯地址。

该方法要在 Init 方法之前调用。

函数原型：

```
void RegisterNameServer (char *pszNsAddress);
```

参数：

pszNsAddress：指向飞马服务端 NameServer 网络通讯地址的指针。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。

注意：此接口保留，但目前并未启用！

4.2.9 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，飞马采用其主题。

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

4.2.10 ReqUserLogin 方法

用户发出登陆请求。

函数原型：

```
int ReqUserLogin(
CUstpFtdcReqUserLoginField *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 CUstpFtdcReqUserLoginField
{
///交易日
TUstpFtdcDateType       TradingDay;
///交易用户代码
TUstpFtdcUserIDType    UserID;
///会员代码
TUstpFtdcParticipantIDType  ParticipantID;
///密码
TUstpFtdcPasswordType       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。

4.2.11 ReqUserLogout 方法

用户发出登出请求。

函数原型：

```
int ReqUserLogout(
CUstpFtdcReqUserLogoutField *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 CUstpFtdcReqUserLogoutField
{
    ///交易用户代码
    TUstpFtdcUserIDType    UserID;
    ///会员代码
    TUstpFtdcParticipantIDType  ParticipantID;
};
```

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。其它值代表失败。

4.2.12 SubMarketData 方法

用户请求订阅的合约号。

函数原型：

```
int SubMarketData(char *ppInstrumentID[], int nCount)
```

参数：

ppInstrumentID：合约 ID 数组地址。

nCount：合约的个数。

返回值：

0，代表成功。其它值代表失败。

4.2.13 UnSubMarketData 方法

用户请求去除订阅的合约号。

函数原型：

```
int UnSubMarketData(char *ppInstrumentID[], int nCount)
```

参数：

ppInstrumentID：退订合约 ID 数组地址。

nCount：合约的个数。

返回值：

0，代表成功。其它值代表失败。

第5章 开发示例

```
// mddemo.cpp :
//一个简单的例子，介绍UstpFtdcMduserApi和UstpFtdcMduserSpi接口的使用。

#include <stdio.h>
#include <string.h>
#include <float.h>
#include "UstpFtdcMduserApi.h"

class CSimpleHandler : public CUstpFtdcMduserSpi
{
public:
    // 构造函数，需要一个有效的指向CUstpFtdcMduserApi实例的指针
    CSimpleHandler(CUstpFtdcMduserApi *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()
    {
        CUstpFtdcReqUserLoginField reqUserLogin;
        strcpy(reqUserLogin.TradingDay, m_pUserApi->GetTradingDay());
        strcpy(reqUserLogin.BrokerID, "0001");
        strcpy(reqUserLogin.UserID, "000101");
        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(CUstpFtdcRspUserLoginField *pRspUserLogin,
CUstpFtdcRspInfoField *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]="IF1212";
        contracts[1]="IF1303";
        contracts[2]="IF1210";
        m_pUserApi->SubMarketData(contracts, 3);

        char * uncontracts[2]={"",""};
        uncontracts[0]="IF1211";
        uncontracts[1]="IF1212";
        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(CUstpFtdcDepthMarketDataField *pMarketData)
    {
        // 客户端按需处理返回的数据

    printf("%s,%s,%d,",pMarketData->InstrumentID,pMarketData->UpdateTime,pMarke
tData->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(CUstpFtdcRspInfoField *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(CUstpFtdcSpecificInstrumentField
*pSpecificInstrument, CUstpFtdcRspInfoField *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(CUstpFtdcSpecificInstrumentField
*pSpecificInstrument, CUstpFtdcRspInfoField *pRspInfo, int nRequestID, bool
bIsLast)
    {
        printf("UnSub 返回订阅合约：%s \n",pSpecificInstrument->InstrumentID);
    }

private:
    // 指向CUstpFtdcMduserApi实例的指针
    CUstpFtdcMduserApi *m_pUserApi;
};

int main()
{
    // 产生一个CUstpFtdcMduserApi实例
    CUstpFtdcMduserApi *pUserApi = CUstpFtdcMduserApi::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, USTP_TERT_RESTART);
    // 设置行情发布服务器的地址
    pUserApi->RegisterFront("tcp://127.0.0.1:9987");
    // 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;
}
```

