XTP关于L2行情数据回补功能的使用说明

Table of Contents

一. 新增的头文件
二. 新增的函数

- 1. QuoteAPI新增接口
- 2. QuoteSpi新增回调函数

三. 新增接口说明及使用示例
四. 行情回补数据逻辑
五. 注意事项

一. 新增的头文件

文件名	详情
xquote_api_rebuild_tbt_struct.h	行情回补所需结构体定义头文件。

二. 新增的函数

1. QuoteAPI新增接口

```
        ///用户登录回补服务器请求
        ///@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，“-2”表示已存在连接，不允许重复登录，如果需要重连，请先logout，“-3”表示输入有错误
        ///@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。回补服务器会在无消息交互后定时断线，请注意仅在需要回补数据时才保持连接，回补完成后请及时logout
        virtual int LoginToRebuildQuoteServer(const char* ip, int port, const char* user, const char* password, XTP_PROTOCOL_TYPE sock_type, const char* local_ip = NULL) = 0;

        ///登出回补服务器请求
        ///@return 登出是否成功，“0”表示登出成功，非“0”表示登出出错，此时用户可以调用GetApiLastError()来获取错误代码
        ///@remark 此函数为同步阻塞式，不需要异步等待登出，当函数返回即可进行后续操作
        virtual int LogoutFromRebuildQuoteServer() = 0;
```

```
        ///请求回补指定行情，包括快照和逐笔
        ///@return 请求回补指定频道的逐笔行情接口调用是否成功，“0”表示接口调用成功，非“0”表示接口调用出错
        ///@param rebuild_param 指定回补的参数信息，注意一次性回补最多1000个数据，超过1000需要分批次请求，一次只能指定一种类型的数据
        ///@remark 仅在逐笔行情丢包时或者确实快照行情时使用，回补的行情数据将从OnRebuildTickByTick或者OnRebuildMarketData()接口回调提供，与订阅的行情数据不在同一个线程内
        virtual int RequestRebuildQuote(XTPQuoteRebuildReq* rebuild_param) = 0;
```

2. QuoteSpi新增回调函数

```
        ///当客户端与回补行情服务器通信连接断开时，该方法被调用。
        ///@param reason 错误原因，请与错误代码表对应
        ///@remark api不会自动重连，当断线发生时，请用户自行选择后续操作。回补服务器会在无消息交互后会定时断线，请注意仅在需要回补数据时才保持连接，无回补需求时，无需登陆。
        virtual void OnRebuildQuoteServerDisconnected(int reason) {};

        ///请求回补指定频道的逐笔行情的总体结果应答
        ///@param rebuild_result 当回补结束时被调用，如果回补结果失败，则msg参数表示失败原因
        ///@remark 需要快速返回，仅在回补数据发送结束后调用，如果请求数据太多，一次性无法回补完，那么rebuild_result.result_code = XTP_REBUILD_RET_PARTLY，此时需要根据回补结果继续发起回补数据请求
        virtual void OnRequestRebuildQuote(XTPQuoteRebuildResultRsp* rebuild_result) {};

        ///回补的逐笔行情数据
        ///@param tbt_data 回补的逐笔行情数据
        ///@remark 需要快速返回，此函数调用与OnTickByTick不在一个线程内，会在OnRequestRebuildQuote()之前回调
        virtual void OnRebuildTickByTick(XTPTBT *tbt_data) {};

        ///回补的快照行情数据
        ///@param md_data 回补的逐笔行情数据
        ///@remark 需要快速返回，此函数调用与OnDepthMarketData不在一个线程内，会在OnRequestRebuildQuote()之前回调
        virtual void OnRebuildMarketData(XTPMD *md_data) {};
```

三. 新增接口说明及使用示例

1. LoginToRebuildQuoteServer

用户登录回补服务器请求。

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。回补服务器会在无消息交互后定时断线，请注意仅在需要回补数据时才保持连接，回补完成后请及时logout。

◇ 1.函数原型

```
virtual int LoginToRebuildQuoteServer(const char* ip, int port, const char* user,
const char* password, XTP_PROTOCOL_TYPE sock_type, const char* local_ip = NULL) =
0;
```

