行情客户端应用程序接口

2023 年 12 月 5 日

目录

第1章 介绍

1.1 MduserAPI 简介

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订阅和接收功能。该类库包含以下 5 个文件：

文件名	版本	文件描述
CFfexFtdcMduserApi.h	V2.0	行情接口头文件
CFfexFtdcMduserApiStruct.h	V2.0	定义了 MduserAPI 所需的一系列数据类型的头文件
CFfexFtdcMduserApiDataType.h	V2.0	定义了一系列业务相关的数据结构的头文件
libCFFEXmduserapi.dll	V2.0	动态链接库二进制文件
libCFFEXmduserapi.lib	V2.0	导入库文件
libCFFEXmduserapi_lnx64.so	V2.0	Linux 动态库

Windows 支持 VS2012 编译器，需要打开多线程编译选项/MT。Linux 版本 api 是基于 Redhat 6.3 版本，gcc 版本为 4.4.6。

1.2 MduserAPI 发行的平台

目前发布了以下操作系统平台的版本：

- Intel X86/Windows7：包括.h 文件、.dll 文件和.lib 文件。
- Linux RedHat6.3：包括.h 文件和.so 文件。

1.3 修改历史

1.3.1 版本 1.5

本版本基于《行情客户端应用程序接口 V12》修改。主要有以下变更：

- 本版本提供了灾备功能：

- 增加【4.10 灾备接口】，简要说明了灾备原理。
- 由于登录报文中增加了数据中心代码， MdUserAPI 修改了 ReqUserLogin 和 OnRspUserLogin 方法的参数。

- 本版本提供了数据流长度的查询功能：
  - 登录交易系统时，应答中会返回当前会员私有流长度和交易员私有流长度。
  - MdUserAPI 增加了对 ReqQryTopic 和 RspQryTpoic 方法说明，用于查询流长度。
- 对以前版本发现问题的修正：
  - MdUserAPI 增加了 GetVersion 方法说明，之前版本虽提供功能但无文档说明。
  - MdUserAPI 增加了 RegisterNameServer 方法说明，之前版本虽提供功能但无文档说明。
- 使用限制
  - 为提高 API 的性能，在 WIN32 环境下，MduserAPI 在初始化时，随机使用了 39601-39650 的 Tcp 端口。Linux 版本没有这个限制。

1.3.2 版本 1.59

本版本基于《行情客户端应用程序接口 V15》修改。接口无变化，主要有以下变更：

- 本版本修改了版本运行和编译说明

1.3.3 版本 2.0

本版本基于《行情客户端应用程序接口 V1.59》修改。主要有以下变更：

- 本版本新增组播行情流
  - 行情 API2.0 支持 2 种行情接收方式:TCP 模式、混合模式。
    - TCP 模式只接收 TCP 行情流。
    - 混合模式根据网络状况选择接收 TCP 行情流或组播行情流。
- 本版本增加了部分行情请求接口和行情回调接口。
  - 为了满足行情商需求，新增公有流订阅接口，订阅该接口可以收到合约、合约状态通知和交易所公告消息。
  - 扩展了部分接口功能
    - 实例创建接口（CreateFtdcMduserApi）新增运行模式参数，创建实例时需要制定 API 端的运行模式。
    - 主题订阅接口（SubscribeMarketDataTopic）新增组播通道地址参数。如果使用 TCP 模式，该参数为 NULL，如果使用混合模式，该参数不能为 NULL。
- 本版本修改了版本运行和编译说明
  - 不再依赖于 openssl 库
  - 变更了 Windows 的编译环境
- 从本版本开始，行情 API 与交易 API 分开发布
  - 为了让行情 API 区别于交易 API，对行情 API 里相应的文件名称、数据结构、类型名称做了更新
- 在深度行情域 CFfexFtdcMdDepthMarketDataField 的末尾添加上带价、下带价两个字段

第2章 接收模式

2.1 行情接收模式

行情 API2.0 提供 2 种行情流接收模式：TCP 模式、混合模式。

TCP 模式是 API 端默认接收方式，接口使用方式和 1.59API 完全兼容，柜台系统只需要更新头文件和数据域字段名称重新编译即可。

混合模式混合了组播和 TCP 两种接收方式。API 刚启动时，TCP 通道接收行情快照及续传行情，组播通道缓存实时行情。TCP 行情序号和组播缓存的序号平衡后（平衡由 API 决策），接收通道会自动切换到组播通道上，如果组播通道出现了丢包或者故障，接收通道会自动的切换到 TCP 上，如果组播通道恢复，接收通道会从 TCP 通道切换到组播通道。使用混合模式需要柜台系统在创建 API 实例时指定混合运行模式，同时在订阅主题时绑定组播地址。

运行模式参数	行情接收模式
MDRM_TCP	TCP 模式
MDRM_MIX	混合模式

2.1.1 数据流订阅方式

TCP 模式和混合模式都支持 3 种行情重传方式：重发（RESTART）、续传（RESUME）和快照（QUICK）。

- 重发（RESTART）方式从行情数据流中的第一个报文开始传输，这时 API 将忽略本地文件中记录的数据流报文序号。
- 续传（RESUME）方式从本地文件中记录的行情流报文序号之后开始传输。
- 快照（QUICK）方式从订阅这一时刻数据流最大序号开始传输。

2.1.2 时序

TCP 模式下，API 的运行时序和 1.59API 保持一致，如下图所示。TCP 行情通道负责快照及 TCP 实时行情推送。席位登录成功后，根据席位的主题订阅权限，使用 TCP 模式的席位从 TCP 通道获取相应的主题行情，而发送端在 TCP 通道上通过轮询的方式发送行情。

