

南京艾科朗克信息科技有限公司

**XELE-MD证券行情系统**

*API接口以及行情结构体说明*

XELE-MD证券行情系统

*API接口以及行情结构体说明*

# 声明

**版权所有 © 南京艾科朗克信息科技有限公司2021。 保留一切权利。**  
非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

**商标声明**  
和其他艾科朗克商标均为南京艾科朗克信息科技有限公司（以下简称艾科朗克）的商标。  
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

**注意**  
您购买的产品、服务或特性等应受艾科朗克商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，艾科朗克对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

**南京艾科朗克信息科技有限公司**

地址：江苏省南京市秦淮区永丰大道8号白下高新技术产业园区三号楼B栋101

邮编：210014

电话：025-88017287

传真：025-85766287

邮件：[support@accelecom.com](mailto:support@accelecom.com)

网址：[www.accelecom.com](http://www.accelecom.com)

# 版本记录

|  |  |  |
| --- | --- | --- |
| **日期** | **修订者** | **备注** |
| 2/22/2021 | 朱军 | 文档初始化完成 |
| 2023/05/31 | 朱军 | 整合上交LDDS L2和Binary L1行情 |
| 2023/11/16 | 朱军 | 添加深交逐笔构建快照功能 |
| 2023/11/23 | 朱军 | 提升行情API易用性 |
| 2024/12/06 | 朱军 | 删除上交老的期权结构体 |

目录

[声明 1](#_Toc184399108)

[版本记录 2](#_Toc184399109)

[一、 系统介绍 6](#_Toc184399110)

[(一)、 系统概述 6](#_Toc184399111)

[(二)、 API集成环境说明 6](#_Toc184399112)

[(三)、 通讯模式 6](#_Toc184399113)

[(四)、 接收行情线程模型 7](#_Toc184399114)

[(五)、 API时序图 7](#_Toc184399115)

[二、 接口类库文件说明 8](#_Toc184399116)

[三、 XeleMd API使用说明 8](#_Toc184399117)

[(一)、 GetVersion方法 8](#_Toc184399118)

[(二)、 CreatInstance方法 9](#_Toc184399119)

[(三)、 Login方法 9](#_Toc184399120)

[(四)、 Init方法 11](#_Toc184399121)

[(五)、 RegisterSseSpi方法 13](#_Toc184399122)

[(六)、 RegisterSzseSpi方法 13](#_Toc184399123)

[(七)、 Start方法 13](#_Toc184399124)

[(八)、 QueryStatic方法 14](#_Toc184399125)

[(九)、 Stop方法 16](#_Toc184399126)

[(十)、 CreateRestore方法 16](#_Toc184399127)

[(十一)、 RestoreQuery方法 16](#_Toc184399128)

[(十二)、 OnRestoreData方法 17](#_Toc184399129)

[四、 XeleMdSseSpi使用说明 18](#_Toc184399130)

[(一)、 onMarketDataSnapshotSse方法 18](#_Toc184399131)

[(二)、 onBestOrdersSse方法 20](#_Toc184399132)

[(三)、 onIndexSse方法 21](#_Toc184399133)

[(四)、 onAfterTradeSse方法 22](#_Toc184399134)

[(五)、 onAfterSnapshotSse方法 23](#_Toc184399135)

[(六)、 onBondSnapshotSse方法 24](#_Toc184399136)

[(七)、 onBondTickSse方法 26](#_Toc184399137)

[(八)、 onTickMergeSse方法 27](#_Toc184399138)

[(九)、 onETFSnapshotSse方法 29](#_Toc184399139)

[(十)、 onMarketDataTreeSnapSse方法 30](#_Toc184399140)

[(十一)、 onBondTreeSnapSse方法 31](#_Toc184399141)

[(十二)、 onStaticInfoSse方法 31](#_Toc184399142)

[(十三)、 onIndexSseL1方法 31](#_Toc184399143)

[(十四)、 onStockSnapSseL1方法 32](#_Toc184399144)

[(十五)、 onBondSnapSseL1方法 34](#_Toc184399145)

[(十六)、 onFundSseL1方法 34](#_Toc184399146)

[(十七)、 onOptionSseL1方法 35](#_Toc184399147)

[(十八)、 onNationalDebtSseL1方法 37](#_Toc184399148)

[(十九)、 onAfterTradeSseL1方法 38](#_Toc184399149)

[(二十)、 onBond201SnapSseL1方法 39](#_Toc184399150)

[五、 XeleMdSzseSpi使用说明 39](#_Toc184399151)

[(一)、 onMarketDataSnapshotSz方法 40](#_Toc184399152)

[(二)、 onMarketDataTreeSnapSz方法 42](#_Toc184399153)

[(三)、 onBestOrdersSz方法 43](#_Toc184399154)

[(四)、 onIndexSz方法 44](#_Toc184399155)

[(五)、 onTradeSz方法 45](#_Toc184399156)

[(六)、 onOrderSz方法 46](#_Toc184399157)

[(七)、 onAfterSnapshotSz方法 47](#_Toc184399158)

[(八)、 onL1MarketDataSnapSz方法 48](#_Toc184399159)

[(九)、 onIOPVSz方法 50](#_Toc184399160)

[(十)、 onBlockTradeSz方法 51](#_Toc184399161)

[(十一)、 onBondSnapshotSz方法 53](#_Toc184399162)

[(十二)、 onBondBestOrdersSz方法 55](#_Toc184399163)

[(十三)、 onBondTreeSnapSz方法 56](#_Toc184399164)

[(十四)、 onBondTradeSz方法 56](#_Toc184399165)

[(十五)、 onBondBlockTradeSz方法 57](#_Toc184399166)

[(十六)、 onBondBidTradeSz方法 58](#_Toc184399167)

[(十七)、 onBondOrderSz方法 59](#_Toc184399168)

[(十八)、 onBondBlockOrderSz方法 60](#_Toc184399169)

[(十九)、 onBondBidOrderSz方法 61](#_Toc184399170)

[(二十)、 onL1BondSnapSz方法 62](#_Toc184399171)

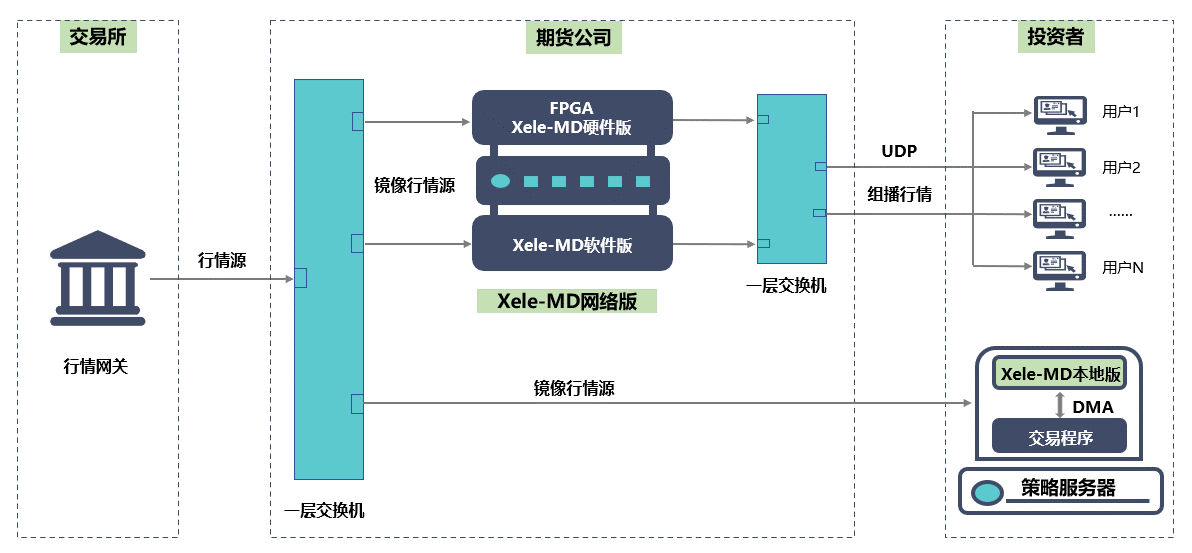
[六、 开发样例 64](#_Toc184399172)

[七、 系统调优 64](#_Toc184399173)

[八、 api补充说明 67](#_Toc184399174)

[(一)、 逐笔保序 67](#_Toc184399175)

1. 系统介绍
2. 系统概述



Xele-MD®证券版，是基于FPGA的纳秒级行情加速系统。

Xele-MD®证券版系统包含了行情解码、行情处理、通路优选和行情转发四大功能。支持股票、指数、债券和期权等行情，助力证券市场极速量化交易的参与者有效提升策略收益。

* **Xele-MD®证券网络版：**部署在证券公司，通过交换机组播分发加速过的行情数据，众多用户通过网络接收。
* **Xele-MD®深交本地版**：直接部署在策略终端服务器上，投资者独享整套行情加速系统。

1. API集成环境说明

API库基于Redhat/Centos7.x操作系统、gcc4.8.5版本编译器进行编译生成，所以建议客户集成API库时，使用Redhat/Centos7.x操作系统、gcc4.8.5以上版本编译器。

1. 通讯模式

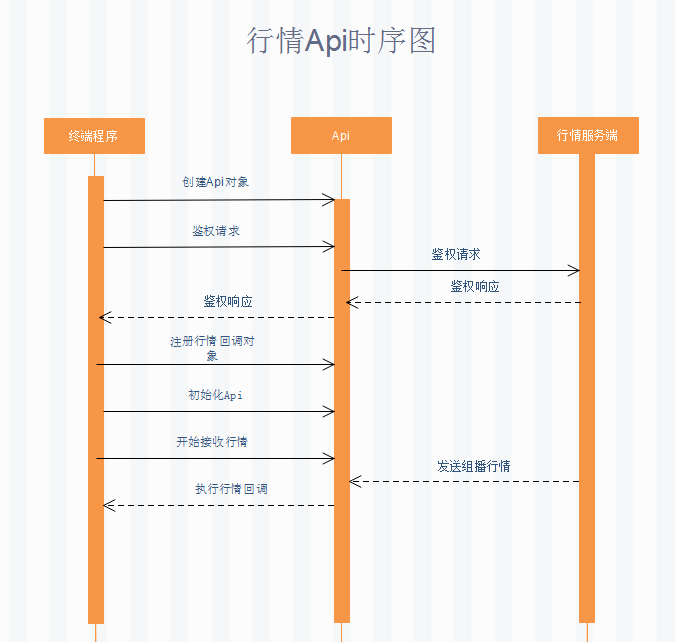
API对外有两种通讯模式:TCP,组播。

客户通过TCP登录鉴权服务器，通过TCP请求行情回补并接收回补响应。通过组播方式接收行情。

1. 接收行情线程模型

API内部将一个组播定义为一个通道，使用一个独立的线程接收，组播中对应的行情回调也在该接收线程中。所以回调函数应尽快返回，避免阻塞导致丢包。如果回调函数处理比较费时，需要将行情缓存起来后另起线程处理，接收线程数和需要接收的组播个数相等。

