

XTP 行情服务接入前指引

一、XTP 行情接入要点概述

XQuote 是 XTP 自主开发的行情解码、数据推送的服务器。

支持的行情源包括 SHL1, SHL2, SZL1, SZL2 行情及北交所行情。

用户可以通过 API 获取股票、基金、债券、指数、期权行情。

目前 XQuote 只提供实时行情，不提供历史行情。

行情推送方式有两种：TCP 推送和 UDP（组播）推送（生产环境使用 UDP 推送）。

推送的数据包括：行情快照、买一卖一委托队列、逐笔及订单簿。

XQuote 提供按市场订阅和按股票代码订阅两种模式。

因为实盘的数据量，比公网测试环境的数据量大得多，所以**用户在接入实盘前，一定要在实盘上做接入测试。**

接入前请下载最新的 API，网址为 <https://xtp.zts.com.cn/download.html>。请勿使用 1.1.14.x 版本的 API，该版本不能用来登录行情服务器。

二、基础操作

1、登录

(1) API 使用 `Login` 函数进行登录。`Login` 函数的 `sock_type` 参数，决定了 API 接收更新数据的方式。请务必与行情服务器一致，否则会收不到行情更新数据。

```
///用户登录请求
///@return 登录是否成功，“0”表示登录成功，“-1”表示连接服务器出错，此时用户可以调用GetApiLastError()来获取错误代码
///@param ip 服务器ip地址，类似“127.0.0.1”
///@param port 服务器端口号
///@param user 登陆用户名
///@param password 登陆密码
///@param sock_type “1”代表TCP，“2”代表UDP
///@param local_ip 本地网卡地址，类似“127.0.0.1”
///@remark 此函数为同步阻塞式，不需要异步等待登录成功，当函数返回即可进行后续操作，此api只能有一个连接
virtual int Login(const char* ip, int port, const char* user, const char* password, XTP_PROTOCOL_TYPE sock_type,
                  const char* local_ip = NULL);
```

(2) API 日志级别

```
///创建QuoteApi
///@param client_id （必须输入）用于区分同一用户的不同客户端，由用户自定义
///@param save_file_path （必须输入）存储订阅信息文件的目录，请设定一个有可写权限的真实存在的路径，如果路径不存在的话，可能会因为
///@param log_level 日志输出级别
///@return 创建出的UserApi
///@remark 如果一个账户需要在多个客户端登录，请使用不同的client_id，系统允许一个账户同时登录多个客户端，但是对于同一账户，相同的c
static QuoteApi *CreateQuoteApi(uint8_t client_id, const char* save_file_path, XTP_LOG_LEVEL log_level=XTP_LOG_LEVEL_DEBUG);
```

可以通过 `CreateQuoteApi` 的 `log_level` 参数来调整 API 的日志级别，默认为 `DEBUG`。但 `DEBUG` 级别的日志会非常多，请慎重使用。

(3) 一个账号在多个客户端登录

XTP 的行情是允许一个账号在多个客户端同时登录的。只需要将 `CreateQuoteApi` 的第 `client_id` 参数设置为不同的值即可。

2、查询

(1) 查询可交易合约信息

```

///获取当前交易日合约部分静态信息
///@return 发送查询请求是否成功，“0”表示发送查询请求成功，非“0”表示发送查询请求不成功
///@param exchange_id 交易所代码，必须提供 1-上海 2-深圳
virtual int QueryAllTickers(XTP_EXCHANGE_TYPE exchange_id);

///查询合约部分静态信息的应答
///@param ticker_info 合约部分静态信息
///@param error_info 查询合约部分静态信息时发生错误时返回的错误信息，当error_info为空，或者error_info
///@param is_last 是否此次查询合约部分静态信息的最后一个应答，当为最后一个的时候为true，如果为false，
virtual void OnQueryAllTickers(XTPQSI* ticker_info, XTPRI *error_info, bool is_last);

```

调用 **QueryAllTickers**，查询对应市场可交易合约（股票/基金/债券/期权）基本信息。

行情服务器内部对从数据库查询的到 ticker 进行过滤，过滤条件为(trade_status=0)，所以该接口，**查询不到**指数的信息。

当行情服务器返回数据时，**OnQueryAllTickers** 函数被调用（异步接口）。

注意：目前（2018-12）生产环境，把停牌股票的 trade_status 配置成了 0，即该接口也返回停牌股票。

返回的数据类型列表：

返回类型	含义	备注
XTP_TICKER_TYPE_STOCK(0)	股票	主板/中小板/创业板
XTP_TICKER_TYPE_TECH_STOCK(5)	科创板	科创板(上海)
XTP_TICKER_TYPE_BOND(3)	债券	国债/企业债/公司债/转换债券/国债逆回购
XTP_TICKER_TYPE_FUND(2)	基金	ETF/分级基金/货币基金/深交所仅申赎基金/上交所货币基金/深交所货币基金
XTP_TICKER_TYPE_OPTION(4)	期权	个股期权/ETF 期权
XTP_TICKER_TYPE_UNKNOWN(6)		客户应该过滤掉返回类型为 6 的 ticker
XTP_TICKER_TYPE_INDEX(1)	指数	本接口不返回指数

(2) 查询最新价接口

```

///获取合约的最新价格信息
///@return 发送查询请求是否成功，“0”表示发送查询请求成功，非“0”表示发送查询请求不成功
///@param ticker 合约ID数组，注意合约代码必须以'\0'结尾，不包含空格
///@param count 要查询的合约个数
///@param exchange_id 交易所代码
virtual int QueryTickersPriceInfo(char *ticker[], int count, XTP_EXCHANGE_TYPE exchange_id);

///获取所有合约的最新价格信息
///@return 发送查询请求是否成功，“0”表示发送查询请求成功，非“0”表示发送查询请求不成功
virtual int QueryAllTickersPriceInfo();

///查询合约的最新价格信息应答
///@param ticker_info 合约的最新价格信息
///@param error_info 查询合约的最新价格信息时发生错误时返回的错误信息，当error_info为空，或者error_info.error
///@param is_last 是否此次查询的最后一个应答，当为最后一个的时候为true，如果为false，表示还有其他后续消息响应
virtual void OnQueryTickersPriceInfo(XTPPTI* ticker_info, XTPRI *error_info, bool is_last);

