中国银河证券格物金融服务平台
金融数据功能开发手册
（C++版）

目录

目录 1
1. 版本说明 1
1.1 文档管理信息表 1
1.2 文档变更记录表 1
2. 功能介绍 2
2.1 术语和缩略语 2
2.2 金融数据服务 2
2.3 两种使用场景 2
2.3.1 托管机房模式使用场景 2
2.3.2 互联网模式使用场景 2
2.4 数据内容 2
3. C++ 开发指南 4
3.1 SDK 4
3.2 运行环境 5
3.2.1 操作系统及编译器版本 5
3.2.2 Linux 推荐运行环境配置 6
3.2.3 Windows 推荐运行环境配置 6
3.3 运行模式说明 6
3.3.1 托管机房模式 6
3.3.1.1 TCP 通道 6
3.3.1.2 QTCP 通道 7
3.3.1.3 RTCP 通道 7
3.3.1.4 TCP、QTCP、RTCP 通道同时使用的通道设置 7
3.3.2 互联网模式 7
3.4 API 开发步骤 8
3.4.1 初始化 TGW 9
3.4.2 异常释放资源处理 10
3.4.3 数据订阅 10
3.4.4 取消订阅 10
3.4.5 获取数据 10
3.5 API 接口详细 11
3.5.1 参数说明 11
3.5.2 基础接口 12
1) GetVersion 方法 12
2) Init 方法 12
3) Release 方法 13
4) FreeMemory 方法 13
5) GetTaskID 方法 13
3.5.3 订阅数据方法 13
6) Subscribe 方法 14
7) UnSubscribe 方法 14

8) SubFactor 方法 14
9) UnSubFactor 方法 15
3.5.4 订阅回调方法 15
1) OnLog 方法 15
2) OnEvent 方法 16
3) OnIndicator 方法 16
4) OnLogon 方法 16
5) OnMDSnapshot 方法 16
6) OnMDIndexSnapshot 方法 17
7) OnMDOptionSnapshot 方法 17
8) OnMDHKTSnapshot 方法 17
9) OnMDAfterHourFixedPriceSnapshot 方法 17
10) OnMDCSIIndexSnapshot 方法 18
11) OnMDCnIndexSnapshot 方法 18
12) OnMDHKTRealtimeLimit 方法 18
13) OnMDHKTProductStatus 方法 18
14) OnMDHKTVCM 方法 18
15) OnMDFutureSnapshot 方法 19
16) OnKLine 方法 19
17) OnSnapshotDerive 方法 19
18) OnFactor 方法 19
3.5.5 查询数据方法 20
3.5.6 查询回调方法 25
1) IGMDKlineSpi 接口 25
2) IGMDSnapshotSpi 接口 25
3) IGMDOrderQueueSpi 接口 26
4) IGMDTickExecutionSpi 接口 27
5) IGMDTickOrderSpi 接口 27
6) IGMDCodeTableSpi 接口 27
7) IGMDSecuritiesInfoSpi 接口 28
8) IGMDExFactorSpi 接口 28
9) IGMDFactorSpi 接口 28
10) IGMDThirdInfoSpi 接口 29
3.5.7 回放数据方法 29
3.5.8 回放回调方法 31
1) OnMDKline 方法 31
2) OnMDSnapshot 方法 31
3) OnMDTickExecution 方法 32
4) OnRspTaskStatus 方法 32
3.6 开发 Demo 示例 32
3.7 编译运行步骤 38
3.8 开发注意事项 38
3.9 演示示例（获取和保存数据） 40
3.9.1 修改参数 40

3.9.2 运行启动脚本 41
3.9.3 开启订阅命令输入终端 41
3.9.4 查看订阅数据 csv 文件 42
3.9.5 tgw_test 程序订阅命令说明 42
4. 公共数据字典 45
4.1 事件级别(EventLevel) 45
4.2 错误码(ErrorCode) 45
4.3 数据定义长度(ConstField) 46
4.4 回放请求任务状态(HistoryTaskStatus) 47
4.5 数据类型(MDDatatype) 47
4.6 日志输出级别(LogLevel) 47
4.7 市场类型定义(MarketType) 48
4.8 API 通道模式(ApiMode) 48
4.9 托管机房模式通道(ColocatChannelMode) 48
4.10 订阅数据类型(SubscribeDataType) 48
4.11 品种类型定义(VarietyCategory) 49
5. 行情数据字典 51
5.1 K 线(MDKLine) 51
5.2 现货衍生(MDSnapshotDerive) 51
5.3 加工因子(Factor) 52
5.4 L1 现货快照(MDSnapshotL1) 52
5.5 指数快照(MDIndexSnapshot) 53
5.6 期权快照(MDOptionSnapshot) 53
5.7 港股通快照(MDHKTSnapshot) 54
5.8 期货快照(MDFutureSnapshot) 55
5.9 盘后定价交易快照(MDAfterHourFixedPriceSnapshot) 56
5.10 中证指数行情(MDCSIIndexSnapshot) 57
5.11 国证指数快照(MDCnIndexSnapshot) 58
5.12 港股通实时额度(MDHKTRealtimeLimit) 58
5.13 港股通产品状态(MDHKTProductStatus) 59
5.14 港股 VCM(MDHKTVCM) 59
5.15 L2 现货快照(MDSnapshotL2) 59
5.16 现货逐笔成交(MDTickExecution) 60
5.17 逐笔委托(MDTickOrder) 61
5.18 委托队列(MDOrderQueue) 61
5.19 代码表(MDCodeTable) 62
5.20 复权因子表(MDExFactorTable) 62
5.21 个股基础信息(MDStockInfo) 62
5.22 代码表(MDCodeTableRecord) 63
5.23 金融资讯数据(ThirdInfoData) 65
6. 补充字段取值说明 66
6.1 证券代码表 security_type 66
6.2 证券代码表的 currency 取值 67
6.3 交易阶段代码取值 68

