Market Data Line TCP 客户端 API 参考

版本	修订时间	修订内容
0.74	2014/12/23	增加深交所 level2 行情
0.8	2015/1/21	增加上证期权 5 档行情

目录

1. 数据类型

1.1 整数

MDL 支持的整数如下表所示:

	C++	Protobuf
8 位无符号整数	uint8_t	uint8
16 位无符号整数	uint16_t	uint16
32 位无符号整数	uint32_t	uint32
64 位无符号整数	uint64_t	uint64
8 位整数	int8_t	int8
16 位整数	int16_t	int16
32 位整数	int32_t	int32
64 位整数	int64_t	int64

1.2 字符串

	C++	Protobuf
Ascii 字符串	MDLAnsiString	string
UTF8 字符串	MDLUTF8String	string

- MDLAnsiString 提供以下方法:

  const char* c_str(): 返回 Asiic 格式以 null 结尾的字符串

  std::string ToUTF8(): 转换为 UTF8(zn_CH)格式标准字符串

  std::string std_str(): 返回 Asiic 格式标准字符串

- MDLUTF8String 提供以下方法:

  const char* c_str(): 返回 UTF8(zn_CH)格式以 null 结尾的字符串

  std::string ToAnsi(): 转换为 MBCS 格式标准字符串

  std::string std_str(): windows 平台下返回 MBCS 格式标准字符串, linux 平台下返回 UTF8(zn_CH)格式标准字符串

1.3 浮点数

	C++	Protobuf
单精度浮点	MDLFloatT<x>	float_x
双精度浮点	MDLDouble<x>	double_x

为避免精度损失，MDL 通过把小数点右移取整的方法记录浮点数，其中 x 表示小数右移位数。

- MDLFloatT<x>提供以下方法

  uint32_t GetDecimalPlace(): 返回小数位数

  uint32_t GetDecimalShift(): 返回 float 格式和整数格式之间相差的倍数，1 位小数为 10,2 位小数 100,以此类推

  uint32_t m_Value: 返回整数格式的浮点数

  float GetFloat(): 返回 float 格式的浮点数，即 m_Value / GetDecimalShift()

  bool IsNull(): 检查字段是否为空

- MDLDoubleT<x>提供以下方法

  uint32_t GetDecimalPlace(): 返回小数位数

  uint32_t GetDecimalShift(): 返回 double 格式和整数格式之间相差的倍数，1 位小数为 10,2 位小数 100,以此类推

  uint64_t m_Value: 返回整数格式的浮点数

  double GetDouble(): 返回 double 格式的浮点数，即 m_Value / GetDecimalShift()

  bool IsNull(): 检查字段是否为空

1.4 日期时间

	C++	**Protobuf**
时间	MDLTime	uint32
日期	MDLDate	uint32

MDL 用 10 进制格式记录日期和时间。其中时间格式为 hhmmssmmm (小时/分钟/秒/毫秒)，日期格式为 yymmdd(年/月/日)

- MDLTime 提供以下方法

  uint8_t GetHour(): 返回小时值

  uint8_t GetMinute(): 返回分钟值

  uint8_t GetSecond():返回秒钟值

  uint16_t GetMilliSec(): 返回毫秒值

- MDLDate 提供以下方法

  uint8_t GetYear(): 返回年

  uint8_t GetMonth(): 返回月

  uint8_t GetDay():返回日

1.5 数组

	C++	Protobuf
数组	MDLList<x>	repeated

MDL 支持数组及嵌套，其中 x 表示数组元素的类型

- MDLListT<x>提供以下方法:

  uint32_t Length: 返回数组元素的个数

  x* operator[]: 通过下标返回数组元素的地址

2. 函数参考

2.1 CreateIOManager

声明: IOManagerPtr CreateIOManager()

返回值: IOManager 对象或空值

说明: 创建并返回 IOManager 对象。

为避免 API 库和 API 头文件不匹配的情况，函数检测头文件宏定义 MDLAPI_VERSION 是否和当前 API 库版本是否兼容，如果不兼容则返回空对象。

头文件: mdl_api.h

3. 类参考

3.1 RefCountedPtr<T>

类说明:

API 库使用引用计数/智能指针包装对象，智能指针的主要作用是自动调用对象的析构函数，用户可以无需关心 API 对象的析构问题。智能指针对象的大多数成员函数，比如复制、构造、重载都为了配合 C++语言语法使用，在这里不一一说明，只介绍重要的成员函数。

头文件:

mdl_api.h

类成员:

- void Reset()

说明: 把智能指针置为空

- bool IsNull() const

说明: 检测智能指针是否为空

3.2 IOManager

类说明:

IOManager 对象表示 API 库的一个 Instance，用户必须保存这个 Instance 直到不再使用 API 库的任何功能。

头文件:

mdl_api.h

类成员:

- SubscriberPtr CreateSubscriber(MessageHandlerBase* handler)

参数: handler， 用户提供的回调类，这个类必须在 IOManager 的生命周期内保持有效。

返回值: 返回订阅对象

说明: 用户可以创建多个订阅对象，每个对象从不同的服务器订阅不同的数据。

- void Shutdown()

说明: 断开网络连接，停止订阅

3.3 Subscriber

类说明:

Subscriber 表示连接到 feeder handler 的一个实例， 用户可以创建多个 Subscriber 以便从不同的 feeder handler 订阅数据。

头文件:

mdl_api.h

类成员:

- void AddSubscription(uint8_t svrID, uint16_t svrVersion, uint16_t msgID)

参数:

svrID: 要订阅的数据服务 ID，当前可订阅的数据服务有:

MDLSID_MDL_SHL1 上交所 Level1 数据

MDLSID_MDL_SHL2 上交所 Level2 数据 (仅做测试用途)

MDLSID_MDL_SZL1 深交所 Level1 数据

MDLSID_MDL_CFFEX: 中金所期货数据源

MDLSID_MDL_CZCE: 郑商所期货数据源

MDLSID_MDL_SHFE: 上期所期货数据源

MDLSID_MDL_DCE: 大商所期货数据源

MDLSID_MDL_HKEX:香港交易所数据源

svrVersion: 要订阅的数据服务版本号

上交所 Level1 数据版本号 MDLVID_MDL_SHL1

上交所 Level2 数据版本号 MDLVID_MDL_SHL2

深交所 Level1 数据版本号 MDLVID_MDL_SZL1

中金所期货数据版本号 MDLVID_MDL_CFFEX

郑商所期货数据版本号 MDLVID_MDL_CZCE

上期所期货数据版本号 MDLVID_MDL_SHFE

大商所期货数据版本号 MDLVID_MDL_DCE

msgID: 要订阅的消息 ID，消息 ID 值请参考头文件定义

上交所 Level1 数据定义头文件 mdl_shl1_msg.h

上交所 Level2 数据定义头文件 mdl_shl2_msg.h

深交所 Level1 数据定义头文件 mdl_szl1_msg.h

中金所期货数据定义头文件 mdl_cffex_msg.h

郑商所期货数据定义头文件 mdl_czce_msg.h

上期所期货数据定义头文件 mdl_shfe_msg.h

大商所期货数据定义头文件 mdl_dce_msg.h

说明:

函数记录用户要订阅的消息内容，然后在登陆到服务器时向服务器提交订阅请求，因此用户必须在调用 Connect 之前添加要订阅的信息内容。

该方法应该在 Connect 之前调用。

- template <class T> void SubcribeMessage()

说明: 功能和 AddSubscription 相同，但用户可以只提供要订阅的消息类型，而不必提供参数。例如：

```
SubscriberPtr subscriber;

subscriber ->SubscribeMessage< mdl_shl1_msg ::SHL1Index>();

subscriber ->SubscribeMessage< mdl_shl1_msg ::SHL2Index>();
```

- void AddSubscriptionByFieldValues(uint8_t svrID, uint16_t svrVersion, uint16_t msgID, const char* fieldName, const char** fieldValues, uint32_t fieldValueCount)

参数:

svrID, svrVersion, msgID: 请参考 AddSubscription

fieldName: 要订阅的消息的字段名

fieldValues: 要订阅的消息的字段值

fieldValueCount: 要订阅的消息的字段值个数

说明: 功能和 AddSubscription 相同，除此之外用户还可以指定要订阅的消息的某个字段必须符合规定的值

该方法应该在 Connect 之前调用。

- template <class T>

void SubcribeMessageByFieldValues(const char* fieldName, const char** fieldValues, size_t fieldValuesCount)