```

该接口可以返回股票/基金/债券/期权等最新价格。

需要注意的是：

- (1) 请使用“订阅/推送”的方式来获取行情最新价格，不要频繁调用查询接口。
- (2) 服务器对“全查询或超过 3000 只股票的查询”频率做了限制。

3、订阅

目前我们支持**按股票代码**和**按市场**进行订阅。快照、逐笔、订单簿(OB)使用不同的订阅接口。

生产环境使用 **UDP 组播**，推荐客户按**市场**进行订阅。

SubscribeAllMarketData 、 SubscribeAllOptionMarketData 、 SubscribeAllTickByTick 、
SubscribeAllOrderBook

SubscribeMarketData、SubscribeTickByTick、SubscribeOrderBook

例如：使用 SubscribeMarketData 订阅快照行情：

```
///订阅行情，包括股票、指数和期权。
///@return 订阅接口调用是否成功，“0”表示接口调用成功，非“0”表示接口调用出错
///@param ticker 合约ID数组，注意合约代码必须以'\0'结尾，不包含空格
///@param count 要订阅/退订行情的合约个数
///@param exchange_id 交易所代码
///@remark 可以一次性订阅同一证券交易所的多个合约，无论用户因为何种问题需要重新登录行情服务器，都需要重新订阅行情
virtual int SubscribeMarketData(char *ticker[], int count, XTP_EXCHANGE_TYPE exchange_id);
```

订阅函数，传入需要订阅的股票代码，订阅股票的个数，以及市场。注意**市场**不要写错误。如股票代码同样是“000001”，深交所表示平安银行，上交所表示上证指数。

OnSubMarketData 返回订阅结果。

```
///订阅行情应答，包括股票、指数和期权
///@param ticker 详细的合约订阅情况
///@param error_info 订阅合约发生错误时的错误信息，当error_info为空，或者error_info.error_id为0时，表明没有错误
///@param is_last 是否此次订阅的最后一个应答，当为最后一个的时候为true，如果为false，表示还有其他后续消息响应
///@remark 每条订阅的合约均对应一条订阅应答，需要快速返回，否则会堵塞后续消息，当堵塞严重时，会触发断线
virtual void OnSubMarketData(XTPST *ticker, XTPRI *error_info, bool is_last);
```

注意：

(1) 生产环境 L1/L2 行情服务器均使用 UDP 推送模式，生产环境推荐使用全市场订阅接口。

(2) 测试环境使用 TCP 的方式推送行情。由于带宽的限制，请客户只订阅少量股票，否则会出现推送延时、断线等问题。

4、推送

订阅成功后，当有行情服务器有数据推过来时，**OnDepthMarketData、OnSubOrderBook、OnTickByTick** 会被回调。用户就可以从该函数中取数据了。

在这些回调函数中，**请不要使用耗时操作，需尽快返回**。否则会造成 API 来不及接收数据而丢行情。

```

///深度行情通知, 包含买一卖一队列
///@param market_data 行情数据
///@param bidl_qty 买一队列数据
///@param bidl_count 买一队列的有效委托笔数, 即bidl_qty数组的长度, 最大为50
///@param max_bidl_count 买一队列总委托笔数
///@param askl_qty 卖一队列数据
///@param askl_count 卖一队列的有效委托笔数, 即askl_qty数组的长度, 最大为50
///@param max_askl_count 卖一队列总委托笔数
///@remark 需要快速返回, 否则会堵塞后续消息, 当堵塞严重时, 会触发断线
virtual void OnDepthMarketData(XTPMD *market_data, int64_t bidl_qty[], int32_t bidl_count, int32_t max_bidl_count,
                                int64_t askl_qty[], int32_t askl_count, int32_t max_askl_count);

```

三、数据结构及字段含义

1、行情快照（XTPMD）赋值字段及含义

有关 ticker 代码及其含义的说明：

<http://www.sse.com.cn/market/price/report/>

<http://www.szse.cn/market/trend/>

从 API 2.2.32.2 开始, XTPMarketDataStruct 结构体, 增加 XTP_MARKETDATA_TYPE_V2 data_type_v2 字段, 来表明是哪种数据类型。

(1) 指数字段说明

data_type_v2 = XTP_MARKETDATA_V2_INDEX;

- exchange_id, 交易所 ID, 枚举值
- ticker, 股票代码, 6 位
- last_price, 最新
- pre_close_price, 昨日收盘
- open_price, 今日开盘
- high_price, 最高
- low_price, 最低
- close_price, 今收。上交所有该字段（大于 0 该值有效）。深交所无该字段, xquote 将其赋值为 last_price。

- data_time
 - SHL1 取自每行的时戳
 - SHL2 取自 FAST 层的时戳(非 STEP 层)
 - SZ 直接由深交所给出
- qty, 成交量, 上交所单位手, 深交所单位股
- turnover, 成交金额, 单位元
- trades_count, 成交笔数, SH 无意义（填 0）, SZ 有值

(2) 股票/基金字段说明

股票的价格精度为 0.01 元, 债券为 0.001 元

data_type_v2 = XTP_MARKETDATA_V2_ACTUAL;

- exchange_id, 交易所 ID, 枚举值
- ticker, 股票代码, 6 位
- last_price, 最新
- pre_close_price, 昨收
- open_price, 今开
- high_price, 最高
- low_price, 最低
- close_price, 今收。上交所有该字段（大于 0 该值有效）。深交所无该字段，xquote 将其赋值为 last_price。

深交所的今收，是在收盘后，通过文件(cashsecurityclosemd_*.xml)的形式下发的。

- data_time
 - SHL1 取自每行的时戳（非第一行 header 的时戳），含义为最近一次的成交时间
 - SHL2 取自 FAST 层的时戳（非 STEP 层），含义为全市场行情时间，会持续更新
 - SZ 直接由深交所给出
- upper_limit_price/lower_limit_price, 涨/跌停价。SZ 该值由深交所实时给出；SH 来源于初始化文件（上交所没有实时给出该值）。
- qty, 成交量，单位股
- turnover, 成交金额，单位元
- avg_price, 无意义
- bid[10]/ask[10], 买/卖价
 - L1 数据源，只显示 5 档。L2 数据源，显示 10 档。
 - 单位元。
- bid_qty[10]/ask_qty[10], 买/卖量
 - L1 数据源，只显示 5 档。L2 数据源，显示 10 档。
 - 单位股。
- trades_count, 成交笔数。SHL1 无意义（填 0），SHL2/SZL1/SZL2 有值。
- ticker_status, 当前交易状态及标志，8 字节。