1. API时序图



1. 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 |
| XeleMd.h | v2.2 | 定义行情接口的头文件 |
| XeleMdSpi.h | v2.2 | 定义行情回调接口的头文件 |
| SseXeleMdStruct.h | v2.2 | 定义上交行情结构体头文件 |
| SzseXeleMdStruct.h | v2.2 | 定义深交行情结构体头文件 |
| libxelemd.a | v2.2 | 行情接口静态链接库二进制文件  红帽系列 linux 64位 |

1. XeleMd API使用说明
2. GetVersion方法

该方法获取API以及对应的行情结构体的版本号。

函数原型：

static void GetVersion(Version\* ver)

参数：

参数一，版本结构体指针。

返回值：无

结构体说明：

typedef struct \_Version

{

char api\_ver[16]; //API版本

char sse\_ver[16]; //上交结构体版本

char sze\_ver[16]; //深交结构体版本

} Version;

1. CreatInstance方法

创建行情接收类的实例。

函数原型：

static XeleMd\* CreatInstance();

参数：

无

返回值：创建的实例指针

1. Login方法

该方法请求登录到鉴权系统,并接收鉴权响应。

函数原型：

bool Login(LoginReq\* login\_req, LoginRsp \*login\_rsp, struct MulticastAddr \*\*addrBegin);

参数：

参数一，登录请求结构体指针

参数二，登录响应结构体指针

参数三，二级指针，保存函数申请的空间首地址，其中存放组播信息

返回值：登录结果,true代表成功，false代表失败

结构体说明：

struct LoginReq

{

char sserver\_ip[16]; //鉴权服务器 IP

uint16\_t server\_port; //鉴权服务器 port

char username[16];

char password[16];

char eth\_name[32]; //本地接收行情的网卡名

};

struct LoginRsp

{

int err\_code;

char err\_msg[40];

uint32\_t numAddr; //返回的有效组播地址数量, 后面跟numAddr个MulticastAddr

};

struct MulticastAddr

{

uint32\_t md\_system; // 组播地址所属的行情系统

char ip[16];

uint16\_t port;

char md\_type[128]; // 组播地址对应的行情类型，取值为输出行情结构体中的type字段, 逗号分割，可包含多个类型

} \_\_attribute\_\_((packed));

md\_system字段值的含义：

|  |  |  |  |
| --- | --- | --- | --- |
| **行情系统** | **md\_system值（16进制）** | **md\_system值（10进制）** | **md\_system值（2进制）** |
| 上海上交主行情 | 0x01 | 1 | 0000 0001 |
| 上海上交备行情 | 0x02 | 2 | 0000 0010 |
| 上海深交主行情 | 0x04 | 4 | 0000 0100 |
| 上海深交备行情 | 0x08 | 8 | 0000 1000 |
| 深圳深交主行情 | 0x10 | 16 | 0001 0000 |
| 深圳深交备行情 | 0x20 | 32 | 0010 0000 |
| 深圳上交主行情 | 0x40 | 64 | 0100 0000 |
| 深圳上交备行情 | 0x80 | 128 | 1000 0000 |

err\_code含义：

|  |  |
| --- | --- |
| **返回码** | **返回码含义** |
| 0 | 登录成功 |
| 100 | 登录请求消息格式错误 |
| 101 | 用户名错误 |
| 102 | 密码错误 |
| 103 | IP错误 |
| 104 | 账户过期 |
| 105 | 行情转发系统错误 |
| 106 | 重复登录 |
| 107 | 账户已注销 |

1. Init方法

API实例初始化函数，收行情前必须先调用该函数。

函数原型：

int Init(MdParam\* const param, const int count);

参数：

参数一，指向初始化参数数组，每个组播通道对应数组中一个元素

参数二，传入数组大小

返回值：初始化结果,0代表成功，-1代表失败

结构体说明：

typedef struct \_MdParam

{

char m\_interfaceName[64]; //接收网卡名

char m\_localIp[32]; //接收网卡ip

char m\_mcastIp[32]; //组播ip

short m\_mcastPort; //组播端口

int m\_bindCpuId; //接收线程绑定cpuid

ENicType m\_nicType; //网卡类型

//是否使用忙等待模式收包，忙等待模式延迟更小，但会增大cpu占用

bool m\_polling = true;

/\*\*

\* api缓存报文的chche大小，单位MB

\* 0代表不使用cache

\* cache模式下cache存满api会丢弃部分包 \*/

int m\_cache = 0;

int m\_cacheCpuId; //cache模式下消费线程绑核cpuid，-1代表不绑核

/\*\*

\* 主备自动切换功能，仅支持盘中切换，

\*在主组播上一定时间内收不到包自动切换到备用组播，

\* 主组播有包后再自动切回来

\*\*/

char m\_backupIntName[64]; //备用接收网卡名

char m\_backupLocalIp[32]; //备用接收网卡ip

char m\_backupMcastIp[32]; //备用组播ip

uint16\_t m\_backupMcastPort; //备用组播端口,配成0时不启用主备切换功能

ENicType m\_backupNicType; //备用网卡类型

int m\_backupCpuId; //备用报文接收线程绑核cpuid，-1代表不绑核

int m\_backupSwitchTime; //多长时间收不到包切换，单位秒

int m\_tickToSnap = 0; //逐笔构建快照， 0代表不构建，1代表上交股票构建，2代表深交股票构建

int m\_tickSnapCpuId = -1; //逐笔转快照绑核cpuid，-1代表不绑核

char m\_tickSnapSubscribe[256]; //逐笔构建快照合约订阅文件路径

char m\_tickSnapLicense[256]; //逐笔构建快照license文件路径,启用该功能需要license文件

  bool m\_lastFlag = false;     //上交逐笔行情是否标记当前报文为该合约在fast压缩包中最后一次出现

} MdParam;

1. RegisterSseSpi方法

注册上交行情回调函数。

函数原型：

void RegisterSseSpi(XeleMdSseSpi\* pSpi);

参数：

参数一，派生自上交回调接口类的实例，基类默认收到行情后不做任何处理，用户可选择关注的行情重写其回调函数

返回值：无

1. RegisterSzseSpi方法

注册深交行情回调函数。

函数原型：

void RegisterSseSpi(XeleMdSzseSpi\* pSpi);

参数：

参数一，派生自深交回调接口类的实例，基类默认收到行情后不做任何处理，用户可选择关注的行情重写其回调函数

返回值：无

1. Start方法

开始接收行情。

函数原型：

int Start();

参数：无

返回值：执行结果,0代表成功，-1代表失败

1. QueryStatic方法

查询上交静态行情，需要券商部署静态行情组播系统才能查询到。

函数原型：

const StaticInfo \*QueryStatic(const char \*securityId);

参数：

参数一，合约id

返回值：静态行情结构体指针，为空代表没查到

结构体定义：

struct StaticInfo

{

  uint8\_t messageType;  //消息类型，静态消息类型为0xf

  uint32\_t sequence;    //udp输出包序号，从1开始

  uint8\_t exchangeID;   //交易所id，上交所：1，深交所：2

  char securityID[9];   //证券代码

 /\*

 涨跌停限制类型

‘N’表示交易规则（2013修订版）3.4.13规定的有涨跌幅限制类型或者权证管理办法第22条规定

‘R’表示交易规则（2013修订版）3.4.15和3.4.16规定的无涨跌幅限制类型

‘S’表示回购涨跌幅控制类型

‘F’表示基于参考价格的涨跌幅控制

‘P’表示IPO上市首日的涨跌幅控制类型

‘U’表示无任何价格涨跌幅控制类型

\*/

  char priceLimitType;

  double upperLimitPrice;       //涨停价

  double lowerLimitPrice;       //跌停价

  uint64\_t buyUnit;          //买数量单位

  uint64\_t sellUnit;         //卖数量单位

  uint64\_t upperQuantityLimitPriceDeclare;  //限价申报数量上限

  uint64\_t lowerQuantityLimitPriceDeclare;  //限价申报数量下限

  double priceGear;          //价格档位,申报价格的最小变动单位

  uint64\_t upperQuantityMarketPriceDeclare; //市价申报数量上限

  uint64\_t lowerQuantityMarketPriceDeclare; //市价申报数量下限

  /\*\*

   \* 证券类别

   \* ‘ES’表示股票；‘EU’表示基金；‘D’表示债券； ‘RWS’表示权证；‘FF’表示期货；

   \* 'CB'表示公募REITs。（参考ISO10962），集合资产管理计划、债券预发行、定向可转债取‘D’

  \*/

  char securityType[7];

  char securityName[9];  //证券名称

  char fileDate[9];  //文件日期(YYYYMMDD)

  char resv[3];      //保留字段

  char securitySubType[4];  //详细证券类别,参考《上海证券市场竞价撮合平台市场参与者接口规格说明书》

  char financeFlag;        //融资标的标志,‘T’表示是融资标的证券,‘F’表示不是融资标的证券

  char shortSaleFlag;       //融券标的标志,‘T’表示是融券标的证券,‘F’表示不是融券标的证券

  char productStatus[21];   //产品状态,参考《上海证券市场竞价撮合平台市场参与者接口规格说明书》

  char listDate[9];        //上市日期(YYYYMMDD)

};

1. Stop方法

停止接收行情。

函数原型：

int Stop();

参数：无

返回值：执行结果,0代表成功，-1代表失败

1. CreateRestore方法

当客户发现逐笔行情有丢包时，可以请求回补服务端重发相应的行情。该功能需要先通过此函数创建一个请求实例。

函数原型：

static XeleMdRestoreApi \*CreateRestore(XeleMdRestoreSpi \*p);