说明: AddSubscriptionByFieldValues 的简化版本

- template <class T>

void SubcribeMessageByUTF8FieldValues(const char* fieldName, const char** fieldValues, size_t fieldValuesCount)

说明: AddSubscriptionByFieldValues 的简化版本。如果客户端想要过滤的字段是 UTF8 格式，但是客户端程序使用的是 gbk, gb2312 等多字节编程环境，那么该函数可以把字段值转换为 UTF8 格式。

- const char* GetUserName()
- void SetUserName(const char* userName)

说明: 设置用户名(当前版本不使用)

该方法应该在 Connect 之前调用。

- const char* GetPassword()
- void SetPassword(const char* passwd)

说明: 设置用户密码(当前版本不使用)

该方法应该在 Connect 之前调用。

- const char* GetServerAddress()
- void SetServerAddress(const char* address)

说明: 服务器 IP 地址列表，IP 地址间用分号隔开

例如 xxx.xxx.xxx.xxx:port;yyyy.yyyy.yyyy.yyyy:port;

API 从 IP 地址列表的首地址开始循环遍历列表，若登陆当前地址失败，则登陆下一个地址

该方法应该在 Connect 之前调用。

- MDLMessageEncoding GetMessageEncoding()

- void SetMessageEncoding(MDLMessageEncoding encoding)

说明: encoding: 数据传输的编码方式，当前支持以下几种编码方式

MDLEID_BINARY：速度最快效率最高 (默认使用)

MDLEID_FAST: 具有数据压缩功能，可用于低速网络

MDLEID_JSON: 使用灵活，无数据 schema (当前版本未使用)

MDLEID_PROTOBUF: 使用广泛，有数据 schema 定义，可用于高级语言处理如 JAVA

该方法应该在 Connect 之前调用。

默认值: MDLEID_BINARY

- uint32_t GetHeartbeatInterval()
- void SetHeartbeatInterval(uint32_t interval)

说明: 服务端发送心跳间隔，单位秒。如果为 0，则服务端不发送心跳。

该方法应该在 Connect 之前调用。

默认值: 10

- uint32_t GetHeartbeatTimeout()
- void SetHeartbeatTimeout(uint32_t interval)

说明: 服务端发送心跳超时值，单位秒。如果为 0，则不检测服务端心跳

该方法应该在 Connect 之前调用。

默认值: 30

- const char* Connect()

说明: Connect 函数遍历地址列表连接到指定的服务器，连接成功后发送登陆请求并且订阅消息，如果登陆成功则返回空字符串""，如果地址列表中的所有服务器都无法登陆则返回错误信息。

一旦成功登陆后，API 会自动维护连接状态。如果连接断开，API 会自动重连到地址列表的下一个地址，并以 sys message 的形式报告重连过程。

返回值: 连接成功返回空值，否则返回错误信息

3.4 MDLMessage

类说明:

MDLMessage 提供访问消息头部和消息体的方法，管理消息占用的内存资源。

头文件:

mdl_api.h

类成员:

- MDLMessageHead* GetHead() const

返回值: 消息头部

- char* GetBody() const

返回值: 消息体地址

说明: 用户应该根据消息头部的 ServiceID, MessageID 决定消息体 Body 的类型。

- uint32_t GetBodySize() const

返回值: 消息体大小

- RefCountedPtrT<MDLMessage> Copy() const

返回值: 消息的一个引用拷贝

说明: 创建一个消息的引用拷贝而无需拷贝内存，当用户需要保存消息对象以便稍后使用时比较有用。

3.5 MessageHandler

类说明:

MessageHandler 是用户提供的回调类，用户通过重载各个回调函数处理消息。

注意，MessageHandler 中的各个成员函数在一个独立的工作线程中被调用，用户需要自己处理数据的线程同步关系。

消息处理过程应该尽量简短，避免磁盘读写、数据库存储等复杂操作，对于消息接收速度太慢的客户端可能会丢失消息，甚至被服务器断开连接。

头文件:

mdl_api.h

类成员:

- void OnMDLAPIMessage(const MDLMessage* msg)

说明: 用户重载此函数处理 API 的网络事件，比如重连，重连失败等

消息类型	说明
mdl_api_msg::ConnectingEvent	正在连接到服务器
mdl_api_msg::ConnectErrorEvent	连接到服务器失败
mdl_api_msg::DisconnectedEvent	从服务器断开

- void OnMDLSysMessage(const MDLMessage* msg)

说明: 用户重载此函数处理服务的登陆响应，检测登陆响应代码，检测订阅结果

消息类型	说明
mdl_sys_msg::LogonResponse	登陆响应

- void OnMDLSHL1Message(const MDLMessage* msg)

说明: 用户重载此函数处理上交所 Level1 行情数据

消息类型	说明
mdl_shl1_msg::SHL1Index	指数快照
mdl_shl1_msg::SHL1Stock	股票快照

- void OnMDLSHL2Message(const MDLMessage* msg)

说明: 用户重载此函数处理上交所 Level2 行情数据

消息类型	说明
mdl_shl2_msg::SHL1Stock	Level1 股票快照 Level2 转发
mdl_shl2_msg::SHL2Transaction	逐笔成交
mdl_shl2_msg::SHL2MarketData	市场行情
mdl_shl2_msg::SHL2VirtualAuctionPrice	虚拟竞价集合
mdl_shl2_msg::SHL2Index	指数快照
mdl_shl2_msg::SHL2MarketOverview	市场总览

mdl_shl2_msg::SHL2Statics	统计消息

- void OnMDLSZL1Message(const MDLMessage* msg)

说明: 用户重载此函数处理深交所 Level1 行情数据

消息类型	说明
mdl_szl1_msg::SZL1Index	指数快照
mdl_szl1_msg::SZL1Stock	股票快照

- void OnMDLDCEMessage(const MDLMessage* msg)

说明: 用户重载此函数处理大商所期货数据

消息类型	说明
mdl_dce_msg::CTPFuture	CTP 格式期货数据

- void OnMDLSHFEMessage(const MDLMessage* msg)

说明: 用户重载此函数处理上期所期货数据

消息类型	说明
mdl_shfe_msg::CTPFuture	CTP 格式期货数据

- void OnMDLCZCEMessage(const MDLMessage* msg)

说明: 用户重载此函数处理郑商所期货数据

消息类型	说明
mdl_czce_msg::CTPFuture	CTP 格式期货数据

- void OnMDLCFFEXMessage(const MDLMessage* msg)

说明: 用户重载此函数处理中金所期货数据

消息类型	说明
mdl_cffex_msg::CTPFuture	CTP 格式期货数据

- void OnMDLHKFEMessage(const MDLMessage* msg)