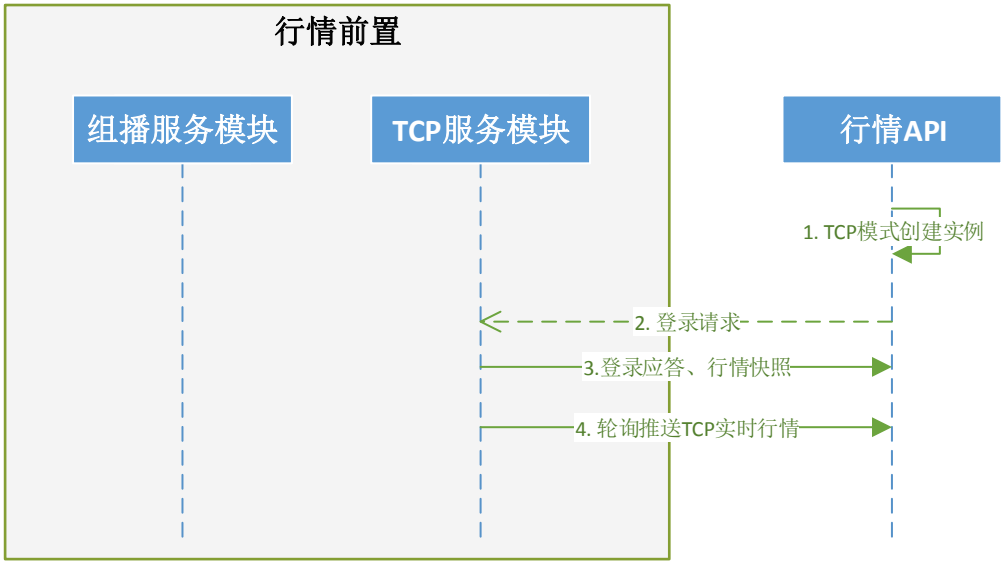


图 1.TCP 模式启动时序

混合模式下，API 刚启动时，会先加入组播组，缓存组播通道中的实时行情报文。系统登录成功后，API 通过 TCP 通道获取快照和历史行情，TCP 行情和组播通道中的行情平衡以后（TCP 和组播序号一致，并且组播通道在最近 5 秒未丢失数据），API 会自动关闭 TCP 接收通道，转为组播通道接收实时行情。如果在组播行情接收过程中，发生通道故障或者丢包的情况，API 会自动切换到 TCP 通道接收实时行情，从而保证行情数据不会丢失。在混合模式下建议同时注册多个组播通道，通过多通道数据冗余降低组播通道数据丢失的风险。

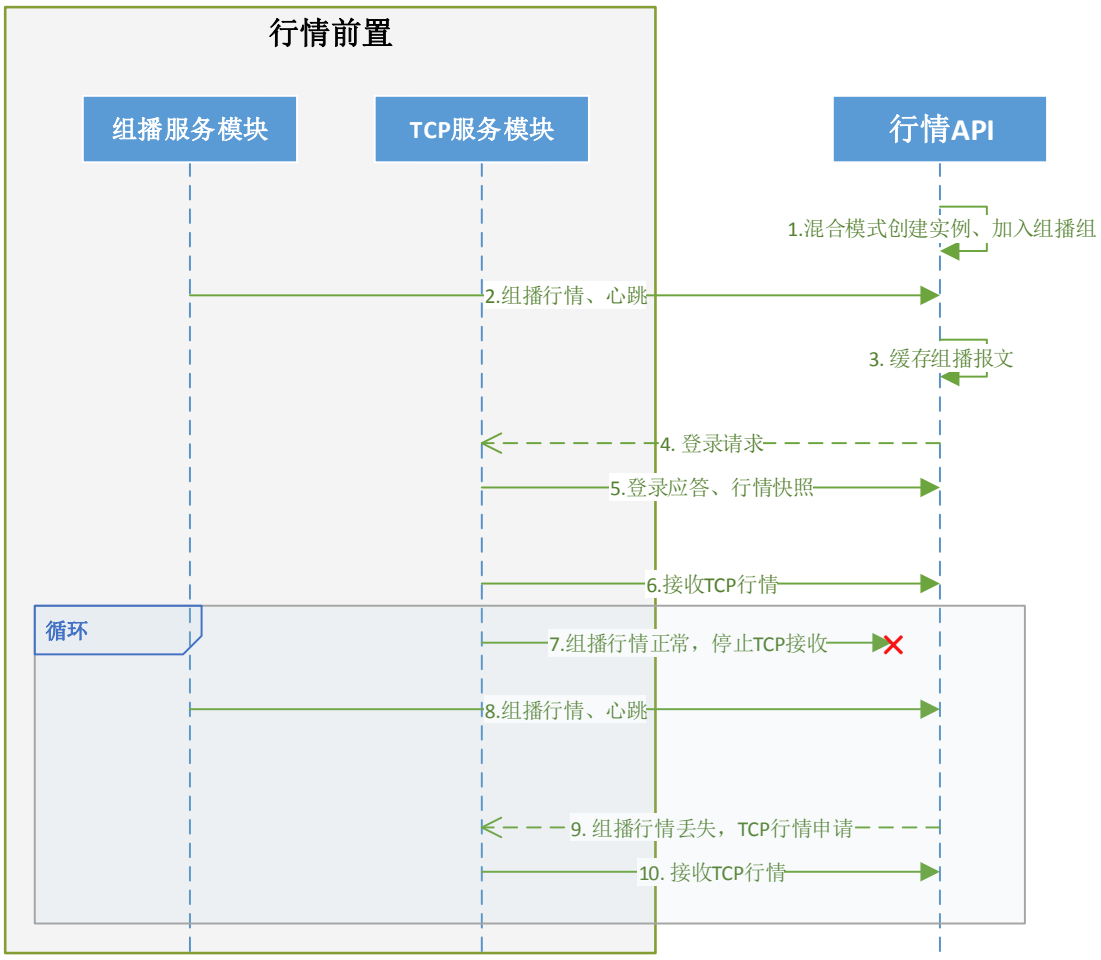


图 2：混合模式启动时序

2.2 数据流

行情 API 有三种数据流：对话流、公共流、行情流。对话流和公共流只能通过 TCP 通道接收，行情流通过 TCP 或组播通道接收。

2.2.1 对话流

对话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。请求、应答、查询等接口是通过对话流交互。

2.2.2 公有流

行情公有流是一个单向的数据流，由行情前置发给会员柜台，包括合约通知、合约状态通知、交易所公告三类消息。会员端柜台必须在 Init 之前调用公有流订阅接口才能收到公有流消息。

2.2.3 行情流

行情 API 的核心功能就是接收行情流。行情数据流也是一个单向数据流，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。交易所会公布各行情主题的具体内容，并设定每个行情用户所能订阅的行情主题。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。

行情流的接收通道有两种，TCP 行情流和组播行情流。TCP 模式的接收方式为 TCP 行情流，而混合模式会根据当前的网络状态选择接收 TCP 行情流或者组播行情流。

TCP 行情流是一个可靠的数据流，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。组播行情流底层使用的是 UDP 通信，在网络状态不佳的情况下，可能存在丢包的情况。在混合模式下，为了保证数据的完整性，如果出现丢包（默认等待 100ms）或组播心跳超时（默认等待 3s）的情况，API 端会自动从组播行情流切换到 TCP 行情流。

第3章 接口模式

3.1 MduserAPI 接口

MduserAPI 提供了两个接口类，分别为 CFfexFtdcMduserApi 和 CFfexFtdcMduserSpi。这两个接口类是对 FTD 协议的封装。

行情接收系统可以通过 CFfexFtdcMduserApi 发出操作请求，通过继承 CFfexFtdcMduserSpi 并重载回调函数来处理交易系统的响应。

3.1.1 对话流编程接口

通过对话流进行通讯的编程接口通常如下：

```
////请求：
int CFfexFtdcMduserApi::ReqXXX(
                CFfexFtdcMdXXXField *pReqXXX,
                int nRequestID)
////响应：
void CFfexFtdcMduserSpi::OnRspXXX(
                CFfexFtdcMdXXXField *pRspXXX,
                CFfexFtdcMdRspInfoField *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。

请求接口的第二个参数为请求号。请求号由行情接收系统应用程序负责维护，正常情况下每个请求的请求号不会重复。在接收交易系统的响应时，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。

当收到交易系统应答时，CFfexFtdcMduserSpi 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。

回调函数一共包含四个参数。其中：

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。

3.1.2 行情流编程接口

行情流中的数据包含了交易系统推出的行情信息。

通过行情流接收回报的编程接口通常如下：