参数：

参数一，继承自XeleMdRestoreSpi的类实例，其中的回调函数会在收到回补行情时调用

返回值：创建的实例指针

1. RestoreQuery方法

当客户发现逐笔行情有丢包时，可以通过该函数请求回补服务端重发相应的行情。

函数原型：

virtual int RestoreQuery(const char\* restoreUrl, const RestoreQueryFeild & logonReq);

参数：

参数一，回补服务器地址，例“tcp://192.168.0.1:2222”

参数二，回补请求参数结构体

返回值：执行结果,0代表成功，-1代表失败

结构体定义：

typedef struct \_RestoreQueryFeild

{

char clientId[11]; //用户ID

char password[11]; //用户密码

uint8\_t exchangeType; //交易所类型 1: sse 2:sz.

uint64\_t appSeqStart; //起始序列号

uint64\_t appSeqEnd; //结束序列号

uint16\_t channelId; //通道号

uint32\_t categoryId; //上交需要填写

} \_\_attribute\_\_((packed)) RestoreQueryFeild;

1. OnRestoreData方法

当API收到逐笔回补的响应时，该函数会被调用。

函数原型：

virtual void OnRestoreData(int errorno, uint8\_t \*pdata, bool bLast);

参数：

参数一，返回的错误码

参数二，行情数据，使用前需要判断是否为空

参数三，是否是最后一条数据

错误码范围为：

0 成功

1 错误的用户名或密码

2 错误的交易所ID

3 连接失败，可能url错误或服务未启动s

100 回补成功

101 回补部分成功

102 没有权限

103 无效数据

104 内部错误

1. XeleMdSseSpi使用说明

使用者要新设计一个类继承自class XeleMdSseSpi，并实现其全部或部分回调函数数。

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，不然导致UDP丢包。请注意，不同行情回调接口是否是同一个线程取决于这些消息类型是否在同一个组播里。

1. onMarketDataSnapshotSse方法

当API接收到上交快照行情时，该方法会被调用。

函数原型：

virtual void onMarketDataSnapshotSse(MarketDataSnapshotSse\* p);

参数：

参数一，上交快照行情

结构体定义：

struct BidAskPriceQtySse {

uint32\_t price; //申买、申卖价格，实际值除以1000

uint64\_t qty; //申买、申卖数量，实际值除以1000

};

struct MarketDataSnapshotSse {

uint8\_t messageType; //消息类型，快照为0x1

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t resv[2]; //保留字段

/\*

产品所处的交易阶段代码

该字段为8位字符串，左起每位表示特定的含义，无定义则填空格。

第1位：‘S’表示启动（开市前）时段，‘C’表示开盘集合竞价时段，‘T’表示连续交易时段，‘E’表示闭市时段，‘P’表示产品停牌，

‘M’表示可恢复交易的熔断时段（盘中集合竞价），‘N’表示不可恢复交易的熔断时段（暂停交易至闭市），‘U’表示收盘集合竞价时段。

第2位：‘0’表示此产品不可正常交易，‘1’表示此产品可正常交易，无意义填空格。

第3位：‘0’表示未上市，‘1’表示已上市。

第4位：‘0’表示此产品在当前时段不接受订单申报，‘1’ 表示此产品在当前时段可接受订单申报。无意义填空格

\*/

char tradingPhaseCode[8];

/\*

当前品种交易状态，详见上交所接口说明文档

\*/

char instrumentStatus[6];

uint8\_t resv2[5]; //保留字段

uint32\_t timeStamp; //最新订单时间,格式HHMMSS，例如143025 表示 14:30:25

uint32\_t preClosePrice; //昨收价（来源消息头),实际值上交除以1000

uint32\_t numTrades; //总成交笔数

uint64\_t totalVolumeTrade; //总成交量,实际值上交除以1000

uint64\_t totalValueTrade; //总成交金额，实际值上交除以100000

uint32\_t lastPrice; //最近价，实际值除以1000

uint32\_t openPrice; //开盘价，实际值除以1000

uint32\_t closePrice; //今日收盘价,实际值除以1000

uint32\_t highPrice; //最高价，实际值除以1000

uint32\_t lowPrice; //最低价，实际值除以1000

uint64\_t IOPV; //ETF净值估值，实际值除以1000

uint64\_t warUpperPx; //IOPV高精度值，实际值除以100000

uint32\_t bidAvgPrice; //买入委托数量加权平均价，实际值除以1000

uint64\_t bidTotalQty; //买入委托总数量，实际值除以1000

uint32\_t askAvgPrice; //卖出委托数量加权平均价，实际值除以1000

uint64\_t askTotalQty; //卖出委托总数量，实际值除以1000

BidAskPriceQtySse bidInfo[SNAPSHOT\_LEVEL]; //申买信息

BidAskPriceQtySse askInfo[SNAPSHOT\_LEVEL]; //申卖信息

uint32\_t msgSeqID; //消息序号

uint32\_t withdrawBuyNum; //买入撤单笔数

uint64\_t withdrawBuyAmount; //买入撤单数量，实际值除以1000

uint64\_t withdrawBuyMoney; //买入撤单金额，实际值除以100000

uint32\_t withdrawSellNum; //卖出撤单笔数

uint64\_t withdrawSellAmount; //卖出撤单数量，实际值除以1000

uint64\_t withdrawSellMoney; //卖出撤单金额，实际值除以100000

};

1. onBestOrdersSse方法

当API接收到上交订单明细行情时，该方法会被调用。

函数原型：

virtual void onBestOrdersSse(BestOrdersSse\* p);

参数：

参数一，上交订单明细行情

结构体定义：

struct BestOrdersSse {

uint8\_t messageType; //消息类型，订单明细为0x2

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

char recv[3]; //保留字段

uint32\_t msgSeqID; //消息序号

uint8\_t side; //买卖标识：买：1，卖：2

uint8\_t number; //明细个数

uint32\_t timeStamp; //最新订单时间，格式HHMMSS，例如143025 表示 14:30:25

uint32\_t price; //委托价格，实际值除以1000

uint64\_t orders; //申买/卖数量，实际值除以1000

uint64\_t volume[50]; //委托数量，最多揭示50笔，实际值除以1000

};

1. onIndexSse方法

当API接收到上交指数行情时，该方法会被调用。

函数原型：

virtual void onIndexSse(IndexSse\* p);

参数：

参数一，上交指数行情

结构体定义：

struct IndexSse {

uint8\_t messageType; //消息类型，指数为0x3

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t resv[5]; //保留字段

uint32\_t timeStamp; //最新订单时间,格式HHMMSS，例如143025 表示 14:30:25

uint32\_t tradeTime; //成交时间，

uint32\_t msgSeqID; //消息序号

uint64\_t preClosePrice; //前盘指数（来源扩展字段）,实际值除以100000

uint64\_t openPrice; //开盘指数,实际值除以100000

uint64\_t lastPrice; //最新指数,实际值除以100000

uint64\_t highPrice; //最高指数,实际值除以100000

uint64\_t lowPrice; //最低指数,实际值除以100000

uint64\_t closePrice; //今日收盘指数,实际值除以100000

uint64\_t totalVolume; //成交总量,实际值除以100000

uint64\_t totalValue; //成交总金额,实际值除以10

};

1. onAfterTradeSse方法

当API接收到上交股票盘后逐笔成交行情时，该方法会被调用。

函数原型：

virtual void onAfterTradeSse(AfterTradeSse\* p);

参数：

参数一，上交股票盘后逐笔成交行情

结构体定义：

struct AfterTradeSse {

uint8\_t messageType; //消息类型，上交盘后逐笔成交为0x8

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

char tradeBSFlag; //内外盘标志 ：B-外盘，主动买，S-内盘，主动买，N-未知

uint32\_t tradeIndex; //上交所：成交序号

uint32\_t channelNo; //成交通道

uint32\_t transactTime; //成交时间

uint32\_t tradePrice; //成交价格,实际值除以1000

uint64\_t tradeQty; //成交数量,实际值除以1000

uint64\_t tradeMoney; //成交金额,实际值除以100000

uint64\_t bidapplSeqnum; //买方委托索引

uint64\_t offerapplSeqnum; //卖方委托索引

uint32\_t msgSeqID; //消息序号

char resv2[4]; //保留字段

};

1. onAfterSnapshotSse方法

当API接收到上交股票盘后快照行情时，该方法会被调用。

函数原型：

virtual void onAfterSnapshotSse(AfterSnapshotSse\* p);

参数：

参数一，上交股票盘后快照行情

结构体定义：

struct AfterSnapshotSse {

uint8\_t messageType; //消息类型，上交快照为0x7

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t resv; //保留字段

char instrumentStatus[6]; //当前品种交易状态,详见上交所接口说明文档

uint32\_t timeStamp; //最新订单时间,格式HHMMSS，例如143025 表示 14:30:25

uint32\_t closePrice; //今日收价,实际值除以1000

uint32\_t numTrades; //总成交笔数

uint64\_t totalVolumeTrade; //总成交量,实际值除以1000

uint64\_t totalValueTrade; //总成交金额，实际值除以100000

uint64\_t totalBidQty; //买入委托总数量，实际值除以1000

uint64\_t totalAskQty; //卖出委托总数量，实际值除以1000

uint32\_t withdrawBuyNumber; //买入撤单笔数

uint64\_t withdrawBuyAmount; //买入撤单数量，实际值除以1000

uint32\_t withdrawSellNumber; //卖出撤单笔数

uint64\_t withdrawSellAmount; //卖出撤单数量，实际值除以1000

uint32\_t msgSeqID; //消息序号

uint32\_t noBidLevel; //买盘价位数量

uint32\_t noOfferLevel; //卖盘价位数量

uint64\_t bidQty; //一档申买量，实际值除以1000

uint64\_t askQty; //一档申卖量，实际值除以1000

uint64\_t bidOrderNum; //买一价申买量揭示笔数

uint64\_t askOrderNum; //卖一价申卖量揭示笔数

};

1. onBondSnapshotSse方法

当API接收到上交债券快照行情时，该方法会被调用。

函数原型：

virtual void onBondSnapshotSse(BondSnapshotSse\* p);

