华锐高速行情平台 AMD

转码 API（AMA）C++ 开发指南

AMD 产品部

2021.06

CONTENTS

目录

修订历史			
日期	版本	AMA 版本号	修订说明
2019.9	0.1		创建
2020.03	0.2		补充精度，出入参说明及缺失的市场行情
2020.06	1.0	3.3.0	基于 V3.3 版本完善该文档
2020.06	1.1	3.4.0	基于 V3.4 版本完善该文档
2020.09	1.2.0	3.5.0	1. MDSnapshot 快照数据结构新增“当前品种交易状态”(instrument_status)、“基金 T-1 日收盘时刻 IOPV”(pre_close_iopv)、“债券加权平均委买价格”(alt_weighted_avg_bid_price)、“债券加权平均委卖价格”(alt_weighted_avg_offer_price)、“ETF 申购笔数”(etf_buy_number)、“ETF 申购数量”(etf_buy_amount)、“ETF 申购金额”(etf_buy_money)、“ETF 赎回笔数”(etf_sell_number)、“ETF 赎回数量”(etf_sell_amount)、“ETF 申购金额”(etf_sell_money)、“权证执行的总数量”（total_warrant_exec_volume)、“债券质押式回购品种加权”(war_lower_price)、“权证涨停价格”(war_upper_price)、“买入撤单笔数”(withdraw_buy_number)、“买入撤单数量”(withdraw_buy_amount)、“买入撤单金额”(withdraw_buy_money)、“卖出撤单笔数”(withdraw_sell_number)、“卖出撤单数量”(withdraw_sell_amount)、“卖出撤单金额”(withdraw_sell_money)、“买入总笔数”(total_bid_number)、“卖出总笔数”(total_offer_number)、“买入委托成交最大等待时间”(bid_trade_max_duration)、“卖出委托成交最大等待时间”(offer_trade_max_duration)、“买方委托价位数”(num_bid_orders)、“卖方委托价位数”(num_offer_orders)。
2020.10	1.2.1	3.5.1	1. 修改了 MDFutureSnapshot 的“合约代码”(security_code)字段长度，由 16 位改成了 32 位。
2020.11	1.3.0	3.6.0	1. 调整了 MDHKTSnapshot 港股快照行情结构中的“申买价”(bid_price)、“申买量”(bid_volume)、“申卖价”(offer_price)、“申卖量”(offer_volume)的档位，由原来的 1 档调成了 5 档 。 2. 增加了 OnMDHKTVCM 港股 VCM 数据回调接口以及 MDHKTVCM 数据结构。

2021.01	1.4.0	3.7.0	1. 增加 OnMDNEEQSecurityInfo 股转系统证券信息数据回调接口以及 MDNEEQSecurityInfo 数据结构。 2. 增加 OnMDNEEQNonPublicTransDeclaredInfo 股转系统非公开申报转让信息数据回调接口以及 MDNEEQNonPublicTransDeclaredInfo 数据结构。 3. 增加 OnMDNEEQHierarchicalInfo 股转系统分层信息数据回调接口以及 MDNEEQHierarchicalInfo 数据结构。 4. MDSnapshot 快照数据结构新增“最近成交时间”(last_trade_time)。 5. MDOptionSnapshot 期权快照数据结构新增“最近成交时间”(last_trade_time)和“参考价”(ref_price)。 6. OnEvent 接口中 EventCode 结构体增加行情数据升降级通知消息。
2021.3	1.5.0	3.8.0	1. 更新了 Cfg 结构体定义：1) “逐笔保序时间”（keep_order_timeout_ms）的单位调整成为毫秒 , 2) “数据压缩标志” tcp_compress_mode 数据压缩标志只有 TCP 模式 3) 删除了“线程工作模式”(polling)和“线程处理工作队列长度”(queue_size)以及各通道的独立扩展配置。 2. MDTickOrder 现货逐笔委托数据结构体中新增“原始订单号”（orig_order_no）和“业务序号”（biz_index）。 3. MDTickExecution 现货逐笔成交数据结构体中新增“业务序号”（biz_index）。
2021.4	1.6.0	3.9.0	1. 新增按市场、证券数据类型、证券品种类型、代码订阅接口(SubscribeData)。 2. 新增获取代码表接口(GetCodeTableList)。 3. 期权(MDOptionSnapshot)期货(MDFutureSnapshot)结构体新增字段，具体参考对应结构体。 4. 所有结构体增加品种类型字段（variety_category）。 5. 新增委托簿(目前只有 RedHat 系统的 c++和 java 接口支持)。
2021.6	1.6.1	3.9.1	1. 新增上海委托薄支持，同时支持新增 python 版本委托薄支持 2. 新增支持数据结构 MDNEEQNegotiableDeclaredInfo 3. 新增支持数据结构 MDNEEQMarketMakerDeclaredInfo 4. 新增支持数据结构 MDNEEQNonPublicTransferDealInfo 5. 新增支持数据结构 MDHKMarketStatus 6. 数据结构 MDHKTRealtimeLimit 新增字段 mkt_status

一、 引言

1. 文档目的

华锐高速行情平台 AMD 是华锐金融技术推出的低时延、高吞吐、高可用的行情分发平台。行情消费者可以通过华锐高速行情转码 API（AMA）对接 AMD，接收转码行情。

本文档是 AMA 的开发指南，包含了对 API 接口的说明以及示例，用于指引行情接收系统的开发人员基于此 AMA 进行行情接收功能的开发。

2. 术语和缩略语

术语、缩写	解释
AMD	Archforce Market Data，华锐高速行情平台
AMA	Archforce Market Data API，华锐高速行情转码 API

二、文件说明

文件	说明
c++/	C++语言接口文件
c++/include/ama.h	API 头文件
c++/include/ama_datatype.h	API 使用的类型定义
c++/include/ama_struct.h	API 使用的数据结构定义
c++/lib/*	API 所依赖的动态链接库
c++/ama_demo.cpp	仅供参考的开发样例程序
c++/Makefile	ama_demo 程序编译脚本
c++/bin/ama_test	编译好的测试可执行程序
c++/etc/*	测试程序的配置文件
c++/run.sh	测试程序执行脚本

三、参数及精度说明

1. 参数影响说明

1.1. channel_mode 参数

参数作用:

用来设置接入上游的方式，即和上游建立数据通道的方式

参数取值:

通道的优先级 kRDMA 到 kPCAP 优先级依次递减(优先级越高，时延越小,kRDMA 优先级最高，kPCAP 优先级最低)

模式	说明
kRDMA	RDMA 抓包方式获取数据
kEXA	EXA 抓包方式获取数据
kMDDP	mddp 网关组播方式获取数据
kAMI	AMI 组播方式获取数据
kTCP	TCP 流方式获取数据
kPCAP	libpcap 抓包方式获取数据

1.2. tcp_compress_mode 参数

参数作用:

如果是以 TCP 方式接入上游,此参数用来设置数据压缩方式,其他接入模式该参数不生效。

参数取值:

0 表示不压缩，1 表示华锐自定义数据压缩方式,2 表示 zstd 数据压缩方式。

1.3. ha_mode 参数说明

参数作用:

如果希望同时使用两种或两种以上的模式接入上游以达到更高级别的高可用效果（例如：

同时使用 AMI 和 TCP 模式接入，若 AMI 模式异常无法使用时仍然可以使用 TCP 模式接入的数据），此参数用来设置多个接入模式之间的高可用切换方式。如果只使用一种模式接入上游，建议设置成 kMasterSlaveA/kMasterSlaveB。

参数取值:

模式	说明
kMasterSlaveA	主备切换模式的 A 模式，该模式下检测到高优先级通道可用时，会立即切换到高优先级通道接收数据
kMasterSlaveB	主备切换模式的 B 模式，该模式下仅在当前使用通道出现问题需要切换时，才会切换到当前可用的最高优先级通道接收数据
kRegularDataFilter	规则数据过滤模式，该模式下上游通道以多活的方式接入，对数据进行滤重后下发

1.4. min_log_level 参数说明

参数作用:

用来设置最小日志级别(接口日志数据通过 OnLog 回调函数返回)

参数取值:

取值范围:kTrace,kDebug,kInfo,kWarn,kError,kFatal,日志严重级别从左到右依次递增

建议取值:kInfo

1.5. is_output_mon_data 参数说明

参数作用:

监控数据回调返回开关(监控数据通过 OnIndicator 回调函数返回)

参数取值:

false: OnIndicator 回调函数不会返回任何数据

true: OnIndicator 回调函数定时返回监控数据

1.6. keep_order 与 keep_order_timeout_ms 参数说明

参数作用:

keep_order 参数用来设置保序开关,keep_order_timeout_ms 用来设置保序超时时间(ms)

参数取值:

keep_order 设置为 true 时，会对收到的逐笔委托和逐笔成交数据进行全局保序，如果上游数据出现乱序或者缺失，会等待 keep_order_timeout_ms 超时时间，同时缓存逐笔数据。如果上游超时时间后，仍旧没有补齐数据，那么默认数据丢失，打印警告日志和增加监控丢失数量统计，下发缓存数据。keep_order 设置为 false 时，不对上游数据进行保序，直接滤重后实时下发数据。

注意事项:

订阅操作会过滤数据，如果只订阅了逐笔数据的部分代码，那么保序操作没有意义，计算出的数据缺失的原因有可能是因为订阅过滤导致序号不连续触发的，因此非全部代码订阅时不能打开逐笔保序开关。

1.7. is_subscribe_full 参数说明

参数作用:

初始订阅操作是否为全部订阅开关

参数取值:

true：初始订阅值为订阅所有数据

false：初始化订阅值为不订阅任何数据

1.8. is_thread_safe 参数说明

参数作用:

设置不同回调接口是否强制为同一线程回调

参数取值:

true：所有数据回调接口强制为同一线程回调

false：不同数据回调接口不保证为同一线程回调

1.9. 委托薄参数说明

enable_order_book	是否启用委托薄(true:启用，false:不启用)，如果启用委托薄，账户也需要有相应的委托薄权限
entry_size	委托薄回调档位数量
thread_num	委托薄并行计算线程数(默认值为 3)
order_queue_size	每个档位委托揭示笔数(每档最多揭示 50 笔)
order_book_deliver_interval_microsecond	委托簿递交时间间隔(微秒级)，设置为 0 代表只要达到平衡状态就可以递交委托薄数据，不需要等待。

2. 精度说明

为兼容多市场，华锐 API 为对各行情字段的数值取值统一了表现方式和相对实际值的倍数关系，具体表现如下:

1. 数值无论真实值是浮点型或是整形，统一倍数转换成整形数值。

2. 倍数转换转换规律如下:

1) 数量类型:实际为浮点数类型，其中小数位数为 2，接口处理将实际数值扩大了 100 倍。

2) 价格类型:实际为浮点数类型，其中小数位数为 6，接口处理将实际数值扩大了 1000000。

3) 总金额类型:实际为浮点数类型，其中小数位数为 5，接口处理将实际数值扩大了 100000。

4) 汇率类型:实际为浮点数类型，其中小数位数为 8，接口处理将实际数值扩大了 100000000。

四、兼容性说明

1. AMA 与服务端系统 AMD 的兼容性

AMA 不兼容低版本的 AMD，此接口不支持接入 3.9.1 之前版本的 AMD 系统，高版本的 AMD 兼容低版本的 AMA 接口。

2. AMA 对于操作系统的兼容性

系统	编译器	备注
RedHat7.2	gcc4.8.5	支持连接上游方式：TCP、AMI、PCAP、EXA、RDMA、MDDP 模式。 支持委托簿
Ubuntu16.04	gcc5.4.0	支持连接上游方式：TCP、AMI、PCAP、EXA、RDMA、MDDP 模式 支持委托簿
Ubuntu18.04	gcc8.4.0	支持连接上游方式：TCP、AMI、PCAP、EXA、RDMA、MDDP 模式 支持委托簿
Ubuntu20.04	gcc9.3.0	支持连接上游方式：TCP、AMI、PCAP、EXA、RDMA、MDDP 模式 支持委托簿
Gentoo2.7	gcc9.3.0	支持连接上游方式：TCP、AMI、PCAP、EXA、RDMA、MDDP 模式 支持委托簿
Windows7 及以上版本（32 和 64 位）	VC++ 2017	支持连接上游方式：TCP、MDDP 模式 支持委托薄

3. AMA 接口开发语言的支持

3.1. 支持的语言接口

C++、JAVA、Python

3.2. 接口效率对比说明

接口 C++语言处理效率比 JAVA、Python 要快，如果数据量较大而且机器性能比较差的情况下建议使用 C++ 语言接口开发。

五、各通道说明

1. TCP 通道模式

1.1. 数据接入方式

通过 TCP 方式和上游保持长连接，上游收到数据后主动推送转码数据给下游。

1.2. 关键参数设置

channel_mode:kTCP

tcp_compress_mode，具体参数使用请参考参数影响说明。

1.3. 数据过滤方式

TCP 接入上游方式数据过滤全部是由上游组件处理

1）账号权限:只会下发账户权限范围内的数据，权限范围外的数据过滤。

2）订阅信息:只会下发订阅范围内的数据，订阅范围外的数据过滤。

1.4. 异常情况说明

1）网络问题导致下游连接断开,接口会自动发起重连，无需调用特殊接口处理。

2）网络堵塞或者下游处理速度太慢会导致消息堆积，如果一直没有改善上游会将处理慢的下游连接断开，保证不同的连接之间不会互相影响。

2. AMI 通道模式说明

2.1. 数据接入方式

组播方式接入上游数据，上游收到数据后主动推送转码数据给下游。(需要确保上游和下游组播网络连通,设置此模式前请先用工具测试组播网络的连通性)