```
void CFfexFtdcMduserSpi::OnRtnXXX(CFfexFtdcMdXXXField *pXXX);
```

当收到交易系统通过行情流发布的行情数据时，CFfexFtdcMduserSpi 的回调函数会被调用。回调函数的参数为通知的具体内容。

3.1.3 公共流编程接口

公共流中的数据中交易所的公共信息，包括合约、公告等。

通过公共流接收回报的编程接口通常如下：

```
void CFfexFtdcTraderSpi::OnRtnXXX(CFfexFtdcXXXField *pXXX)
```

当收到交易后台通过公共流发布的回报数据时，CFfexFtdcTraderSpi 的回调函数会被调用。回调函数的参数为通知的具体内容。

第4章 运行模式

4.1 工作流程

行情接收系统和交易系统的交互过程分为 2 个阶段：初始化阶段和功能调用阶段。

4.1.1 初始化阶段

在初始化阶段，行情接收系统的程序必须完成如下步骤（具体代码请参考开发实例）：

顺序	行情接收系统
1	创建一个 CFfexFtdcMduserApi 实例，指定 API 运行模式
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	TCP 模式：订阅主题行情流，组播地址为 NULL 混合模式：订阅主题行情流，绑定组播地址。
5	设置行情服务 NameServer 的网络地址。[1]
6	初始化

[1] 为了保持与上一版的兼容性，API 仍然提供了注册行情服务的接口，但交易所建议不要使用这些接口，这些接口将在下一版本中取消。有关 NameServer 的说明参见 0 前置机列列表。

4.1.2 功能调用阶段

在功能调用阶段，会员系统可以调用行情接收系统中的请求方法，如 ReqUserLogin，同时提供回调函数以响应回报信息。注意事项：

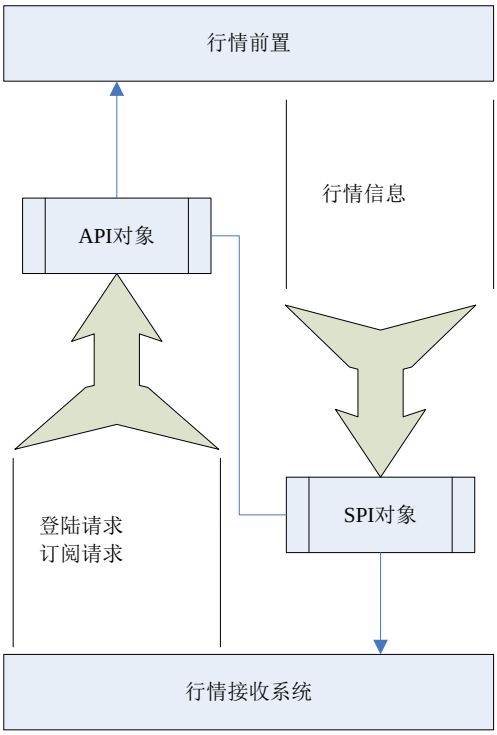
1. API 请求的输入参数不能为 NULL。
2. API 请求的返回参数，0 表示正确，其他表示错误，详细错误编码请查表。

4.2 工作线程

行情接收系统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 API 工作线程。应用程序与行情前置的通讯是由 API 工作线程驱动的。

CFfexFtdcMduserApi 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。

CFfexFtdcMduserSpi 提供的接口回调是由 MduserAPI 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 CFfexFtdcMduserSpi 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。



4.3 与交易所前置系统的连接

MduserAPI 使用建立在 TCP 协议之上的 FTD 协议与交易所的行情前置系统进行通信。MduserAPI 使用 CFfexFtdcMduserApi:: RegisterFront 方法注册交易所行情前置系统的网络通讯地址。

交易所拥有多个行情前置系统，用于负载均衡且互为备份，从而提高系统的性能和可靠性。为保证交易时通信的可靠性， MduserAPI 可以注册多个前置。

API 在初始化后，会从已注册的前置中随机挑选一个前置，尝试建立网络连接，如果不成功，则依次逐个尝试其它前置，直到连接成功为止。如果在交易过程中网络连接出现故障， API 依然使用上述过程，尝试连接其它前置。

中国金融期货交易所将公布至少两台前置机的网络地址，因此会员系统应该至少注册两个前置机的网络地址以防止所连交易前置发生故障从而引发单点隐患。

中国金融期货交易所建议使用 NameServer 方式接入，MdUserAPI 使用 CFfexFtdcMduserApi:: RegisterNameServer 方法注册交易所名字服务器网络地址。可以多次调用该函数注册多个名字服务器网络地址。MdUserAPI 会自动连接 NameServer 获取一组前置地址然后连接前置。

4.4 本地文件

MduserAPI 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"。 交易 API 与行情 API 的本地文件必须分目录存放。

4.5 前置机列表

会员系统连接交易所行情前置机后方能接入中金所交易系统。为了容错和负载均衡，交易所将在主用数据中心和备用数据中心部署两组、每组多台前置机。交易所将公布各前置机的网络地址的列表，会员系统从列表中随机选出一个尝试与其建立连接；会员系统在某一时刻只与一个前置机相连，如果由该前置机出现故障导致连接中断或者超时，则会员系统需尝试连接列表中的其它前置机。

会员系统得到前置列表的方式有两种：

- 交易所公布前置列表，会员系统通过 API 的 RegisterFront 接口，将前置机逐一注册进 API。
- 交易系统提供 NameServer，其功能是向 API 公布前置机列表。交易所首先公布 NameServer 列表，会员系统通过 API 的 RegisterNameServer 接口，将 NameServer 注册进 API。API 先尝试从 NameServer 中得到前置

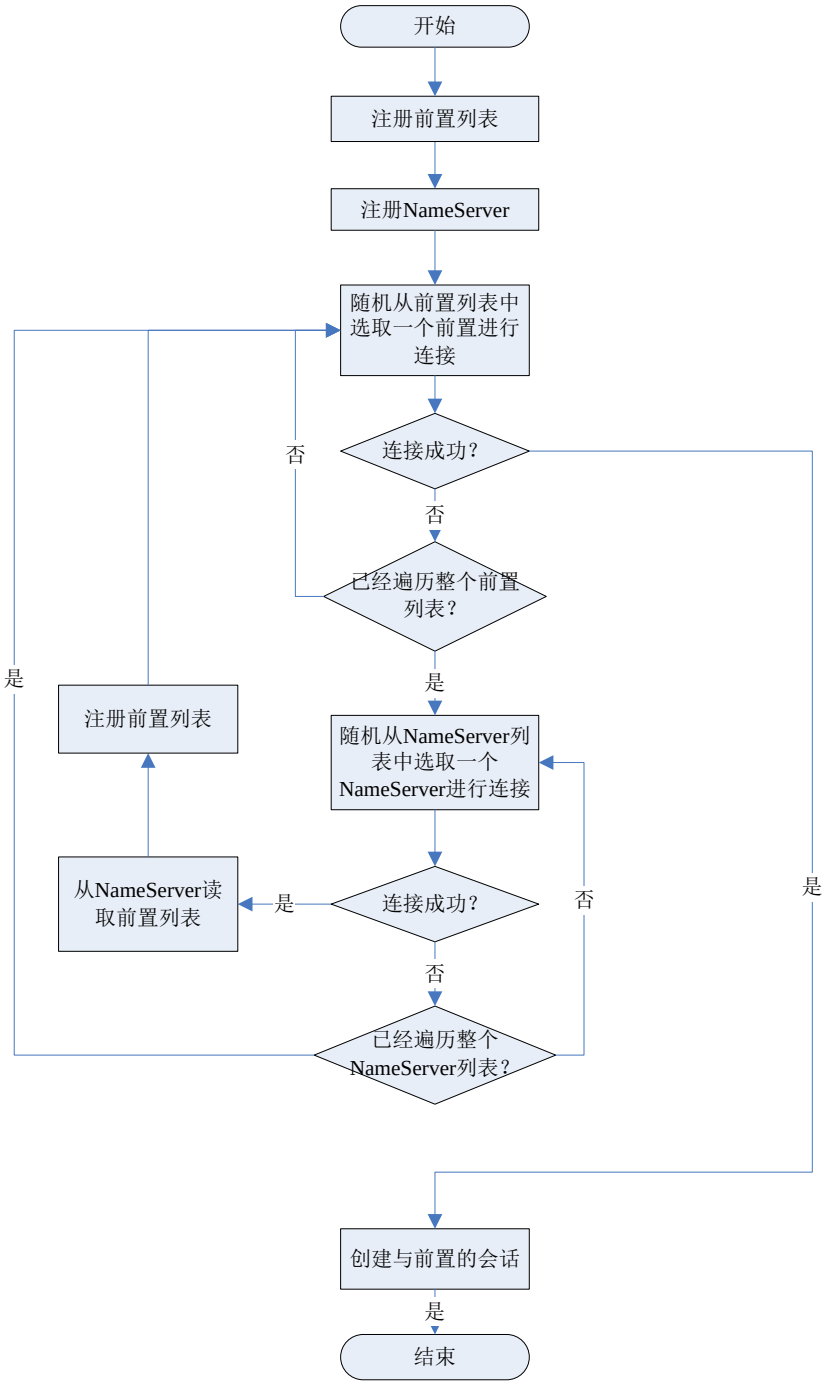