第 0 位：

- 'S', 启动（开市前）时段(SH/SZ)
- 'C', 集合竞价(SH/SZ)
- 'T', 连续竞价(SH/SZ)
- 'B', 休市（SZ）
- 'E', 闭市(SH/SZ)
- 'P', 产品停牌(SH/SZ)
- 'M', 表示可恢复交易的熔断时段（盘中集合竞价）(SH)
- 'N', 表示不可恢复交易的熔断时段（暂停交易至闭市）(SH)
- 'U', 表示收盘集合竞价时段(SH)
- 'D', 开盘结合竞价结束到连续竞价开始前（SH）
- 'A', 盘后交易（SZ）
- 'V', 波动性中断（SZ，SH 期权）

第 1 位：

- '0', 此产品不可正常交易

- ‘1’，此产品可以正常交易
- 无意义填充格

第 2 位：

- ‘0’，未上市
- ‘1’，已上市

第 3 位：

- ‘0’，此产品在当前时段，不接受进行新订单申报
- ‘1’，此产品在当前时段，可接受进行新订单申报
- 无意义填充格

深交所只有第 0、1 位，没有第 2、3 位

上海	0	9:15	9:25	9:30	11:30	13:00	14:57	15:00
市场	-	-	-	-	-	-	-	-
	9:15	9:25	9:30	11:30	13:00	14:57	15:00	
非停牌	S 11	C111	T111	T111	T111	T111	U111	E111
停牌	P011	P011	P011	P011	P011	P011	P011	P011

上海	0		9:25	9:30	10:03	11:30	13:00	14:57	15:00
市场	-	9:25	-	-	-	-	-	-	-
	9:25		9:30	10:00	11:30	13:00	14:57	15:00	
SH603192		C111	T111	P011	T111	T111	T111	U111	E111
2018. 08. 28									
[9:30-10:00]									
盘中停牌									

注意：从 2018-12 月份开始，上交所停止发送开盘集合竞价消息（UA3107）。开盘/收盘集合竞价消息，直接从 UA3202 发出。

深圳	0	9:15	9:25	9:30	11:30	13:00	14:57	15:00
市场	-	-	-	-	-	-	-	-
	9:15	9:25	9:30	11:30	13:00	14:57	15:00	
非停牌	S1	C1	B1	T1	B1	T1	C1	E1
停牌	S0	C0	B0	T0	B0	T0	C0	E0

深圳	0	9:15	9:25	9:30	9:30	10:00	11:30	13:00	14:57	15:00
市场	-	-	-	-	-	-	-	-	-	-
	9:15	9:25	9:30		10:00	11:30	13:00	14:57	15:00	
SZ300747	S1	C1	B1	T1	V1	T1	B1	T1	C1	E1
2018.05.25										
[9:30-10:00]										
盘中停牌										

- data_type, 股票时为 0。
- stk, 股票扩展字段。
 - total_bid_qty, 委托买入总量, SHL2/SZ 有值。
 - total_ask_qty, 委托卖出总量, SHL2/SZ 有值。
 - ma_bid_price, 加权平均委买价格, SHL2/SZ 有值。
 - ma_ask_price, 加权平均委卖价格, SHL2/SZ 有值。
 - ma_bond_bid_price, 债券加权平均委买价格, SHL2 有值。
 - ma_bond_ask_price, 债券加权平均委卖价格, SHL2 有值。
 - yield_to_maturity, 债券到期收益率, SHL2 有值。
 - iopv, 基金实时参考净值, SH/SZ 有值。
 - etf_buy_count, ETF 申购笔数, SHL2 有值。
 - etf_sell_count, ETF 赎回笔数, SHL2 有值。
 - etf_buy_qty, ETF 申购数量, SHL2 有值。
 - etf_buy_money, ETF 申购数量, SHL2 有值。
 - etf_sell_qty, ETF 赎回数量, SHL2 有值。
 - etf_sell_money, ETF 赎回金额, SHL2 有值。
 - total_warrant_exec_qty, 权证执行的总数量, SHL2 有值。
 - warrant_lower_price, 权证跌停价格, SHL2 有值。
 - warrant_upper_price, 权证涨停价格, SHL2 有值。
 - cancel_buy_count, 买入撤单笔数, SHL2 有值。
 - cancel_sell_count, 卖出撤单笔数, SHL2 有值。
 - cancel_buy_qty, 买入撤单数量, SHL2 有值。
 - cancel_sell_qty, 卖出撤单数量, SHL2 有值。
 - cancel_buy_money, 买入撤单金额, SHL2 有值。
 - cancel_sell_money, 卖出撤单金额, SHL2 有值。
 - total_buy_count, 买入总笔数, SHL2 有值。
 - total_sell_count, 卖出总笔数, SHL2 有值。
 - duration_after_buy, 买入委托成交最大等待时间, SHL2 有值。
 - duration_after_sell, 卖出委托成交最大等待时间, SHL2 有值。
 - num_bid_orders, 买方委托价位数, SHL2 有值。
 - num_ask_orders, 卖方委托价位数, SHL2 有值。
 - pre_iopv, 基金 T-1 日净值。SZ 有值。

（3）期权字段说明

期权的价格精度为 0.0001 元

`data_type_v2 = XTP_MARKETDATA_V2_OPTION;`

- `exchange_id`, 交易所 ID, 枚举值
- `ticker`, 股票代码, 8 位
- `last_price`, 最新成交价
- `pre_close_price`, 昨日收盘价
 - SH mkttdt 数据源, 在行情文件中不提供, 初始化时从 `reff03MMDD.txt` 取;
 - SH LDDS 数据源, 在流式行情中提供该字段(`PrevClosePx`);
 - SZ 数据源, 在流式行情中提供该字段(`PrevClosePx`).
- `open_price`, 今开
- `high_price`, 当日最高价
- `low_price`, 当日最低价
- `close_price`, 今收价, SH mkttdt 总为 0。SH LDDS 大于 0 该值有效 (观察到盘中为 0, 收盘后大于 0)。SZ 该值总等于 `last_price`。
- `pre_total_long_positon`, SH 和 SZ 市场, 该值总为 0
- `total_long_positon`, 未平仓合约数量 (张或股)
- `pre_settl_price`, 昨日结算价
- `settl_price`, 今日结算价, SH mkttdt 该值为 0, SH LDDS 数据源有值; SZ 数据源该值为 0
- `upper_limit_price/lower_limit_price`, 涨/跌停价。SH 来源于初始化文件 (上交所没有实时给出该值)。
- `data_time`
 - SH mkttdt 数据源, 取自每行的时戳 (观察到为上交所最近一次的成交时间, 若委托发生变化, 该值不变)
 - SH LDDS 数据源, 取自 FAST 层最里面的时戳 (观察到为上交所最近一次的成交时间, 若委托发生变化, 该值不变)
- `qty`, 当日累计成交量
- `turnover`, 当日累计成交金额
- `avg_price`, 均价, 计算方法为 `turnover/qty`, 无意义
- `bid[10]/ask[10]`, 买/卖价, SH/SZ 均只有 5 档
- `bid_qty[10]/ask_qty[10]`, 买卖量, SH/SZ 均只有 5 档
- `trades_count`, 成交笔数, SH mkttdt 数据源为 0, SH LDDS 数据源有值
- `ticker_status`, 当前交易状态及标志, 8 字节