2.2. 关键参数设置

channel_mode:kAMI

2.3. 数据过滤方式

1)账号权限:只会下发账户权限范围内的数据，权限范围外的数据过滤(由上游处理)

2)订阅信息:下游过滤掉订阅范围以外的数据(由下游处理)

3. DAS 抓包通道模式说明

3.1. 数据接入方式

通过抓取网卡数据来获取原始数据，解码后通过回调递交转码数据。

3.2. 关键参数设置

channel_mode:kRDMA/kEXA/kPCAP

模式网卡要求说明:

kRDMA	网卡要求为支持 RDMA 功能的网卡
kEXA	网卡要求为 EXA 低延迟网卡
kPCAP	普通网卡

3.3. 数据过滤方式

1)账号权限:只会下发账户权限范围内的数据（权限由上游配置），权限范围外的数据过滤(由下游处理)

2)订阅信息:下游过滤掉订阅范围以外的数据(由下游处理)

六、C++开发接口

1. IAMDApi 接口

AMA 接口操作类。该类不需要创建实例，直接调用类方法即可。如 IAMDApi::GetVersion()。

1.1. GetVersion 方法

获取 AMA 版本信息。

函数原型：

static const char* GetVersion();

返回值：

版本信息字符串

1.2. Init 方法

初始化 AMA。

函数原型：

static int32_t Init(const IAMDSpi* pSpi, const Cfg& cfg);

参数：

参数	解释
pSpi(in)	IAMDSpi 的派生类实例指针，必须在调用 Release 函数之后才能销毁该实例
cfg (in)	AMA 内部需要的配置参数

返回值：

错误代码，详见 ama_datatype.h 中的 ErrorCode

Cfg 相关的结构定义在 ama_struct.h 中，如下所示：

```
/**
 * @name 配置定义
 * @{ */
struct Cfg
```

```
{
    //------------------------------全局配置信息------------------------------------------------------------
    uint64_t    channel_mode;                           // 通道模式的集合，请参考 ChannelMode，该配置为各通道模式的集合
    uint32_t    ha_mode;                                // 高可用工作模式，请参考 HighAvailableMode
    int32_t     min_log_level;                          // 日志最小级别，请参考 LogLevel
    bool        is_output_mon_data;                     // 是否输出监控数据的配置，true-输出监控数据，false-不输出监控数据
    bool        is_thread_safe;                         // 回调接口是否保证线程安全，true-启用线程安全模式执行回调接口，
false-非线程安全模式执行回调接口

    bool        keep_order;                             // 逐笔保序标志，true-开启保序，false-开启不保序
    uint32_t    keep_order_timeout_ms;                  // 逐笔保序超时时间（单位：毫秒），keep_order=true 时有效
    bool        is_subscribe_full;                      // 默认是否订阅全部数据，true-默认订阅全部，false-默认不订阅任何数据

    //------------------------------UMS 服务的连接信息------------------------------------------------------
    UMSItem     ums_servers[ConstField::kUMSItemLen];           // UMS 的服务信息项，该信息不能超过 10 个
    uint32_t    ums_server_cnt;                                 // UMS 的服务信息项个数，小于 1 将启动失败

    char        username[ConstField::kUsernameLen];             // 用户名
    char        password[ConstField::kPasswordLen];             // 用户密码，明文填入，密文使用

    uint32_t    tcp_compress_mode;              // TCP 模式传输数据压缩标志,0:不压缩 1:自定义压缩 2:zstd 压缩(仅 TCP 模式有效)
    Cfg()
    {
        channel_mode = 0;
        ha_mode = HighAvailableMode::kRegularDataFilter;
        min_log_level = LogLevel::kInfo;
        is_output_mon_data = false;
        is_thread_safe = false;
        keep_order = false;
        keep_order_timeout_ms = 3000;
        is_subscribe_full = false;
        ums_server_cnt = 0;
        tcp_compress_mode = 0;
    }
};
/**  @} */
```

示例代码如下：

```
    amd::ama::Cfg cfg; // 准备 AMA 配置
    /*
```

```
        通道模式设置及各个通道说明:
        cfg.channel_mode = amd::ama::ChannelMode::kTCP;   ///< TCP 方式计入上游行情系统
        cfg.channel_mode = amd::ama::ChannelMode::kAMI;   ///< AMI 组播方式接入上游行情系统
        cfg.channel_mode = amd::ama::ChannelMode::kRDMA;  ///< 开启硬件加速 RDMA 通道,抓取网卡数据包数据
        cfg.channel_mode = amd::ama::ChannelMode::kEXA;   ///< 开启硬件加速 EXA 通道,抓取网卡数据包数据
        cfg.channel_mode = amd::ama::ChannelMode::kPCAP;  ///< 开启硬件加速 PCAP 通道,抓取网卡数据包数据
        cfg.channel_mode = amd::ama::ChannelMode::kMDDP;  ///< 直接接入交易所网关组播数据，现在只有深圳交易所开通了此服务
        cfg.channel_mode = amd::ama::ChannelMode::kTCP|amd::ama::ChannelMode::kAMI;  ///< 同时通过 TCP 方式和 AMI 组播方式接入上游，通过 cfg.ha_mode 设置对应的高可用设置模式
    */
    cfg.channel_mode = amd::ama::ChannelMode::kTCP;
    cfg.tcp_compress_mode = 0;  //TCP 传输数据方式: 0 不压缩 1 华锐自定义压缩 2 zstd 压缩(仅 TCP 模式有效)

    /*
        通道高可用模式设置
        1. cfg.channel_mode 为单通道时,建议设置值为 kMasterSlaveA / kMasterSlaveB
        2. cfg.channel_mode 混合开启多个通道时,根据需求设置不同的值
            1) 如果需要多个通道为多活模式接入,请设置 kRegularDataFilter 值
            2) 如果需要多个通道互为主备接入，请设置值为 kMasterSlaveA / kMasterSlaveB,kMasterSlaveA / kMasterSlaveB 差别请参看注释说明
                通道优先级从高到低依次为 kRDMA/kEXA/kMDDP/kAMI/kTCP/kPCAP
    */
    cfg.ha_mode = amd::ama::HighAvailableMode::kMasterSlaveA;
    cfg.min_log_level = amd::ama::LogLevel::kInfo; // 设置日志最小级别：Info 级, AMA 内部日志通过 OnLog 回调函数返回

    /*
        设置是否输出监控数据: true(是), false(否), 监控数据通过 OnIndicator 回调函数返回
        监控数据格式为 json, 主要监控数据包括订阅信息，数据接收数量统计
        数据量统计:包括接收数量和成功下发的数量统计,两者差值为过滤的数据量统计
        eg: "RecvSnapshot": "5926", "SuccessSnapshot": "5925",表示接收了快照数据 5926 个,成功下发 5925 个，过滤数据为 5926 - 5925 = 1 个
            过滤的数据有可能为重复数据或者非订阅数据
    */
    cfg.is_output_mon_data = false;

    /*
        设置逐笔保序开关: true(开启保序功能) , false(关闭保序功能)
        主要校验逐笔成交数据和逐笔委托数据是否有丢失,如果丢失会有告警日志,缓存逐笔数据并等待 keep_order_timeout(单位 s)时间等待上游数据重传,
        如果超过此时间,直接下发缓存数据,默认数据已经丢失,如果之后丢失数据过来也会丢弃。
        同时由于深圳和上海交易所都是通道内序号连续,如果打开了保序开关,必须订阅全部代码的逐笔数据,否则一部分序号会被订阅过滤,导致数据超时等待以及丢失告警。
    */
    cfg.keep_order = false;
```

```
    cfg.keep_order_timeout_ms = 3000;

   cfg.is_subscribe_full = false; //设置默认订阅: true 代表默认全部订阅, false 代表默认全部不订阅

    /*
        配置 UMS 信息:
        username/password 账户名/密码, 一个账户只能保持一个连接接入
        ums 地址配置:
            1) ums 地址可以配置 1-8 个 建议值为 2 互为主备, ums_server_cnt 为此次配置 UMS 地址的个数
            2) ums_servers 为 UMS 地址信息数据结构:
                local_ip 为本地地址,填 0.0.0.0 或者本机 ip
                server_ip 为 ums 服务端地址
                server_port 为 ums 服务端端口
    */
    strcpy(cfg.username, "user1");
    strcpy(cfg.password, "pass1234");
    cfg.ums_server_cnt = 2;
    strcpy(cfg.ums_servers[0].local_ip, "0.0.0.0");
    strcpy(cfg.ums_servers[0].server_ip, "192.168.1.101");
    cfg.ums_servers[0].server_port = 6001;

    strcpy(cfg.ums_servers[1].local_ip, "0.0.0.0");
    strcpy(cfg.ums_servers[1].server_ip, "192.168.1.102");
    cfg.ums_servers[1].server_port = 6001;

    /*
        业务数据回调接口(不包括 OnIndicator/OnLog 等功能数据回调)的线程安全模式设置:
        true: 所有的业务数据接口为接口集线程安全
        false: kTCP 通道模式情况下,所有业务数据接口为接口集线程安全
               kAMI 通道模式下, 如果 domainserver 不配置主题多线程下发的情况下所有业务数据接口为接口集线程安全(默认
domainserver 不开启)
               kMDDP 通道模式下,所有业务数据接口为接口集线程安全
               kRDMA/KEXA/kPCAP 抓包模式下, 业务接口单接口为线程安全,接口集非线程安全
    */
    cfg.is_thread_safe = false;


    MySpi spi;   //初始化回调类

    /*
        初始化回调以及配置信息,此函数为异步函数, 如果通道初始化以及登陆出现问题会通过 onLog / onEvent 返回初始化结果信息
    */
    if (amd::ama::IAMDApi::Init(&spi, cfg)
        != amd::ama::ErrorCode::kSuccess)
```

```
    {
        std::lock_guard<std::mutex> _(g_mutex);
        std::cout << "Init AMA failed" << std::endl;
        amd::ama::IAMDApi::Release();
        return -1;
    }

    /*
        订阅信息设置:
        1. 订阅信息分三个维度 market:市场, flag:数据类型(比如现货快照, 逐笔成交, 指数快照等), 证券代码
        2. 订阅操作有三种:
            kSet 设置订阅, 以市场为单位覆盖订阅信息
            kAdd 增加订阅, 在前一个基础上增加订阅信息(ps: 现阶段接口如果之前订阅了某个类型(如 kSnapshot)的全部代码,不支持其余类型(kTickOrder)的部分代码增加)
            kDel 删除订阅, 在前一个基础上删除订阅信息(ps: 现阶段接口如果之前订阅了某个类型的全部代码,不支持删除其中部分代码)
            kCancelAll 取消所有订阅信息
    */
    amd::ama::SubscribeItem sub[3];
    memset(sub, 0, sizeof(sub));

    /* 订阅深交所全部证券的逐笔委托和逐笔成交数据 */
    sub[0].market = amd::ama::MarketType::kSZSE;
    sub[0].flag = amd::ama::SubscribeDataType::kTickOrder
                | amd::ama::SubscribeDataType::kTickExecution;
    sub[0].security_code[0] = '\0';

    /* 订阅上交所 600000、600004 两只证券的全部数据 */
    sub[1].market = amd::ama::MarketType::kSSE;
    sub[1].flag = amd::ama::SubscribeDataType::kNone;
    strcpy(sub[1].security_code, "600000");
    sub[2].market = amd::ama::MarketType::kSSE;
    sub[2].flag = amd::ama::SubscribeDataType::kNone;
    strcpy(sub[2].security_code, "600004");

    /* 发起订阅 */
    if (amd::ama::IAMDApi::SubscribeData(
                amd::ama::SubscribeType::kSet, sub, 3)
        != amd::ama::ErrorCode::kSuccess)
    {
        std::lock_guard<std::mutex> _(g_mutex);
        std::cout << "Subscribe data failed" << std::endl;
        amd::ama::IAMDApi::Release();
        return -1;
```

```
}
```

1.3. Join 方法

Join AMA 内部工作线程。非必调用的函数。仅当应用将各回调中的首地址直接转移至其他工作线程，而不在当前回调直接调用 IAMDApi::FreeMemory(snapshots)删除数据的情况下，需先调用 Join 将 AMA 内部工作线程停止掉，然后再将转移的数据调用 IAMDApi::FreeMemory(snapshots)删除，最后再调用 Release 彻底关闭 AMA。具体请看一下代码示例。

函数原型：

static void Join();

参数：

无

1.4. Release 方法

释放 AMA。

函数原型：

static int32_t Release();

返回值：

错误代码，详情请参考 ErrorCode

1.5. FreeMemory 方法

释放内存。

AMA 行情数据回调后，其数据指针所有权归应用所有，所以调用该接口释放内存。回调用除 OnLog、OnIndicator、OnEvent 外，其余都需要显示的调用 FreeMemroy 释放内存，否则会造成内存泄漏。

函数原型：

static void FreeMemory(void* data);