列表，再尝试根据前置列表来连接前置机。

使用 NameServer 的优点在于：

- 增加交易所前置机部署的灵活性，可以根据业务需要和负载短期内新增前置机并无需对会员系统做任何修改。
- NameServer 可以较好地完成主用系统和灾备系统的切换。
- NameServer 功能单一，结构简单，负载也很低，不用考虑负载均衡，可以灵活部署。

会员系统可以同时使用 RegisterFront()方法注册前置机列表和使用 RegisterNameServer()方法注册 NameServer 列表。API 会首先尝试连接已注册的前置，如果连接不成功，再尝试连接 NameServer。

API 连接前置的流程图：


开始
注册前置列表
注册NameServer
随机从前置列表中选取一个前置进行连接
连接成功？
否
否
已经遍历整个前置列表？
是
是
注册前置列表
随机从NameServer列表中选取一个NameServer进行连接
从NameServer读取前置列表
是
连接成功？
否
否
已经遍历整个NameServer列表？
是
创建与前置的会话
是
结束

第5章 MduserAPI 接口分类

5.1 管理接口

MduserAPI 的管理接口是对 API 的生命周期和运行参数进行控制。

接口类型	接口名称	说明
生命周期管理接口	CFfexFtdcMduserApi:: CreateFtdcMduserApi	指定行情接收模式，创建 MduserApi 实例
	CFfexFtdcMduserApi::GetVersion	获取 API 版本
	CFfexFtdcMduserApi::GetTradingDay	获取交易日
	CFfexFtdcMduserApi:: Release	删除接口实例
	CFfexFtdcMduserApi:: Init	初始化
	CFfexFtdcMduserApi:: Join	等待接口线程结束运行
参数管理接口	CFfexFtdcMduserApi:: RegisterSpi	注册回调接口
	CFfexFtdcMduserApi:: RegisterFront	注册前置机网络地址
	CFfexFtdcMduserApi:: RegisterNameServer	注册 NameServer 网络地址
	CFfexFtdcMduserApi:: SetHeartbeatTimeout	设置 TCP 心跳超时时间
订阅接口	CFfexFtdcMduserApi:: SubscribeMarketDataTopic	订阅主题行情，绑定组播地址
	CFfexFtdcMduserApi:: SubscribePublicTopic	订阅公共流
通信状态接口	CFfexFtdcMduserSpi:: OnFrontConnected	与交易系统建立起通信连接时（还未登录前），该方法被调用
	CFfexFtdcMduserSpi:: OnFrontDisconnected	与交易系统通信连接断开时，该方法被调用
	CFfexFtdcMduserSpi:: OnHeartBeatWarning	当 TCP 通道长时间未收到报文时，该方法被调用
	CFfexFtdcMduserSpi::OnMultiHeartBeat	组播心跳接口，服务端定时推送组播心跳。
	CFfexFtdcMduserSpi::OnMultiHeartBeatWarning	当组播通道长时间未收到报文时，该方法被调用。
	CFfexFtdcMduserSpi::OnReportEvent	本地事件报告接口

5.2 业务接口

业务类型	业务	请求接口 / 响应接口	数据流
登录	登录	CFfexFtdcMduserApi:: ReqUserLogin CFfexFtdcMduserSpi::OnRspUserLogin	对话流
	登出	CFfexFtdcMduserApi::ReqUserLogout CFfexFtdcMduserSpi::OnRspUserLogout	对话流
订阅	订阅主题	CFfexFtdcMduserApi::ReqSubscribeTopic CFfexFtdcMduserSpi::OnRspSubscribeTopic	对话流
	查询主题	CFfexFtdcMduserApi::ReqQryTopic CFfexFtdcMduserSpi::OnRspQryTopic	查询流
合约	合约通知	CFfexFtdcMduserSpi:: OnRtnInsInstrument	公共流
	合约状态通知	CFfexFtdcMduserSpi:: OnRtnInstrumentStatus	公共流
公告	交易所公告	CFfexFtdcMduserSpi ::OnRtnMarketDataBulletin	公共流
行情	行情通知	CFfexFtdcMduserSpi::OnRtnDepthMarketData	行情流

第6章 MduserAPI 参考手册

行情客户端系统 API 提供了两个接口类，分别为 CFfexFtdcMduserApi 和 CFfexFtdcMduserSpi。

6.1 CFfexFtdcMduserSpi 接口

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

6.1.1 OnFrontConnected 方法

当行情接收系统与行情发布服务器建立起 TCP 虚链路（连接）后，该方法被调用。

函数原型：