上交所标志含义

第 0 位:

- 'S', 启动 (开市前) 阶段
- 'C', 集合竞价
- 'T', 连续交易
- 'B', 休市
- 'E', 闭市
- 'V', 波动性中断
- 'P', 临时停牌
- 'U', 收盘集合竞价

- ‘M’，可恢复交易的熔断（盘中集合竞价）
- ‘N’，不可恢复交易的熔断（暂停交易至闭市）

第 1 位：

- ‘0’，未连续停牌
- ‘1’，连续停牌
- 预留则填空格

第 2 位：

- ‘0’，不限制开仓
- ‘1’，限制备兑开仓
- ‘2’，卖出开仓
- ‘3’，限制卖出开仓、备兑开仓
- ‘4’，限制买入开仓
- ‘5’，限制买入开仓、备兑开仓
- ‘6’，限制买入开仓、卖出开仓
- ‘7’，限制买入开仓、卖出开仓、备兑开仓

第 3 位：

- ‘0’，此产品在当前时段不接受进行新订单申报
- ‘1’，此产品在当前时段可接受进行新订单申报

深交所标志含义

同股票字段说明

- data_type，期权为 1
- opt，期权扩展字段（仅 SH 市场有意义，SZ 市场值均无意义）
 - auction_price，波段性中断参考价
 - auction_qty，波段性中断集合竞价虚拟匹配量
 - last_enquiry_time，最近询价时间，格式为 YYYYMMDDHHMMSSsss，若无询价

时，HHMMSSsss 部分为 0。总为 0。

（4）债券字段说明

`data_type_v2 = XTP_MARKETDATA_V2_BOND;`

从 2021.10.25 后，上交所启用新的债券交易系统，修改了债券行情的协议。

上交所 L1 时，ticker_status 有值；

上交所 L2 时，ticker_status 无意义；bond.instrument_status 有值；

深交所 L1/L2 时，ticker_status 有值。

（5）买一卖一委托队列字段及含义

L1 没有委托队列，L2 有委托队列。

在订阅行情快照后，若有行情数据推过来，`OnDepthMarketData` 函数就会被回调。

```

///深度行情通知, 包含买一卖一队列
///@param market_data 行情数据
///@param bidl_qty 买一队列数据
///@param bidl_count 买一队列的有效委托笔数, 即bidl_qty数组的长度, 最大为50
///@param max_bidl_count 买一队列总委托笔数
///@param askl_qty 卖一队列数据
///@param askl_count 卖一队列的有效委托笔数, 即askl_qty数组的长度, 最大为50
///@param max_askl_count 卖一队列总委托笔数
///@remark 需要快速返回, 否则会堵塞后续消息, 当堵塞严重时, 会触发断线
virtual void OnDepthMarketData(XTPMD *market_data, int64_t bidl_qty[], int32_t bidl_count, int32_t max_bidl_count,
int64_t askl_qty[], int32_t askl_count, int32_t max_askl_count);

```

卖十	9.20	686170	卖一队列 (28/28)
卖九	9.19	1003328	75136 600 1300 200 300 1900
卖八	9.18	660940	30000 100 900 720 6000 600
卖七	9.17	474260	2000 1400 5300 100 2900 1000
卖六	9.16	1015417	100 500 100 1900 4000 900 1000
卖五	9.15	566359	100 300 5000
卖四	9.14	296100	
卖三	9.13	431532	
卖二	9.12	410026	50,揭示的买一委托长度
卖一	9.11	144356	88,买一委托队列长度
买一	9.10	1164390	买一队列 (50/88)
买二	9.09	1287804	586690 100 1000 400 1300 600
买三	9.08	1174400	1700 100 800 2400 5000 200000
买四	9.07	1004900	500 1500 600 2000 1000 100 100
买五	9.06	613800	4000 1000 5000 200 3300 500
买六	9.05	739000	400 500 300 130600 2200 3500
买七	9.04	1128300	5000 100 100 1800 1000 1000
买八	9.03	269900	500 1800 5600 100 5000 3300
买九	9.02	194700	1000 4000 11000 200 500 100
买十	9.01	322200	10000

2、逐笔（XTPTBT）赋值字段及含义

(1) 深交所逐笔

深交所只有逐笔委托和逐笔成交，没有逐笔状态。

- exchange_id, 交易所 ID
- ticker, 股票代码, 6 字节
- seq, 无意义, 预留
- data_time, 委托时间或成交时间, 格式 YYYYMMDDHHMMSSsss
- type, 委托或成交标识
- entrust, 逐笔委托
 - channel_no, 频道代码
 - seq, 在该 channel_no 内的委托序号
 - price, 委托价格
 - qty, 委托数量
 - side, 买卖方向。'1':买; '2':卖; 'G':借入; 'F':出借。

➤ **ord_type**, 订单类别。'1': 市价; '2': 限价; 'U': 本方最优。注: 交易所没有给出转限、转撤等更加详细的信息

- **trade**, 逐笔成交
 - **channel_no**, 频道代码
 - **seq**, 在该 **channel_no** 内的成交序号
 - **price**, 成交价格
 - **qty**, 成交数量
 - **money**, 成交金额, 无意义
 - **bid_no**, 买方订单号, 可以追溯到 **entrust** 中的 **seq**
 - **ask_no**, 卖方订单号, 可以追溯到 **entrust** 中的 **seq**
 - **trade_flag**, 成交标志。SZL2, 成交标识 ('4':撤; 'F':成交)