参数：

参数一，上交债券快照行情

结构体定义：

struct BondSnapshotSse {

  uint8\_t  messageType;       //消息类型，债券快照为0x0C

  uint32\_t sequence;         //udp输出包序号，从1开始

  uint8\_t  exchangeID;        //交易所id，上交所：1，深交所：2

  char   securityID[9];      //证券代码

  char   resv[2];         //保留字段

  char  instrumentStatus[6]; //当前品种交易状态,详见上交所接口说明文档

  char  resv2;          //保留字段

  uint32\_t timeStamp;        //最新订单时间，精确到毫秒，格式HHMMSSmmm，例如143025001 表示 14:30:25:001

  uint32\_t preClosePrice;      //昨收价（来源消息头)，实际值除以1000

  uint32\_t openPrice;        //开盘价，实际值除以1000

  uint32\_t highPrice;        //最高价，实际值除以1000

  uint32\_t lowPrice;         //最低价，实际值除以1000

  uint32\_t lastPrice;        //最近价，实际值除以1000

  uint32\_t closePrice;        //今日收盘价，实际值除以1000

  uint32\_t numTrades;        //总成交笔数

  uint64\_t totalVolumeTrade;     //总成交量，实际值除以1000

  uint64\_t totalValueTrade;     //总成交金额，实际值除以100000

  uint64\_t bidTotalQty;       //买入委托总数量，实际值除以1000

  uint32\_t bidAvgPrice;       //买入委托加权平均价，实际值除以1000

  uint64\_t askTotalQty;       //卖出委托总数量，实际值除以1000

  uint32\_t askAvgPrice;       //卖出委托加权平均价，实际值除以1000

  uint32\_t avgPrice;         //加权平均价，实际值除以1000

  BidAskPriceQtySse bidInfo[SNAPSHOT\_LEVEL];  //申买信息

  BidAskPriceQtySse askInfo[SNAPSHOT\_LEVEL];  //申卖信息

  uint32\_t msgSeqID;              //消息序号

uint32\_t withdrawBuyNum; //买入撤单笔数

uint64\_t withdrawBuyAmount; //买入撤单数量，实际值除以1000

uint64\_t withdrawBuyMoney; //买入撤单金额，实际值除以100000

uint32\_t withdrawSellNum; //卖出撤单笔数

uint64\_t withdrawSellAmount; //卖出撤单数量，实际值除以1000

uint64\_t withdrawSellMoney; //卖出撤单金额，实际值除以100000

};

1. onBondTickSse方法

当API接收到上交债券逐笔行情时，该方法会被调用。

函数原型：

virtual void onBondTickSse(BondTickSse\* p);

参数：

参数一，上交债券逐笔行情

结构体定义：

struct BondTickSse {

  uint8\_t  messageType;   //消息类型，债券逐笔为0x0D

  uint32\_t sequence;    //udp输出包序号，从1开始

  uint8\_t  exchangeID;   //交易所id，上交所：1，深交所：2

  char   securityID[9];  //证券代码

  char   tickType;    //类型：A=新增委托订单 D=删除委托订单 S=产品状态订单 T=成交

  /\*

  订单状态

  为新增或删除委托订单时：

    0-买单  1-卖单

  为产品状态订单时；

    0=产品未上市 1=启动 2=开市集合竞价 3=连续自动撮合 4=停牌 5=闭市 6=交易结束

  为成交时：

    0-外盘，主动买 1-内盘，主动卖 2-未知

  \*/

  char   tickBSFlag;

  uint32\_t tickIndex;  //逐笔序号,从1开始，按channel连续

  uint32\_t channelNo;  //频道代码

  uint32\_t tickTime;   //订单或成交时间，精确到毫秒，格式HHMMSSmmm，例如143025001 表示 14:30:25:001

  uint64\_t buyOrderNo;  //买方订单号

  uint64\_t sellOrderNo; //卖方订单号

  uint32\_t price;    //价格，实际值除以1000

  uint64\_t qty;     //数量，实际值除以1000

  uint64\_t tradeMoney; //成交金额，仅适用于TickType=T时，实际值除以100000

  uint32\_t msgSeqID;  //消息序号（仅定位调试使用）

  /\* 标记当前包是否是同一个fast消息中解出的最后一笔行情

   如果开启了标记合约在fast压缩包中最后一次出现的功能，

   则含义变为当前包是否为该合约在fast压缩包中最后一次出现

   0-非最后  1-最后 \*/

  uint8\_t  isLast;

  char     resv[2];        //保留字段

};

1. onTickMergeSse方法

当API接收到上交逐笔合并行情时，该方法会被调用。

函数原型：

virtual void onTickMergeSse(TickMergeSse\* p);

参数：

参数一，上交逐笔合并行情

结构体定义：

struct TickMergeSse {

  uint8\_t  messageType;  //消息类型，竞价逐笔合并为0x11

  uint32\_t sequence;    //udp输出包序号，从1开始

  uint8\_t  exchangeID;  //交易所id，上交所：1，深交所：2

  char    securityID[9]; //证券代码

  char    tickType;   //类型：A=新增委托订单 D=删除委托订单 S=产品状态订单 T=成交

  /\*

  订单状态

  为新增或删除委托订单时：

    0-买单  1-卖单

  为产品状态订单时；

    0=产品未上市 1=启动 2=开市集合竞价 3=连续自动撮合 4=停牌 5=闭市 6=交易结束

  为成交时：

    0-外盘，主动买 1-内盘，主动卖 2-未知

  \*/

  char   tickBSFlag;

  uint64\_t tickIndex;  //逐笔序号,从1开始，按channel连续

  uint32\_t channelNo;  //频道代码

  uint32\_t tickTime;  //订单或成交时间，精确到10毫秒，格式HHMMSSmm

  uint64\_t buyOrderNo; //买方订单号

  uint64\_t sellOrderNo; //卖方订单号

  uint32\_t price;    //价格，实际值除以1000

  uint64\_t qty;     //数量，实际值除以1000

  // TickType=T时，代表成交金额，实际值除以100000；TickType=A时，代表已成交委托数量，实际值除以1000，其他无意义

uint64\_t tradeMoney;

  uint32\_t msgSeqID;  //消息序号（仅定位调试使用）

  /\* 标记当前包是否是同一个fast消息中解出的最后一笔行情

   如果开启了标记合约在fast压缩包中最后一次出现的功能，

   则含义变为当前包是否为该合约在fast压缩包中最后一次出现

   0-非最后  1-最后 \*/

  uint8\_t  isLast;

  char     resv[6];        //保留字段

};

1. onETFSnapshotSse方法

当API接收到上交ETF快照行情时，该方法会被调用。

函数原型：

virtual void onETFSnapshotSse(ETFSnapshotSse\* p);

参数：

参数一，上交ETF快照行情

结构体定义：

struct ETFSnapshotSse {

  uint8\_t  messageType;    //消息类型，ETF快照为0x10

  uint32\_t sequence;      //udp输出包序号，从1开始

  uint8\_t  exchangeID;     //交易所id，上交所：1，深交所：2

  char   securityID[9];   //证券代码

  uint32\_t timeStamp;   //更新时间，格式HHMMSS

  uint32\_t etfBuyNum;     //ETF申购笔数

  uint64\_t etfBuyAmount;    //ETF申购数量，实际值除以1000

  uint64\_t etfBuyMoney;    //ETF申购金额，实际值除以100000

  uint32\_t etfSellNum;     //ETF赎回笔数

  uint64\_t etfSellAmount;   //ETF赎回数量，实际值除以1000

  uint64\_t etfSellMoney;    //ETF赎回金额，实际值除以100000

  uint32\_t msgSeqID;      //消息序号（仅定位调试使用）

  char   resv[1];      //保留字段

};

1. onMarketDataTreeSnapSse方法

当API接收到上交股票逐笔转快照行情时，该方法会被调用。

函数原型：

virtual void onMarketDataTreeSnapSse(MarketDataTreeSnapSse\* p);

参数：

参数一，逐笔转快照行情

结构体定义：

struct MarketDataTreeSnapSse {

uint8\_t messageType; //消息类型，股票快照为0x12，债券为0x13

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t signLoss; //丢包标志,1表示丢包， 0表示未丢包

uint64\_t applSeqNum; //构建行情的逐笔序号

uint32\_t timeStamp; //最新订单时间，精确到10毫秒，格式HHMMSSmm

uint32\_t preClosePrice; //昨收价（来源消息头)，实际值除以1000

uint32\_t numTrades; //总成交笔数

uint64\_t totalVolumeTrade; //总成交量，实际值除以1000

uint64\_t totalValueTrade; //总成交金额，实际值除以100000

uint32\_t lastPrice; //最近价，实际值除以1000

uint32\_t openPrice; //开盘价，实际值除以1000

uint32\_t highPrice; //最高价，实际值除以1000

uint32\_t lowPrice; //最低价，实际值除以1000

uint32\_t bidAvgPrice; //买入委托加权平均价，实际值除以1000

uint64\_t bidTotalQty; //买入委托总数量，实际值除以1000

uint32\_t askAvgPrice; //卖出委托加权平均价，实际值除以1000

uint64\_t askTotalQty; //卖出委托总数量，实际值除以1000

BidAskPriceQtySse bidInfo[10]; //10档申买信息

BidAskPriceQtySse askInfo[10]; //10档申卖信息

};

1. onBondTreeSnapSse方法

当API接收到上交债券逐笔转快照行情时，该方法会被调用。

函数原型：

virtual void onMarketDataTreeSnapSse(MarketDataTreeSnapSse\* p);

参数：

参数一，逐笔转快照行情

1. onStaticInfoSse方法

当API接收到上交静态行情时，该方法会被调用。静态行情也可以通过QueryStatic方法查询。

函数原型：

virtual void onStaticInfoSse(StaticInfoSse\* p);

参数：

参数一，上交静态行情

1. onIndexSseL1方法

当API接收到上交L1指数行情时，该方法会被调用。

函数原型：

virtual void onIndexSseL1(IndexSseL1\* p);