参数：

参数	解释
data (in)	需要被释放的内存首地址

1.6. SubscribeData 方法

1.6.1. 权限订阅类型

根据市场、数据权限类型、股票代码订阅行情数据。

函数原型：

static int32_t SubscribeData(

int32_t subscribe_type,

const SubscribeItem* item,

int32_t cnt);

参数：

参数	解释
subscribe_type(in)	订阅类型，取值参见 SubscribeType 类型定义
item(in)	订阅信息数据项首地址
cnt(in)	订阅信息数据项个数

SubscribeItem 结构定义如下：

```
/**
 * @name 订阅数据项定义
 * @{ */
struct SubscribeItem
{
    int32_t market;                                 // 市场类型，参考 MarketType，为 0 表示订阅所有支持的市场
    uint64_t flag;                                  // 各数据权限类型的集合，参考 SubscribeDataType，为 0 表示订阅所有支持
```

```
的数据类型
    char security_code[ConstField::kSecurityCodeLen];    // 证券代码，为空表示订阅所有代码
};
/**  @} */
```

权限订阅数据类型结构体定义：

```
class SubscribeDataType
{
public:
    static const uint64_t kNone                                = 0x000000000000;    ///< 订阅全部数据
    static const uint64_t kSnapshot                            = 0x000000000001;    ///< 订阅现货快照数据
    static const uint64_t kTickExecution                       = 0x000000000002;    ///< 订阅逐笔成交数据
    static const uint64_t kTickOrder                           = 0x000000000004;    ///< 订阅逐笔委托数据
    static const uint64_t kOrderQueue                          = 0x000000000008;    ///< 订阅委托队列数据
    static const uint64_t kIndexSnapshot                       = 0x000000000010;    ///< 订阅指数快照数据
    static const uint64_t kFutureSnapshot                      = 0x000000000020;    ///< 订阅期货连线数据
    static const uint64_t kOptionSnapshot                      = 0x000000000040;    ///< 订阅期权快照数据
    static const uint64_t kHKTSnapshot                         = 0x000000000080;    ///< 订阅港股快照数据
    static const uint64_t kAfterHourFixedPriceSnapshot         = 0x000000000100;    ///< 订阅上交所盘后定价快照数据
    static const uint64_t kAfterHourFixedPriceTickExecution    = 0x000000000400;    ///< 订阅上交所盘后定价逐笔成交数据
    static const uint64_t kCSIIndexSnapshot                    = 0x000000000800;    ///< 订阅中证指数快照数据
    static const uint64_t kNEEQSnapshot                        = 0x000000001000;    ///< 订阅北交所快照数据
};
```

设置订阅示例代码如下：

```
    /*
        订阅信息设置:
        1. 订阅信息分三个维度 market:市场, flag:数据类型(比如快照, 逐笔成交, 逐笔成交等), 证券代码
        2. 订阅操作有三种:
            kSet 设置订阅, 以市场为单位覆盖订阅信息
            kAdd 增加订阅, 在前一个基础上增加订阅信息(ps: 现阶段接口如果之前订阅了某个类型(如 kSnapshot)的全部代码,不支持其余类型(kTickOrder)的部分代码增加)
            kDel 删除订阅, 在前一个基础上删除订阅信息(ps: 现阶段接口如果之前订阅了某个类型的全部代码,不支持删除其中部分代码)
            kCancelAll 取消所有订阅信息
    */
    amd::ama::SubscribeItem sub[3];
    memset(sub, 0, sizeof(sub));

    /* 订阅深交所全部证券的逐笔委托和逐笔成交数据 */
```

```
sub[0].market = amd::ama::MarketType::kSZSE;
sub[0].flag = amd::ama::SubscribeDataType::kTickOrder
              | amd::ama::SubscribeDataType::kTickExecution;
sub[0].security_code[0] = '\0';

/* 订阅上交所 600000、600004 两只证券的全部数据 */
sub[1].market = amd::ama::MarketType::kSSE;
sub[1].flag = amd::ama::SubscribeDataType::kNone;
strcpy(sub[1].security_code, "600000");
sub[2].market = amd::ama::MarketType::kSSE;
sub[2].flag = amd::ama::SubscribeDataType::kNone;
strcpy(sub[2].security_code, "600004");

/* 发起订阅 */
if (amd::ama::IAMDApi::SubscribeData(
            amd::ama::SubscribeType::kSet, sub, 3)
    != amd::ama::ErrorCode::kSuccess)
{
    std::lock_guard<std::mutex> _(g_mutex);
    std::cout << "Subscribe data failed" << std::endl;
    amd::ama::IAMDApi::Release();
    return -1;
}
```

添加订阅代码示例如下：

```
// 添加订阅信息
amd::ama::SubscribeItem sub;

// 订阅上交所 600005 证券的逐笔委托和逐笔成交数据
sub.market = amd::ama::MarketType::kSSE;
sub.flag = amd::ama::SubscribeDataType::kTickOrder
              | amd::ama::SubscribeDataType::kTickExecution;
strcpy(sub.security_code, "600005");

// 订阅数据
if (amd::ama::IAMDApi::SubscribeData(
            amd::ama::SubscribeType::kAdd, &sub, 1)
    != amd::ama::ErrorCode::kSuccess)
{
    std::lock_guard<std::mutex> _(g_mutex);
```

```
    std::cout << "Subscribe data failed" << std::endl;
    return -1;
}
```

删除订阅代码示例如下：

```
// 添加订阅信息
amd::ama::SubscribeItem sub;

// 订阅上交交所 600005 证券的逐笔委托和逐笔成交数据
sub.market = amd::ama::MarketType::kSSE;
sub.flag = amd::ama::SubscribeDataType::kTickOrder
           | amd::ama::SubscribeDataType::kTickExecution;
strcpy(sub.security_code, "600005");

// 订阅数据
if (amd::ama::IAMDApi::SubscribeData(
        amd::ama::SubscribeType::kDel, &sub, 1)
    != amd::ama::ErrorCode::kSuccess)
{
    std::lock_guard<std::mutex> _(g_mutex);
    std::cout << "Subscribe data failed" << std::endl;
    return -1;
}
```

取消所有订阅如下：

```
// 取消所有订阅信息
amd::ama::SubscribeItem sub;
// 订阅数据
if (amd::ama::IAMDApi::SubscribeData(
        amd::ama::SubscribeType::kCancelAll, &sub, 1)
    != amd::ama::ErrorCode::kSuccess)
{
    std::lock_guard<std::mutex> _(g_mutex);
    std::cout << "Subscribe data failed" << std::endl;
    return -1;
}
```

各订阅数据类型与数据类型对应关系如下：

订阅权限数据类型	数据类型
kSnapshot	MDSnapshot MDIndicatorOfTradingVolumeSnapshot
kTickExecution	MDTickExecution MDRefinancingTickExecution MDNegotiableTickExecution
kTickOrder	MDTickOrder MDRefinancingTickOrder MDNegotiableTickOrder
kOrderQueue	MDOrderQueue
kIndexSnapshot	MDIndexSnapshot MDCnIndexSnapshot
kFutureSnapshot	MDFutureSnapshot
kOptionSnapshot	MDOptionSnapshot
kHKTSnapshot	MDHKTSnapshot MDHKTRealtimeLimit MDHKTProductStatus MDHKTVCM
kAfterHourFixedPriceSnapshot	MDAfterHourFixedPriceSnapshot
kAfterHourFixedPriceTickExecution	MDAfterHourFixedPriceTickExecution
kCSIIndexSnapshot	MDCSIIndexSnapshot
kNEEQSnapshot	OnMDNEEQSnapshot OnMDNEEQNonPublicTransDeclaredInfo OnMDNEEQHierarchicalInfo

1.6.2. 品种订阅类型

根据市场类型、证券数据类型、证券品种类型、股票代码订阅行情数据

函数原型：

```
static int32_t SubscribeData(
                    int32_t subscribe_type,
```

const SubscribeCategoryItem* item,

int32_t cnt);

参数：

参数	解释
subscribe_type(in)	订阅类型，取值参见 SubscribeType 类型定义
item(in)	订阅信息数据项首地址
cnt(in)	订阅信息数据项个数

订阅条目结构体定义：

```
struct SubscribeCategoryItem
{
    int32_t market;                                    // 市场类型，参考 MarketType，为 0 表示订阅所有支持的市场
    uint64_t data_type;                                // 各数据证券数据类型集合，为 0 表示订阅所有支持的证券数据类型
    uint64_t category_type;                            // 各数据证券品种类型集合，为 0 表示订阅所有支持的证券品种数据类型
    char security_code[ConstField::kFutureSecurityCodeLen]; // 证券代码，为空表示订阅所有代码
};
```

证券数据类型定义：

```
//证券数据类型
class  SubscribeSecuDataType
{
public:
        static  const  uint64_t  kNone           =  0x000000000000;        ///<  订阅全部证券数据类别
        static  const  uint64_t  kSnapshot       =  0x000000000001;        ///<  订阅快照数据类别
        static  const  uint64_t  kTickExecution  =  0x000000000002;        ///<  订阅逐笔成交数据
        static  const  uint64_t  kTickOrder      =  0x000000000004;        ///<  订阅逐笔委托数据
        static  const  uint64_t  kOrderQueue     =  0x000000000008;        ///<  订阅委托队列数据
};
```

证券品种类型定义：

```
//证券品种类型
class  SubscribeCategoryType
```

```
{
public:
        static  const  uint64_t  kNone           =  0x000000000000;        ///<  订阅全部证券品种类别
        static  const  uint64_t  kStock          =  0x000000000001;        ///<  订阅股票证券品种类别
        static  const  uint64_t  kFund           =  0x000000000002;        ///<  订阅基金证券品种类别
        static  const  uint64_t  kBond           =  0x000000000004;        ///<  订阅债券证券品种类别
        static  const  uint64_t  kIndex          =  0x000000000008;        ///<  订阅指数证券品种类别
        static  const  uint64_t  kHKT            =  0x000000000010;        ///<  订阅港股通证券品种类别
        static  const  uint64_t  kOption         =  0x000000000020;        ///<  订阅期权证券品种类别
        static  const  uint64_t  kFutureOption   =  0x000000000040;        ///<  订阅期货期权证券品种类别
        static  const  uint64_t  kOthers         =  0x100000000000;        ///<  订阅其他证券品种类别
};
```

示例：

```
//订阅示例 2
{
    /*
    按品种类型订阅信息设置:
    1. 订阅信息分三个维度 market:市场, data_type:证券数据类型(), category_type:品种类型, security_code:证券代码
    2. 订阅操作有三种:
        kSet 设置订阅, 以市场为单位覆盖订阅信息
        kAdd 增加订阅, 在前一个基础上增加订阅信息
        kDel 删除订阅, 在前一个基础上删除订阅信息
        kCancelAll 取消所有订阅信息
    */
    amd::ama::SubscribeCategoryItem sub1[3];
    memset(sub1, 0, sizeof(sub1));

    /* 订阅深交所全部证券代码的股票逐笔委托和逐笔成交数据 */
    sub1[0].market = amd::ama::MarketType::kSZSE;
    sub1[0].data_type = amd::ama::SubscribeDataType::kSnapshot
                    | amd::ama::SubscribeDataType::kTickExecution;
    sub1[0].category_type = amd::ama::SubscribeCategoryType::kStock;
    sub1[0].security_code[0] = '\0';

    /* 订阅上交所全部证券代码的基金逐笔委托和逐笔成交数据 */
    sub1[1].market = amd::ama::MarketType::kSSE;
    sub1[1].data_type = amd::ama::SubscribeDataType::kSnapshot
                    | amd::ama::SubscribeDataType::kTickExecution;
    sub1[1].category_type = amd::ama::SubscribeCategoryType::kFund;
    sub1[1].security_code[0] = '\0';
```

```
    /* 订阅上交所全部证券代码的基金逐笔委托和逐笔成交数据 */
    sub1[2].market = amd::ama::MarketType::kSSE;
    sub1[2].data_type = amd::ama::SubscribeDataType::kSnapshot
                | amd::ama::SubscribeDataType::kTickExecution;
    sub1[2].category_type = amd::ama::SubscribeCategoryType::kFund;
    memset(sub1[2].security_code, 0 , sizeof(sub1[2].security_code));
    strcpy(sub1[2].security_code, "600004");

    /* 发起订阅 */
    if (amd::ama::IAMDApi::SubscribeData(
                amd::ama::SubscribeType::kSet, sub1, 3)
        != amd::ama::ErrorCode::kSuccess)
    {
        std::cout << "Subscribe data failed" << std::endl;
        amd::ama::IAMDApi::Release();
        return -1;
    }
}
```

1.7. GetCodeTableList

代码表请求操作

函数原型：

static bool GetCodeTableList(CodeTableRecordList& list);

