DataFeed
API 接口文档

Dalian Futures Information
Technology Co., Ltd.

文档信息

项目名称	Level 2 行情系统项目
文档名称	《level 2 行情系统应用程序接口说明》
文档版本号	<v1.1.8>
作者	Level 2 行情系统开发组

文档修订历史

版本	日期	修订内容	修订者
1.0.0	08,28,2012	创建	Level 2 行情系统开发组
1.0.1	10,15,2012	加入心跳内容，加入新报文	Level 2 行情系统开发组
1.0.2	02,18,2013	加入期权字段	Level 2 行情系统开发组
1.0.3	07.24.2013	修复非交易阶段无限重连等 bug	Level 2 行情系统开发组
1.0.4	08.14.2013	增加新回调接口，解决已知 bug	Level 2 行情系统开发组
1.0.5	10.25.2013	接口使用自定义类型，减少与服务器断开相应时间	Level 2 行情系统开发组
1.0.6	03.14.2014	变更结构体	Level 2 行情系统开发组
1.0.7	03.17.2015	增加组播接收行情模式	Level 2 行情系统开发组
1.0.8	08.01.2016	增加 OnDisconnected 断开原因	Level 2 行情系统开发组
1.1.4	05.26.2017	修改版本号	Level 2 行情系统开发组
1.1.5	06.12.2017	spi 回调类增加备注	Level 2 行情系统开发组
1.1.6	10.18.2017	增加常见错误码	Level 2 行情系统开发组
1.1.7	07.02.2018	增加行情序列号，部分接口变化	Level 2 行情系统开发组
1.1.8	31.08.2018	增加针对组播模式下网络调优的说明	Level 2 行情系统开发组

目录

一、 简介

大连飞创深度行情(level2)应用程序接口（API）是一个基于C++的类库，通过使用和扩展类库提供的功能来实现相关的深度行情获取功能，行情包括5级深度委托行情、最佳买卖价位上前10笔分笔委托量、加权平均委买价格、加权平均委卖价格、委托买总量、委托卖总量、实时结算价以及分价位成交量统计等。

本文档的主要内容包括：

- 接口类库文件说明
- 接口的系统架构
- 接口开发规范
- 业务与接口的对照表
- 接口定义
- 接口使用时的一些注意事项

二、 接口类库文件说明

接口类库包含如下文件：

文件名	版本	文件描述
DFITCL2ApiDataType.h	v1.1.6.5	定义接口所需的接口数据类型的头文件
DFITCL2Api.h	v1.1.6.5	定义行情接口的头文件
level2Api.dll	v1.1.6.5	行情接口动态链接库二进制文件
level2Api.lib	v1.1.6.5	行情接口导入库文件
liblevel2Api.so	v1.1.6.5	行情接口动态链接库二进制文件(linux)
logging.properties	v1.1.6.5	日志系统配置文件