参数：

参数一，上交L1指数行情

结构体定义：

struct IndexSseL1 {

  uint8\_t  messageType;       //消息类型，指数为0x1

  uint32\_t sequence;         //udp输出包序号，从1开始

  uint8\_t  exchangeID;       //交易所id，上交所：1，深交所：2

  char     securityID[9];    //证券代码

  uint16\_t  flag;

  char     recv[3];

  uint8\_t  securityType;

  uint8\_t  tradSesMode;

  uint64\_t sendingTime;      //发送时间，格式： YYYYMMDDHHmmSSsss

  uint32\_t tradeDate;       //交易日期 YYYYMMDD

  uint32\_t lastUpdateTime;    //最新更新时间 HHMMSSsss

  uint64\_t preClosePrice;     //实际值除以100000

  uint64\_t totalVolumeTraded;

  uint64\_t tradeNum;

  uint64\_t totalValueTraded;   //实际值除以100

  char tradingPhaseCode[8];

  uint64\_t lastIndex;       //实际值除以100000

  uint64\_t openIndex;       //实际值除以100000

  uint64\_t closeIndex;       //实际值除以100000

  uint64\_t highIndex;       //实际值除以100000

  uint64\_t lowIndex;        //实际值除以100000

};

1. onStockSnapSseL1方法

当API接收到上交L1股票快照行情时，该方法会被调用。

函数原型：

virtual void onStockSnapSseL1(StockBondSnapSseL1\* p);

参数：

参数一，上交L1股票快照行情

结构体定义：

struct priceQty {

  uint64\_t price;           //实际值除以100000

  uint64\_t qty;

};

struct StockBondSnapSseL1 {

uint8\_t messageType; //消息类型，股票快照类型为0x2,债券为0x3

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint16\_t flag;

char recv[3];

uint8\_t securityType;

uint8\_t tradSesMode;

uint64\_t sendingTime; //发送时间，格式： YYYYMMDDHHmmSSsss

uint32\_t tradeDate; //交易日期 YYYYMMDD

uint32\_t lastUpdateTime; //最新更新时间 HHMMSSsss

uint64\_t preClosePrice; //实际值除以100000

uint64\_t totalVolumeTraded;

uint64\_t tradeNum;

uint64\_t totalValueTraded; //实际值除以100

char tradingPhaseCode[8];

uint64\_t lastPrice; //实际值除以100000

uint64\_t openPrice; //实际值除以100000

uint64\_t closePrice; //实际值除以100000

uint64\_t settlePrice; //实际值除以100000

uint64\_t highPrice; //实际值除以100000

uint64\_t lowPrice; //实际值除以100000

uint64\_t preSettlePrice; //实际值除以100000

struct priceQty bidPriceQty[5];

struct priceQty askPriceQty[5];

};

1. onBondSnapSseL1方法

当API接收到上交L1债券快照行情时，该方法会被调用。

函数原型：

virtual void onBondSnapSseL1(StockBondSnapSseL1\* p);

参数：

参数一，上交L1债券快照行情

1. onFundSseL1方法

当API接收到上交L1基金快照行情时，该方法会被调用。

函数原型：

virtual void onFundSseL1(FundSseL1\* p);

参数：

参数一，上交L1基金快照行情

结构体定义：

struct FundSseL1 {

uint8\_t messageType; //消息类型，基金为0x4

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint16\_t flag;

char recv[3];

uint8\_t securityType;

uint8\_t tradSesMode;

uint64\_t sendingTime; //发送时间，格式： YYYYMMDDHHmmSSsss

uint32\_t tradeDate; //交易日期 YYYYMMDD

uint32\_t lastUpdateTime; //最新更新时间 HHMMSSsss

uint64\_t preClosePrice; //实际值除以100000

uint64\_t totalVolumeTraded;

uint64\_t tradeNum;

uint64\_t totalValueTraded; //实际值除以100

char tradingPhaseCode[8];

uint64\_t lastPrice; //实际值除以100000

uint64\_t openPrice; //实际值除以100000

uint64\_t closePrice; //实际值除以100000

uint64\_t settlePrice; //实际值除以100000

uint64\_t highPrice; //实际值除以100000

uint64\_t lowPrice; //实际值除以100000

uint64\_t preSettlePrice; //实际值除以100000

uint64\_t IOPV; //实际值除以100000

uint64\_t preCloseIOPV; //实际值除以100000

struct priceQty bidPriceQty[5];

struct priceQty askPriceQty[5];

};

1. onOptionSseL1方法

当API接收到上交L1期权行情时，该方法会被调用。

函数原型：

virtual void onOptionSseL1 (OptionSseL1\* p);

参数：

参数一，上交L1期权行情

结构体定义：

struct OptionSseL1 {

uint8\_t messageType; //消息类型，期权为0x5

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint16\_t flag;

char recv[3];

uint8\_t securityType;

uint8\_t tradSesMode;

uint64\_t sendingTime; //发送时间，格式： YYYYMMDDHHmmSSsss

uint32\_t tradeDate; //交易日期 YYYYMMDD

uint32\_t lastUpdateTime; //最新更新时间 HHMMSSsss

uint64\_t preClosePrice; //实际值除以100000

uint64\_t totalVolumeTraded;

uint64\_t tradeNum;

uint64\_t totalValueTraded; //实际值除以100

char tradingPhaseCode[8]; //实际值除以100000

uint64\_t lastPrice; //实际值除以100000

uint64\_t openPrice; //实际值除以100000

uint64\_t closePrice; //实际值除以100000

uint64\_t settlePrice; //实际值除以100000

uint64\_t highPrice; //实际值除以100000

uint64\_t lowPrice; //实际值除以100000

uint64\_t preSettlePrice; //实际值除以100000

uint64\_t auctionPrice; //实际值除以100000

uint64\_t auctionQty; //实际值除以100000

uint64\_t totalLongPosition;

struct priceQty bidPriceQty[5];

struct priceQty askPriceQty[5];

};

1. onNationalDebtSseL1方法

当API接收到上交L1国债行情时，该方法会被调用。

函数原型：

virtual void onNationalDebtSseL1(NationalDebtSseL1\* p);

参数：

参数一，上交L1国债行情

结构体定义：

struct NationalDebtSseL1 {

uint8\_t messageType; //消息类型，国债为0x6

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint16\_t flag;

char recv[3];

uint8\_t securityType;

uint8\_t tradSesMode;

uint64\_t sendingTime; //发送时间，格式： YYYYMMDDHHmmSSsss

uint32\_t tradeDate; //交易日期 YYYYMMDD

uint32\_t lastUpdateTime; //最新更新时间 HHMMSSsss

uint64\_t preClosePrice; //实际值除以100000

uint64\_t totalVolumeTraded;

uint64\_t tradeNum;

uint64\_t totalValueTraded; //实际值除以100

char tradingPhaseCode[8];

uint64\_t lastPrice; //实际值除以100000

uint64\_t openPrice; //实际值除以100000

uint64\_t closePrice; //实际值除以100000

uint64\_t settlePrice; //实际值除以100000

uint64\_t highPrice; //实际值除以100000

uint64\_t lowPrice; //实际值除以100000

uint64\_t preSettlePrice; //实际值除以100000

struct priceQty bidPriceQty[5];

struct priceQty askPriceQty[5];

};

1. onAfterTradeSseL1方法

当API接收到上交L1盘后固定价格行情时，该方法会被调用。

函数原型：

virtual void onAfterTradeSseL1(AfterTradeSseL1\* p);

参数：

参数一，上交L1盘后固定价格行情

结构体定义：

struct AfterTradeSseL1 {

uint8\_t messageType; //消息类型，盘后为0x7

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint16\_t flag;

char recv[3];

uint8\_t securityType;

uint8\_t tradSesMode;

uint64\_t sendingTime; //发送时间，格式： YYYYMMDDHHmmSSsss

uint32\_t tradeDate; //交易日期 YYYYMMDD

uint32\_t lastUpdateTime; //最新更新时间 HHMMSSsss

uint64\_t preClosePrice; //实际值除以100000

uint64\_t totalVolumeTraded;

uint64\_t tradeNum;

uint64\_t totalValueTraded; //实际值除以100

char tradingPhaseCode[8];

uint64\_t notTradedBidVolume;

uint64\_t notTradedAskVolume;

uint64\_t closePrice; //实际值除以100000

};

1. onBond201SnapSseL1方法

当API接收到上交L1新债券行情时（交易所MD201消息），该方法会被调用。

函数原型：

virtual void onBond201SnapSseL1(StockBondSnapSseL1\* p);

参数：

参数一，上交L1新债券行情

1. XeleMdSzseSpi使用说明

使用者要新设计一个类继承自class XeleMdSzseSpi，并实现其全部或部分回调函数数。

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，不然导致UDP丢包。请注意，不同行情回调接口是否是同一个线程取决于这些消息类型是否在同一个组播里。

1. onMarketDataSnapshotSz方法

当API接收到深交快照行情时，该方法会被调用。

函数原型：

virtual void onMarketDataSnapshotSz(MarketDataSnapshotSz\* p);

参数：

参数一，深交快照行情

结构体定义：

struct BidAskPriceQtySz {

uint64\_t price; //申买、申卖价格，实际值除以1000000

uint64\_t qty; //申买、申卖数量，实际值除以100

};

struct MarketDataSnapshotSz {

uint8\_t messageType; //消息类型，快照为0x1

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t resv[2]; //保留字段

/\*

产品所处的交易阶段代码

第0 位：S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

第1 位：0=正常状态,1=全天停牌

\*/

char tradingPhaseCode[8];

uint8\_t resv2[7]; //保留字段

uint64\_t timeStamp; //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm, 精确到毫秒，示例：20190411102939000

uint64\_t preClosePrice; //昨收价,实际值除以10000

uint64\_t numTrades; //总成交笔数

uint64\_t totalVolumeTrade; //总成交量,实际值除以100

uint64\_t totalValueTrade; //总成交金额，实际值除以10000

uint64\_t lastPrice; //最近价，实际值除以1000000

uint64\_t openPrice; //开盘价，实际值除以1000000

uint64\_t openInterest; //合约持仓量，实际值除以100

uint64\_t highPrice; //最高价，实际值除以1000000

uint64\_t lowPrice; //最低价，实际值除以1000000

uint64\_t upperlmtPrice; //涨停价，实际值除以1000000

uint64\_t lowerlmtPrice; //跌停价，实际值除以1000000

uint64\_t bidAvgPrice; //买入委托数量加权平均价，实际值除以1000000

uint64\_t bidTotalQty; //买入委托总数量，实际值除以100

uint64\_t askAvgPrice; //卖出委托数量加权平均价，实际值除以1000000

uint64\_t askTotalQty; //卖出委托总数量，实际值除以100

BidAskPriceQtySz bidInfo[SNAPSHOT\_LEVEL]; //申买信息

BidAskPriceQtySz askInfo[SNAPSHOT\_LEVEL]; //申卖信息

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

uint32\_t resv3; //保留字段

uint64\_t IOPV; //基金实时参考值，实际值除以1000000

uint64\_t bidNumOfOrders[SNAPSHOT\_LEVEL]; //买10档价位总委托笔数

uint64\_t askNumOfOrders[SNAPSHOT\_LEVEL]; //卖10档价位总委托笔数

};