```
void OnFrontConnected();
```

注意：OnFrontConnected 被调用仅说明 TCP 连接成功，行情接收系统必须自行登录，才能进行后续的业务操作。登录失败不会回调该方法。

6.1.2 OnFrontDisconnected 方法

当行情接收系统与交易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，行情接收系统可不做处理。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

6.1.3 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

6.1.4 OnMultiHeartBeat 方法

组播心跳接口。行情服务端定时推送组播心跳。

函数原型：

```
void OnMultiHeartBeat(CFfexFtdcMdMultiHeartBeatField *pMultiHeartBeat);
```

参数：

pMultiHeartBeat：组播心跳域

组播心跳域包括组播通道的主题号，组播通道已经接收的最大行情序号，组播心跳产生的时间，组播地址等信息。

```
Struct CFfexFtdcMdMultiHeartBeatField{
        ///行情主题号
        TFfexFtdcMdSequenceSeriesType  TopicID;
        ///最新序号
        TFfexFtdcMdSequenceNoType  LastSeqNo;
        ///当前时间
        TFfexFtdcMdTimeType CurrTime;
        ///组播通道地址
        TFfexFtdcMdMultiIPAddrType  MultiIPAddr;
}
```

6.1.5 OnMultiHeartBeatWarning 方法

组播心跳告警接口，组播通道 10s 收不到任何消息会回调该接口。

函数原型：

```
void OnMultiHeartBeatWarning(const char* MultiIPAddr, int nTimeLapse);
```

参数：

MultiIPAddr：组播通道地址

nTimeLapse：距离上次接收报文的时间

6.1.6 OnReportEvent 方法

本地 API 事件报告接口。工作在混合模式下，如果发生通道切换会回调该接口。

函数原型：

```
void OnReportEvent(const char* EventMsg);
```

参数：

EventMsg： 事件详细信息。

6.1.7 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：

nTopicID：主题代码（如私有流、公共流、行情流等）。

nSequenceNo：报文序号。

6.1.8 OnPackageEnd 方法

报文回调结束通知。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：

nTopicID：主题代码（如私有流、公共流、行情流等）。

nSequenceNo：报文序号。

6.1.9 OnRspUserLogin 方法

当行情接收系统发出登录请求之后，交易系统返回响应时，该方法会被调用，通知行情接收系统登录是否成功。

函数原型：

```
void OnRspUserLogin(
        CFfexFtdcMdRspUserLoginField *pRspUserLogin,
        CFfexFtdcMdRspInfoField *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 CFfexFtdcMdRspUserLoginField
    {
        ///交易日
        TFfexFtdcMdDateType TradingDay;
        ///登录成功时间
        TFfexFtdcMdTimeType LoginTime;
        ///最大本地报单号
        TFfexFtdcMdOrderLocalIDTypeMaxOrderLocalID;
        ///交易用户代码
        TFfexFtdcMdUserIDType   UserID;
        ///会员代码
```

```
        TFfexFtdcMdParticipantIDType    ParticipantID;
        ///交易系统名称
        TFfexFtdcMdTradingSystemNameType    TradingSystemName;
        ///数据中心代码
        TFfexFtdcMdDataCenterIDTypeDataCenterID;
    };
```

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 CFfexFtdcMdRspInfoField
    {
            ///错误代码
            TFfexFtdcMdErrorIDType  ErrorID;
            ///错误信息
            TFfexFtdcMdErrorMsgTypeErrorMsg;
    };
```

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。

6.1.10 OnRspUserLogout 方法

当行情接收系统发出登出请求之后，交易系统返回响应时，该方法会被调用，通知行情接收系统登出是否成功。

函数原型：

```
void OnRspUserLogout(
      CFfexFtdcMdRspUserLogoutField *pRspUserLogout,
      CFfexFtdcMdRspInfoField *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 CFfexFtdcMdRspUserLogoutField
    {
```

```
        ///交易用户代码
        TFfexFtdcMdUserIDType   UserID;
        ///会员代码
        TFfexFtdcMdParticipantIDType    ParticipantID;
    };
```

pRspInfo：返回用户响应信息的地址。

响应信息结构：

```
struct CFfexFtdcMdRspInfoField
    {
            ///错误代码
            TFfexFtdcMdErrorIDType  ErrorID;
            ///错误信息
            TFfexFtdcMdErrorMsgType 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。

6.1.11 OnRspSubscribeTopic 方法

订阅主题应答。当会员系统发出订阅主题指令后，交易系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspSubscribeTopic(
    CFfexFtdcMdDisseminationField *pDissemination,
    CFfexFtdcMdRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

参数：

pDissemination：指向订阅主题结构的地址，包含了要订阅的主题和起始报文的序号。订阅主题结构：

```
    struct CFfexFtdcMdDisseminationField {
        ///序列系列号
        TFfexFtdcMdSequenceSeriesType  SequenceSeries;
```

```
    ///序列号
    TFfexFtdcMdSequenceNoType   SequenceNo;
};
```

pRspInfo：指向响应信息结构的地址。响应信息结构：

```
struct CFfexFtdcMdRspInfoField {
    ///错误代码
    TFfexFtdcMdErrorIDType  ErrorID;
    ///错误信息
    TFfexFtdcMdErrorMsgType ErrorMsg;
};
```

可能出现的错误：

错误代码	错误提示	可能的原因
66	用户尚未登录	尚未登录

nRequestID：返回订阅主题请求的 ID，该 ID 由用户在订阅主题时指定。

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

6.1.12 OnRspQryTopic 方法

查询主题应答。当会员系统发出查询主题指令后，交易系统返回响应时，该方法会被调用。

函数原形：

```
void OnRspQryTopic (
        CFfexFtdcMdDisseminationField *pDissemination,
        CFfexFtdcMdRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

参数：

pDissemination：指向查询主题结构的地址，包含了要查询的主题和该主题报文个数。查询主题结构：

```
struct CFfexFtdcMdDisseminationField {
    ///序列系列号
    TFfexFtdcMdSequenceSeriesType   SequenceSeries;
    ///序列号
    TFfexFtdcMdSequenceNoType   SequenceNo;
};
```

pRspInfo：指向响应信息结构的地址。响应信息结构：

```
struct CFfexFtdcMdRspInfoField {
    ///错误代码
    TFfexFtdcMdErrorIDType  ErrorID;
    ///错误信息
    TFfexFtdcMdErrorMsgType ErrorMsg;
};
```