支持windows 32（Microsoft VisualC++6.0），linux 32/ 64集成开发环境。

三、 系统架构

(一)、 接口与其他系统关系

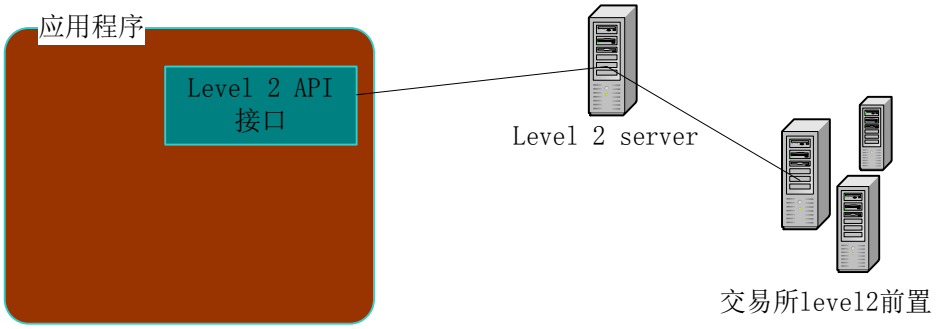

图 1 Level2API 与其他系统的关系

说明：

本API接口用于应用程序获取大连商品交易所level2深度行情；

- 应用程序可以是人工交易客户端或者程序化交易平台；
- 交易API接口负责与level2 server连接。

(二)、 通讯模式

API与level 2 server的通讯协议是建立在TCP/UDP协议上的通讯协议，其中：

- 控制信息只能通过TCP传输，此TCP连接为长连接；
- 行情信息可以通过TCP/UDP/组播三种方式获得，API中的传输方式选择只针对行情接收方式。

四、 接口开发规范

(一)、 开发流程

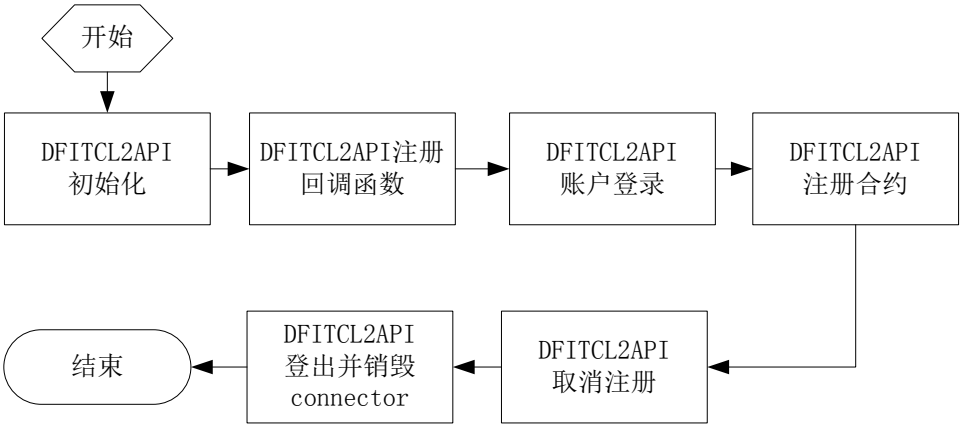

图 2 Level2API 开发流程

说明：

使用windows api时，编译时需要level2Api.lib, 运行时需要level2Api.dll。

使用linux api时，编译和运行时都需要指定liblevel2Api.so的路径。

在centos 6.5 32/64位，举例：

g++ -L 来指定编译时liblevel2Api.so的路径；

export LD_LIBRARY_PATH指定运行时liblevel2Api.so的路径，或者将其加入/usr/lib (或/usr/lib64)。

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。

五、 DFITC_L2::DFITCL2Api 使用说明

(一)、 CreateDFITCL2Api 方法

NEW_CONNECTOR（）

该方法产生一个行情API实例。

函数原型：

```
static DFITCL2Api *CreateDFITCL2Api();
```

成功：返回一个指向类型为DFITC_L2::DFITCL2Api的对象的指针。

失败：返回空指针。

(二)、 DestoryDFITCL2Api 方法

该方法销毁一个api实例。

函数原型：

```
static int DestoryDFITCL2Api(DFITCL2Api * & L2Api);
```

参数：

L2Api：指向api实例的指针。

返回值：

成功：返回0

失败：返回非0值

(三)、 Connect 方法

该方法进行连接level 2 server操作，注册回调函数集。

函数原型：

```
virtual int Connect(char * pszSvrAddr, DFITCL2Spi *pSpi, unsigned int RecvQuoteType)
```

参数：

pszSvrAddr：level2服务器地址；

pSpi：回调函数集指针；

RecvQuoteType：接收行情方式，0为udp，1为tcp，2为组播。

返回值：

0：连接到level 2 server成功；

非0：连接到level 2 server失败。

(四)、 ReqUserLogin 方法

用户发出请求登录。

函数原型：

```
int ReqUserLogin(struct DFITCUserLoginField *pUserLoginData);
```

参数：

pUserLoginData：指向用户登录请求结构的地址。

用户请求登录结构：

```
struct DFITCReqUserLoginField
{
    long lRequestID;                    //请求id
    DFITCAccountIDType accountID;       //资金帐号
    DFITCPasswdType passwd;             //密码
}
```

返回值：

0：发送登录请求成功；

非0：发送登录请求失败。

(五)、 ReqUserLogout 方法

用户发出退出请求。

函数原型：

```
int ReqUserLogout( struct DFITCUserLogoutField *pUserLogoutData );
```

参数：

pUserLogoutData：指向用户退出请求结构的地址。

用户请求退出结构：

```
Struct DFITCUserLogoutField
{
    long lRequestID;                    //请求ID
```

```
    DFITCAccountIDType    accountID;          //资金帐号
};
```