1. onMarketDataTreeSnapSz方法

当API接收到深交股票逐笔构建快照行情时，该方法会被调用。

函数原型：

virtual void onMarketDataTreeSnapSz(MarketDataTreeSnapSz\* p);

参数：

参数一，深交逐笔构建快照行情

结构体定义：

struct MarketDataTreeSnapSz {

uint8\_t messageType; //消息类型，股票逐笔转行情快照为0x10，债券为0x14

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t signLoss; //丢包标志,1表示丢包， 0表示未丢包

uint64\_t applSeqNum; //构建行情的逐笔序号

uint64\_t timeStamp; //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm，精确到毫秒。

uint64\_t preClosePrice; //昨收价，实际值除以10000

uint64\_t numTrades; //总成交笔数

uint64\_t totalVolumeTrade; //总成交量，实际值除以100

uint64\_t totalValueTrade; //总成交金额，实际值除以10000

uint64\_t lastPrice; //最近价，实际值除以1000000

uint64\_t openPrice; //开盘价，实际值除以1000000

uint64\_t highPrice; //最高价，实际值除以1000000

uint64\_t lowPrice; //最低价，实际值除以1000000

uint64\_t upperlmtPrice; //涨停价，实际值除以1000000

uint64\_t lowerlmtPrice; //跌停价，实际值除以1000000

uint64\_t bidAvgPrice; //买入委托数量加权平均价，实际值除以1000000

uint64\_t bidTotalQty; //买入委托总数量，实际值除以100

uint64\_t askAvgPrice; //卖出委托数量加权平均价，实际值除以1000000

uint64\_t askTotalQty; //卖出委托总数量，实际值除以100

BidAskPriceQtySz bidInfo[10]; //10档申买信息

BidAskPriceQtySz askInfo[10]; //10档申卖信息

};

1. onBestOrdersSz方法

当API接收到深交订单明细行情时，该方法会被调用。

函数原型：

virtual void onBestOrdersSz(BestOrdersSz\* p);

参数：

参数一，深交订单明细行情

结构体定义：

struct BestOrdersSz {

uint8\_t messageType; //消息类型，订单明细为0x2

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

char recv; //保留字段

uint64\_t timeStamp; //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm, 精确到毫秒，示例：20190411102939000

uint8\_t side; //买卖标识:买：1，卖：2

uint8\_t number; //明细个数

uint64\_t price; //委托价格，实际值除以1000000

uint64\_t orders; //申买/卖数量,实际值除以100

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

char resv[2]; //保留字段

uint64\_t volume[0]; //委托数量，变长数组，最多揭示50笔,实际值除以100

};

1. onIndexSz方法

当API接收到深交指数行情时，该方法会被调用。

函数原型：

virtual void onIndexSz(IndexSz\* p);

参数：

参数一，深交指数行情

结构体定义：

struct IndexSz {

uint8\_t messageType; //消息类型，指数为0x3

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t resv; //保留字段

/\*

产品所处的交易阶段代码(仅深交所有效)

第0 位：

S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,

E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

第1 位：

0=正常状态,1=全天停牌

\*/

char tradingPhaseCode[8];

uint64\_t timeStamp; //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm, 精确到毫秒，示例：20190411102939000

uint64\_t preClosePrice; //前盘指数（来源扩展字段）,实际值除以1000000

uint64\_t openPrice; //开盘指数,实际值除以1000000

uint64\_t lastPrice; //最新指数,实际值除以1000000

uint64\_t highPrice; //最高指数,实际值除以1000000

uint64\_t lowPrice; //最低指数,实际值除以1000000

uint64\_t closePrice; //今日收盘指数,实际值除以1000000

uint64\_t tradeNum; //成交笔数

uint64\_t totalVolume; //成交总量,实际值除以100

uint64\_t totalValue; //成交总金额,实际值除以10000

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

};

1. onTradeSz方法

当API接收到深交逐笔成交行情时，该方法会被调用。

函数原型：

virtual void onTradeSz(TradeSz\* p);

参数：

参数一，深交逐笔成交行情

结构体定义：

struct TradeSz {

uint8\_t messageType; //消息类型，逐笔成交为0x4

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t execType; //成交类别 ： 0x1--撤销；0x2--成交

uint64\_t applSeqNum; //消息记录号

uint64\_t transactTime; //成交时间,格式YYYYMMDDHHMMSSmmm, 精确到毫秒，示例：20190411102957670

uint64\_t tradePrice; //成交价格,实际值除以10000

uint64\_t tradeQty; //成交数量,实际值除以100

uint64\_t bidapplSeqnum; //买方委托索引

uint64\_t offerapplSeqnum; //卖方委托索引

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

};

1. onOrderSz方法

当API接收到深交逐笔委托行情时，该方法会被调用。

函数原型：

virtual void onOrderSz(OrderSz\* p);

参数：

参数一，深交逐笔委托行情

结构体定义：

struct OrderSz {

uint8\_t messageType; //消息类型，逐笔委托为0x5

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

char rsvd; //保留字段

char side; //买卖方向：1=买，2=卖，G=借入，F=出借

char orderType; //订单类别：1=市价，2=限价，U=本方最优

uint64\_t applSeqNum; //消息记录号

uint64\_t transactTime; //委托时间,格式YYYYMMDDHHMMSSmmm, 精确到毫秒，示例：20190411102957670

uint64\_t price; //价格,实际值除以10000

uint64\_t qty; //数量，实际值除以100

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

};

1. onAfterSnapshotSz方法

当API接收到深交盘后快照行情时，该方法会被调用。

函数原型：

virtual void onAfterSnapshotSz (AfterSnapshotSz\* p);

参数：

参数一，深交盘后快照行情

结构体定义：

struct AfterSnapshotSz {

uint8\_t messageType; //消息类型，深交盘后快照为0x6

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

/\*

字段有效标识:

bit0:买入有效，bit1:卖出有效，其余位保留

\*/

uint8\_t flag;

/\*

产品所处的交易阶段代码

深交:

第0 位：S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

第1 位：0=正常状态,1=全天停牌

\*/

char tradingPhaseCode[8];

uint64\_t timeStamp; //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm, 精确到毫秒，示例：20190411102957670

uint64\_t preClosePrice; //昨收价，实际值除以1000000

uint64\_t bidPrice; //买入价格，实际值除以1000000

uint64\_t bidQty; //买入数量，实际值除以1000000

uint64\_t askPrice; //卖出价格，实际值除以1000000

uint64\_t askQty; //卖出数量，实际值除以1000000

uint64\_t numTrades; //总成交笔数

uint64\_t totalVolumeTrade; //总成交量,实际值除以100

uint64\_t totalValueTrade; //总成交金额，实际值除以10000

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

};

1. onL1MarketDataSnapSz方法

当API接收到深交L1快照行情时，该方法会被调用。

函数原型：

virtual void onL1MarketDataSnapSz(L1MarketDataSnapSz\* p);

参数：

参数一，深交L1快照行情

结构体定义：

struct L1MarketDataSnapSz {

uint8\_t messageType; //消息类型，深交L1快照为0xf

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t resv[2]; //保留字段

/\*

产品所处的交易阶段代码

深交:

第0 位：S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

第1 位：0=正常状态,1=全天停牌

\*/

char tradingPhaseCode[8];

uint8\_t resv2[7]; //保留字段

uint64\_t timeStamp; //数据生成时间（切片时间）,格式YYYYMMDDHHMMSSmmm, 精确到毫秒，示例：20190411102957670

uint64\_t preClosePrice; //昨收价（来源消息头),实际值除以10000

uint64\_t numTrades; //总成交笔数

uint64\_t totalVolumeTrade; //总成交量,实际值除以100

uint64\_t totalValueTrade; //总成交金额，实际值除以10000

uint64\_t lastPrice; //最近价，实际值除以1000000

uint64\_t openPrice; //开盘价，实际值除以1000000

uint64\_t openInterest; //合约持仓量，实际值除以100

uint64\_t highPrice; //最高价，实际值除以1000000

uint64\_t lowPrice; //最低价，实际值除以1000000

uint64\_t upperlmtPrice; //涨停价，实际值除以1000000

uint64\_t lowerlmtPrice; //跌停价，实际值除以1000000

uint64\_t bidAvgPrice; //买入委托数量加权平均价，实际值除以1000000

uint64\_t bidTotalQty; //买入委托总数量，实际值除以100

uint64\_t askAvgPrice; //卖出委托数量加权平均价，实际值除以1000000

uint64\_t askTotalQty; //卖出委托总数量，实际值除以100

BidAskPriceQtySz bidInfo[5]; //申买信息

BidAskPriceQtySz askInfo[5]; //申卖信息

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

uint32\_t resv3;              //保留字段

 uint64\_t IOPV;              //基金实时参考值，实际值除以1000000

 uint64\_t bidNumOfOrders[5];        //买5档价位总委托笔数

 uint64\_t askNumOfOrders[5];        //卖5档价位总委托笔数

};

1. onIOPVSz方法

当API接收到深交基金实时参考值行情时，该方法会被调用。

函数原型：

virtual void onIOPVSz(IOPVSnapshotSz\* p);

参数：

参数一，深交基金实时参考值行情

结构体定义：

struct IOPVSnapshotSz {

uint8\_t messageType; //消息类型，iopv为0x13

uint32\_t sequence; //udp输出包序号，从1开始

uint8\_t exchangeID; //交易所id，上交所：1，深交所：2

char securityID[9]; //证券代码

uint8\_t resv; //保留字段

/\*

\* 产品所处的交易阶段代码

\* 第0位：S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

\* 第1位：0=正常状态,1=全天停牌

\*/

char tradingPhaseCode[8];

uint64\_t timeStamp; //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm，精确到毫秒。

//示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

uint64\_t IOPV; //基金实时参考值，实际值除以1000000

uint16\_t channelNo; //频道代码

uint16\_t mdstreamid; //行情类别

};