(2) 上交所逐笔

跟深交所相比, 上交所增加了逐笔状态消息。

- **exchange_id**, 交易所 ID
- **ticker**, 股票代码, 6 字节
- **seq**, 对应上交所 **bizIndex** 字段, 在同一个 **channel** 内唯一
- **data_time**, 委托时间或成交时间, 格式 YYYYMMDDHHMMSSsss
- **type**, 委托或成交标识
- **entrust**, 逐笔委托
 - **channel_no**, 频道代码
 - **seq**, 在该 **channel_no** 内的委托序号, 单调递增
 - **price**, 委托价格
 - **qty**, 委托数量
 - **side**, 买卖方向。'B':买; 'S':卖
 - **ord_type**, 订单类别。'A':增加; 'D':删除
 - **order_no**, 原始订单编号
- **trade**, 逐笔成交
 - **channel_no**, 频道代码
 - **seq**, 在该 **channel_no** 内的成交序号, 单调递增
 - **price**, 成交价格
 - **qty**, 成交数量
 - **money**, 成交金额, SHL2 有该值, SZL2 没有该值 (总为 0)
 - **bid_no**, 买方订单号, 可以追溯到 **entrust** 中的 **order_no**
 - **ask_no**, 卖方订单号, 可以追溯到 **entrust** 中的 **order_no**
 - **trade_flag**, 成交标志。SHL2, 内外盘标识 ('B':主动买; 'S':主动卖; 'N':未知)
- **state**, 状态订单 <债券才有该类型消息>
 - **channel_no**, 频道代码
 - **seq**, 在该 **channel_no** 内的序号, 单调递增
 - **flag**, 状态

(3) 对数据的解释

SHL1、SZL1 没有逐笔。

SHL2 的撤单信息在逐笔委托中(order_type=' D'); SZL2 的撤单信息在逐笔成交中(trade_flag=' 4')。

channel_no, 频道代码。同一只股票的逐笔委托和逐笔成交行情数据中, channel_no 都相同。不同股票的逐笔委托和逐笔成交, channel_no 可能不同。同一个 channel_no 会有多只股票的逐笔委托和逐笔成交数据。

SZL2 的逐笔委托和逐笔成交, 放到一起编号。可通过逐笔成交里的订单号 (bid_no 或 ask_no) 追溯到对应的委托逐笔。

SHL2 债券, 逐笔委托&逐笔成交&逐笔状态, 放到一起编号。

SHL2 股票/基金, 逐笔委托单独编号(entrust.seq), 逐笔成交单独编号(trade.seq)。逐笔委托&逐笔成交一起有编号 (tbt.seq)。

期权无逐笔数据。

SH 现有逐笔 channel: [1, 6] [20] [103][801] (共 9 个, 20 为 B 股, 103 为盘后固定价格交易, 801 为债券)

SZ 现有逐笔 channel: [2011, 2014] [2021, 2024] [2031, 2034] [2061][2071] (共 14 个, 2061 为国债逆回购, 2071 为债券)

3、订单簿 (XTPOB) 赋值字段及含义

(1) 结构字段说明

- exchange_id, 交易所 ID
- ticker, 股票代码
- last_price, 最近成交价
- qty, 成交总量
- turnover, 成交总金额
- trades_count, 成交笔数
- bid[10]/ask[10], 十档买/卖价
- bid_qty[10]/ask_qty[10], 十档买/卖量
- data_time, 最近成交时间, 格式 YYYYMMDDHHMMSSsss

(2) 对数据的解释

订单簿, 是根据逐笔委托和逐笔成交数据重建的。

在集合竞价阶段 (包括开盘和收盘), 不推送该数据, 这个时间段的行情快照, 可以从 XTPMD 结构中获取。

使用该数据的用户, 应该与行情快照 (XTPMD) 结合起来使用。最新的时间戳, 便是最新的数据。

快照				10:16:21.000	订单簿				10:16:22.760
今开:	9.15	最高:	9.15	成交量:	13425653	最新价:	9.12	量:	13427153
昨收:	9.16	最低:	9.07	成交额:	122263603	笔数:	4283	额:	122277283
卖十	9.22	425600	卖一队列 (50/61)			卖十	9.22	425600	
卖九	9.21	500412	34881 1800 300 1000 106680			卖九	9.21	500412	
卖八	9.20	673170	100000 300 700 1500 14400 800			卖八	9.20	673170	
卖七	9.19	441228	1100 1000 1700 5500 3000 3000			卖七	9.19	441228	
卖六	9.18	675120	2000 2000 70000 1500 300 2000			卖六	9.18	675120	
卖五	9.17	436060	4000 2700 10000 1000 5000			卖五	9.17	436060	
卖四	9.16	959940	20000 100 100 1500 35573 100			卖四	9.16	959940	
卖三	9.15	389642	2000 100 1900 6000 2500 100			卖三	9.15	389642	
卖二	9.14	239300	100 100 100 1000 400 1500 600			卖二	9.14	239300	
卖一	9.13	506954	100 5000 100			卖一	9.13	507954	
买一	9.12	44949	买一队列 (17/17)			买一	9.12	42649	
买二	9.11	695972	2949 300 1800 2900 3000 2000			买二	9.11	695972	
买三	9.10	597886	1300 1500 500 11500 12500 600			买三	9.10	597886	
买四	9.09	648940	400 1200 1000 1000 500			买四	9.09	648940	
买五	9.08	842500				买五	9.08	843400	
买六	9.07	840800				买六	9.07	840800	
买七	9.06	628000				买七	9.06	628000	
买八	9.05	656900				买八	9.05	656900	
买九	9.04	1117400				买九	9.04	1117400	
买十	9.03	187300				买十	9.03	187300	

深交所债券:

债券上市首日开盘集合竞价的有效竞价范围为发行价的上下 30%，连续竞价、收盘集合竞价的有效竞价范围为最近成交价的上下 10%；非上市首日开盘集合竞价的有效竞价范围为前收盘价的上下 10%，连续竞价、收盘集合竞价的有效竞价范围为最近成交价的上下 10%。