6.4 Security_status 68
6.5 K 线算法说明 69

1. 版本说明

1.1 文档管理信息表

主题	中国银河证券格物金融服务平台金融数据功能开发手册（Python 版）
文档版本	V1.0.0
C++ SDK 版本	V1.0.1
创建时间	2023 年 1 月 28 日
创建人	中国银河证券信息技术部量化交易服务研发团队
最新发布日期	2023 年 3 月 10 日

1.2 文档变更记录表

修改人	修改时间	修改内容
张德高	2023 年 3 月 10 日	文档整体结构调整和细节表述优化

2. 功能介绍

本文档是 tgw 的 SDK 开发指南，包含了对 API 接口的说明以及示例，用于指引开发人员通过 tgw 金融数据功能接口进行数据接收和查询的开发。

2.1 术语和缩略语

术语、缩写	解释
tgw	tgw，格物机构金融服务平台
SDK	Software Development Kit，金融数据功能软件开发工具包

2.2 金融数据服务

金融数据功能，是指用户使用 C++、Python 以及其他本功能可支持的程序设计语言或用户端页面，获取公司通过对证券交易所等渠道的公开信息加工而成的行情数据、金融资讯数据等金融数据的功能。

2.3 两种使用场景

tgw 包含了托管机房模式和互联网模式两种使用场景。

2.3.1 托管机房模式使用场景

在银河证券机房内部网络环境内，用户除 Level-1、K 线、资讯数据、因子数据等中低频数据外，通过 SDK 提供获取需要增强 Level-2 等高频数据。

2.3.2 互联网模式使用场景

在互联网网络环境内下，用户通过 SDK 提供获取 Level-1、K 线、资讯数据、因子数据等中低频数据。

2.4 数据内容

托管机房模式和互联网模式都能提供的数据包括：

- 实时推送 K 线数据（1/3/5/10/15/30/60/120 分钟线）
- K 线历史数据（日/周/月/季年线、1/3/5/10/15/30/60/120 分钟线）
- 实时推送快照衍生指标数据
- 实时推送加工因子数据

- 证券个股基本信息查询
- 证券代码表信息查询
- 复权因子表信息查询
- 加工因子历史数据
- 金融资讯数据查询

托管机房模式下的数据包括：

- Level2 现货快照历史数据
- 指数快照历史数据
- Level2 历史委托队列数据
- Level2 历史逐笔委托数据
- Level2 历史逐笔成交数据

互联网模式下的数据包括：

- 实时推送 Level1 现货快照数据
- 实时推送指数快照数据
- 实时推送期权快照数据
- 实时推送期货快照数据
- 实时推送港股通快照数据
- 实时推送盘后定价交易快照数据
- 实时推送中证指数快照数据
- 实时推送深交所国证指数快照数据
- 实时推送港股通实时额度数据
- 实时推送港股通产品状态快照数据
- 港股市场波动调节机制(VCM)推送数据
- Level1 快照历史数据