1. onBlockTradeSz方法

当API接收到深交盘后大宗成交快照行情时，该方法会被调用。

函数原型：

virtual void onBlockTradeSz(BlockTradeSz\* p);

参数：

参数一，深交盘后大宗成交快照行情

结构体定义：

struct BlockTradeSz {

  uint8\_t  messageType;      //消息类型，盘后定价大宗交易为0x7

  uint32\_t  sequence;        //udp输出包序号，从1开始

  uint8\_t  exchangeID;       //交易所id，上交所：1，深交所：2

  char    securityID[9];    //证券代码

  uint8\_t  flag;          //字段有效标识：bit0，买入有效，bit1，卖出有效，其余位保留

  /\*

   \* 产品所处的交易阶段代码

   \* 深交所:

   \* 第0位：S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

   \* 第1位：0=正常状态,1=全天停牌

   \*/

  char tradingPhaseCode[8];

  uint64\_t timeStamp;     //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t preClosePrice;   //昨收价（来源消息头），实际值除以10000

  uint64\_t bidPrice;      //买入价格，实际值除以1000000

  uint64\_t bidQty;       //买入数量，实际值除以1000000

  uint64\_t askPrice;      //卖出价格，实际值除以1000000

  uint64\_t askQty;       //卖出数量，实际值除以1000000

  uint64\_t numTrades;     //总成交笔数

  uint64\_t totalVolumeTrade;  //总成交量，实际值除以100

  uint64\_t totalValueTrade;   //总成交金额，实际值除以10000

  uint16\_t channelNo;      //频道代码

  uint16\_t mdstreamid;     //行情类别

};

1. onBondSnapshotSz方法

当API接收到深交债券快照行情时，该方法会被调用。

函数原型：

virtual void onBondSnapshotSz(BondSnapshotSz\* p);

参数：

参数一，深交债券快照行情

结构体定义：

struct BondSubStatus {

  char     SubTradingPhaseCode[7];    //细分交易阶段(除byte0、byte1外，其余字节全部是0x00)

  uint8\_t  tradingType;           //交易方式

};

struct BondSnapshotSz {

  uint8\_t  messageType;     //消息类型，债券快照为0x0C

  uint32\_t  sequence;       //udp输出包序号，从1开始

  uint8\_t  exchangeID;      //交易所id，上交所：1，深交所：2

  char    securityID[9];   //证券代码

  uint8\_t  resv[2];        //保留字段

  /\*

   \* 产品所处的交易阶段代码

   \* 第0位：S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

   \* 第1位：0=正常状态,1=全天停牌

   \*/

  char tradingPhaseCode[8];

  uint8\_t resv2[7];       //保留字段

  uint64\_t timeStamp;      //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t preClosePrice;   //昨收价（来源消息头），实际值除以10000

  uint64\_t numTrades;      //总成交笔数

  uint64\_t totalVolumeTrade;   //总成交量，实际值除以100

  uint64\_t totalValueTrade;   //总成交金额，实际值除以10000

  uint64\_t lastPrice;      //最近价，实际值除以1000000

  uint64\_t openPrice;      //开盘价，实际值除以1000000

  uint64\_t resv3;        //保留字段

  uint64\_t highPrice;      //最高价，实际值除以1000000

  uint64\_t lowPrice;       //最低价，实际值除以1000000

  uint64\_t closePrice;      //收盘价，实际值除以1000000

  uint64\_t latestMatchPrice;   //匹配成交最近价，实际值除以1000000

  uint64\_t bidAvgPrice;    //买入委托数量加权平均价，实际值除以1000000

  uint64\_t bidTotalQty;     //买入委托总数量，实际值除以100

  uint64\_t askAvgPrice;     //卖出委托数量加权平均价，实际值除以1000000

  uint64\_t askTotalQty;     //卖出委托总数量，实际值除以100

  BidAskPriceQtySz bidInfo[SNAPSHOT\_LEVEL]; //申买信息

  BidAskPriceQtySz askInfo[SNAPSHOT\_LEVEL]; //申卖信息

  BondSubStatus   subStatus[5];       //细分交易状态

  uint64\_t auctionVolumeTrade; //匹配成交成交量，实际值除以100

  uint64\_t auctionValueTrade; //匹配成交成交金额，实际值除以10000

  uint16\_t channelNo;      //频道代码

  uint16\_t mdstreamid;     //行情类别

uint32\_t resv4;        //保留字段

  uint64\_t IOPV;        //基金实时参考值，实际值除以1000000

  uint64\_t bidNumOfOrders[SNAPSHOT\_LEVEL]; //买10档价位总委托笔数

  uint64\_t askNumOfOrders[SNAPSHOT\_LEVEL]; //卖10档价位总委托笔数

};

1. onBondBestOrdersSz方法

当API接收到深交债券订单明细行情时，该方法会被调用。

函数原型：

virtual void onBondBestOrdersSz(BondBestOrdersSz\* p);

参数：

参数一，深交债券订单明细行情

结构体定义：

struct BondBestOrdersSz {

  uint8\_t  messageType;  //消息类型，订单明细为0x0D

  uint32\_t sequence;    //udp输出包序号，从1开始

  uint8\_t  exchangeID;   //交易所id，上交所：1，深交所：2

  char   securityID[9]; //证券代码

  char   recv;  //保留字段

  uint64\_t timeStamp;  //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint8\_t  side;    //买卖标识：买：1，卖：2

  uint8\_t  number;   //明细个数

  uint64\_t price;    //委托价格，实际值除以1000000

  uint64\_t orders;   //申买/卖数量，实际值除以100

  uint16\_t channelNo;  //频道代码

  uint16\_t mdstreamid; //行情类别

  char   resv[2];   //保留字段

  uint64\_t volume[0];  //委托数量，变长，最多揭示50笔，实际值除以100

};

1. onBondTreeSnapSz方法

当API接收到深交债券逐笔构建快照行情时，该方法会被调用。

函数原型：

virtual void onBondTreeSnapSz(MarketDataTreeSnapSz\* p);

参数：

参数一，深交逐笔构建快照行情

1. onBondTradeSz方法

当API接收到深交债券逐笔成交行情时，该方法会被调用。

函数原型：

virtual void onBondTradeSz(BondTradeSz\* p);

参数：

参数一，深交债券逐笔成交行情

结构体定义：

struct BondTradeSz {

  uint8\_t  messageType;  //消息类型，债券逐笔成交为0x8

  uint32\_t sequence;    //udp输出包序号，从1开始

  uint8\_t  exchangeID;   //交易所id，上交所：1，深交所：2

  char   securityID[9]; //证券代码

  uint8\_t  execType;    //成交类别：0x1--撤销；0x2--成交

  uint64\_t applSeqNum;   //消息记录号

  uint64\_t transactTime;  //成交时间，格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t tradePrice;   //成交价格，实际值除以10000

  uint64\_t tradeQty;    //成交数量，实际值除以100

  uint64\_t bidapplSeqnum;  //买方委托索引

  uint64\_t offerapplSeqnum; //卖方委托索引

  uint16\_t channelNo;    //频道代码

  uint16\_t mdstreamid;    //行情类别

};

1. onBondBlockTradeSz方法

当API接收到深交债券大额逐笔成交行情时，该方法会被调用。

函数原型：

virtual void onBondBlockTradeSz(BondBlockTradeSz\* p);

参数：

参数一，深交债券大额逐笔成交行情

结构体定义：

struct BondBlockTradeSz {

  uint8\_t  messageType;   //消息类型，债券大额逐笔成交为0x9

  uint32\_t sequence;     //udp输出包序号，从1开始

  uint8\_t  exchangeID;    //交易所id，上交所：1，深交所：2

  char   securityID[9];  //证券代码

  uint8\_t  execType;     //成交类别：0x1--撤销；0x2--成交

  uint64\_t applSeqNum;    //消息记录号

  uint64\_t transactTime;   //成交时间，格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t tradePrice;    //成交价格，实际值除以10000

  uint64\_t tradeQty;     //成交数量，实际值除以100

  uint64\_t bidapplSeqnum;  //买方委托索引

  uint64\_t offerapplSeqnum; //卖方委托索引

  uint8\_t  settlePeriod;  //结算周期

  uint16\_t settleType;    //结算方式

  uint16\_t channelNo;    //频道代码

  uint16\_t mdstreamid;    //行情类别

};

1. onBondBidTradeSz方法

当API接收到深交债券竞买逐笔成交行情时，该方法会被调用。

函数原型：

virtual void onBondBidTradeSz (BondBidTradeSz\* p);

参数：

参数一，深交债券竞买逐笔成交行情

结构体定义：

struct BondBidTradeSz {

  uint8\_t  messageType;    //消息类型，债券大额逐笔成交为0x11

  uint32\_t sequence;     //udp输出包序号，从1开始

  uint8\_t  exchangeID;    //交易所id，上交所：1，深交所：2

  char   securityID[9];   //证券代码

  uint8\_t  execType;     //成交类别：0x1--撤销；0x2--成交

  uint64\_t applSeqNum;    //消息记录号

  uint64\_t transactTime;   //成交时间，格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t tradePrice;    //成交价格，实际值除以10000

  uint64\_t tradeQty;     //成交数量，实际值除以100

  uint64\_t bidapplSeqnum;   //买方委托索引

  uint64\_t offerapplSeqnum;  //卖方委托索引

  uint64\_t marginPrice;    //成交的边际价格，实际值除以10000

  char   secondaryOrderID[17]; //竞买场次编号

  uint8\_t  settlePeriod;   //结算周期

  uint16\_t settleType;    //结算方式

  uint16\_t bidExecInstType;  //竞买成交方式

  uint16\_t channelNo;     //频道代码

  uint16\_t mdstreamid;    //行情类别

};

1. onBondOrderSz方法

当API接收到深交债券逐笔委托行情时，该方法会被调用。

函数原型：

virtual void onBondOrderSz(BondOrderSz\* p);