SZOB 在揭示的时候，并没有对有效竞价范围外的委托进行过滤。

四、FAQ

1、公网测试环境

公网测试环境是为了方便测试，在公网上搭建的一套测试环境。7*24 小时提供行情服务。

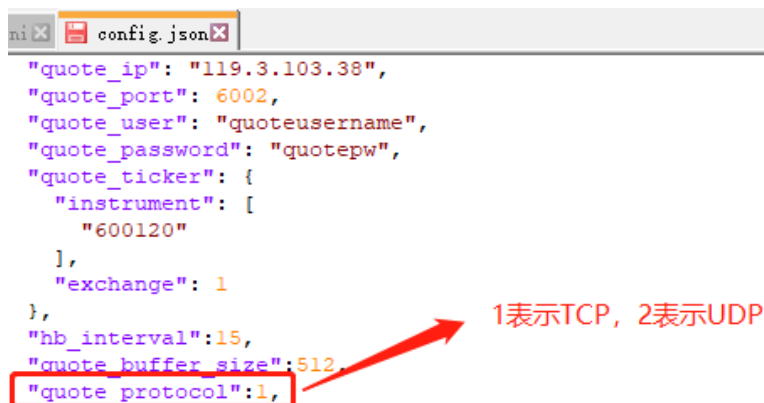
公网测试环境仅供客户调试 API 接口，提供的 L2 测试数据包括：10 档价/量、买一卖一委托队列、逐笔委托、逐笔成交，但是没有订单簿 OB 行情。

在公网测试环境，不建议全订阅。因为带宽有限。

所有用户在上实盘前，请务必在实盘上先做行情接入测试。

公网测试环境，使用 TCP 的方式推送数据。在该环境下测试时，API 和客户端需要配置成 TCP 的方式接收行情，否则会收不到行情的更新数据。

(1) 若使用 XTP 官网上的测试用例，请按下图方式进行配置



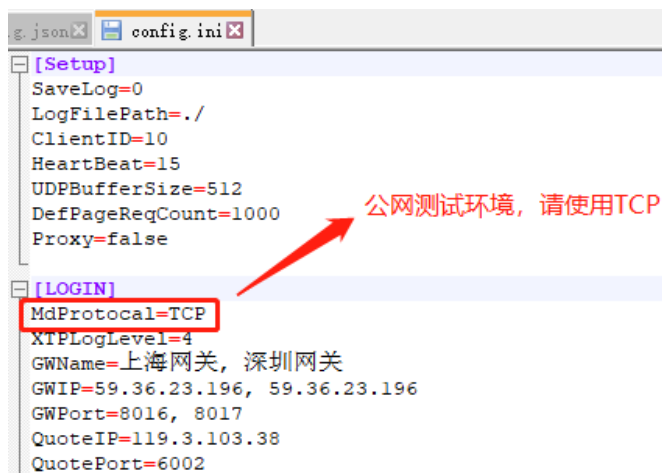
```

{
  "quote_ip": "119.3.103.38",
  "quote_port": 6002,
  "quote_user": "quoteusername",
  "quote_password": "quotepw",
  "quote_ticker": {
    "instrument": [
      "600120"
    ],
    "exchange": 1
  },
  "hb_interval": 15,
  "quote_buffer_size": 512,
  "quote_protocol": 1,
}

```

1表示TCP, 2表示UDP

(2) 若使用 XTP 客户端，请按下图进行配置



```

[Setup]
SaveLog=0
LogFilePath=./
ClientID=10
HeartBeat=15
UDPBufferSize=512
DefPageReqCount=1000
Proxy=false

[LOGIN]
MdProtocol=TCP
XTPLogLevel=4
GWName=上海网关, 深圳网关
GWIP=59.36.23.196, 59.36.23.196
GWPort=8016, 8017
QuoteIP=119.3.103.38
QuotePort=6002

```

公网测试环境，请使用TCP

(3) 若使用 API，请按下图方式进行配置

```

///用户登录请求
///@return 登录是否成功，“0”表示登录成功，“-1”表示连接服务器出错，此时用户可以调用GetApiLastError()来获取错误代码。
///@param ip 服务器ip地址，类似“127.0.0.1”
///@param port 服务器端口号
///@param user 登陆用户名
///@param password 登陆密码
///@param sock_type “1”代表TCP，“2”代表UDP
///@param local_ip 本地网卡地址，类似“127.0.0.1”
///@remark 此函数为同步阻塞式，不需要异步等待登录成功，当函数返回即可进行后续操作，此api只能有一个连接
virtual int Login(const char* ip, int port, const char* user, const char* password, XTP_PROTOCOL_TYPE sock_type,
const char* local_ip = NULL);

```

公网测试环境，请使用TCP

2、生产环境

XTP 在每个机房，都部署有 L1 行情服务器和 L2 行情服务器，均使用 UDP 方式对外推送行情。客户请勿跨机房连接行情服务器。

L1，只提供 5 档快照。

L2，提供 10 档快照、买一卖一委托队列、逐笔。

另外，在每个机房部署有 OB 服务器（不跨机房提供）。

期权(SH/SZ)，无论哪个环境，交易所只提供 5 档快照，不提供逐笔。

买一卖一委托队列，跟行情快照是同时发过来的。

SHL2 的逐笔数据，在开盘时间段，延时比较大。

SZL2 的逐笔数据，深交所一般是实时发过来的。即交易下单后，行情马上可以看到逐笔委托。

每条行情快照加上买一卖一委托队列的大小为，约为 1.5KB（按最大算）。

每条逐笔的大小约为 112B。

在数据高峰时期（9:15, 9:30, 13:00, 15:00），数据量非常大，约 20W 笔 / 秒。推送的带宽约为 22MB / 秒。

请不要阻塞 API 行情接收线程。

若 API 使用 TCP 方式收行情数据：API 接收线程收数据慢，行情服务器会断开与 API 的连接。

若 API 使用**组播(UDP)**方式收行情数据（生产 L2）：API 接收线程收数据慢（接收线程阻塞，或者在接收线程处理完数据后再接收下一条数据），可能会丢数据。

为了避免丢数据的问题发生，建议用“生产者 / 消费者”的模式来处理行情数据。接收线程只管收数据，另启动一个线程处理数据。

提示：测试发现，格式化类的操作（如 `snprintf`, `printf`）比较耗时，请勿在接收线程中使用。

用组播时，还需要确保 API 所在的机器，能够收到组播数据。**防火墙开启时，组播数据可能会被过滤掉。**请关闭防火墙，或允许行情的组播数据通过防火墙。

行情服务器一般在 8:30 至 8:45 就绪，交易所从 9:15 开始集合竞价。但在集合竞价前，交易所会发一些数据（只有行情快照，是为了测试），我们也会把这些数据发给用户。这些数据属于盘前数据。

假如 API 在 9:10 启动，可能会丢失一部分盘前数据（API 日志中会提示 WARNING），这个是没有关系的。

但若有盘中大量提示丢数据的日志，需要查找原因。

所有用户在上实盘前，请务必在实盘做行情接入测试。

3、UDP（组播）推送方式及问题排查

生产环境 Level-2、Level-1 行情数据，都是通过**组播**的方式推送给 API 的，请使用 UDP 连接，请关闭防火墙。有些机器收不到组播数据，请按照以下步骤排查。