CodeTableRecordList 定义：

```
struct CodeTableRecord
{
    char security_code[ConstField::kFutureSecurityCodeLen];          // 证券代码
    uint8_t market_type;                                             // 证券市场
    char symbol[ConstField::kSymbolLen];                             // 简称
    char english_name[ConstField::kSecurityAbbreviationLen];         // 英文名
    char security_type[ConstField::kMaxTypesLen];                    // 交易所证券类别
    char    currency[ConstField::kTypesLen];                         // 货币种类
    uint8_t variety_category;                                        // 品种类型
    int64_t pre_close_price;                                         // 昨收价
    int64_t close_price;                                             // 收盘价
    char underlying_security_id[ConstField::kSecurityCodeLen];       // 标的代码
    char    contract_type[ConstField::kMaxTypesLen];                 // 合约类别
```

```
    int64_t exercise_price;                                     // 行权价
    uint32_t expire_date;                                       // 行权日期
    int64_t high_limited;                                       // 涨停价
    int64_t low_limited;                                        // 跌停价
};

struct CodeTableRecordList
{
    uint32_t list_nums;
    CodeTableRecord* records;
};
```

示例：

```
// 代码表接口获取示例
void GetCodeList()
{
    amd::ama::CodeTableRecordList list;
    bool ret = amd::ama::IAMDApi::GetCodeTableList(list);
    if(ret)
    {
        for(uint32_t i=0; i< list.list_nums; i++)
        {
            /*
                handle list.records
                records 是代码表数据头指针
            */
            std::lock_guard<std::mutex> _(g_mutex);
            std::cout << "Receive tick execution: " << std::endl
                      << "    market:        "
                      << (uint32_t)list.records[i].market_type << std::endl
                      << "    security_code: "
                      << list.records[i].security_code << std::endl
                      << "    variety_category: "
                      << (uint32_t)list.records[i].variety_category << std::endl;
        }
        if(list.list_nums > 0)
            amd::ama::IAMDApi::FreeMemory(list.records);  //释放代码表内存池数据
    }
}
```

2. IAMDSpi 接口

AMA 中的接收数据的回调基类。使用 AMA 时需要继承该类，并将派生类的实例传递给 IAMDApi::Init 函数，供 AMA 回调使用。为保证程序正确运行，必须保证该实例生命周期长于 AMA。

2.1. OnLog 方法

接收日志数据回调。

函数原型:

```
void OnLog(const int32_t& level, const char* log, uint32_t len);
```

参数:

参数	解释
level（out）	日志数据级别，(详见 ama_datatype.h 中的 LogLevel)
log(out)	日志内容
len(out)	日志内容长度

LogLevel 结构定义如下：

```
class LogLevel
{
public:
    /**
     * @brief 日志输出级别定义
     */
    enum { kTrace = 0, kDebug = 1, kInfo  = 2, kWarn  = 3, kError = 4, kFatal = 5, };
};
```

2.2. OnIndicator 方法

接收监控数据回调。

函数原型：

void OnIndicator(const char* indicator, uint32_t len);

参数：

参数	解释
indicator（out)	监控数据内容，格式为 JSON 字符串
len(out)	监控内容长度

1、json 字符串的内容

Json 字符串主要分为三个部分

（1）、通道状态信息

通道状态信息主要包含通道状态信息和接收数据统计的数量信息。

（2）、订阅信息

订阅信息主要包含订阅的市场类型、数据结构类型，品种类型，订阅的代码。

（3）、逐笔保序信息

逐笔保序信息主要包含市场收到的逐笔数和市场遗失的逐笔数。

2、json 字符串示例：

```
"class_name": "AppHandlerImpl",
    "class_objects": [
        {
            "object_name": "AMA",
            "TCP": {
                "[TCP|AMA_L2|]": {
                    "RemoteIp": "",
                    "RemotePort": "0",
                    "LocalIp": "",
                    "LocalPort": "0",
                    "Status": "kDisConnect",
                    "RecvTotalBytes": "0",
                    "RecvTotalPkg": "0",
                    ......
```

```
                "RecvSnapshotCnt": "0",
                "SuccessSnapshotCnt": "0",
                ......
            }
        },
        "SubscribeInfo": [
        {
            "Market": "SSE",
            "SubscribeDataTypes": [
                {
                    "SubDataType": "Snapshot",
                    "CategorysType": [
                        {
                            "CategoryType": "Stock",
                            "SubCodeList": "[ 600000 ]"
                        }
                    ]
                }
        ],
        "TickSerialize": {
            "SSERecvTicks": "0",
            ......
        }
    ]
}
```


2.3. OnEvent 方法

接收事件通知回调，使用者可根据该回调事件做相应的处理。

函数原型：

void OnEvent(uint32_t level, uint32_t code,const char* event_msg, uint32_t len);

参数：

参数	解释
level（out)	事件级别，参考 ama_datatype.h 中 EventLevel
code(out)	事件代码，参考 ama_datatype.h 中 EventCode
event_msg(out)	事件具体信息
len(out)	事件具体信息长度

EventLevel 结构定义如下：

```
class EventLevel
{
public:
    /**
     * @brief 事件级别定义
     */
    enum
    {
        kInfo  = 1, ///< 普通事件
        kWarn  = 2, ///< 告警事件
        kError = 3  ///< 错误事件，比较严重，需要介入处理
    };
};
```

EventCode 结构定义如下：

```
class EventCode
{
public:
    enum
    {
        /*-------------------------与 UMS 会话层事件信息----------------------*/
        kUMSConnectSuccess                  = 1,                    // 连接 UMS 成功
        kUMSConnectFailed,                                          // 连接 UMS 失败
        kUMSLogonSuccess,                                           // 登录 UMS 成功
        kUMSLogonFailed,                                            // 登录 UMS 失败
        kUMSHeartbeatTimeout,                                       // 与 UMS 心跳超时

        /*-------------------------硬件加速通道相关事件----------------------*/
```

```
        kChannelRDMAInitSuccess,                                    // RDMA 通道开启成功
        kChannelRDMAInitFailed,                                     // RDMA 通道开启失败
        kChannelEXAInitSuccess,                                     // EXA 通道开启成功
        kChannelEXAInitFailed,                                      // EXA 通道开启失败
        kChannelPCAPInitSuccess,                                    // PCAP 通道开启成功
        kChannelPCAPInitFailed,                                     // PCAP 通道开启失败

        /*-----------------------CTP 硬件加速相关事件-----------------------*/
        kCTPDASStreamStart,                                         // 通道数据流开始
        kCTPDASStreamFinished,                                      // 通道数据流结束

        /*-----------------------SzseBinary 硬件加速相关事件-----------------------*/
        kSzseBinaryDASStreamStart,                                  // 通道数据流开始
        kSzseBinaryDASStreamFinished,                               // 通道数据流结束

        /*-----------------------SseLDDS 硬件加速相关事件-----------------------*/
        kSseLDDSDASStreamStart,                                     // 通道数据流开始
        kSseLDDSDASStreamFinished,                                  // 通道数据流结束

        /*-----------------------AMI 组播通道相关事件-----------------------*/
        kChannelAMIInitSuccess,                                             // AMI 通道开启成功
        kChannelAMIInitFailed,                                              // AMI 通道开启失败

        /*-----------------------TCP 通道相关事件-----------------------*/
        kChannelTCPInitSuccess,                                 // TCP 通道开启成功
        kChannelTCPInitFailed,                                  // TCP 通道开启失败
        kChannelTCPConnectSuccess,                              // TCP 通道连接成功
        kChannelTCPConnectFailed,                               // TCP 通道连接失败
        kChannelTCPLogonSuccess,                                // TCP 通道登录成功
        kChannelTCPLogonFailed,                                 // TCP 通道登录失败
        kChannelTCPSessionClosed,                               // TCP 通道连接断开
        kChannelTCPHeartbeatTimeout,                            // TCP 通道会话心跳失败
        kChannelTCPMarketDataDegrade,                           // TCP 通道行情数据降级
        kChannelTCPMarketDataUpgrade,                           // TCP 通道行情数据升级
        /*-----------------------主备切换相关事件-----------------------*/
        kSourceMasterSlaveChanged,                                  // 主备源切换
    };
};
```

2.4. OnMDSnapshot 方法

接收现货快照数据回调。

函数原型：

void OnMDSnapshot(MDSnapshot* snapshot, uint32_t cnt);

参数：

参数	解释
snapshot(out)	现货快照数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDSnapshot 结构体定义如下：

```
/**
 * @name 现货快照数据信息结构定义
 * @{ */
struct MDSnapshot
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    int64_t orig_time;                                          // 时间（YYYYMMDDHHMMSSsss）
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];   // 产品实时阶段及标志
    //上海现货快照交易状态
    //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 0 位：‘S’表示启动（开市前）时段，‘C’表示集合竞价时段，‘T’表示连续交易时段，
    //‘E’表示闭市时段 ，‘P’表示临时停牌，
    //‘M’表示可恢复交易的熔断（盘中集合竞价），‘N’表示不可恢复交易的熔断（暂停交易至闭市）
    //‘U’表示收盘集合竞价
    //第 1 位：‘0’表示此产品不可正常交易，‘1’表示此产品可正常交易。
    //第 2 位：‘0’表示未上市，‘1’表示已上市
    //第 3 位：‘0’表示此产品在当前时段不接受进行新订单申报，‘1’ 表示此产品在当前时段可接受进行新订单申报。

    //深圳现货快照交易状态
    //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
    int64_t pre_close_price;                                    // 昨收价，实际值需除以 1000000
    int64_t open_price;                                         // 开盘价，实际值需除以 1000000
    int64_t high_price;                                         // 最高价，实际值需除以 1000000
    int64_t low_price;                                          // 最低价，实际值需除以 1000000
    int64_t last_price;                                         // 最新价，实际值需除以 1000000
    int64_t close_price;                                        // 收盘价，实际值需除以 1000000
    int64_t bid_price[ConstField::kPositionLevelLen];           // 申买价，实际值需除以 1000000
    int64_t bid_volume[ConstField::kPositionLevelLen];          // 申买量，实际值需除以 100
    int64_t offer_price[ConstField::kPositionLevelLen];         // 申卖价，实际值需除以 1000000
```

```
    int64_t offer_volume[ConstField::kPositionLevelLen];          // 申卖量，实际值需除以 100
    int64_t num_trades;                                           // 成交笔数
    int64_t total_volume_trade;                                   // 成交总量，实际值需除以 100
    int64_t total_value_trade;                                    // 成交总金额，实际值需除以 100000
    int64_t total_bid_volume;                                     // 委托买入总量，实际值需除以 100
    int64_t total_offer_volume;                                   // 委托卖出总量，实际值需除以 100
    int64_t weighted_avg_bid_price;                               // 加权平均为委买价格，实际值需除以 1000000
    int64_t weighted_avg_offer_price;                             // 加权平均为委卖价格，实际值需除以 1000000
    int64_t IOPV;                                                 // IOPV 净值估产，实际值需除以 1000000
    int64_t yield_to_maturity;                                    // 到期收益率，实际值需除以 1000
    int64_t high_limited;                                         // 涨停价，实际值需除以 1000000
    int64_t low_limited;                                          // 跌停价，实际值需除以 1000000
    int64_t price_earning_ratio1;                                 // 市盈率 1，实际值需除以 1000000
    int64_t price_earning_ratio2;                                 // 市盈率 2，实际值需除以 1000000
    int64_t change1;                                              // 升跌 1（对比昨收价），实际值需除以 1000000
    int64_t change2;                                              // 升跌 2（对比上一笔），实际值需除以 1000000
    int32_t channel_no;                                           // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];
    char    instrument_status[ConstField::kTradingPhaseCodeLen];  // 当前品种交易状态
    int64_t pre_close_iopv;                                       // 基金 T-1 日收盘时刻 IOPV，实际值需除以
1000000
    int64_t alt_weighted_avg_bid_price;                           // 债券加权平均委买价格，实际值需除以
1000000
    int64_t alt_weighted_avg_offer_price;                         // 债券加权平均委卖价格，实际值需除以
1000000
    int64_t etf_buy_number;                                       // ETF 申购笔数
    int64_t etf_buy_amount;                                       // ETF 申购数量,实际值需除以 100
    int64_t etf_buy_money;                                        // ETF 申购金额,实际值需除以 100000
    int64_t etf_sell_number;                                      // ETF 赎回笔数
    int64_t etf_sell_amount;                                      // ETF 赎回数量,实际值需除以 100
    int64_t etf_sell_money;                                       // ETF 申购金额,实际值需除以 100000
    int64_t total_warrant_exec_volume;                            // 权证执行的总数量,实际值需除以 100
    int64_t war_lower_price;                                      // 债券质押式回购品种加权平均价，实际值需除
以 1000000
    int64_t war_upper_price;                                      // 权证涨停价格，实际值需除以 1000000
    int64_t withdraw_buy_number;                                  // 买入撤单笔数
    int64_t withdraw_buy_amount;                                  // 买入撤单数量,实际值需除以 100
    int64_t withdraw_buy_money;                                   // 买入撤单金额,实际值需除以 100000
    int64_t withdraw_sell_number;                                 // 卖出撤单笔数
    int64_t withdraw_sell_amount;                                 // 卖出撤单数量,实际值需除以 100
    int64_t withdraw_sell_money;                                  // 卖出撤单金额,实际值需除以 100000
    int64_t total_bid_number;                                     // 买入总笔数
    int64_t total_offer_number;                                   // 卖出总笔数
    int32_t bid_trade_max_duration;                               // 买入委托成交最大等待时间
```

```
    int32_t offer_trade_max_duration;                                   // 卖出委托成交最大等待时间
    int32_t num_bid_orders;                                             // 买方委托价位数
    int32_t num_offer_orders;                                           // 卖方委托价位数
    int64_t last_trade_time;                                            // 最近成交时间（为 YYYYMMDDHHMMSSsss 仅上海
mkdt00 文件，LDDS 生效）
    uint8_t  variety_category;                                        //  品种类别对应 VarietyCategory;
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDSnapshot(amd::ama::MDSnapshot* snapshot,uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
              ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(snapshot);
    }
```