返回值：

0：发送登出请求成功；

非0：发送登出请求失败。

(六)、 SubscribeMarketData 方法

该方法发出订阅某个/某些合约行情请求。组播模式下不支持订阅指定行情。

函数原型：

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

参数：

ppInstrument：指针数组，每个指针指向一个合约；

nCount：订阅合约个数。

注： 1、nCount必须等于ppInstrument中的指针个数

(七)、 UnSubscribeMarketData 方法

该方法发出退订某个/某些合约行情请求。组播模式下不支持退订合约行情。

函数原型：

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

参数：

同上。

(八)、 SubscribeAll 方法

该方法发出订阅全部行情。组播模式下收到的行情即为全部合约的行情，不需再进行订阅全部行情操作。

函数原型：

```
virtual int SubscribeAll();
```

参数：

无

返回值：

0：成功

非0：失败

(九)、 UnSubscribeAll 方法

该方法发出解除全部行情订阅。组播模式下支持解除全部行情订阅。

函数原型：

```
virtual int UnSubscribeAll();
```

参数：

无

返回值：

0：成功

非0：失败

(十)、 ReqChangePassword 方法

函数原型：

```
virtual int ReqChangePassword(DFITCPasswdChangeField * pReqUserPasswdChangeField )
```

参数：

pReqUserPasswdChangeField：指向用户修改密码请求结构的地址。

用户请求登录结构：

```
struct DFITCPasswdChangeField
{
    long lRequestID;
    char AccountID[30];    // 帐户名称
    char OldPassword[30];  // 当前密码
```

```
    char NewPassword[30]; //  新密码
};
```

六、 DFITCL2Spi 使用说明

使用者要新设计一个类继承自class DFITCL2Spi，并实现其全部回调函数集接口成员函数。下图为应用程序使用API简要流程。

注意：如果回调函数耗时过长，需要将从回调函数中取得的行情信息放入一个缓冲区中处理，不然udp接收行情有可能产生行情丢失，tcp接收行情有可能产生行情滞后。请注意，行情回调接口和其他回调接口不是线程安全的。