- (1) 启动 API。
- (2) 确保 `Login` 的 `sock_type` 参数使用的是 `XTP_PROTOCOL_UDP`。
- (3) 订阅股票，确保回调函数 `OnSubMarketData` 返回成功。
- (4) 进行 **socket 参数优化**，可登录 xms 至公共信息里查看优化说明。
- (5) 若还没有收到数据（`OnDepthMarketData` 没有被回调），请使用**抓包工具**（如 wireshark, tcpdump），看看组播数据有没有到达 API 所在的机器。组播的地址与端口，请查看 API 的日志，也可以与 XTP 运维人员联系。

```
[INFO][XTP:0]Receive trading day 20180314 from quote server success.
[INFO][XTP:0]Receive udp ip 230.1.1.2 from quote server success.
[INFO][XTP:0]Receive udp port 17778 from quote server success.
[INFO][XTP:0]Connect to quote server 10.25.24.48:6661 success.
```


- (6) `tcpdump -i 网卡 udp port 端口号` (如 `tcpdump -i p5p2 udp port 17778`)，看看组播数据有没有到达主机。
- (7) 若**没有抓到行情组播数据**，请调整系统参数。

windows 系统

打开注册表编辑器 regedit，
 路径：`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters`，
 在该路径下新建两个 DWORD 变量，一个命名为 **IGMPVersion**，
 值设定为 3，第二个命名为 **IGMPLevel**，值设定为 2。
 路径：`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Afd\Parameters`，
 增加 **DefaultReceiveWindow** 字段，DWORD 类型，大小设置为 134217728 (128M)。
 卸掉杀毒软件（或者杀毒软件放行组播数据）。
 重启系统。
 关掉防火墙（或允许组播使用的端口通过）

linux 系统

确保组播地址绑定到了正确的网卡上，`netstat -gn`：



```

[ll@rw bin]$ netstat -gn
IPv6/IPv4 Group Memberships
Interface    RefCnt Group
-----
lo           1      224.0.0.1
eno1         1      230.1.1.2
eno1         1      224.0.0.251
  
```

组播地址

关掉防火墙（或允许组播使用的端口通过）：`systemctl stop firewalld.service`。

下面命令是临时的，操作系统重启后会**失效**。需要写在操作系统的启动脚本中。

```

sysctl -w net.core.rmem_default="33554432"
sysctl -w net.core.rmem_max="67108864"
sysctl -w net.core.wmem_default="33554432"
sysctl -w net.core.wmem_max="67108864"
sysctl -w net.ipv4.udp_mem="33554432 67108864 67108864"
sysctl -w net.ipv4.udp_rmem_min="33554432"
sysctl -w net.ipv4.udp_wmem_min="33554432"
  
```

- (8) 若还不能收到数据，添加临时路由：`route add -net 组播地址 掩码 dev 设备`，（如 `route add -net 230.1.1.0/24 dev p5p2`）
- (9) Windows 下用 SmartX 客户端(Windows/macOS)，订阅一整天行情，看 API 的 quote.log 日志，看看有没有丢包；若丢包，需要查找原因。
- (10) API 若**发生频繁丢包**，调用 `SetUDPBufferSize` 函数，调整参数到 512MB（或更大）。

```

///设置采用UDP方式连接时的单个队列接收缓冲区大小，目前可能最大使用4个缓冲区队列
///@remark 需要在Login之前调用，默认大小和最小设置均为64MB。此缓存大小单位为MB，请输入2的次方数，例如128MB请输入128。
virtual void SetUDPBufferSize(uint32_t buff_size);

```

(11) API 的默认日志级别是 DEBUG，需要调整到 INFO（或 ERROR），否则可能会导致丢包。CreateQuoteApi 的 `log_level` 参数。

4、TCP 推送方式及问题排查

TCP 模式下，若用户收不到行情数据，请按以下方式排查。

- (1) 启动 API。
- (2) 确保 Login 的 `sock_type` 参数使用的是 XTP_PROTOCOL_TCP。
- (3) 订阅股票，确保回调函数 OnSubMarketData 返回成功。
- (4) 若还没有收到数据(OnDepthMarketData 没有被回调)，请使用抓包工具(如 wireshark, tcpdump)，看看数据有没有到达 API 所在的机器。
- (5) tcpdump -i 网卡 tcp port 端口号（如 tcpdump -i p5p2 tcp port 6666），看看行情数据有没有到达主机。

5、行情数据更新频率

种类	集合竞价	连续竞价	午间休市
SH 指数	约 5 秒一次 (不准)	约 5 秒一次 (不准)	60 秒推一次 L2 时间戳跟着变 (201901 后，不准)
SH 股票	有变化 3 秒一次 无变化 60 秒一次	有变化 3 秒一次 无变化 60 秒一次	60 秒推一次 L2 时间戳跟着变 (201901 后)
SH 逐笔委托	9:15-9:25，可以下单，但交易所不推逐笔委托 9:25，开始推送逐笔委托 9:25-9:30，无数据（开盘集合竞价结束到连续竞价开始前） 9:30，开始推（连续竞价）		
SH 逐笔成交	9:25，有大量逐笔成交（集合竞价） 9:25-9:30，无数据（开盘集合竞价结束到连续竞价开始前） 9:30，开始推（连续竞价）		
SH 期权	有变化 15 秒一次 无变化 30 秒一次	有变化 0.5 秒一次 无变化 30 秒一次	30 秒一次
SH 盘后固定价格（快照）	参与盘后固定价格交易的股票，15:00 开始发布 15:00 - 15:05，集中撮合 15:05 - 15:30，连续交易 15:30 后，闭市		
SH 盘后固定价格（逐笔成交）	参与盘后固定价格交易的股票，15:05 开始发布		

SZ 指数	60 秒一次	3 秒一次	60 秒推一次 时间戳跟着变
SZ 股票	60 秒一次	有变化 3 秒一次 无变化 60 秒一次	60 秒推一次 时间戳跟着变
SZ 逐笔委托	9:15-9:25, 推逐笔委托(期间可以下单, 可以撤单) 9:25-9:30, 无数据(开盘集合竞价结束到连续竞价开始前) 9:30, 开始推(连续竞价)		
SZ 逐笔成交	9:15-9:25, 只有逐笔撤单, 无逐笔成交单 9:25, 有大量逐笔成交(集合竞价) 9:25-9:30, 无数据(开盘集合竞价结束到连续竞价开始前) 9:30, 开始推(连续竞价)		
SZ 期权	6 秒一次	有变化 0.5 秒一次 无变化 60 秒一次	6 秒一次