2.5. OnMDOptionSnapshot 方法

接收期权快照数据回调。

函数原型：

void OnMDOptionSnapshot(MDOptionSnapshot* snapshot,uint32_t cnt);

参数：

参数	解释
snapshot(out)	期权快照数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDOptionSnapshot 结构体定义如下：

```
/**
 * @name 期权快照数据信息结构定义
 * @{ */
struct MDOptionSnapshot
```

```
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 期权代码
    int64_t orig_time;                                          // 时间（YYYYMMDDHHMMSSsss）
    int64_t pre_settle_price;                                   // 昨结算价（仅上海有效），实际值需除以 1000000
    int64_t pre_close_price;                                    // 昨收盘价（仅深交所有效），实际值需除以 1000000
    int64_t open_price;                                         // 今开盘价，实际值需除以 1000000
    int64_t auction_price;                                      // 动态参考价（波动性中断参考价），实际值需除以
1000000
    int64_t auction_volume;                                     // 虚拟匹配数量，实际值需除以 100
    int64_t high_price;                                         // 最高价，实际值需除以 1000000
    int64_t low_price;                                          // 最低价，实际值需除以 1000000
    int64_t last_price;                                         // 最新价，实际值需除以 1000000
    int64_t close_price;                                        // 收盘价 （仅上海有效），实际值需除以 1000000
    int64_t high_limited;                                       // 涨停价，实际值需除以 1000000
    int64_t low_limited;                                        // 跌停价，实际值需除以 1000000
    int64_t bid_price[5];                                       // 申买价，实际值需除以 1000000
    int64_t bid_volume[5];                                      // 申买量，实际值需除以 100
    int64_t offer_price[5];                                     // 申卖价，实际值需除以 1000000
    int64_t offer_volume[5];                                    // 申买量，实际值需除以 100
    int64_t settle_price;                                       // 今日结算价，实际值需除以 1000000
    int64_t total_long_position;                                // 总持仓量，实际值需除以 100
    int64_t total_volume_trade;                                 // 总成交数，实际值需除以 100
    int64_t total_value_trade;                                  // 总成交额，实际值需除以 100000
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];       // 产品实时阶段及标志
    //上海期权交易状态
    //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 0 位：‘S’表示启动（开市前）时段，‘C’表示集合竞价时段，‘T’表示连续交易时段，
    //‘B’表示休市时段，‘E’表示闭市时段，‘V’表示波动性中断，‘P’表示临时停牌，
    //‘U’表示收盘集合竞价 ‘M’表示可恢复交易的熔断（盘中集合竞价），‘N’表示不可恢复交易的熔断（暂停交易至闭市）
    //第 1 位：‘0’表示未连续停牌，‘1’表示连续停牌。（预留，暂填空格）
    //第 2 位：‘0’表示不限制开仓，‘1’表示限制备兑开仓，‘2’表示卖出开仓，‘3’表示限制卖出开仓、备兑开仓，‘4’表示限制
买入开仓，‘5’表示限制买入开仓、备兑开仓，‘6’表示限制买入开仓、卖出开仓，‘7’表示限制买入开仓、卖出开仓、备兑开仓
    //第 3 位：‘0’表示此产品在当前时段不接受进行新订单申报，‘1’ 表示此产品在当前时段可接受进行新订单申报。

    //深圳期权交易状态
    //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
    int32_t channel_no;                                         // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];                                    // 行情类别
    int64_t last_trade_time;                                    // 最近成交时间（为 YYYYMMDDHHMMSSsss 仅上海
mkdt03 文件，LDDS 生效）
    int64_t ref_price;                                          //参考价，实际值需除以 1000000
```

```
    uint8_t variety_category;                                   // 品种类别对应 VarietyCategory
    char    contract_type;                                      // 合约类别
    int32_t expire_date;                                        // 到期日
    char    underlying_security_code[ConstField::kSecurityCodeLen];  // 标的代码
    int64_t exercise_price;
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDOptionSnapshot(amd::ama::MDOptionSnapshot* snapshot,uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(snapshot);
    }
```

2.6. OnMDHKTSnapshot 方法

接收港股快照数据回调。

函数原型：

```
void OnMDHKTSnapshot(MDHKTSnapshot* snapshot, uint32_t cnt);
```

参数：

参数	解释
snapshot(out)	港股快照数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDHKTSnapshot 结构体定义如下：

```
/**
```

```
 * @name 港股通快照行情
 * @{ */
struct MDHKTSnapshot
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 港股代码
    int64_t orig_time;                                          // 时间（YYYYMMDDHHMMSSsss）
    int64_t pre_close_price;                                    // 昨收价，实际值需除以 1000000
    int64_t nominal_price;                                      // 按盘价，实际值需除以 1000000
    int64_t high_price;                                         // 最高价，实际值需除以 1000000
    int64_t low_price;                                          // 最低价，实际值需除以 1000000
    int64_t last_price;                                         // 最新价，实际值需除以 1000000
    int64_t bid_price[5];                                       // 申买价，实际值需除以 1000000
    int64_t bid_volume[5];                                      // 申买量，实际值需除以 100
    int64_t offer_price[5];                                     // 申卖价，实际值需除以 1000000
    int64_t offer_volume[5];                                    // 申卖量，实际值需除以 100
    int64_t total_volume_trade;                                 // 总成交数，实际值需除以 100
    int64_t total_value_trade;                                  // 总成交额，实际值需除以 100000
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];  // 产品实时阶段及标志
    //上海港股通交易状态
    //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 0 位：‘0’表示正常，‘1’表示暂停交易。

    //深圳港股通交易状态
    //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
    int32_t channel_no;                                         // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];        // 行情类别
    int64_t ref_price;                                          // 参考价格
    int64_t high_limited;                                       // 涨停价
    int64_t low_limited;                                        // 跌停价
    int64_t bid_price_limit_up;                                 // 买盘上限价，实际值需除以 1000000 （仅深圳有效）
    int64_t bid_price_limit_down;                               // 买盘下限价，实际值需除以 1000000 （仅深圳有效）
    int64_t offer_price_limit_up;                               // 卖盘上限价，实际值需除以 1000000 （仅深圳有效）
    int64_t offer_price_limit_down;                             // 卖盘下限价，实际值需除以 1000000 （仅深圳有效）
    uint8_t  variety_category;                                  // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
// 定义快照数据回调处理方法
void OnMDHKTSnapshot(amd::ama::MDHKTSnapshot* snapshot,uint32_t cnt) override
{
    for (uint32_t i = 0; i < cnt; ++i)
    {
        ///< TODO
    }
    // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
    // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
    amd::ama::IAMDApi::FreeMemory(snapshot);
}
```

2.7. OnMDIndexSnapshot 方法

接收指数快照数据回调。

函数原型：

```
void OnMDIndexSnapshot(MDIndexSnapshot* snapshot,uint32_t cnt);
```

参数：

参数	解释
snapshot(out)	指数快照数据首指针，需要显示的调用 FreeMemory
cnt(out)	指数快照个数

MDIndexSnapshot 结构体定义如下：

```
/**
 * @name 现货指数快照数据信息结构定义
 * @{ */
struct MDIndexSnapshot
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
```

```
    int64_t orig_time;                                                  // 时间（YYYYMMDDHHMMSSsss）
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];       // 产品实时阶段及标志（仅深圳有效）
    //深圳指数快照交易状态
    //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
    int64_t pre_close_index;                                            // 前收盘指数，实际值需除以 1000000
    int64_t open_index;                                                 // 今开盘指数，实际值需除以 1000000
    int64_t high_index;                                                 // 最高指数，实际值需除以 1000000
    int64_t low_index;                                                  // 最低指数，实际值需除以 1000000
    int64_t last_index;                                                 // 最新指数，实际值需除以 1000000
    int64_t close_index;                                                // 收盘指数（仅上海有效),实际值需除以 1000000
    int64_t total_volume_trade;                                         // 参与计算相应指数的交易数量，实际值需除以 100
    int64_t total_value_trade;                                          // 参与计算相应指数的成交金额，实际值需除以 100000
    int32_t channel_no;                                                 // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];                // 行情类别
    uint8_t  variety_category;                                          // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDIndexSnapshot (amd::ama::MDIndexSnapshot* snapshot,uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(snapshot);
    }
```

2.8. OnMDTickOrder 方法

接收逐笔委托数据回调。

函数原型：

void OnMDTickOrder(MDTickOrder* tick, uint32_t cnt);

参数：

参数	解释
tick（out）	逐笔委托数据首指针，需要显示的调用 FreeMemory
cnt（out）	数据个数

MDTickOrder 结构体定义如下：

```
/**
 * @name 现货逐笔委托数据信息结构定义
 * @{ */
struct MDTickOrder
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int32_t channel_no;                                     // 频道号
    int64_t appl_seq_num;                                   // 频道索引
    int64_t order_time;                                     // 时间（YYYYMMDDHHMMSSsss）
    int64_t order_price;                                    // 委托价格，实际值需除以 1000000
    int64_t order_volume;                                   // 深圳市场:委托数量，上海市场:剩余委托数量 ,实际值需除以 100
    uint8_t side;                                           // 买卖方向 深圳市场:(1-买 2-卖 G-借入 F-出借) 上海市场:(B:买单, S:卖单)
    uint8_t order_type;                                     // 订单类别  深圳市场:(1-市价 2-限价 U-本方最优) 上海市场:(A:增加委托，D:删除委托)
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];    // 行情类别
    int64_t orig_order_no;                                  // 原始订单号
    int64_t biz_index;                                      // 业务序号
    uint8_t variety_category;                               // 品种类别 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
// 定义快照数据回调处理方法
void OnMDTickOrder(MDTickOrder* ticks, uint32_t cnt) override
{
    for (uint32_t i = 0; i < cnt; ++i)
    {
        ///< TODO
    }
    // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
    // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
    amd::ama::IAMDApi::FreeMemory(ticks);
}
```

2.9. OnMDTickExection 方法

接收逐笔成交数据回调。

函数原型：

void OnMDTickExection(MDTickExecution* ticks, uint32_t cnt);

参数：

参数	解释
ticks（out）	逐笔委托数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDTickExecution 结构体定义如下：

```
/**
 * @name 现货逐笔成交数据信息结构定义
 * @{ */
struct MDTickExecution
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t exec_time;                                      // 时间（YYYYMMDDHHMMSSsss）
    int32_t channel_no;                                     // 频道号
```

```
    int64_t appl_seq_num;                                   // 频道编号
    int64_t exec_price;                                     // 委托价格，实际值需除以 1000000
    int64_t exec_volume;                                    // 委托数量，实际值需除以 100
    int64_t value_trade;                                    // 成交金额，实际值需除以 100000
    int64_t bid_appl_seq_num;                               // 买方委托索引
    int64_t offer_appl_seq_num;                             // 卖方委托索引
    uint8_t side;                                           // 买卖方向（仅上海有效 B-外盘，主动买 S-内盘，主动
卖 N-未知）
    uint8_t exec_type;                                      // 成交类型（仅深圳有效 4-撤销 F-成交）
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];    // 行情类别
    int64_t biz_index;                                      // 业务序号
    uint8_t  variety_category;                              // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDTickExecution(MDTickExecution* ticks, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
              ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(ticks);
    }
```

2.10. OnMDOrderQueue 方法

接收委托队列数据回调。

函数原型：

void OnMDOrderQueue(MDOrderQueue* orderqueues,uint32_t cnt);

参数：

参数	解释

orderqueues（out）	委托队列数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDOrderQueue 结构体定义如下：

```
/**
 * @name 现货委托队列数据信息结构定义
 * @{ */
struct MDOrderQueue
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    int64_t order_time;                                         // 委托时间（YYYYMMDDHHMMSSsss）
    uint8_t side;                                               // 买卖方向 (B-买 S-卖)
    int64_t order_price;                                        // 委托价格，实际值需除以 1000000
    int64_t order_volume;                                       // 订单数量，实际值需除以 100
    int32_t num_of_orders;                                      // 总委托笔数
    int32_t items;                                              // 明细个数
    int64_t volume[50];                                         // 订单明细
    int32_t channel_no;                                         //频道号
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];        // 行情类别
    uint8_t  variety_category;                                 // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDOrderQueue(MDOrderQueue* orderqueues, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(orderqueues);
    }
```

2.11. OnMDAfterHourFixedPriceSnapshot 方法

接收盘后定价交易快照数据回调。

函数原型：

```
void OnMDAfterHourFixedPriceSnapshot(
        MDAfterHourFixedPriceSnapshot* snapshots,
        uint32_t cnt);
```