可能出现的错误：

错误代码	错误提示	可能的原因
66	用户尚未登录	尚未登录

nRequestID：返回查询主题请求的 ID，该 ID 由用户在订阅主题时指定。

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

6.1.13 OnRtnInstrumentStatus 方法

合约交易状态改变回报。当合约交易状态发生变化时，交易系统会主动通知会员系统，该方法会被调用。

函数原型：

```
void OnRtnInstrumentStatus(CFfexFtdcMdInstrumentStatusField *pInstrumentStatus);
```

参数：

pInstrumentStatus：指向合约状态结构的地址。

合约状态结构：

```
struct CFfexFtdcMdInstrumentStatusField
    {
        ///结算组代码
        TFfexFtdcMdSettlementGroupIDType    SettlementGroupID;
        ///合约代码
        TFfexFtdcMdInstrumentIDType InstrumentID;
        ///合约交易状态
        TFfexFtdcMdInstrumentStatusType InstrumentStatus;
        ///交易阶段编号
        TFfexFtdcMdTradingSegmentSNType TradingSegmentSN;
        ///进入本状态时间
        TFfexFtdcMdTimeType EnterTime;
        ///进入本状态原因
        TFfexFtdcMdInstStatusEnterReasonType    EnterReason;
        ///交易时段
        TFfexFtdcMdTradePhaseIDType TradePhaseID;
    };
```

6.1.14 OnRtnInsInstrument 方法

增加合约通知。当系统登录成功后时，交易系统会把系统中的增加的合约通过公共流通知给客户端。

函数原型：

```
void OnRtnInsInstrument(CFfexFtdcMdInstrumentField *pInstrument);
```

参数：

pInstrument：指向合约结构的地址。

合约结构：

```
struct CFfexFtdcMdInstrumentField
    {
        ///结算组代码
        TFfexFtdcMdSettlementGroupIDType    SettlementGroupID;
        ///产品代码
        TFfexFtdcMdProductIDType    ProductID;
        ///产品组代码
        TFfexFtdcMdProductGroupIDType   ProductGroupID;
        ///基础商品代码
        TFfexFtdcMdInstrumentIDType UnderlyingInstrID;
        ///产品类型
        TFfexFtdcMdProductClassType ProductClass;
        ///持仓类型
        TFfexFtdcMdPositionTypeType PositionType;
        ///执行价
        TFfexFtdcMdPriceType    StrikePrice;
        ///期权类型
        TFfexFtdcMdOptionsTypeType  OptionsType;
        ///合约数量乘数
        TFfexFtdcMdVolumeMultipleType   VolumeMultiple;
        ///合约基础商品乘数
        TFfexFtdcMdUnderlyingMultipleType   UnderlyingMultiple;
        ///合约代码
        TFfexFtdcMdInstrumentIDType InstrumentID;
        ///合约名称
        TFfexFtdcMdInstrumentNameType   InstrumentName;
        ///交割年份
        TFfexFtdcMdYearType DeliveryYear;
        ///交割月
        TFfexFtdcMdMonthType    DeliveryMonth;
```

```
        ///提前月份
        TFfexFtdcMdAdvanceMonthType AdvanceMonth;
        ///当前是否交易
        TFfexFtdcMdBoolType IsTrading;
        ///期权系列代码
        TFfexFtdcMdOptionSeriesIDType  OptionSeriesID;
```

6.1.15 OnRtnMarketDataBulletin 方法

公告通知。交易所通过交易系统发布公告时，由交易系统主动通知会员系统，该方法会被调用。

函数原型：

```
void OnRtnMarketDataBulletin(CFfexFtdcMdMarketDataBulletinField *pMarketDataBulletin);
```

参数：

pMarketDataBulletin：指向公告结构的地址。

公告结构：

```
struct CFfexFtdcMdMarketDataBulletinField
    {
        ///交易日
        TFfexFtdcMdDateType TradingDay;
        ///公告编号
        TFfexFtdcMdBulletinIDType  BulletinID;
        ///公告类型
        TFfexFtdcMdNewsTypeType NewsType;
        ///紧急程度
        TFfexFtdcMdNewsUrgencyType  NewsUrgency;
        ///发送时间
        TFfexFtdcMdTimeType SendTime;
        ///消息摘要
        TFfexFtdcMdAbstractType Abstract;
        ///消息正文
    TFfexFtdcMdContentType  Content;
    };
```

6.1.16 OnRtnDepthMarketData 方法

行情通知。当行情发生变化时，交易系统会主动通知行情接收系统，该方法会被调用。

函数原型：

```
voidOnRtnDepthMarketData(CFfexFtdcMdDepthMarketDataField *pDepthMarketData);
```

参数：

pDepthMarketData：返回市场行情信息的地址。注意：行情中部分字段是未使用的。

深度市场行情信息结构：

```
struct CFfexFtdcMdDepthMarketDataField
    {
        ///交易日
        TFfexFtdcMdDateType TradingDay;
        ///结算组代码
        TFfexFtdcMdSettlementGroupIDType   SettlementGroupID;
        ///结算编号
        TFfexFtdcMdSettlementIDType SettlementID;
        ///最新价
        TFfexFtdcMdPriceType    LastPrice;
        ///昨结算
        TFfexFtdcMdPriceType    PreSettlementPrice;
        ///昨收盘
        TFfexFtdcMdPriceType    PreClosePrice;
        ///昨持仓量
        TFfexFtdcMdLargeVolumeType  PreOpenInterest;
        ///今开盘
        TFfexFtdcMdPriceType    OpenPrice;
        ///最高价
        TFfexFtdcMdPriceType    HighestPrice;
        ///最低价
        TFfexFtdcMdPriceType    LowestPrice;
        ///数量
        TFfexFtdcMdVolumeType   Volume;
        ///成交金额
        TFfexFtdcMdMoneyType    Turnover;
        ///持仓量
        TFfexFtdcMdLargeVolumeType  OpenInterest;
        ///今收盘
        TFfexFtdcMdPriceType    ClosePrice;
        ///今结算
        TFfexFtdcMdPriceType    SettlementPrice;
        ///涨停板价
        TFfexFtdcMdPriceType    UpperLimitPrice;
        ///跌停板价
```

```
        TFfexFtdcMdPriceType    LowerLimitPrice;
        ///昨虚实度
        TFfexFtdcMdRatioType    PreDelta;
        ///今虚实度
        TFfexFtdcMdRatioType    CurrDelta;
        ///最后修改时间
        TFfexFtdcMdTimeType UpdateTime;
        ///最后修改毫秒
        TFfexFtdcMdMillisecType UpdateMillisec;
        ///合约代码
        TFfexFtdcMdInstrumentIDType InstrumentID;
        ///申买价一
        TFfexFtdcMdPriceType    BidPrice1;
        ///申买量一
        TFfexFtdcMdVolumeType   BidVolume1;
        ///申卖价一
        TFfexFtdcMdPriceType    AskPrice1;
        ///申卖量一
        TFfexFtdcMdVolumeType   AskVolume1;
        ///申买价二
        TFfexFtdcMdPriceType    BidPrice2;
        ///申买量二
        TFfexFtdcMdVolumeType   BidVolume2;
        ///申卖价二
        TFfexFtdcMdPriceType    AskPrice2;
        ///申卖量二
        TFfexFtdcMdVolumeType   AskVolume2;
        ///申买价三
        TFfexFtdcMdPriceType    BidPrice3;
        ///申买量三
        TFfexFtdcMdVolumeType   BidVolume3;
        ///申卖价三
        TFfexFtdcMdPriceType    AskPrice3;
        ///申卖量三
        TFfexFtdcMdVolumeType   AskVolume3;
        ///申买价四
        TFfexFtdcMdPriceType    BidPrice4;
        ///申买量四
        TFfexFtdcMdVolumeType   BidVolume4;
        ///申卖价四
        TFfexFtdcMdPriceType    AskPrice4;
        ///申卖量四
        TFfexFtdcMdVolumeType   AskVolume4;
        ///申买价五
```

```
        TFfexFtdcMdPriceType    BidPrice5;
        ///申买量五
        TFfexFtdcMdVolumeType   BidVolume5;
        ///申卖价五
        TFfexFtdcMdPriceType    AskPrice5;
        ///申卖量五
        TFfexFtdcMdVolumeType   AskVolume5;
        ///上带价
        TFfexFtdcMdPriceType    BandingUpperPrice;
        ///下带价
        TFfexFtdcMdPriceType    BandingLowerPrice;
    };
```