SH 和 SZ 均无期权的逐笔

6、丢包问题 API 日志排查

udpseq0_0.YYYYMMDD 是普通股票的 MarketData 的 Seq
 udpseq0_1.YYYYMMDD 是普通股票的 OrderBook 的 Seq
 udpseq0_2.YYYYMMDD 是普通股票的 TickByTick 的 Seq

 udpseq1_0.YYYYMMDD 是期权的 MarketData 的 Seq
 udpseq1_1.YYYYMMDD 是期权的 OrderBook 的 Seq
 udpseq1_2.YYYYMMDD 是期权的 TickByTick 的 Seq

 udpseq2_0.YYYYMMDD 是接收 buffer 满的时候的错误代码
 udpseq2_1.YYYYMMDD 是普通股票和期权 TickByTick 的来晚了、需要丢弃的 Seq
 udpseq2_2.YYYYMMDD 是普通股票和期权 TickByTick 的乱序 Seq

 如果 tick 丢包的话, quote.log.YYYYMMDD 里会有提示日志
 如果 tick 乱序, udpseq2_2.YYYYMMDD 里会有 seq
 如果接收 buffer 满, udpseq2_0.YYYYMMDD 里会有错误码
 看 marketdata 丢不丢包, 可以看 udpseq0_0.YYYYMMDD 是否 seq 丢失(seq 乱序没关系)
 看 OrderBook 丢不丢包, 可以看 udpseq0_1.YYYYMMDD 是否 seq 丢失(seq 乱序没关系)

7、对 seq is discrete N to M 的解释

组播 (UDP) 推送行情模式下。

行情服务器启动后, 若有数据需要推送, 就会往外推数据 (组播)。UDP 包, 是统一连续编号的。

若 API 启动前行情服务器已经往外推送了组播数据, API 就会打印一条 WARNIG 日志。

```
[WARNING][XTP:10200202]MarketData udp seq is discrete 1 - 654.[xapi_parse_data_thread.cpp:666]
[INFO][XTP:0]End to subscribe market data with tcp.
[INFO][XTP:0]Begin to subscribe order book with tcp.
[WARNING][XTP:10200202]Tick by tick udp seq is discrete 1 - 375.[xapi_parse_data_thread.cpp:772]
[WARNING][XTP:10200202]Option MarketData udp seq is discrete 1 - 360.[xapi_parse_data_thread.cpp:1050]
```

上面的日志表明, 丢失了行情快照数据 (1-654)、逐笔数据 (1-375)、期权行情快照数据 (1-360)。这种情况可以忽略。

但若在收行情的过程中, 发现大量下图所示的日志, 需要查找问题。

```
[10295][WARNING][XTP:10200202]Tick by tick udp seq is discrete 663689 - 666246.[xapi_parse_data_thread.cpp:765]
[10295][WARNING][XTP:10200202]OrderBook udp seq is discrete 237300 - 239606.[xapi_parse_data_thread.cpp:711]
[10295][WARNING][XTP:10200202]MarketData udp seq is discrete 100909 - 101526.[xapi_parse_data_thread.cpp:659]
[10295][WARNING][XTP:10200202]Tick by tick udp seq is discrete 23520892 - 23521054.[xapi_parse_data_thread.cpp:765]
[10295][WARNING][XTP:10200202]OrderBook udp seq is discrete 17562598 - 17562718.[xapi_parse_data_thread.cpp:711]
[10295][WARNING][XTP:10200202]OrderBook udp seq is discrete 17906934 - 17906998.[xapi_parse_data_thread.cpp:711]
[10295][WARNING][XTP:10200202]Tick by tick udp seq is discrete 23993965 - 23994047.[xapi_parse_data_thread.cpp:765]
[10295][WARNING][XTP:10200202]Tick by tick udp seq is discrete 24952764 - 24952970.[xapi_parse_data_thread.cpp:765]
[10295][WARNING][XTP:10200202]OrderBook udp seq is discrete 18616791 - 18616959.[xapi_parse_data_thread.cpp:711]
[10295][WARNING][XTP:10200202]OrderBook udp seq is discrete 18616995 - 18617062.[xapi_parse_data_thread.cpp:711]
```

另外, 行情快照、逐笔数据、订单簿数据, 是分开编号的。

对日志” MarketData udp seq XXX is time out, discrete xxx-xxx”的解释:

行情服务器会对发出去的每个 udp 包按顺序编号 (行情快照、逐笔、订单簿各自独立编号)。以行情快照(MarketData)为例, API 正常收到包的顺序应该为 1, 2, 3, 4, 5, 但若 API 收到 1, 2, 3, 然后收到 5, 就会去等 4 的包。若在一段时间内, API 还没有收到 4 的包, 就会打印上面的日志。并认为 4 丢掉了。

若发现丢包较多, 可以把 UDP 的接收缓存开大点。

Windows:

打开注册表的

“HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Afd\Parameters”选项, 增加 “DefaultReceiveWindow” 字段, DWORD 类型, 大小设置为 113246208(128M)。

发送端 (行情服务器), 可以增加 “DefaultSendWindow” 字段, DWORD 类型, 大小设置为 113246208(128M)。

8、其它问题

(1) SHL2 的**最新价**和**收盘价**, 分别是怎么计算得到的。发现收盘后, 有些股票的收盘价和最新价不相同。

最新价 = 最新一支逐笔成交的价格

收盘价 = 证券的收盘价为当日该证券**最后一笔交易**前一分钟所有交易的成交量加权平均价 (含最后一笔交易)。当日无成交的, 以前收盘价为当日收盘价。

债券质押式回购的收盘价为当日该证券**最后一笔交易**前一小时所有交易的成交量加权平均价 (含最后一笔交易)。当日无成交的, 以前收盘价为当日收盘价。

收盘集合竞价阶段产生交易的, 这两个价格相同; 否则可能会不同。

(2) SHL1 和 SHL2 每天的快照数量不一样。SHL1 和 SHL2 分别在什么时候推出快照, 所有档位, 只要发生变化就推?

SHL1, 买卖 5 档盘口发生变化时推。

SHL2, 买卖 10 档盘口发生变化时推。

(3) SHL2 逐笔提速后, 逐笔实时发送, 逐笔约有 250ms 时间间隔。