参数：

参数	解释
snapshots(out)	盘后定价交易快照数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDAfterHourFixedPriceSnapshot 结构体定义如下：

```
/**
 * @name 盘后快照定义
 * @{ */
struct MDAfterHourFixedPriceSnapshot
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    int64_t orig_time;                                          // 时间（为 YYYYMMDDHHMMSSsss）
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];   // 交易阶段代码
    //上海盘后快照交易状态
    //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 0 位：‘I’表示启动（开市前）时段，  ‘A’表示集中撮合时段，‘H’表示连续交易时段，‘D’表示闭市时段，‘F’表示停牌

    //深圳盘后快照交易状态
    // 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
    int64_t close_price;                                    // 今日收盘价（仅上海有效），实际值需除以 1000000
    int64_t bid_price;                                      // 申买价 （仅深圳有效），实际值需除以 1000000
    int64_t bid_volume;                                     // 申买量，实际值需除以 100
    int64_t offer_price;                                    // 申卖价 （仅深圳有效），实际值需除以 1000000
    int64_t offer_volume;                                   // 申卖量，实际值需除以 100
```

```
    int64_t pre_close_price;                                    // 昨收价，实际值需除以 1000000
    int64_t num_trades;                                         // 成交笔数
    int64_t total_volume_trade;                                 // 成交总量，实际值需除以 100
    int64_t total_value_trade;                                  // 成交总金额，实际值需除以 100000
    int64_t total_bid_volume;                                   // 委托买入总量，实际值需除以 100
    int64_t total_offer_volume;                                 // 委托卖出总量，实际值需除以 100
    int32_t channel_no;                                         // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];        // 行情类别
    uint8_t  variety_category;                                 // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDAfterHourFixedPriceSnapshot(MDAfterHourFixedPriceSnapshot* snapshots, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(snapshots);
    }
```

2.12. OnMDAfterHourFixedPriceTickExecution 方法

接收盘后定价交易逐笔成交数据回调。

函数原型：

void OnMDAfterHourFixedPriceTickExecution(

MDAfterHourFixedPriceTickExecution* ticks,

uint32_t cnt);

参数：

参数	解释
ticks (out)	盘后定价交易逐笔成交数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDAfterHourFixedPriceTickExecution 结构体定义如下：

```
/**
 * @name 盘后逐笔成交定义
 * @{ */
struct MDAfterHourFixedPriceTickExecution
{
    int32_t market_type;
    int64_t appl_seq_num;                                   // 消息记录号
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t exec_time;                                      // 成交时间 （YYYYMMDDHHMMSSsss）
    int64_t exec_price;                                     // 委托价格，实际值需除以 1000000
    int64_t exec_volume;                                    // 委托数量，实际值需除以 100
    int64_t value_trade;                                    // 成交金额，实际值需除以 100000
    int64_t bid_appl_seq_num;                               // 买方委托索引
    int64_t offer_appl_seq_num;                             // 卖方委托索引
    uint8_t side;                                           // 买卖方向（仅上海有效 B-外盘，主动买 S-内盘，主
动卖 N-未知）
    uint8_t exec_type;                                      // 成交类型
    int32_t channel_no;                                     // 频道代码
    uint8_t  variety_category;                              // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDAfterHourFixedPriceTickExecution(MDAfterHourFixedPriceTickExecution* ticks, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
```

```
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(ticks);
    }
```

2.13. OnMDFutureSnapshot 方法

接收期货快照数据回调。

函数原型：

```
void OnMDFutureSnapshot(
        amd::ama::MDFutureSnapshot* snapshots, uint32_t cnt)
```

参数：

参数	解释
snapshots (out)	期货快照成交数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDFutureSnapshot 结构体定义如下：

```
/**
* @name MDFutureSnapshot 期货快照数据结构
* @{ */
struct MDFutureSnapshot
{
    int32_t market_type;                                        // 市场类型
    char security_code[ConstField::kSecurityCodeLen];           // 合约代码
    int32_t action_day;                                         // 业务日期
    int64_t orig_time;                          // 交易日 YYYYMMDDHHMMSSsss(TradingDay + UpdateTime + UpdateMillisec)
    char exchange_inst_id[ConstField::kExChangeInstIDLen];      // 合约在交易所的代码
    int64_t last_price;                                         // 最新价，实际值需除以 1000000
    int64_t pre_settle_price;                                   // 上次结算价，实际值需除以 1000000
    int64_t pre_close_price;                                    // 昨收价，实际值需除以 1000000
    int64_t pre_open_interest;                                  // 昨持仓量，实际值需除以 100
    int64_t open_price;                                         // 开盘价，实际值需除以 1000000
```

```
    int64_t high_price;                                          // 最高价，实际值需除以 1000000
    int64_t low_price;                                           // 最低价，实际值需除以 1000000
    int64_t total_volume_trade;                                  // 数量，实际值需除以 100
    int64_t total_value_trade;                                   // 成交金额，实际值需除以 100000
    int64_t open_interest;                                       // 持仓量，实际值需除以 100
    int64_t close_price;                                         // 今收盘，实际值需除以 1000000
    int64_t settle_price;                                        // 本次结算价，实际值需除以 1000000
    int64_t high_limited;                                        // 涨停板价，实际值需除以 1000000
    int64_t low_limited;                                         // 跌停板价，实际值需除以 1000000
    int64_t pre_delta;                                           // 昨虚实度，实际值需除以 1000000
    int64_t curr_delta;                                          // 今虚实度，实际值需除以 1000000
    int64_t bid_price[5];                                        // 申买价，实际值需除以 1000000
    int64_t bid_volume[5];                                       // 申买量，实际值需除以 100
    int64_t offer_price[5];                                      // 申卖价，实际值需除以 1000000
    int64_t offer_volume[5];                                     // 申卖量，实际值需除以 100
    int64_t average_price;                                       // 当日均价，实际值需除以 1000000
    int32_t trading_day;                                         // 交易日期
    uint8_t variety_category;                                    // 品种类别对应 VarietyCategory
    char exchange_inst_groupid[ConstField::kSecurityCodeLen];    // 结算组代码
    int64_t his_high_price;                                      // 历史最高价，实际值需除以 1000000
    int64_t his_low_price;                                       // 历史最低价，实际值需除以 1000000
    int64_t latest_volume_trade;                                 // 最新成交量，实际值需除以 100
    int64_t init_volume_trade;                                   // 初始持仓量，实际值需除以 100
    int64_t change_volume_trade;                                 // 持仓量变化，实际值需除以 100
    int64_t bid_imply_volume;                                    // 申买推导量，实际值需除以 100
    int64_t offer_imply_volume;                                  // 申卖推导量，实际值需除以 100
    char arbi_type;                                              // 策略类别
    char instrument_id_1[ConstField::kFutureSecurityCodeLen];    // 第一腿合约代码
    char instrument_id_2[ConstField::kFutureSecurityCodeLen];    // 第二腿合约代码
    char instrument_name[ConstField::kFutureSecurityCodeLen];    // 合约名称
    int64_t total_bid_volume_trade;                              // 总买入量，实际值需除以 100
    int64_t total_ask_volume_trade;                              // 总卖出量，实际值需除以 100
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDFutureSnapshot(MDFutureSnapshot* snapshots, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
```

```
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(snapshots);
    }
```

2.14. OnMDCSIIndexSnapshot 方法

接收中证指数快照数据回调。

函数原型：

```
void OnMDCSIIndexSnapshot(
        amd::ama::MDCSIIndexSnapshot* snapshots, uint32_t cnt)
```

参数：

参数	解释
snapshots (out)	中证指数快照成交数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDCSIIndexSnapshot 结构体定义如下：

```
/**
* @name MDCSIIndexSnapshot 中证指数行情信息
* @{ */
struct MDCSIIndexSnapshot
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t orig_time;                                      // 时间（YYYYMMDDHHMMSSsss）
    int64_t last_index;                                     // 最新指数，实际值需除以 1000000
    int64_t open_index;                                     // 今开盘指数，实际值需除以 1000000
    int64_t high_index;                                     // 最高指数，实际值需除以 1000000
    int64_t low_index;                                      // 最低指数，实际值需除以 1000000
    int64_t close_index;                                    // 收盘指数，实际值需除以 1000000
    int64_t pre_close_index;                                // 前收盘指数，实际值需除以 1000000
```

```
    int64_t change;                                                    // 涨跌，实际值需除以 1000000
    int64_t ratio_of_change;                                           // 涨跌幅，实际值需除以 1000000
    int64_t total_volume_trade;                                        // 成交量，实际值需除以 100
    int64_t total_value_trade;                                         // 成交金额，实际值需除以 100000
    int64_t exchange_rate;                                             // 汇率，实际值需除以 100000000
    char    currency_symbol;                                           // 币种标志（0-人民币 1-港币 2-美元 3-台币 4-日
元）
    int64_t close_index2;                                              // 当日收盘 2，实际值需除以 1000000
    int64_t close_index3;                                              // 当日收盘 3，实际值需除以 1000000
    uint8_t index_market;                                              // 指数市场
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];               // 行情类别  JLLX
    uint8_t  variety_category;                                        // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDCSIIndexSnapshot(MDCSIIndexSnapshot* snapshots, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(snapshots);
    }
```

2.15. OnMDIndicatorOfTradingVolumeSnapshot 方法

接收深交所成交量统计快照数据回调。

函数原型：

```
void OnMDIndicatorOfTradingVolumeSnapshot(
        MDIndicatorOfTradingVolumeSnapshot* snapshots, uint32_t cnt);
```

参数：

参数	解释
snapshots(out)	深交所成交量统计快照数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDIndicatorOfTradingVolumeSnapshot 结构体定义如下：

```
/**
* @name MDIndicatorOfTradingVolumeSnapshot 成交量统计指标快照行情（仅深交所）
* @{ */
struct MDIndicatorOfTradingVolumeSnapshot
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    int64_t orig_time;                                          // 时间（YYYYMMDDHHMMSSsss）
    int64_t total_volume_trade;                                 // 总成交数，实际值需除以 100
    int64_t total_value_trade;                                  // 总成交额，实际值需除以 100000
    int64_t pre_close_price;                                    // 昨收价，实际值需除以 1000000
    uint32_t stock_num;                                         // 统计量指标样本个数
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];  // 产品实时阶段及标志
    //深圳成交量统计指标快照交易状态
    //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
    int32_t channel_no;                                         // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];        // 行情类别
    uint8_t  variety_category;                                  // 品种类别对应 VarietyCategory
};
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDIndicatorOfTradingVolumeSnapshot(MDIndicatorOfTradingVolumeSnapshot* snapshots, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
```

```
        amd::ama::IAMDApi::FreeMemory(snapshots);
    }
```

2.16. OnMDCnIndexSnapshot 方法

接收深交所国证指数快照数据回调。

函数原型：

```
void OnMDCnIndexSnapshot (
        MDCnIndexSnapshot* snapshots, uint32_t cnt);
```

参数：

参数	解释
snapshots(out)	深交所国证指数快照数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDCnIndexSnapshot 结构体定义如下：

```
/**
* @name MDCnIndexSnapshot 国证指数快照行情 （仅深交所）
* @{ */
struct MDCnIndexSnapshot
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    int64_t orig_time;                                          // 时间（YYYYMMDDHHMMSSsss）
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];  // 产品实时阶段及标志
    //深圳成交量统计指标快照交易状态
    //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
```

```
    int64_t pre_close_index;                                    // 前收盘指数，实际值需除以 1000000
    int64_t open_index;                                         // 今开盘指数，实际值需除以 1000000
    int64_t high_index;                                         // 最高指数，实际值需除以 1000000
    int64_t low_index;                                          // 最低指数，实际值需除以 1000000
    int64_t last_index;                                         // 最新指数，实际值需除以 1000000
    int64_t close_index;                                        // 收盘指数，实际值需除以 1000000
    int64_t close_index2;                                       // 收盘指数 2，实际值需除以 1000000
    int64_t close_index3;                                       // 收盘指数 3，实际值需除以 1000000
    int64_t total_volume_trade;                                 // 参与计算相应指数的交易数量，实际值需除以 100
    int64_t total_value_trade;                                  // 参与计算相应指数的成交金额，实际值需除以 100000
    int32_t channel_no;                                         // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];        // 行情类别
    uint8_t  variety_category;                                  // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDCnIndexSnapshot(MDCnIndexSnapshot* snapshots, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(snapshots);
    }
```

2.17. OnMDRefinancingTickOrder 方法

接收深交所转融通逐笔委托数据回调。

函数原型：

```
void OnMDRefinancingTickOrder (
        MDRefinancingTickOrder* ticks, uint32_t cnt);
```

参数：

参数	解释
ticks (out)	深交所转融通逐笔委托数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDRefinancingTickOrder 结构体定义如下：

```
/**
* @name MDRefinancingTickOrder 转融通证券出借逐笔委托
* @{ */
struct MDRefinancingTickOrder
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int32_t channel_no;                                     // 频道编号
    int64_t appl_seq_num;                                   // 消息记录号
    int64_t order_time;                                     // 委托时间 （YYYYMMDDHHMMSSsss）
    int64_t order_price;                                    // 委托价格，实际值需除以 1000000
    int64_t order_volume;                                   // 委托数量，实际值需除以 100
    uint8_t side;                                           // 买卖方向 (1-买 2-卖 G-借入 F-出借)
    uint16_t expiration_days;                               // 期限
    uint8_t expiration_type;                                // 期限类型（1-固定期限）
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];    // 行情类别
    uint8_t  variety_category;                             // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDRefinancingTickOrder(MDRefinancingTickOrder* ticks, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
              ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(ticks);
```

```
    }
```