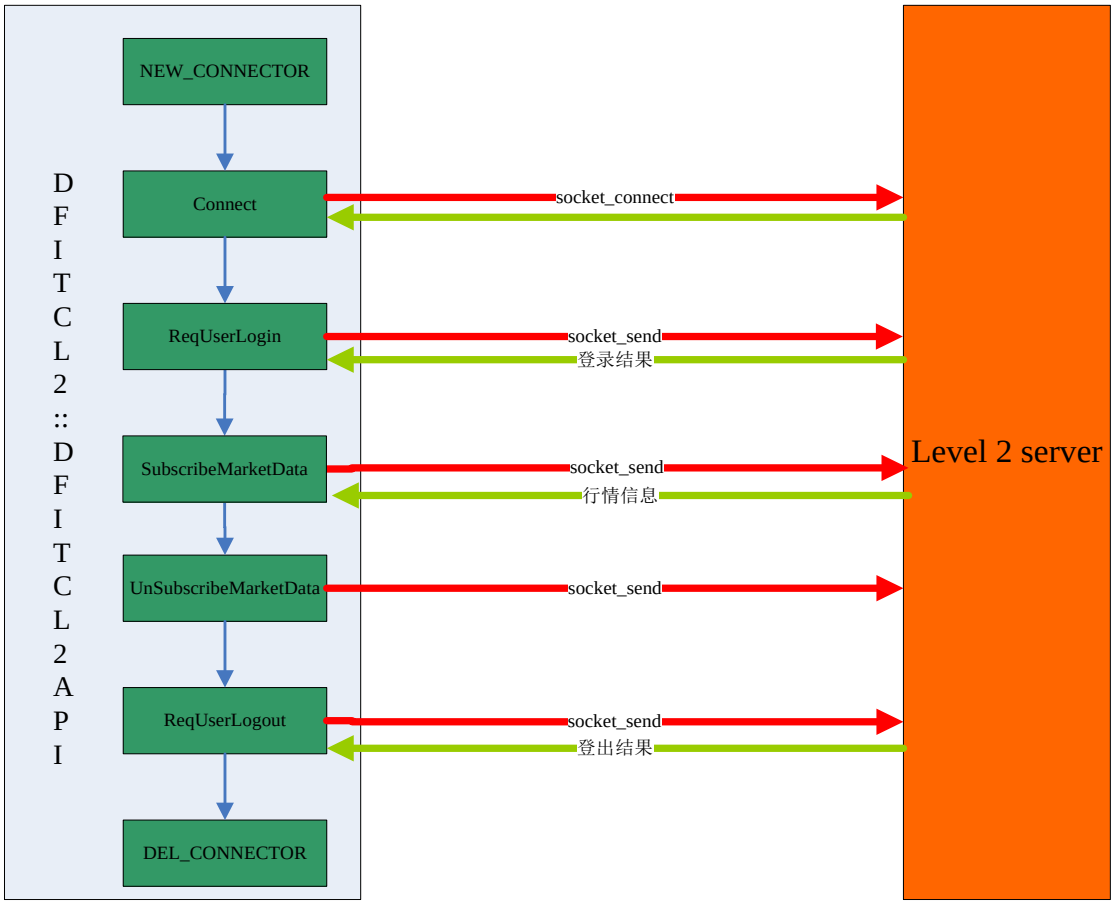

图 3 Level2 API 调用简要流程

(一)、 OnRspUserLogin 方法

当应用程序调用ReqUserLogin方法时，OnRspUserLogin会被调用，通知用户登录

是否成功，登录错误时，会返回登录错误原因在ErrorFiled.ErrMsg中。

函数原型：

```
virtual void OnRspUserLogin(struct ErrorRtnFiled * pErrorFiled)
```

参数： 参数一，登录返回信息

```
struct ErrorRtnFiled
{
    long LocalOrderID;
    int   ErrorID;            // 0为登录成功，非0为登录失败
    char ErrorMsg[128];   //  登录失败时的失败原因信息
};
```

(二)、 OnRspUserLogout 方法

当应用程序调用ReqUserLogout方法时，OnRspUserLogout会被调用，通知用户登出是否成功，登出错误时，会返回登录错误原因在ErrorFiled.ErrMsg中。

通知用户退出状态。

函数原型：

```
virtual void OnRspUserLogout(struct ErrorRtnFiled * pErrorFiled)
```

参数：参数一，登出返回信息

```
struct ErrorRtnFiled
{
    long LocalOrderID;
    int   ErrorID; // 0为登出成功，非0为登录失败
    char ErrorMsg[128]; //  登出失败时的失败原因信息
};
```

(三)、 OnRspSubscribeMarketData 方法

当应用程序调用SubscribeMarketData方法时，OnRspSubscribeMarketData会被调用，通知用户订阅是否成功，订阅错误时，会返回登录错误原因在

ErrorFiled.ErrMsg中。

函数原型：

```
virtual void OnRspSubscribeMarketData(struct ErrorRtnFiled * pErrorFiled)
```

参数： 参数一，订阅返回信息

```
struct ErrorRtnFiled
{
    long LocalOrderID;
    int   ErrorID; // 0为订阅成功，非0为订阅失败
    char ErrorMsg[128]; //  订阅失败时的失败原因信息
};
```

(四)、 OnRspUnSubscribeMarketData 方法

当应用程序调用UnSubscribeMarketData方法时，OnRspUnSubscribeMarketData会被调用，通知用户取消订阅是否成功。

函数原型：

```
virtual void OnRspUnSubscribeMarketData(struct ErrorRtnFiled * pErrorFiled)
```

(五)、 OnRspSubscribeAll 方法

当应用程序调用SubscribeAll方法时，OnRspSubscribeAll会被调用，通知用户全部订阅是否成功，全部订阅错误时，会返回全部订阅错误原因在ErrorFiled.ErrMsg中。

函数原型：

```
virtual void OnRspSubscribeAll(struct ErrorRtnFiled * pErrorFiled)
```

参数： 参数一，登录返回信息

```
struct ErrorRtnFiled
{
    long LocalOrderID;
```

```
    int    ErrorID; // 0为全部订阅成功，非0为失败
    char ErrorMsg[128]; //  全部订阅失败时的失败原因信息
};
```

(六)、 OnRspUnSubscribeAll 方法

当应用程序调用ReqUnSubscribeAll方法时，OnRspUnSubscribeAll会被调用，通知用户取消全部订阅是否成功。

函数原型：

```
virtual void OnRspUnSubscribeAll(struct ErrorRtnFiled * pErrorFiled)
```