参数：

参数一，深交债券逐笔委托行情

结构体定义：

struct BondOrderSz {

  uint8\_t  messageType;  //消息类型，债券逐笔委托为0x0A

  uint32\_t  sequence;    //udp输出包序号，从1开始

  uint8\_t  exchangeID;   //交易所id，上交所：1，深交所：2

  char    securityID[9]; //证券代码

  char rsvd;        //保留字段

  char side;        //买卖方向：1=买，2=卖，G=借入，F=出借

  char  orderType;     //订单类别：1=市价，2=限价，U=本方最优

  uint64\_t applSeqNum;   //消息记录号

  uint64\_t transactTime;  //委托时间，格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t price;      //价格，实际值除以10000

  uint64\_t qty;       //数量，实际值除以100

  uint16\_t channelNo;    //频道代码

  uint16\_t mdstreamid;   //行情类别

};

1. onBondBlockOrderSz方法

当API接收到深交债券大额逐笔委托行情时，该方法会被调用。

函数原型：

virtual void onBondBlockOrderSz(BondBlockOrderSz\* p);

参数：

参数一，深交债券大额逐笔委托行情

结构体定义：

struct BondBlockOrderSz {

  uint8\_t  messageType;   //消息类型，债券逐笔委托为0x0B

  uint32\_t sequence;     //udp输出包序号，从1开始

  uint8\_t  exchangeID;    //交易所id，上交所：1，深交所：2

  char   securityID[9];  //证券代码

  char  rsvd[2];     //保留字段

  char  side;       //买卖方向：1=买，2=卖，G=借入，F=出借

  uint64\_t applSeqNum;    //消息记录号

  uint64\_t transactTime;   //委托时间，格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t price;      //价格，实际值除以10000

  uint64\_t qty;       //数量，实际值除以100

  char   quoteID[11];   //报价消息编号

  char   memberID[7];   //交易商代码

  char   investorType[3]; //交易主体类型

  char   investorID[11];  //交易主体代码

  char   investName[121]; //客户名称

  char   traderCode[9];  //交易员代码

  uint8\_t  settlePeriod;   //结算周期

  uint16\_t settleType;    //结算方式

  uint16\_t channelNo;    //频道代码

  uint16\_t mdstreamid;    //行情类别

};

1. onBondBidOrderSz方法

当API接收到深交债券竞买逐笔委托行情时，该方法会被调用。

函数原型：

virtual void onBondBidOrderSz (BondBidOrderSz\* p);

参数：

参数一，深交债券竞买逐笔委托行情

结构体定义：

struct BondBidOrderSz {

  uint8\_t  messageType;   //消息类型，债券逐笔委托为0x12

  uint32\_t sequence;     //udp输出包序号，从1开始

  uint8\_t  exchangeID;    //交易所id，上交所：1，深交所：2

  char   securityID[9];  //证券代码

  char  rsvd[2];     //保留字段

  char  side;       //买卖方向：1=买，2=卖，G=借入，F=出借

  uint64\_t applSeqNum;    //消息记录号

  uint64\_t transactTime;   //委托时间，格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t price;      //价格，实际值除以10000

  uint64\_t qty;       //数量，实际值除以100

  char   rsvd2[11];    //保留字段

  char   memberID[7];   //交易商代码

  char   investorType[3]; //交易主体类型

  char   investorID[11];  //交易主体代码

  char   investName[121]; //客户名称

  char   traderCode[9];  //交易员代码

  uint16\_t bidTransType;   //竞买业务类别

  uint16\_t bidExecInstType; //竞买成交方式

  uint64\_t lowLimitPrice;  //价格下限，实际值除以10000

  uint64\_t highLimitPrice;  //价格上限，实际值除以10000

  uint64\_t minQty;      //最低成交数量，实际值除以100

  char   secondaryOrderID[17]; //竞买场次编号

  uint32\_t tradeData;    //成交日期

  uint8\_t  settlePeriod;   //结算周期

  uint16\_t settleType;    //结算方式

  uint16\_t channelNo;    //频道代码

  uint16\_t mdstreamid;    //行情类别

};

1. onL1BondSnapSz方法

当API接收到深交债券竞买逐笔委托行情时，该方法会被调用。

函数原型：

virtual void onL1BondSnapSz (L1BondSnapSz\* p);

参数：

参数一，深交债券竞买逐笔委托行情

结构体定义：

struct L1BondSnapSz {

  uint8\_t  messageType;    //消息类型，深交债券L1快照为0x0E

  uint32\_t  sequence;      //udp输出包序号，从1开始

  uint8\_t  exchangeID;     //交易所id，上交所：1，深交所：2

  char    securityID[9];   //证券代码

  uint8\_t  resv[2];      //保留字段

  /\*

   \* 产品所处的交易阶段代码

   \* 深交所:

   \* 第0位：S=启动（开市前）,O=开盘集合竞价,T=连续竞价,B=休市,C=收盘集合竞价,E=已闭市,H=临时停牌,A=盘后交易,V=波动性中断

   \* 第1位：0=正常状态,1=全天停牌

   \*/

  char  tradingPhaseCode[8];

  uint8\_t  resv2[7];      //保留字段

  uint64\_t timeStamp;      //数据生成时间（切片时间），格式YYYYMMDDHHMMSSmmm，精确到毫秒。示例：20190411102939120表示2019年04月11日10点29分39秒120毫秒

  uint64\_t preClosePrice;    //昨收价（来源消息头），实际值除以10000

  uint64\_t numTrades;      //总成交笔数

  uint64\_t totalVolumeTrade;  //总成交量，实际值除以100

  uint64\_t totalValueTrade;   //总成交金额，实际值除以10000

  uint64\_t lastPrice;      //最近价，实际值除以1000000

  uint64\_t openPrice;      //开盘价，实际值除以1000000

  uint64\_t resv3;        //保留字段

  uint64\_t highPrice;      //最高价，实际值除以1000000

  uint64\_t lowPrice;      //最低价，实际值除以1000000

  uint64\_t upperlmtPrice;    //涨停价，实际值除以1000000

  uint64\_t lowerlmtPrice;    //跌停价，实际值除以1000000

  uint64\_t bidAvgPrice;     //买入委托数量加权平均价，实际值除以1000000

  uint64\_t bidTotalQty;     //买入委托总数量，实际值除以100

  uint64\_t askAvgPrice;     //卖出委托数量加权平均价，实际值除以1000000

  uint64\_t askTotalQty;     //卖出委托总数量，实际值除以100

  BidAskPriceQtySz bidInfo[5]; //申买信息

  BidAskPriceQtySz askInfo[5]; //申卖信息

  BondSubStatus  subStatus[5]; //细分交易状态

  uint64\_t auctionVolumeTrade; //匹配成交成交量

  uint64\_t auctionValueTrade;  //匹配成交成交金额

  uint16\_t channelNo;      //频道代码

  uint16\_t mdstreamid;     //行情类别

uint32\_t resv4;        //保留字段

  uint64\_t IOPV;        //基金实时参考值，实际值除以1000000

  uint64\_t bidNumOfOrders[5];  //买5档价位总委托笔数

  uint64\_t askNumOfOrders[5];  s//卖5档价位总委托笔数

};

1. 开发样例

具体样例请查看发布包中demo工程。

1. 系统调优

因Level2行情数据量巨大，网络负担重，建议客户端在使用Level2API前进行调优。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，通过一些调优手段，可最大限度的避免或减少丢包现象。

目前艾科的行情API支持普通网卡模式、SF网卡EFVI模式、Exablaze网卡exanic模式接收UDP行情数据，数据量较大时，普通网卡容易出现丢包的情况。因此，建议使用SF网卡或Exablaze网卡，同时也建议服务器CPU主频3.4G以上，避免行情密集时来不及处理导致丢包。常用调优方法如下：

1. 减少系统中断影响和cpu切换

措施：隔核绑核，用API每个通道绑定不同的核，并预先对要绑的核做隔核。

sa.查询隔核工具是否已安装

rpm -qa | grep tuned

如果已安装，会出现类似如下信息：

tuned-2.7.1-5.el7.centos.xele.noarch

tuned-profiles-realtime-2.7.1-5.el7.centos.xele.noarch

b.配置隔阂，隔阂不能隔0核

vi /etc/tuned/realtime-variables.conf

# Examples:

# isolated\_cores=2,4-7

isolated\_cores=6-7

c.重启服务器

1. 降低行情延时

措施：将API接收模式设为非阻塞模式，通过CPU高占用率降低行情延时，即行情初始话时m\_polling参数设为 true

1. 提升服务器性能

措施：修改服务器模式为实时模式或其他高性能模式，具体操作指令代码为“tuned-adm profile network-latencys”。

1. 优化内存性能

措施一：打开操作系统中的“巨页”功能，具体操作步骤：

1.在“/etc/sysctl.conf”文件尾部添加“vm.nr\_hugepages=2048”

2.执行操作指令“sysctl  -p”，让修改生效

3.最后执行操作指令“cat  /proc/meminfo | grep Huge”， 验证修改效果

措施二：确保绑核的cpu和接收网卡在同一个numa节点下，具体操作指令代码：

1.执行操作指令“cat /sys/class/net/p2p1/device/numa\_node”（p2p1具体以实际接收行情网卡名为准），查看网卡numa。

2.执行操作指令“lscpu”查看cpu numa，从而绑核时选择和网卡同一个numa的cpu核心

1. 提高CPU单核性能

措施：关闭CPU的超线程功能

1. 增大网卡缓冲区

措施：修改网卡缓冲区，具体操作步骤：

a.查看RX最大是多少，具体执行操作指令“ethtool -g p2p1”（p2p1具体以实际接收行情网卡名为准） ,

b.通过 执行操作指令“ethtool -G p2p1 rx 4096”把RX设成最大

c. 执行“ethtool -C p2p1 rx-usecs 0 rx-usecs-irq 0 adaptive-rx off” 修改网卡聚合信息

1. 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 |

建议增加接收数据包队列到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

1. api补充说明
2. 逐笔保序

API自带了逐笔保序功能，该功能默认不开启，开启该功能需要确保股票的逐笔（消息类型0x04,0x05）在同一个组播中，同时债券的逐笔也在同一个组播中，股票和债券的逐笔组播可以相同，也可以不同。

通过修改“XELE\_MD\_KEEP\_ORDER”环境变量决定是否开启保序功能：

0：上交、深交均不保序

1：上交保序，深交不保序

2：深交保序、上交不保序

3：上交、深交均保序

乱序发生时默认最多等待10000条同一通道的行情，如果等不到想要的行情，则输出已经收到的行情。等待条数可以通过“export XELE\_MD\_WAIT\_CNT=xxx”修改，xxx是一个整数。