◇ 2.参数

ip：服务器ip地址，类似"127.0.0.1"

port：服务器端口号

user：登录用户名

password：登录密码

sock_type："1"代表TCP，"2"代表UDP，**此处仅支持TCP**

local_ip：本地网卡地址，类似"127.0.0.1"，可以为NULL

```
// XTP_PROTOCOL_TYPE是通讯传输协议方式
typedef enum XTP_PROTOCOL_TYPE
{
    XTP_PROTOCOL_TCP = 1,    ///<采用TCP方式传输
    XTP_PROTOCOL_UDP         ///<采用UDP方式传输(仅行情接口支持)
}XTP_PROTOCOL_TYPE;
```

◇ 3.返回

登陆是否成功，"0"表示登陆成功，非"0"表示登陆不成功。

◇ 4.调用示例

```
// 回补服务器仅能采用TCP方式连接
if (user_quote_api_)
{
    // 登录前参数设置

    std::string quote_rebuild_server_ip = "xxx.xxx.xxx.xxx";
    int quote_rebuild_server_port = xxxx;
    std::string quote_rebuild_username = "xxxxxxxx";
    std::string quote_rebuild_password = "xxxxxx";

    // 登录
    int ret = user_quote_api_-
>LoginToRebuildQuoteServer(quote_rebuild_server_ip.c_str(),
```

```
quote_rebuild_server_port, quote_rebuild_username.c_str(),
quote_rebuild_password.c_str(), XTP_PROTOCOL_TCP);
    if (0 != ret)
    {
        // 登录失败，获取错误信息
        XTPRI* error_info = m_pQuoteApi->GetApiLastError();
        std::cout << "login to quote rebuild server error, " << error_info-
>error_id << " : " << error_info->error_msg << std::endl;

        return;
    }

    // 登录成功
    //TODO: 用户逻辑，例如请求回补数据等
}
```

2. LogoutFromRebuildQuoteServer

用户登出回补服务器请求。

此函数为同步阻塞式，不需要异步等待登出，当函数返回即可进行后续操作。

◇ 1.函数原型

```
virtual int LogoutFromRebuildQuoteServer() = 0;
```

◇ 2.参数

无

◇ 3.返回

登出是否成功，"0"表示登出成功，非"0"表示登出不成功。

◇ 4.调用示例

```
// 回补服务器仅能采用TCP方式连接
if (user_quote_api_)
{

    // 登出
    user_quote_api_->LogoutFromRebuildQuoteServer();

}
```

3. RequestRebuildQuote

请求回补指定行情，包括快照和逐笔。

此函数为同步阻塞式，不需要异步等待登出，当函数返回即可进行后续操作。

◇ 1.函数原型

```
virtual int RequestRebuildQuote(XTPQuoteRebuildReq* rebuild_param) = 0;
```

◇ 2.参数

rebuild_param: 指定的需要回补数据的参数信息。

```
///实时行情回补请求结构体
typedef struct XTPQuoteRebuildReq
{
    ///请求id 请求端自行管理定义
    int32_t                     request_id;
    ///请求数据类型 1-快照 2-逐笔
    XTP_QUOTE_REBUILD_DATA_TYPE data_type;
    ///请求市场 1-上海  2-深圳
    XTP_EXCHANGE_TYPE           exchange_id;
    ///合约代码 以'\0'结尾  沪深A股6位  期权8位
    char                        ticker[16];
    ///data_type=逐笔 表示逐笔通道号
    int16_t                     channel_number;
    ///data_type=逐笔 表示序列号begin;  =快照 表示时间begin(格式为YYYYMMDDHHMMSSsss)
    int64_t                     begin;
    ///data_type=逐笔 表示序列号end;  =快照 表示时间end(格式为YYYYMMDDHHMMSSsss)   逐
笔区间：[begin，end]前后闭区间  快照区间：[begin，end)  前闭后开区间
    int64_t                     end;
}XTPQuoteRebuildReq;
```

◇ 3.返回

请求回补指定频道的逐笔行情接口调用是否成功，“0”表示接口调用成功，非“0”表示接口调用出错。

◇ 4.调用示例

（1） 请求回补快照数据

下面以请求回补沪市600000快照数据为例：

```
//请求回补沪市600000快照数据
if (user_quote_api_)
{
    //回补请求参数赋值
    XTPQuoteRebuildReq req;
    memset(&req, 0, sizeof(XTPQuoteRebuildReq));
    req.request_id = 1;//用户自定义，用来标识此次查询请求
    req.data_type = XTP_QUOTE_REBUILD_MD;//回补数据类型
    req.exchange_id = XTP_EXCHANGE_SH;//交易所类型，用户根据实际情况修改
    std::string reqticker = "600000";//请求回补的股票代码，此处以600000为例
    strcpy_s(req.ticker, XTP_TICKER_LEN, reqticker.c_str());
    req.channel_number = 0;//仅逐笔订阅时生效，对于快照可不赋值
    req.begin = YYYYMMDDHHMMSSsss;//快照开始时间，闭区间，用户根据实际情况修改
    req.end = YYYYMMDDHHMMSSsss;//快照结束时间，开区间，用户根据实际情况修改
    // 发送请求
    int ret = user_quote_api_->RequestRebuildQuote(&req);

}
```