6.1.17 OnRspError 方法

针对用户请求的出错通知。

函数原型：

```
void OnRspError(
       CFfexFtdcMdRspInfoField *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 CFfexFtdcMdRspInfoField
    {
            ///错误代码
            TFfexFtdcMdErrorIDType  ErrorID;
            ///错误信息
            TFfexFtdcMdErrorMsgType 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。

6.2 CFfexFtdcMduserApi 接口

CFfexFtdcMduserApi 接口提供给用户的功能包括：登入/登出、地址注册、

行情主题订阅、组播地址绑定、行情查询、行情订阅等功能。

6.2.1 CreateFtdcMduserApi 方法

设置 API 行情接收模式，同时产生一个 CFfexFtdcMduserApi 的一个实例，不能通过 new 来产生。

函数原型：

```
static CFfexFtdcMduserApi *CreateFtdcMduserApi(const char *pszFlowPath = "" ,
MD_RUN_MODE nMdRunMode = MDRM_TCP);
```

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。

nMdRunMode：行情接收模式

- MDRM_TCP：TCP 模式，只接收 TCP 行情流。
- MDRM_MIX：混合模式，根据网络情况自动切换 TCP 和组播

返回值：

返回一个指向 CFfexFtdcMduserApi 实例的指针。

6.2.2 GetVersion 方法

获取 API 版本号。

函数原型：

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

参数：

nMajorVersion：返回主版本号

nMajorVersion：返回次版本号

返回值：

返回一个指向版本标识字符串的常量指针。

6.2.3 Release 方法

释放一个 CFfexFtdcMduserApi 实例。不能使用 delete 方法。

函数原型：

```
void Release();
```

Release 方法会完全释放 API 实例，调用 Release 方法后，如果需要继续使用 API，必须重新通过 CreateFtdcMduserApi 来创建新的 API 实例。

6.2.4 Init 方法

使行情接收系统开始与交易系统建立连接，连接成功后可以进行登陆。

函数原型：

```
void Init();
```

6.2.5 Join 方法

行情接收系统等待一个接口实例线程的结束。

函数原型：

```
void Join();
```

6.2.6 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。

6.2.7 RegisterSpi 方法

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

理。

函数原型：

```
void RegisterSpi(CFfexFtdcMduserSpi *pSpi) ;
```

参数：

pSpi：实现了 CFfexFtdcMduserSpi 接口的实例指针。

6.2.8 RegisterFront 方法

设置交易所行情前置系统的网络通讯地址。交易系统拥有多个行情前置系统，用户可以同时注册多个行情前置系统的网络通讯地址。

该方法要在 Init 方法之前调用。

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。

6.2.9 RegisterNameServer 方法

设置交易所 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：指向交易所 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。

6.2.10 SetHeartbeatTimeout 方法

设置网络通信心跳的超时时间。当 MduserAPI 与交易系统的 TCP 连接建立后，连接会定时发送心跳，用以检测连接是否正常。该方法用于设置检测心跳超时的时间。**交易所建议会员系统将 timeout 值设置为 10 至 30 秒之间。**

函数原型：

```
virtual void SetHeartbeatTimeout(unsigned int timeout);
```

参数：

timeout：心跳超时时间（秒）。若超过 timeout/2 秒未收到交易系统的任何信息时，将触发 CFfexFtdcMduserApi::OnHeartBeatWarning()。若超过 timeout 秒未收到交易系统的任何信息时，连接将会中断，将触发 CFfexFtdcMduserApi ::OnFrontDisconnected()。

6.2.11 SubscribeMarketDataTopic 方法

柜台系统根据交易所分配的主题权限订阅主题行情。pszMultiAddr 默认为 NULL，如果使用 TCP 模式，pszMultiAddr 参数会被忽略，如果使用混合模式，该地址必须配置，否则 API 会自动退出。

函数原型：

```
void SubscribeMarketDataTopic(int nTopicID, MD_TE_RESUME_TYPE nResumeType, char
*pszMultiAddr = NULL);
```

参数：

nTopicID：代表深度行情的主题，由交易所公布。

nResumeType：市场行情重传方式

- MD_TERT_RESTART:从本交易日开始重传
- MD_TERT_RESUME:从上次收到的续传
- MD_TERT_QUICK:先传送当前行情快照,再传送登录后市场行情的内容。交易所建议会员使用此方式快速恢复行情。

pszMultiAddr：多播地址相关信息，地址格式举例为："multi://172.31.112.116@42.24.2.35@224.0.0.2:22305"。"172.31.112.116"代表接

收组播行情的本地网卡地址，**该网卡必须存在，否则进程会直接退出**。“42.24.2.35”代表组播源地址，“224.0.0.2:22305”代表组播组地址和端口，**组播源地址和组播组地址由交易所统一对外公布。**

6.2.12 ReqUserLogin 方法

用户发出登陆请求。

函数原型：

```
int ReqUserLogin(
     CFfexFtdcMdReqUserLoginField *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 CFfexFtdcMdReqUserLoginField
    {
        ///交易日
        TFfexFtdcMdDateType TradingDay;
        ///交易用户代码
        TFfexFtdcMdUserIDType  UserID;
        ///会员代码
        TFfexFtdcMdParticipantIDType   ParticipantID;
        ///密码
        TFfexFtdcMdPasswordType Password;
        ///用户端产品信息
        TFfexFtdcMdProductInfoType  UserProductInfo;
        ///接口端产品信息，未使用
        TFfexFtdcMdProductInfoType  InterfaceProductInfo;
        ///协议信息，未使用
        TFfexFtdcMdProtocolInfoType ProtocolInfo;
        ///数据中心代码
        TFfexFtdcMdDataCenterIDType DataCenterID;