2.18. OnMDRefinancingTickExecution 方法

接收深交所转融通逐笔成交数据回调。

函数原型：

```
void OnMDRefinancingTickExecution (
        MDRefinancingTickExecution* ticks, uint32_t cnt);
```

参数：

参数	解释
ticks (out)	深交所转融通逐笔成交数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDRefinancingTickExecution 结构体定义如下：

```
/**
* @name MDRefinancingTickExecution 转融通证券出借逐笔成交
* @{ */
struct MDRefinancingTickExecution
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t exec_time;                                      // 成交时间 YYYYMMDDHHMMSSsss
    int32_t channel_no;                                     // 频道编号
    int64_t appl_seq_num;                                   // 消息记录号
    int64_t exec_price;                                     // 委托价格，实际值需除以 1000000
    int64_t exec_volume;                                    // 委托数量，实际值需除以 100
    int64_t value_trade;                                    // 成交金额，实际值需除以 100000
    int64_t bid_appl_seq_num;                               // 买方委托索引
```

```
    int64_t offer_appl_seq_num;                                    // 卖方委托索引
    uint8_t side;                                                  // 买卖方向
    uint8_t exec_type;                                             // 成交类型（仅深圳有效 4-撤销 F-成交）
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];           // 行情类别
    uint8_t  variety_category;                                     // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDRefinancingTickExecution (MDRefinancingTickExecution* ticks, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(ticks);
    }
```

2.19. OnMDNegotiableTickOrder 方法

接收深交所协议交易逐笔委托数据回调。

函数原型：

```
void OnMDNegotiableTickOrder(
        MDNegotiableTickOrder* ticks, uint32_t cnt);
```

参数：

参数	解释
ticks (out)	深交所协议交易逐笔委托数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNegotiableTickOrder 结构体定义如下：

```
/**
* @name MDNegotiableTickOrder 协议交易逐笔委托
* @{ */
struct MDNegotiableTickOrder
{
    int32_t market_type;
    char    security_code[ConstField::kSecurityCodeLen];                    // 证券代码
    int32_t channel_no;                                                     // 频道编号
    int64_t appl_seq_num;                                                   // 消息记录号
    int64_t order_time;                                                     // 委托时间 （YYYYMMDDHHMMSSsss）
    int64_t order_price;                                                    // 委托价格，实际值需除以 1000000
    int64_t order_volume;                                                   // 委托数量，实际值需除以 100
    uint8_t side;                                                           // 买卖方向 (1-买 2-卖 G-借入 F-出借)
    char    confirm_id[ConstField::kConfirmIdLen];                          // 定价行情约定号；为空表示是意向行情，否则为定
价行情
    char    contactor[ConstField::kContactorLen];                           // 联系人
    char    contact_info[ConstField::kContactInfoLen];                      // 联系方式
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];                    // 行情类别
    uint8_t  variety_category;                                              // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDNegotiableTickOrder(MDNegotiableTickOrder* ticks, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(ticks);
    }
```

2.20. OnMDNegotiableTickExecution 方法

接收深交所协议交易逐笔成交数据回调。

函数原型：

```
void OnMDNegotiableTickExecution(
        MDNegotiableTickExecution* ticks, uint32_t cnt);
```

参数：

参数	解释
ticks (out)	深交所协议交易逐笔成交数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNegotiableTickExecution 结构体定义如下：

```
/**
* @name MDNegotiableTickExecution 协议交易逐笔成交
* @{ */
struct MDNegotiableTickExecution
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t exec_time;                                      // 成交时间 YYYYMMDDHHMMSSsss
    int32_t channel_no;                                     // 频道编号
    int64_t appl_seq_num;                                   // 消息记录号
    int64_t exec_price;                                     // 委托价格，实际值需除以 1000000
    int64_t exec_volume;                                    // 委托数量，实际值需除以 100
    int64_t value_trade;                                    // 成交金额，实际值需除以 100000
    int64_t bid_appl_seq_num;                               // 买方委托索引
    int64_t offer_appl_seq_num;                             // 卖方委托索引
    uint8_t side;                                           // 买卖方向
    uint8_t exec_type;                                      // 成交类型（仅深圳有效 4-撤销 F-成交）
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];    // 行情类别
    uint8_t  variety_category;                             // 品种类别对应 VarietyCategory
```

```
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDNegotiableTickExecution(MDNegotiableTickExecution* ticks, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
              ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(ticks);
    }
```

2.21. OnMDHKTRealtimeLimit 方法

接收港股通实时额度数据回调。

函数原型：

```
void OnMDHKTRealtimeLimit(
        MDHKTRealtimeLimit* limits, uint32_t cnt);
```

参数：

参数	解释
limits(out)	港股通实时额度数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDHKTRealtimeLimit 结构体定义如下：

```
/**
* @name MDHKTRealtimeLimit 港股通实时额度
* @{ */
struct MDHKTRealtimeLimit
```

```
{
    int32_t market_type;                                    // 市场类型
    int64_t orig_time;                                      // 时间(YYYYMMDDHHMMSSsss)
    int64_t threshold_amount;                               // 每日初始额度，实际值需除以 100000
    int64_t pos_amt;                                        // 日中剩余额度，实际值需除以 100000
    char    amount_status;                                  // 额度状态 (1-额度用完或其他原因全市场禁止买入
2-额度可用)
    int32_t channel_no;                                     // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];    // 行情类别
    uint8_t  variety_category;                             // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDHKTRealtimeLimit(MDHKTRealtimeLimit* limits, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(limits);
    }
```

2.22. OnMDHKTProductStatus 方法

接收港股通产品状态数据回调。

函数原型：

```
void OnMDHKTProductStatus(
        MDHKTProductStatus* status, uint32_t cnt);
```

参数：

参数	解释
status (out)	港股通产品状态数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDHKTProductStatus 结构体定义如下：

```
/**
* @name MDHKTProductStatus 港股通可接收订单并转发的产品状态数据
* @{ */
struct MDHKTProductStatus
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    int64_t orig_time;                                          // 时间(YYYYMMDDHHMMSSsss)
    char    trading_status1[ConstField::kTradingStatusLen];     // 证券交易状态（整手订单）
    //港股通整手订单
    //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：‘0’表示限制买入，‘1’表示正常无此限制。
    //第 2 位：‘0’表示限制卖出，‘1’表示正常无此限制。
    char    trading_status2[ConstField::kTradingStatusLen];     // 证券交易状态（零股订单）
    //港股通零股订单
    //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：‘0’表示限制买入，‘1’表示正常无此限制。
    //第 2 位：‘0’表示限制卖出，‘1’表示正常无此限制。
    int32_t channel_no;                                         // 频道代码
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];        // 行情类别
    uint8_t  variety_category;                                 // 品种类别对应 VarietyCategory
};
/**  @} */
```

示例代码如下：

```
    // 定义快照数据回调处理方法
    void OnMDHKTProductStatus(MDHKTProductStatus* status, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
              ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
```

```
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(status);
    }
```

2.23. OnMDHKTVCM 方法

接收港股 VCM 数据回调。

函数原型：

```
void OnMDHKTVCM(MDHKTVCM* vcms, uint32_t cnt);
```

参数：

参数	解释
vcms(out)	vcm 数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDHKTVCM 结构体定义如下：

```
struct MDHKTVCM
{
    uint8_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 港股代码
    int64_t orig_time;                                      // 时间（YYYYMMDDHHMMSSsss）
    int64_t start_time;                                     // 市调机制开始时间
    int64_t end_time;                                       // 市调机制结束时间
    int64_t ref_price;                                      // 市调机制参考价格
    int64_t low_price;                                      // 市调机制最低价格
    int64_t high_price;                                     // 市调机制最高价格
    char    md_stream_id[ConstField::kMDStreamIDMaxLen];    // 行情类别
    uint8_t  variety_category;                              // 品种类别对应 VarietyCategory
};
```

示例代码如下：

```
// 定义快照数据回调处理方法
void OnMDHKTVCM(MDHKTVCM* vcms, uint32_t cnt) override
{
    for (uint32_t i = 0; i < cnt; ++i)
    {
        ///< TODO
    }
    // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
    // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
    amd::ama::IAMDApi::FreeMemory(vcms);
}
```

2.24. OnMDNEEQSnapshot 方法

接收股转系统证券行情信息回调。

函数原型：

```
void OnMDNEEQSnapshot(MDNEEQSnapshot* infos, uint32_t cnt);
```

参数：

参数	解释
infos(out)	股转系统证券行情信息数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNEEQSnapshot **结构体定义如下：**

```
/**
 * @name MDNEEQSnapshot 股转系统证券行情
 * @{ */
struct MDNEEQSnapshot
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t orig_time;                                      // 时间 CCYYMMDD + HHMMSS * 1000
    int64_t pre_close_price;                                // 昨收价，实际值需除以 1000000
    int64_t open_price;                                     // 开盘价，实际值需除以 1000000
    int64_t last_price;                                     // 最新价，实际值需除以 1000000
    int64_t total_volume_trade;                             // 成交总量，实际值需除以 100
```

```
    int64_t total_value_trade;                                  // 成交总金额，实际值需除以 100000
    int64_t num_trades;                                         // 成交笔数
    int64_t high_price;                                         // 最高价，实际值需除以 1000000
    int64_t low_price;                                          // 最低价，实际值需除以 1000000
    int64_t price_earning_ratio1;                               // 市盈率 1，实际值需除以 1000000
    int64_t price_earning_ratio2;                               // 市盈率 2，实际值需除以 1000000
    int64_t change1;                                            // 升跌 1（对比昨收价），实际值需除以 1000000
    int64_t change2;                                            // 升跌 2（对比上一笔），实际值需除以 1000000
    int64_t open_interest;                                      // 合约持仓量，实际值需除以 100
    int64_t bid_price[5];                                       // 申买价，实际值需除以 1000000
    int64_t bid_volume[5];                                      // 申买量，实际值需除以 100
    int64_t offer_price[5];                                     // 申卖价，实际值需除以 1000000
    int64_t offer_volume[5];                                    // 申卖量，实际值需除以 100
    int64_t index_factor;                                       // 指数因子，实际值需除以 1000000
    char    trading_phase_code[ConstField::kTradingPhaseCodeLen];  // 交易阶段代码
    //北交所证券行情状态
    //个位数存放收市行情标志（0：非收市行情；1：收市行情；2：盘后行情）
    //十位数存放正式行情与测试行情标志（0：正式行情；1：测试行情）
    uint8_t variety_category;                                   // 品种类别
};
/**  @} */
```

示例代码如下：

```
    // 定义股转系统证券信息数据回调处理方法
    void OnMDNEEQSnapshot(MDNEEQSnapshot* infos, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(vcms);
    }
```

2.25. OnMDNEEQSecurityInfo 方法

接收股转系统证券信息回调。

函数原型：

```
void OnMDNEEQSecurityInfo(MDNEEQSecurityInfo* infos, uint32_t cnt);
```

参数：

参数	解释
infos(out)	股转系统证券信息数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNEEQSecurityInfo 结构体定义如下：

```
/**
* @name MDNEEQSecurityInfo 北交所证券信息
* @{ */
struct MDNEEQSecurityInfo
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t orig_time;                                      // 时间 CCYYMMDD + HHMMSSss * 10
    char    security_abbreviation[ConstField::kSecurityNameLen];  // 证券简称
    char    english_abbreviation[ConstField::kSecurityNameLen];   // 英文简称
    char    underlying_security[ConstField::kSecurityCodeLen];    // 基础证券
    char    ISIN[ConstField::kSecurityCodeLen];             // ISIN 编码
    int32_t trading_unit;                                   // 交易单位
    char    industry_type[ConstField::kTypesLen];           // 行业种类
    char    currency[ConstField::kTypesLen];                // 货币种类 (00-人民币，02-美元)
    int64_t par_value;                                      // 每股面值，实际值需除以 1000000
    int64_t general_capital;                                // 总股本
    int64_t unrestricted_capital;                           // 非限售股本
    int64_t last_year_earning;                              // 上年每股收益，实际值需除以 1000000
    int64_t cur_year_earning;                               // 本年每股收益，实际值需除以 1000000
    int64_t brokerage_rate;                                 // 经手费率，实际值需除以 1000000
    int64_t stamp_duty_rate;                                // 印花税率，实际值需除以 1000000
    int64_t transfer_fee_rate;                              // 过户费率，实际值需除以 1000000
    char    listing_date[ConstField::kDateLen];             // 挂牌日期
    char    value_date[ConstField::kDateLen];               // 起息日
    char    expiring_date[ConstField::kDateLen];            // 到期日
    int64_t every_limited;                                  // 每笔限量，实际值需除以 100
    int32_t buy_amount_unit;                                // 买数量单位，实际值需除以 100
    int32_t sell_amount_unit;                               // 卖数量单位，实际值需除以 100
    int64_t mini_dec_amount;                                // 最小申报数量，实际值需除以 100
    int32_t price_level;                                    // 价格档位，实际值需除以 1000000
    int64_t first_trade_limit;                              // 首笔交易限价参数，实际值需除以 1000000
```

```
    int64_t follow_trade_limit;                                    // 后续交易限价参数，实际值需除以 1000000
    uint8_t limit_param_nature;                                    // 限价参数性质
    int64_t high_limited;                                          // 涨停价，实际值需除以 1000000
    int64_t low_limited;                                           // 跌停价，实际值需除以 1000000
    int64_t block_trade_ceiling;                                   // 大宗交易价格上限，实际值需除以 1000000
    int64_t block_trade_floor;                                     // 大宗交易价格下限，实际值需除以 1000000
    char    component_mark;                                        // 成分股标志
    int32_t conver_ratio;                                          // 折合比例，实际值需除以 1000000
    char    trade_status;                                          // 交易状态
    char    security_level;                                        // 证券级别
    char    trade_type;                                            // 交易类型
    int64_t market_maker_num;                                      // 做市商数量，实际值需除以 100
    char    suspen_sign;                                           // 停牌标志
    char    ex_sign;                                               // 除权除息标志
    char    net_vote_sign;                                         // 网络投票标志
    char    other_buss_sign[ConstField::kTypesLen];                // 其他业务标志
    char    record_update_time[ConstField::kTimeLen];              // 记录更新时间
    uint8_t  variety_category;                                    // 品种类别对应 VarietyCategory
};
```