（2） 请求回补逐笔数据

下面以请求回补沪市频道号为2011，逐笔序列号在[20,78]区间内的逐笔数据为例：

```
//请求回补沪市频道号为2011，逐笔序列号在[20,78]区间内的逐笔数据
if (user_quote_api_)
{
    //回补请求参数赋值
    XTPQuoteRebuildReq req;
    memset(&req, 0, sizeof(XTPQuoteRebuildReq));
    req.request_id = 1;//用户自定义，用来标识此次查询请求
    req.data_type = XTP_QUOTE_REBUILD_TBT;//回补数据类型
    req.exchange_id = XTP_EXCHANGE_SH;//交易所类型，用户根据实际情况修改
    req.channel_number = 2011;//仅逐笔订阅时生效，对于快照可不赋值，用户根据实际情况修
改
    req.begin = 20;//逐笔开始的序列号，闭区间，用户根据实际情况修改
    req.end = 78;//逐笔结束的序列号，闭区间，用户根据实际情况修改
    // 发送请求
    int ret = user_quote_api_->RequestRebuildQuote(&req);

}
```

◇ 5.响应函数

```
virtual void OnRequestRebuildQuote(XTPQuoteRebuildResultRsp* rebuild_result);
```

◇ 6.通知函数

```
virtual void OnRebuildTickByTick(XTPTBT *tbt_data);
virtual void OnRebuildMarketData(XTPMD *md_data);
```

4. OnRebuildQuoteServerDisconnected

当客户端与回补行情服务器通信连接断开时，该方法被调用。

Api不会自动重连，当断线发生时，请用户自行选择后续操作。回补服务器会在无消息交互后会定时断线，请注意仅在需要回补数据时才保持连接，无回补需求时，无需登陆。

◇ 1.函数原型

```
virtual void OnRebuildQuoteServerDisconnected(int reason);
```

◇ 2.参数

Reason：错误原因，现阶段无需关心

◇ 3.返回
无

5. OnRequestRebuildQuote

请求回补指定频道的逐笔行情的总体结果应答。

仅在回补数据发送结束后调用，如果请求数据太多，一次性无法回补完，那么rebuild_result.result_code = XTP_REBUILD_RET_PARTLY，此时需要根据回补结果继续发起回补数据请求。

◇ 1.函数原型

```
virtual void OnRequestRebuildQuote(XTPQuoteRebuildResultRsp* rebuild_result);
```

◇ 2.参数

rebuild_result: 当回补结束时被调用，如果回补结果失败，则msg参数表示失败原因

```
///实时行情回补响应结构体
typedef struct XTPQuoteRebuildResultRsp
```

```
{
    ///请求id 请求包带过来的id
    int32_t                     request_id;
    ///市场类型 上海 深圳
    XTP_EXCHANGE_TYPE           exchange_id;
    ///总共返回的数据条数
    uint32_t                    size;
    ///逐笔数据 通道号
    int16_t                     channel_number;
    ///逐笔 表示序列号begin; 快照 表示时间begin(格式为YYYYMMDDHHMMSSsss)
    int64_t                     begin;
    ///逐笔 表示序列号end; 快照 表示时间end(格式为YYYYMMDDHHMMSSsss)
    int64_t                     end;
    ///结果类型码
    XTP_REBUILD_RET_TYPE        result_code;
    ///结果信息文本
    char                        msg[64];
}XTPQuoteRebuildResultRsp;

//交易所类型
typedef enum XTP_EXCHANGE_TYPE
{
    XTP_EXCHANGE_SH = 1,    ///<上证
    XTP_EXCHANGE_SZ,        ///<深证
    XTP_EXCHANGE_UNKNOWN    ///<不存在的交易所类型
}XTP_EXCHANGE_TYPE;

//回补返回结果类型
typedef enum XTP_REBUILD_RET_TYPE {
    XTP_REBUILD_RET_COMPLETE    = 1,    ///<全部数据
    XTP_REBUILD_RET_PARTLY      = 2,    ///<部分数据
    XTP_REBUILD_RET_NO_DATA     = 3,    ///<没有数据
    XTP_REBUILD_RET_PARAM_ERR   = 4,    ///<参数错误
    XTP_REBUILD_RET_FREQUENTLY  = 5,    ///<请求频繁
}XTP_REBUILD_RET_TYPE;
```