    };
```

用户需要填写 UserProductInfo 字段，即行情接收系统的产品信息，如软件开发商、版本号等，例如：”Ffex Mduser V100”代表中国金融期货交易所开发的行情接收程序和版本号。

如果是本交易日第一次登录，DataCenterID 可以填 0 或交易所公布的主数据中心代码，以后再登录要填写登录应答返回的 DataCenterID。

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。
- -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。

6.2.13 ReqUserLogout 方法

用户发出登出请求。

函数原型：

```
int ReqUserLogout(
        CFfexFtdcMdReqUserLogoutField *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 CFfexFtdcMdReqUserLogoutField
{
    ///交易用户代码
    TFfexFtdcMdUserIDType   UserID;
    ///会员代码
    TFfexFtdcMdParticipantIDType    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。
- -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。

6.2.14 ReqSubscribeTopic 方法

订阅主题请求。应当在登录完成后调用。

函数原型：

```
int ReqSubscribeTopic (
    CFfexFtdcMdDisseminationField * pDissemination,
    int nRequestID);
```

参数：

pDissemination：指向订阅主题结构的地址，包含了要订阅的主题和起始报文的序号。订阅主题结构：

```
struct CFfexFtdcMdDisseminationField {
    ///序列系列号
    TFfexFtdcMdSequenceSeriesType  SequenceSeries;
    ///序列号
    TFfexFtdcMdSequenceNoType  SequenceNo;
};
SequenceSeries：要订阅的主题
SequenceNo：=-1 表示使用快照方式，其它值表示从该序号开始续传
```

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。

6.2.15 ReqQryTopic 方法

查询主题请求。应当在登录完成后调用。

函数原形：

```
int ReqQryTopic (
    CFfexFtdcMdDisseminationField * pDissemination,
    int nRequestID);
```

参数：

pDissemination：指向查询主题结构的地址，包含了要查询的主题。订阅主题结构：

```
struct CFfexFtdcMdDisseminationField {
    ///序列系列号
    TFfexFtdcMdSequenceSeriesType  SequenceSeries;
    ///序列号
    TFfexFtdcMdSequenceNoType  SequenceNo;
};
SequenceSeries：要查询的主题
SequenceNo：不用填写
```

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。

第7章 开发示例

以下为行情 API 开发示例，介绍 CFfexFtdcMduserApi 和 CFfexFtdcMduserSpi 接口的使用，可以在 linux 环境下直接编译执行。

示例一：TCP 模式

```
// TCP 模式
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
#include "CFFEXFtdcMduserApi.h"

class CSimpleHandler : public CFfexFtdcMduserSpi
{
public:
	// 构造函数，需要一个有效的指向CFfexFtdcMduserApi实例的指针
	CSimpleHandler(CFfexFtdcMduserApi *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() {
		CFfexFtdcMdReqUserLoginField reqUserLogin;
		strcpy(reqUserLogin.TradingDay, m_pUserApi->GetTradingDay());
		strcpy(reqUserLogin.ParticipantID, "P001");
		strcpy(reqUserLogin.UserID, "U001");
		strcpy(reqUserLogin.Password, "P001");

		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(CFfexFtdcMdRspUserLoginField *pRspUserLogin, CFfexFtdcMdRspInfoField
*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) {
                // 登录失败，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);
            }
        }

        // 深度行情通知，行情服务器会主动通知客户端
        void OnRtnDepthMarketData(CFfexFtdcMdDepthMarketDataField *pMarketData) {
            // 客户端按需处理返回的数据
        }

        // 针对用户请求的出错通知
        void OnRspError(CFfexFtdcMdRspInfoField *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
        }

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

int main()
{
        // 产生一个CFfexFtdcMduserApi实例，默认TCP模式
        CFfexFtdcMduserApi *pUserApi = CFfexFtdcMduserApi::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
        ///            MD_TERT_RESTART:从本交易日开始重传
        ///            MD_TERT_RESUME:从上次收到的续传
        ///            MD_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 (102, MD_TERT_RESUME);
        //设置心跳超时时间
        pUserApi->SetHeartbeatTimeout(30);
```

```
        // 设置交易系统域名服务NameServer的地址
        pUserApi->RegisterNameServer("tcp://172.31.197.1:60125");
        // 设置行情前置的地址
        // pUserApi->RegisterFront("tcp://172.31.197.1:60125");
        // 使客户端开始与行情发布服务器建立连接
        pUserApi->Init();

        // 等待行情接收
        pUserApi->Join();

        return 0;
}
```

示例二：混合模式

```
// 混合模式
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
#include "CFFEXFtdcMduserApi.h"

class CSimpleHandler : public CFfexFtdcMduserSpi
{
public:
        // 构造函数，需要一个有效的指向 CFfexFtdcMduserApi 实例的指针
        CSimpleHandler(CFfexFtdcMduserApi *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() {
                CFfexFtdcMdReqUserLoginField reqUserLogin;
                strcpy(reqUserLogin.TradingDay, m_pUserApi->GetTradingDay());
                strcpy(reqUserLogin.ParticipantID, "P001");
                strcpy(reqUserLogin.UserID, "U001");
                strcpy(reqUserLogin.Password, "P001");

                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(CFfexFtdcMdRspUserLoginField *pRspUserLogin, CFfexFtdcMdRspInfoField *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) {
            // 登录失败，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);
        }
    }

    // 深度行情通知，行情服务器会主动通知客户端
    void OnRtnDepthMarketData(CFfexFtdcMdDepthMarketDataField *pMarketData) {
        // 客户端按需处理返回的数据
    }

    // 针对用户请求的出错通知
    void OnRspError(CFfexFtdcMdRspInfoField *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
    }

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

int main()
{
    // 产生一个 CFfexFtdcMduserApi 实例，混合模式
    CFfexFtdcMduserApi *pUserApi = CFfexFtdcMduserApi::CreateFtdcMduserApi("",MDRM_MIX);
    // 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
```

```
        ///            MD_TERT_RESTART:从本交易日开始重传
        ///            MD_TERT_RESUME:从上次收到的续传
        ///            MD_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 (102,
MD_TERT_RESUME,"multi://172.31.197.1@172.31.197.2@224.0.1.1:38888");
        //设置心跳超时时间
        pUserApi->SetHeartbeatTimeout(30);
        // 设置交易系统域名服务 NameServer 的地址
        pUserApi->RegisterNameServer("tcp://172.31.197.1:60125");
        // 设置行情前置的地址
        // pUserApi->RegisterFront("tcp://172.31.197.1:60125");
        // 使客户端开始与行情发布服务器建立连接
        pUserApi->Init();

        // 等待行情接收
        pUserApi->Join();

        return 0;
}
```