(七)、 OnBestAndDeep 方法

当api接收到最优与深度行情时，该方法会被调用，通知应用程序基本行情与五档深度行情信息。

函数原型：

```
virtual void OnBestAndDeep(MDBestAndDeep * const pQuote,UINT4 SequenceNo)
```

参数：参数一，基本行情，五档行情

参数二，基本行情/五档行情序列号

(八)、 OnTenEntrust 方法

当api接收到最优价十笔委托行情时，该方法会被调用，通知应用程序最优价十笔委托行情信息。

函数原型：

```
virtual void OnTenEntrust(MDTenEntrust * const pQuote,UINT4 SequenceNo)
```

参数：参数一，最优价十笔委托行情

参数二，最优价十笔委托行情序列号

(九)、 OnRealtime 方法

当api接收到实时结算价行情时，该方法会被调用，通知应用程序实时结算价行情信息。

函数原型：

```
virtual void OnRealtime(MDRealTimePrice * const pQuote, UINT4 SequenceNo)
```

参数：参数一，实时结算价行情

参数二，实时结算价行情序列号

(十)、 OnOrderStatistic 方法

当api接收到委托统计行情域，加权平均及委托总量行情时，该方法会被调用，通知应用程序该行情信息。

函数原型：

```
virtual void OnOrderStatistic(MDOrderStatistic * const pQuote, UINT4 SequenceNo)
```

参数：参数一，委托统计行情域

参数二，委托统计行情序列号

(十一)、 OnMarchPrice 方法

当api接收到分价位成交行情时，该方法会被调用，通知应用程序分价位成交行情信息。

函数原型：

```
virtual void OnMarchPrice(MDMarchPriceQty * const pQuote, UINT4 SequenceNo)
```

参数：参数一，分价位成交行情

参数二，分价位成交行情序列号

(十二)、 OnRspModifyPassword 方法

当应用程序调用ReqModifyPassword方法时，OnRspModifyPassword会被调用，通

知用户修改密码是否成功，修改密码失败时，会返回修改密码错误原因在ErrorFiled.ErrMsg中。

函数原型：

```
virtual void OnRspModifyPassword(struct ErrorRtnFiled * pErrorFiled)
```

参数：参数一，登出返回信息

```
struct ErrorRtnFiled
{
    long LocalOrderID;
    int    ErrorID; // 0为修改密码成功，非0为登录失败
    char ErrorMsg[128]; //  修改密码失败时的失败原因信息
};
```

(十三)、 OnHeartBeatLost 方法

当应用程序接收不到从level 2 server发来的心跳包时，OnHeartBeatLost会被调用，通知用户心跳丢失。

函数原型：

```
virtual void OnHeartBeatLost() { }
```

(十四)、 OnConnected 方法

当应用程序连接level 2 server成功后，该方法被调用，通知用户连接成功。

函数原型：

```
virtual void OnConnected() { }
```

(十五)、 OnDisconnected 方法

当应用程序与level 2 server失去连接后，该方法被调用，通知用户失去连接。

函数原型：

```
virtual void OnDisconnected(int pReason) { }
```

参数：参数一，断开原因代码

1 跳超时

2 网络断开

3 收到错误报文

4 组播行情模式下，销毁实例时通知客户

5 发送报文过长

七、 日志系统使用说明

日志系统是为了方便定位问题所在，如果需要日志，请按下面的方法操作；如果不需要，请忽略。

(一)、 日志部署方法

将logging.properties放在conf目录下，conf目录在程序的上级目录，具体位置如下：

conf:　logging.properties

Release(可以为任何名字):　程序

(二)、 日志配置方法

在conf/logging.properties中有如下配置项：

1. **log4j.logger.level2Server=DEBUG, B**

这一行的作用是调整输入级别，红色字体代表debug日志输出级别，如果想让日志输出量变小可改为INFO.