示例代码如下：

```
    // 定义股转系统证券信息数据回调处理方法
    void OnMDNEEQSecurityInfo(MDNEEQSecurityInfo* infos, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
              ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(vcms);
    }
```

2.26. OnMDNEEQNonPublicTransDeclaredInfo 方法

接收股转系统非公开申报转让信息数据回调。

函数原型：

```
void OnMDNEEQNonPublicTransDeclaredInfo(
        MDNEEQNonPublicTransDeclaredInfo* infos,
        uint32_t cnt);
```

参数：

参数	解释
infos(out)	股转系统非公开申报转让信息数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNEEQNonPublicTransDeclaredInfo 结构体定义如下：

```
/**
 * @name MDNEEQNonPublicTransDeclaredInfo 北交所非公开转让申报信息库
 * @{ */
struct MDNEEQNonPublicTransDeclaredInfo
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t orig_time;                                      // 时间 CCYYMMDD + HHMMSS * 1000
    char    transaction_unit[ConstField::kTypesLen];        // 交易单元
    char    security_category[ConstField::kTypesLen];       // 证券类别
    char    declare_category[ConstField::kTypesLen];        // 申报类别
    int64_t declare_volume;                                 // 申报数量，实际值需除以 100
    int64_t declare_price;                                  // 申报价格，实际值需除以 1000000
    int32_t deal_agreement_num;                             // 成交约定号
    char    declare_time[ConstField::kTimeLen];             // 申报时间
    char    record_status;                                  // 记录状态
    char    backup_sign;                                    // 备用标志
    uint8_t  variety_category;                             // 品种类别对应 VarietyCategory
};
```

示例代码如下：

```
// 定义股转系统非公开申报转让信息数据回调处理方法
void OnMDNEEQNonPublicTransDeclaredInfo(MDNEEQNonPublicTransDeclaredInfo* infos, uint32_t cnt) override
{
    for (uint32_t i = 0; i < cnt; ++i)
    {
        ///< TODO
    }
    // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
    // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
    amd::ama::IAMDApi::FreeMemory(vcms);
}
```

2.27. OnMDNEEQHierarchicalInfo 方法

接收股转系统分层信息数据回调。

函数原型：

```
void OnMDNEEQHierarchicalInfo(MDNEEQHierarchicalInfo* infos, uint32_t cnt);
```

参数：

参数	解释
infos(out)	股转系统分层信息数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNEEQHierarchicalInfo 结构体定义如下：

```
/**
 * @name MDNEEQHierarchicalInfo 北交所分层信息库
 * @{ */
struct MDNEEQHierarchicalInfo
{
    int32_t market_type;                                        // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    char    trade_date[ConstField::kDateLen];                   // 交易日期 CCYYMMDD
    char    security_abbreviation[ConstField::kSecurityNameLen];// 证券简称
    char    layered_sign;                                       // 分层标志
    char    layered_effective_date[ConstField::kDateLen];       // 分层生效日期 CCYYMMDD
    char    backup_sign;                                        // 备用标志
    uint8_t  variety_category;                                  // 品种类别对应 VarietyCategory
```

```
};
```

示例代码如下：

```
// 定义股转系统分层信息数据回调处理方法
void OnMDNEEQHierarchicalInfo(MDNEEQHierarchicalInfo* infos, uint32_t cnt) override
{
    for (uint32_t i = 0; i < cnt; ++i)
    {
        ///< TODO
    }
    // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
    // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
    amd::ama::IAMDApi::FreeMemory(vcms);
}
```

2.28. OnMDHKMarketStatus 方法

接收港股市场状态数据回调。

函数原型：

```
void OnMDHKMarketStatus (MDHKMarketStatus* status, uint32_t cnt);
```

参数：

参数	解释
status(out)	港股市场状态数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDHKMarketStatus 结构体定义如下：

```
/**  @} */


/**
* @name MDHKMarketStatus 港股市场状态
* @{
* */
struct MDHKMarketStatus
{
```

```
    int32_t market_type;                                                    // 市场类型
    int64_t orig_time;                                                      // 时间(YYYYMMDDHHMMSSsss)
    char    trading_session_sub_id[ConstField::kTradingStatusLen];          // 市场状态
    uint8_t variety_category;                                               // 品种类别
};
```

示例代码如下：

```
    // 定义港股市场状态数据回调处理方法
    void OnMDHKMarketStatus (MDHKMarketStatus * infos, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(vcms);
    }
```

2.29. OnMDNEEQNegotiableDeclaredInfo 方法

接收股转协议转让申报信息库数据回调。

函数原型：

void OnMDNEEQNegotiableDeclaredInfo (MDNEEQNegotiableDeclaredInfo* infos, uint32_t cnt);

参数：

参数	解释
infos(out)	股转协议转让申报信息库数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNEEQNegotiableDeclaredInfo 结构体定义如下：

```
/**
```

```
 * @name MDNEEQNegotiableDeclaredInfo 北交所协议转让申报信息库
 * @{ */
struct MDNEEQNegotiableDeclaredInfo
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t orig_time;                                      // 时间 CCYYMMDD + HHMMSS * 1000
    char    transaction_unit[ConstField::kTypesLen];        // 交易单元
    char    md_stream_id[ConstField::kMDStreamIDLen];       // 业务类别
    int64_t declare_volume;                                 // 申报数量
    int64_t declare_price;                                  // 申报价格
    int32_t deal_agreement_num;                             // 成交约定号
    char    declare_time[ConstField::kTimeLen];             // 申报时间
    char    record_status;                                  // 记录状态
    char    backup_sign;                                    // 备用标志
    uint8_t variety_category;                               // 品种类别
};
```

示例代码如下：

```
    // 定义股转协议转让申报信息库数据回调处理方法
    void OnMDNEEQNegotiableDeclaredInfo(MDNEEQNegotiableDeclaredInfo* infos, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
              ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(vcms);
    }
```

2.30. OnMDNEEQMarketMakerDeclaredInfo 方法

接收股转业务申报信息库数据回调。

函数原型：

void OnMDNEEQMarketMakerDeclaredInfo (MDNEEQMarketMakerDeclaredInfo* infos, uint32_t cnt);

参数：

参数	解释
infos(out)	股转业务申报信息库数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNEEQMarketMakerDeclaredInfo 结构体定义如下：

```
/**
 * @name MDNEEQMarketMakerDeclaredInfo 北交所做市业务申报信息库
 * @{ */
struct MDNEEQMarketMakerDeclaredInfo
{
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t orig_time;                                      // 时间 CCYYMMDD + HHMMSS * 1000
    char    md_stream_id[ConstField::kMDStreamIDLen];       // 业务类别
    int64_t declare_volume;                                 // 申报数量
    int64_t declare_price;                                  // 申报价格
    char    data_type;                                      // 数据类型
    char    declare_time[ConstField::kTimeLen];             // 申报时间
    int64_t backup_field;                                   // 备用字段（预留）
    uint8_t variety_category;                               // 品种类别
};
```

示例代码如下：

```
    // 定义股转做市业务申报信息库数据回调处理方法
    void OnMDNEEQMarketMakerDeclaredInfo(MDNEEQMarketMakerDeclaredInfo* infos, uint32_t cnt) override
    {
        for (uint32_t i = 0; i < cnt; ++i)
        {
            ///< TODO
        }
        // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
        // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
        amd::ama::IAMDApi::FreeMemory(vcms);
    }
```

2.31. OnMDNEEQNonPublicTransferDealInfo 方法

接收股转非公开转让成交信息库数据回调。

函数原型：

void OnMDNEEQNonPublicTransferDealInfo (MDNEEQNonPublicTransferDealInfo* infos, uint32_t cnt);

参数：

参数	解释
infos(out)	股转非公开转让成交信息库数据首指针，需要显示的调用 FreeMemory
cnt(out)	数据个数

MDNEEQNonPublicTransferDealInfo 结构体定义如下：

```
/**
 * @name MDNEEQNonPublicTransferDealInfo 北交所非公开转让成交信息库
 * @{ */
struct MDNEEQNonPublicTransferDealInfo
{
    int32_t market_type;                                        // 市场类型
    int64_t serial_num;                                         // 序号
    char    security_code[ConstField::kSecurityCodeLen];        // 证券代码
    int64_t orig_time;                                          // 时间 CCYYMMDD + HHMMSS * 1000
    char    security_abbreviation[ConstField::kSecurityAbbreviationLen];  // 证券简称
    char    security_category[ConstField::kTypesLen];           // 证券类别
    char    bid_transaction_unit[ConstField::kTypesLen];        // 买入交易单元（预留）
    char    bid_transaction_unit_name[ConstField::kUnitName];   // 买入营业部名称/交易单元名称
    char    offer_transaction_unit[ConstField::kTypesLen];      // 卖出交易单元（预留）
    char    offer_transaction_unit_name[ConstField::kUnitName]; // 卖出营业部名称/交易单元名称
    int64_t deal_volume;                                        // 成交数量
    int64_t deal_price;                                         // 成交价格
    char    deal_time[ConstField::kTimeLen];                    // 成交时间
    char    backup_sign;                                        // 备用标志
    uint8_t variety_category;                                   // 品种类别
};
```

示例代码如下：

```
// 定义股转非公开转让成交信息库数据回调处理方法
void OnMDNEEQNonPublicTransferDealInfo  (MDNEEQNonPublicTransferDealInfo* infos, uint32_t cnt) override
{
    for (uint32_t i = 0; i < cnt; ++i)
    {
        ///< TODO
    }
    // 应用可将数据指针复制到回调函数之外，数据在释放之前一直可用
    // 应用必须在适当的地方释放内存，否则可能造成数据阻塞
    amd::ama::IAMDApi::FreeMemory(vcms);
}
```

2.32. OnMDOrderBook 方法

委托薄数据回调。

函数原型：

```
void OnMDOrderBook(std::vector<amd::ama::MDOrderBook>& order_book);
```

参数：

参数	解释
order_book(out)	委托薄数据结构

MDOrderBook 结构体定义如下：

```
/**
* @name MDOrderBook 委托薄
* @{ */
struct MDOrderBook
{
    int32_t channel_no;                                     // 频道号
    int32_t market_type;                                    // 市场类型
    char    security_code[ConstField::kSecurityCodeLen];    // 证券代码
    int64_t last_tick_time;                                 // 最新逐笔生成时间
    int64_t last_snapshot_time;                             // 最新快照生成时间
    int64_t last_tick_seq;                                  // 最新逐笔序列号
    std::vector<MDOrderBookItem> bid_order_book;            // 买委托薄
    std::vector<MDOrderBookItem> offer_order_book;          // 卖委托薄
};
```

```
/**  @} */
```

示例代码如下：

```
virtual void OnMDOrderBook(std::vector<amd::ama::MDOrderBook>& order_book)
{
      for (const auto& book: order_book)
      {
            ///< TODO
      }
      //不需要手动释放内存
    }
```

3. 线程模型

接口方法集	线程安全
GetVersion Init Join Release FreeMemory SubscribeData	接口集线程安全
OnLog	单独线程安全
OnIndicator	单独线程安全
OnEvent	单独线程安全
OnMDSnapshot OnMDOptionSnapshot OnMDHKTSnapshot OnMDIndexSnapshot OnMDTickOrder OnMDTickExecution OnMDOrderQueue	受 cfg.is_thread_safe 参影响具体请参考 is_thread_safe 参数说明

OnMDAfterHourFixedPriceSnapshot OnMDAfterHourFixedPriceTickExecution OnMDFutureSnapshot OnMDCSIIndexSnapshot OnMDIndicatorOfTradingVolumeSnapshot OnMDCnIndexSnapshot OnMDRefinancingTickOrder OnMDRefinancingTickExecution OnMDNegotiableTickOrder OnMDNegotiableTickExecution OnMDHKTRealtimeLimit OnMDHKTProductStatus OnMDNEEQSnapshot OnMDHKTVCM OnMDNEEQSecurityInfo OnMDNEEQNonPublicTransDeclaredInfo OnMDNEEQHierarchicalInfo OnMDHKMarketStatus OnMDNEEQNegotiableDeclaredInfo OnMDNEEQMarketMakerDeclaredInfo OnMDNEEQNonPublicTransferDealInfo	
OnMDOrderBook	不受 is_thread_safe 参数影响，线程非安全，深圳市场和上海市场分别由不同线程回调