◇ 3.返回

无

◇ 4.触发函数

```
virtual int RequestRebuildQuote(XTPQuoteRebuildReq* rebuild_param) = 0;
```

6. OnRebuildTickByTick

回补的逐笔行情数据通知。

◇ 1.函数原型

```
virtual void OnRebuildTickByTick(XTPTBT *tbt_data);
```

◇ 2.参数

tbt_data: 回补的逐笔行情数据

```
///逐笔委托
struct XTPTickByTickEntrust {
    ///频道代码
    int32_t channel_no;
    ///SH: 委托序号(委托单独编号，同一channel_no内连续)
    ///SZ: 委托序号(委托成交统一编号，同一channel_no内连续)
    int64_t seq;
    ///委托价格
    double  price;
    ///SH: 剩余委托数量(balance)
    ///SZ: 委托数量
    int64_t qty;
    ///SH: 'B':买; 'S':卖
    ///SZ: '1':买; '2':卖; 'G':借入; 'F':出借
    char  side;
    ///SH: 'A': 增加; 'D': 删除
    ///SZ: 订单类别: '1': 市价; '2': 限价; 'U': 本方最优
    char ord_type;
    ///SH: 原始订单号
    ///SZ: 无意义
    int64_t order_no;
};

///逐笔成交
struct XTPTickByTickTrade {
    ///频道代码
    int32_t channel_no;
    ///SH: 成交序号(成交单独编号，同一channel_no内连续)
    ///SZ: 成交序号(委托成交统一编号，同一channel_no内连续)
    int64_t seq;
    ///成交价格
    double price;
    ///成交量
    int64_t qty;
    ///成交金额(仅适用上交所)
    double money;
    ///买方订单号
    int64_t bid_no;
    ///卖方订单号
    int64_t ask_no;
    /// SH: 内外盘标识('B':主动买; 'S':主动卖; 'N':未知)
```

```
    /// SZ: 成交标识('4':撤; 'F':成交)
    char trade_flag;
};

///逐笔状态订单
struct XTPTickByTickStatus {
    ///频道代码
    int32_t channel_no;
    ///同一channel_no内连续
    int64_t seq;
    ///状态信息
    char flag[8];
};

///逐笔数据信息
typedef struct XTPTickByTickStruct {
    ///交易所代码
    XTP_EXCHANGE_TYPE exchange_id;
    ///合约代码（不包含交易所信息），不带空格，以'\0'结尾
    char ticker[XTP_TICKER_LEN];
    /// SH: 业务序号（委托成交统一编号，同一个channel_no内连续，此seq区别于联合体内的
seq，channel_no等同于联合体内的channel_no）
    /// SZ: 无意义
    int64_t seq;
    ///委托时间 or 成交时间
    int64_t data_time;
    ///委托 or 成交
    XTP_TBT_TYPE type;

    union {
        XTPTickByTickEntrust entrust;
        XTPTickByTickTrade     trade;
        XTPTickByTickStatus    state;
    };
} XTPTBT;

//交易所类型
typedef enum XTP_EXCHANGE_TYPE
{
    XTP_EXCHANGE_SH = 1,    ///<上证
    XTP_EXCHANGE_SZ,        ///<深证
    XTP_EXCHANGE_UNKNOWN    ///<不存在的交易所类型
}XTP_EXCHANGE_TYPE;
```

◇ 3.返回

无

◇ 4.请求函数

```
virtual int RequestRebuildQuote(XTPQuoteRebuildReq* rebuild_param) = 0;
```