2. **#log4j.appender.B=org.apache.log4j.RollingFileAppender**

这一行行首的‘#’代表注释的意思，这一行的意义是将日志都生成到一个文件中

3. **log4j.appender.B=org.apache.log4j.DailyRollingFileAppender**

这一行的意义是将日志按天存储

4. **log4j.appender.B.File=/log/level2Server.log**

这一行的红色字体为日志输出路径，用户可以自行配置路径，建议使用绝对路径。

5. **#log4j.appender.B.MaxFileSize=50000KB**
6. **#log4j.appender.B.MaxBackupIndex=10**

以上两行联合第二行有作用(2、5、6行)，这样每一个文件50000KB可以达到，共生成10，如果是个文件都达到上限则最早的日志会被覆盖。

7. log4j.appender.B.DatePattern='.'yyyy-MM-dd
8. log4j.appender.B.layout=org.apache.log4j.PatternLayout
9. log4j.appender.B.layout.ConversionPattern=%d{ISO8601} %5p %-11c{1} - %m%n
注：实现每天生成一个文件的方法：注释掉2、5、6 行，打开其他行

(三)、 日志生成路径

日志文件默认保存在当前程序路径的/log/下，可以根据需求修改conf/logging.properties文件中如下行：log4j.appender.B.File=/log/level2Server.log，‘=’后面即为日志生成路径。用户可根据自己的需求进行修改。

(四)、 日志输出级别

日志输出级别分为三种

1) DEBUG：调试日志输出，用于调试程序以及查找错误是使用，DEBUG模式输出日志大，占用空间，但比INFO模式更利于定位问题所在
2) INFO：用于生成环境使用（建议部署后日志改为info输出模式）
3) ERROR：只输出错误信息

(五)、 日志配置示例

```
log4j.logger.recordlevel2quot=DEBUG, B
log4j.appender.B=org.apache.log4j.RollingFileAppender
log4j.appender.B.File=log/recordlevel2quot.log
log4j.appender.B.MaxFileSize=512MB
log4j.appender.B.MaxBackupIndex=10
log4j.appender.B.DatePattern='.'yyyy-MM-dd
log4j.appender.B.layout=org.apache.log4j.PatternLayout
log4j.appender.B.layout.ConversionPattern=%d{ISO8601} %5p %-11c{1} - %m%n

log4j.logger.level_2_api=DEBUG, N
log4j.appender.N=org.apache.log4j.DailyRollingFileAppender
log4j.appender.N.File=../log/level_2_api_debug.log
log4j.appender.N.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.N.layout=org.apache.log4j.PatternLayout
log4j.appender.N.layout.ConversionPattern=%d{ISO8601} %5p %-11c{1} - %m%n

log4j.logger.level_2_api_debug=INFO, M
```

```
log4j.appender.M=org.apache.log4j.DailyRollingFileAppender
log4j.appender.M.File=../log/level_2_api_debug.log
log4j.appender.M.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.M.layout=org.apache.log4j.PatternLayout
log4j.appender.M.layout.ConversionPattern=%d{ISO8601} %5p %-11c{1} - %m%n

log4j.logger.level_2_api_record=INFO, O
log4j.appender.O=org.apache.log4j.DailyRollingFileAppender
log4j.appender.O.File=../log/level_2_api_record.log
log4j.appender.O.MaxFileSize=128MB
log4j.appender.O.MaxBackupIndex=10
log4j.appender.O.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.O.layout=org.apache.log4j.PatternLayout
log4j.appender.O.layout.ConversionPattern=%d{ISO8601} %5p %-11c{1} - %m%n
```

八、 组播接收行情模式

- Level2 API v1.1.6.x及以上版本支持组播接收行情模式，该模式操作方法与TCP接收模式基本一致，需要依次调用Connect方法、ReqUserLogin方法，无需订阅操作即可接收全部行情。
- 组播模式接收所有行情，不支持用户订阅行情，需要用户自行过滤合约品种。
- TCP模式和组播模式可以在不同的应用实例中同时使用，在一个实例只能使用一种连接模式。因组播连接将独占组播数据端口，在每个客户端（计算机）对于同一个QuotServer服务器只能运行一个组播实例。
- 由于组播通信方式是基于UDP不可靠通信连接，存在极微量的行情丢失现象，用户可根据行情序列号SequenceNo是否连续递增来判断行情是否丢失。

九、 开发样例

具体样例请查看发布包中demo工程。

十、 网络调优

因 Level2 行情数据量巨大，网络负担重，建议客户端在使用 Level2API 前进行网络调优，

以下设置的参数仅为参考，不同服务器之间的差异可能导致参数的差异，请针对个例进行优化处理。UDP 和组播方式接收行情因使用 UDP 协议传送行情数据，理论上存在数据包丢失的可能性，通过修改优化一些网络参数，可最大限度的避免或减少丢包现象。