说明: 用户重载此函数处理中金所期货数据

消息类型	说明
mdl_hkfe_msg::OMDMarketData	港股行情

4. 代码示例

4.1 C++客户端

```
#include "mdl_api.h"
#include "mdl_sh11_msg.h"
#include <string>
#include <stdio.h>

class MyMessageHandler : public MessageHandler {

    // 处理登陆响应
    void OnMDLSysMessage(const MDLMessage* msg) {
        MDLMessageHead* head = msg->GetHead();
        if (head->MessageID == mdl_sys_msg::LogonResponse::MessageID) {
            mdl_sys_msg::LogonResponse* resp = (mdl_sys_msg::LogonResponse*)msg-
>GetBody();
            if (resp->ReturnCode != MDLEC_OK) {
                // 当应用程序首次调用Connect连接到服务器时，API检测服务器ReturnCode，如果失败
则Connect失败，应用程序不会收到Logon失败通知，但会收到Logon成功通知
                // 当已经建立的连接意外断开，API自动做重连时，无论Logon成功或失败应用程序都会收
到消息通知，原则上应用程序的错误处理方式是打印日志，让API自动做下一次重连
                printf("Logon failed, return code = %d\n", resp->ReturnCode);
                return;
            }

            for (uint32_t i = 0; i < resp->Services.Length; ++i) {
                for (uint32_t j = 0; j < resp->Services[i]->Messages.Length; ++j) {
                    // API不处理消息订阅错误，如果发生错误，应用程序应该检查Connect以前输入的订
阅参数是否正确，以及服务器是否支持此类消息推送
                    if (resp->Services[i]->Messages[j]->MessageStatus != MDLEC_OK) {
                        printf("Subscriber message %d failed, return code = %d\n",
                            resp->Services[i]->Messages[j]->MessageID,
                            resp->Services[i]->Messages[j]->MessageStatus);
                    }
                }
            }
        }
    }

    // 处理上交所Level1股票行情
```

```
    Void OnMDLSHL1Message(const MDLMessage* msg) {
        MDLMessageHead* head = msg->GetHead();
        if (head->MessageID == mdl_shl1_msg::SHL1Index::MessageID) {
            mdl_shl1_msg::SHL1Index* resp = (mdl_shl1_msg::SHL1Index*)msg->GetBody();
            printf("[%d:%d:%d]%s,%s:Turnover(%.3f),HighIndex(%.3f),LastIndex(%.3f)\n",
                head->LocalTime.GetHour(), head->LocalTime.GetMinute(), head-
>LocalTime.GetSecond(),
                resp->IndexID.c_str(), resp->IndexName.c_str(),
                resp->Turnover.GetDouble(), resp->HighIndex.GetDouble(), resp-
>LastIndex.GetDouble());

        }
        else if (head->MessageID == mdl_shl1_msg::SHL1Stock::MessageID) {
            mdl_shl1_msg::SHL1Stock* resp = (mdl_shl1_msg::SHL1Stock*)msg->GetBody();
            printf("[%d:%d:%d]%s,%s:Turnover(%.3f),OpenPrice(%.3f),LastPrice(%.3f)\n",
                head->LocalTime.GetHour(), head->LocalTime.GetMinute(), head-
>LocalTime.GetSecond(),
                resp->SecurityID.c_str(), resp->SecurityName.c_str(),
                resp->Turnover.GetDouble(),
                resp->OpenPrice.GetFloat(), resp->LastPrice.GetFloat());

        }
    }
};


int main(int argc, char** argv) {
    // 创建客户端库实例
    IOManagerPtr ioManager = CreateIOManager();
    if (ioManager.IsNull()) {
        printf("Invalid api dll or so version\n");
        return 0;
    }

    // 创建一个订阅对象,然后把订阅对象和消息处理类关联起来
    // 注意: 消息处理类一旦和订阅对象建立关联，必须维持生命周期直到ioManager被释放
    MyMessageHandler myMessageHandler;
    Subscriber subscriber = ioManager->CreateSubscriber(&myMessageHandler);

    // 配置要订阅的消息
    //订阅上交所的所有指数和股票
    subscriber->SubscribeMessage<mdl_shl1_msg::SHL1Index>();
    subscriber->SubscribeMessage<mdl_shl1_msg::SHL1Stock>();
    //订阅深交所的所有指数和股票
    subscriber->SubscribeMessage<mdl_szl1_msg::SZL1Index>();
    subscriber->SubscribeMessage<mdl_szl1_msg::SZL1Stock>();
    //订阅四大期货行情
    subscriber->SubscribeMessage<mdl_cffex_msg::CTPFuture>();
    subscriber->SubscribeMessage<mdl_czce_msg::CTPFuture>();
    subscriber->SubscribeMessage<mdl_dce_msg::CTPFuture>();
    subscriber->SubscribeMessage<mdl_shfe_msg::CTPFuture>();

    // 配置服务器地址
    subscriber->SetServerAddress("127.0.0.1:9010");

    // 建立连接
    std::string err = subscriber->Connect();
```

```
    if (!err.empty()) {
        printf("Connect failed %s\n", err.c_str());
        return 0;
    }

    // 一旦连接建立成功，客户端API就在后台线程运行，调用应用程序的消息处理类
    // 应用程序可以继续在当前线程做其它处理
    printf("Receiving message, press enter to exit.");
    getchar();

    // 不需要接收消息的时候应用程序调用Shutdown，断开已建立的连接，停止后台线程，释放ioManager对
象
    ioManager->Shutdown();

}
```