7. OnRebuildMarketData

回补的逐笔行情数据通知。

◇ 1.函数原型

```
virtual void OnRebuildMarketData(XTPMD *md_data);
```

◇ 2.参数

md_data: 回补的快照行情数据

```
///股票、基金 等额外数据
struct XTPMarketDataStockExData {
    ///委托买入总量(SH,SZ)
    int64_t total_bid_qty;
    ///委托卖出总量(SH,SZ)
    int64_t total_ask_qty;
    ///加权平均委买价格(SH,SZ)
    double ma_bid_price;
    ///加权平均委卖价格(SH,SZ)
    double ma_ask_price;
    ///债券加权平均委买价格(SH)
    double ma_bond_bid_price;
    ///债券加权平均委卖价格(SH)
    double ma_bond_ask_price;
    ///债券到期收益率(SH)
    double yield_to_maturity;
    ///基金实时参考净值(SH,SZ)
    double iopv;
    ///ETF申购笔数(SH)
    int32_t etf_buy_count;
    ///ETF赎回笔数(SH)
    int32_t etf_sell_count;
    ///ETF申购数量(SH)
    double etf_buy_qty;
    ///ETF申购金额(SH)
    double etf_buy_money;
    ///ETF赎回数量(SH)
    double etf_sell_qty;
    ///ETF赎回金额(SH)
    double etf_sell_money;
    ///权证执行的总数量(SH)
    double total_warrant_exec_qty;
    ///权证跌停价格（元）(SH)
```

```
    double warrant_lower_price;
    ///权证涨停价格（元）(SH)
    double warrant_upper_price;
    ///买入撤单笔数(SH)
    int32_t cancel_buy_count;
    ///卖出撤单笔数(SH)
    int32_t cancel_sell_count;
    ///买入撤单数量(SH)
    double cancel_buy_qty;
    ///卖出撤单数量(SH)
    double cancel_sell_qty;
    ///买入撤单金额(SH)
    double cancel_buy_money;
    ///卖出撤单金额(SH)
    double cancel_sell_money;
    ///买入总笔数(SH)
    int64_t total_buy_count;
    ///卖出总笔数(SH)
    int64_t total_sell_count;
    ///买入委托成交最大等待时间(SH)
    int32_t duration_after_buy;
    ///卖出委托成交最大等待时间(SH)
    int32_t duration_after_sell;
    ///买方委托价位数(SH)
    int32_t num_bid_orders;
    ///卖方委托价位数(SH)
    int32_t num_ask_orders;

    ///基金T-1日净值(SZ)
    double pre_iopv;
    ///预留
    int64_t r1;
    ///预留
    int64_t r2;
};

///债券额外数据
struct XTPMarketDataBondExData {
    ///委托买入总量(SH,SZ)
    int64_t total_bid_qty;
    ///委托卖出总量(SH,SZ)
    int64_t total_ask_qty;
    ///加权平均委买价格(SH,SZ)
    double ma_bid_price;
    ///加权平均委卖价格(SH,SZ)
    double ma_ask_price;
    ///债券加权平均委买价格(SH)
    double ma_bond_bid_price;
    ///债券加权平均委卖价格(SH)
    double ma_bond_ask_price;
    ///债券到期收益率(SH)
    double yield_to_maturity;
    ///匹配成交最近价(SZ)
    double match_lastpx;
```

```
    ///债券加权平均价格(SH)
    double ma_bond_price;
    ///预留
    double r2;
    ///预留
    double r3;
    ///预留
    double r4;
    ///预留
    double r5;
    ///预留
    double r6;
    ///预留
    double r7;
    ///预留
    double r8;
    ///买入撤单笔数(SH)
    int32_t cancel_buy_count;
    ///卖出撤单笔数(SH)
    int32_t cancel_sell_count;
    ///买入撤单数量(SH)
    double cancel_buy_qty;
    ///卖出撤单数量(SH)
    double cancel_sell_qty;
    ///买入撤单金额(SH)
    double cancel_buy_money;
    ///卖出撤单金额(SH)
    double cancel_sell_money;
    ///买入总笔数(SH)
    int64_t total_buy_count;
    ///卖出总笔数(SH)
    int64_t total_sell_count;
    ///买入委托成交最大等待时间(SH)
    int32_t duration_after_buy;
    ///卖出委托成交最大等待时间(SH)
    int32_t duration_after_sell;
    ///买方委托价位数(SH)
    int32_t num_bid_orders;