(一)、 网卡缓冲区调优

查看网卡缓冲区设置

ethtool -g <网卡设备>

示例
[root@localhost LINUX64]# ethtool -g eth0 Ring parameters for eth0: Pre-set maximums: RX: 4096 RX Mini: 0 RX Jumbo: 0 TX: 4096 Current hardware settings: RX: 512 RX Mini: 0 RX Jumbo: 0 TX: 512

设置网卡缓冲区参数

ethtool -G <网卡设备> rx NEW-BUFFER-SIZE 设置 RX ring 缓冲区大小

示例
[root@localhost LINUX64]# ethtool -G eth0 rx 1024 [root@localhost LINUX64]# ethtool -g eth0 Ring parameters for eth0: Pre-set maximums: RX: 4096 RX Mini: 0 RX Jumbo: 0 TX: 4096 Current hardware settings: RX: 1024 RX Mini: 0 RX Jumbo: 0 TX: 1024

建议客户端计算机将 Rx 设置为 1024 以上

(二)、 操作系统内核缓冲区调优

查看当前操作系统缓冲区的设置

sysctl -A|grep net|grep 'mem\|backlog'|grep 'udp_mem\|rmem_max\|max_backlog'

示例
[root@localhost log]# sysctl -A | grep net | grep 'mem|backlog' | grep 'udp_mem|rmem_max|max_backlog' net.core.rmem_max = 124928　　socket 最大接收缓冲区 net.core.netdev_max_backlog = 1000　　接收数据包队列大小 net.ipv4.udp_mem = 753792　1005056　1507584　　最大可分配的缓冲区空间总量

建议增加最大 Socket 接收缓冲区大小为 32MB：

sysctl -w net.core.rmem_max=33554432

示例 ： 增加最大 Socket 接收缓冲区大小为 32MB
[root@localhost log]# sysctl -w net.core.rmem_max=33554432 net.core.rmem_max = 33554432

建议增加到 2000 以上

示例 ： 增加接收数据包队列大小
[root@localhost log]# sysctl -w net.core.netdev_max_backlog=2000 net.core.netdev_max_backlog = 2000

指令执行完毕后运行 /sbin/sysctl -p 使之生效

十一、 注意事项

(一)、 判断 float、double 无效值

float无效值在windows下为1. #INF，在linux下为inf，请用户注意显示的问题

Double无效值在windows和linux下均为1.79769e+308

(二)、 API 库与头文件的版本一致性

更新Level2API v1.1.6.5及以上版本时，请注意同步更新对应的头文件，否则在应用中可能接收不到行情数据。

十二、 附录

(一)、 常见错误码

-1：其他错误。

1：已登录，请登出后再执行连接操作。

2：已连接组播，不需要再连接。

3：连接模式填写错误。

4：连接服务器失败，请检查网络。

5：版本不匹配。

0：成功。

(二)、 API v1.1.6.x 和 v1.1.5.2 新旧差异对照表

Level2 V1.1.6.x	Level2 V1.1.5.2
支持组播方式获取数据	非正式支持组播模式
新版 API 部分接口新增序列号参数，如下：	旧版 API 定义，部分如下：
void OnBestAndDeep(struct MDBestAndDeep * const pQuote , UINT4 SequenceNo)	void OnBestAndDeep(struct MDBestAndDeep * const pQuote)
void OnArbi(struct MDBestAndDeep * const pQuote, UINT4 SequenceNo)	void OnArbi(struct MDBestAndDeep * const pQuote)
void OnTenEntrust(struct MDTenEntrust * const pQuote, UINT4SequenceNo)	void OnTenEntrust(struct MDTenEntrust * const pQuote)
void OnRealtime(struct MDRealTimePrice * const pQuote, UINT4 SequenceNo)	void OnRealtime(struct MDRealTimePrice * const pQuote)
void OnOrderStatistic(struct MDOrderStatistic * const pQuote, UINT4 SequenceNo)	void OnOrderStatistic(struct MDOrderStatistic * const pQuote)
void OnMarchPrice(struct MDMarchPriceQty * const pQuote, UINT4 SequenceNo)	void OnMarchPrice(struct MDMarchPriceQty * const pQuote)
组播方式接收全部行情，不需要订阅/退订操作，用户登陆成功后即开始接收行情数据。	TCP/UDP方式接收行行情，需要订阅指定合约或全部订阅