    ///卖方委托价位数(SH)
    int32_t num_ask_orders;
    ///时段(SHL2)，L1快照数据没有此字段，具体字段说明参阅《上海新债券Level2行情说
明.doc》文档
    char instrument_status[8];
};

/// 期权额外数据
struct XTPMarketDataOptionExData {
    ///波段性中断参考价(SH)
    double  auction_price;
    ///波段性中断集合竞价虚拟匹配量(SH)
    int64_t auction_qty;
    ///最近询价时间(SH)
    int64_t last_enquiry_time;
};
```

```
///行情快照数据类型，2.2.32以前版本所用
enum XTP_MARKETDATA_TYPE {
    XTP_MARKETDATA_ACTUAL = 0, // 现货(股票/基金/债券等)
    XTP_MARKETDATA_OPTION = 1, // 期权
};

///行情快照数据类型，2.2.32版本新增字段
enum XTP_MARKETDATA_TYPE_V2 {
    XTP_MARKETDATA_V2_INDEX  = 0, // 指数
    XTP_MARKETDATA_V2_OPTION = 1, // 期权
    XTP_MARKETDATA_V2_ACTUAL = 2, // 现货(股票/基金等)
    XTP_MARKETDATA_V2_BOND   = 3, // 债券
};

///行情
typedef struct XTPMarketDataStruct
{
    // 代码
    ///交易所代码
    XTP_EXCHANGE_TYPE exchange_id;
    ///合约代码（不包含交易所信息），不带空格，以'\0'结尾
    char    ticker[XTP_TICKER_LEN];

    // 价格
    ///最新价
    double  last_price;
    ///昨收盘
    double  pre_close_price;
    ///今开盘
    double  open_price;
    ///最高价
    double  high_price;
    ///最低价
    double  low_price;
    ///今收盘
    double  close_price;

    // 期权数据
    ///昨日持仓量(张)(目前未填写)
    int64_t pre_total_long_positon;
    ///持仓量(张)
    int64_t total_long_positon;
    ///昨日结算价（SH）
    double  pre_settl_price;
    ///今日结算价（SH）
    double  settl_price;

    // 涨跌停
    ///涨停价
    double  upper_limit_price;
    ///跌停价
    double  lower_limit_price;
    ///预留
```

```
    double  pre_delta;
    ///预留
    double  curr_delta;

    /// 时间类，格式为YYYYMMDDHHMMSSsss
    int64_t data_time;

    // 量额数据
    ///数量，为总成交量（单位股，与交易所一致）
    int64_t qty;
    ///成交金额，为总成交金额（单位元，与交易所一致）
    double  turnover;
    ///预留(无意义)
    double  avg_price;

    // 买卖盘
    ///十档申买价
    double bid[10];
    ///十档申卖价
    double  ask[10];
    ///十档申买量
    int64_t bid_qty[10];
    ///十档申卖量
    int64_t ask_qty[10];

    // 额外数据
    ///成交笔数
    int64_t trades_count;
    ///当前交易状态说明，参阅《XTP API常见问题.doc》文档
    char ticker_status[8];
    ///数据
    union {
        XTPMarketDataStockExData  stk;
        XTPMarketDataOptionExData opt;
        XTPMarketDataBondExData  bond;
    } ;
    ///决定了union是哪种数据类型（2.2.32版本以前所用字段，仅为了保持兼容，不建议使用该字
段)
    XTP_MARKETDATA_TYPE data_type;
    ///决定了union是哪种数据类型（2.2.32版本新增字段，更详细区分了行情快照数据类型）
    XTP_MARKETDATA_TYPE_V2 data_type_v2;
} XTPMD;
```

◇ 3.返回

无

◇ 4.请求函数

```
virtual int RequestRebuildQuote(XTPQuoteRebuildReq* rebuild_param) = 0;
```

四. 行情回补数据逻辑

当用户在UDP连接下订阅行情发现行情丢包时，可以使用回补数据服务器将丢失的数据回补。具体逻辑如下：

- （1）调用LoginToRebuildQuoteServer()登陆数据回补服务器。
- （2）调用RequestRebuildQuote()请求回补数据。
- （3）回补数据通过OnRebuildTickByTick()或者OnRebuildMarketData()回调给用户。
- （4）回补结果通过OnRequestRebuildQuote()通知结果。
  - 当回补结果成功，调用LogoutFromRebuildQuoteServer()登出。
  - 当回补结果失败，且rebuild_result.result_code = XTP_REBUILD_RET_PARTLY时，再次发起回补数据请求。

五. 注意事项

- 请仅在UDP连接下发生行情丢包时连接。
- 服务器有定时断线机制，请不要一直保持连接或者在断线后重连。
- 一次请不要请求太多数据，行情回补服务器有请求数据上限。