3. C++ 开发指南

金融数据功能提供 Windows 和 linux 两种操作系统下 C++和 Python 各两种 SDK 版本供投资者开发。

请参考以下指南进行 c++开发使用。

3.1 SDK

各操作系统版本 SDK 下载解压即可使用，目录结构如下：

Linux 系统下 **C++版本 SDK 目录**

```
.\c++
├──demo   (样例程序目录)
│        make.sh              (编译样例程序脚本)
│        Makefile             (样例程序 tgw_demo 编译的 Makefile 文件)
│        tgw_demo.cpp         (样例程序)
│        tgw_push_spi.cpp     (实时推送 spi 及回调接口实现)
│        tgw_push_spi.h       (实时推送 spi 及回调接口实现)
│        tgw_query_spi.cpp    (查询 spi 及回调接口实现)
│        tgw_query_spi.h      (查询 spi 及回调接口实现)
│        tgw_replay_spi.cpp   (回放 spi 及回调接口实现)
│        tgw_replay_spi.h     (回放 spi 及回调接口实现)
├──include          (头文件目录。使用时可直接 #include "tgw.h")
│        tgw.h                (API 推送和查询接口头文件)
│        tgw_datatype.h       (API 使用的类型定义)
│        tgw_export.h
│        tgw_history_spi.h    (API 回放接口头文件)
│        tgw_struct.h         (API 使用的数据结构定义)
├──test_tool
│        bin/tgw_test         (已编译好演示程序，无代码，使用时可直接运
│                              行./run.sh，运行前需更改配置文件)
│        etc/tgw.json         （tgw_test 演示程序配置文件目录）
│        run.sh               （tgw_test 演示程序运行脚本）
└──lib              (依赖库文件目录)
```

Windows 系统下 **C++版本 SDK 目录**

```
.\c++
├──demo   (样例程序目录)
│        tgw_demo.cpp         (样例程序)
│        tgw_push_spi.cpp     (实时推送 spi 及回调接口实现)
│        tgw_push_spi.h       (实时推送 spi 及回调接口实现)
│        tgw_query_spi.cpp    (查询 spi 及回调接口实现)
```

```
│       tgw_query_spi.h     (查询 spi 及回调接口实现)
│       tgw_replay_spi.cpp(回放 spi 及回调接口实现)
│       tgw_replay_spi.h    (回放 spi 及回调接口实现)
│       └─project
│               tgw_demo.sln          (VC++2017 解决方案文件)
│               └─tgw_demo
│                       tgw_demo.vcxproj   (VC++2017 工程文件)
│                       tgw_demo.vcxproj.filters (VC++2017 工程文件)
│                       tgw_demo.vcxproj.user (VC++2017 工程文件)
├──include          (头文件目录。使用时可直接 #include "tgw.h")
│       tgw.h                  (API 推送和查询接口头文件)
│       tgw_datatype.h         (API 使用的类型定义)
│       tgw_export.h
│       tgw_history_spi.h    (API 回放接口头文件)
│       tgw_struct.h          (API 使用的数据结构定义)
├──test_tool
│       bin/tgw_test.exe      (已编译好演示程序，无代码，使用时可直接运
│                                 行./run.sh，运行前需更改配置文件)
│       etc/tgw.json          （tgw_test 演示程序配置文件目录）
│       run.bat               （tgw_test 演示程序运行脚本）
└──lib                  (依赖库文件目录)
```

使用前必须有中国银河证券股份有限公司授权的登录账户密码以及服务 IP 地址和端口。建议使用 Telnet 命令验证网络连通性。

3.2 运行环境

3.2.1 操作系统及编译器版本

支持系统版本	编译器	备注
RedHat7.2 RedHat7.4 RedHat7.6	gcc4.8.5	支持多种 Level 1 或 Level 2 及自定义因子数据的订阅推送，支持快照，k 线，因子等数据查询，代码表查询，金融资讯数据查询
Windows10（64 位）	VC++ 2017	支持多种 Level 1 或 Level 2 及自定义因子数据的订阅推送，支持快照，k 线，因子等数据查询，代码表查询，金融资讯数据查询

3.2.2 Linux 推荐运行环境配置

类型	最低配置	推荐配置
处理器	2.10GHz,4 核	2.10GHz,8 核
内存	DDR4 4GB	DDR4 4GB
硬盘	200G 机械硬盘/SSD	480G 机械硬盘/SSD
网卡	普通网卡	普通万兆网卡
操作系统	REDHAT 7.2/7.4/7.6	REDHAT 7.2/7.4/7.6

3.2.3 Windows 推荐运行环境配置

类型	最低配置	推荐配置
处理器	2.60GHz，4 核	2.60GHz，8 核
内存	DDR4 4GB	DDR4 4GB
硬盘	200G 机械硬盘/SSD	480G 机械硬盘/SSD
网卡	普通网卡	普通万兆网卡
操作系统	Windows 10(64 位)	Windows 10(64 位)

3.3 运行模式说明

tgw 包含托管机房模式和互联网模式。

通过在初始化启动时设置 IGMDApi::Init()接口中 api_mode 区分。

3.3.1 托管机房模式

三种通道 TCP、QTCP、RTCP，可单独使用，可两两结合，也可三个同时使用。

参数设置：

（1） 通道模式 api_mode 设置

tgw.ApiMode.kColocationMode 为托管机房模式

（2） 通道线程数 coloca_cfg.qtcp_channel_thread

最小值为 1，最大值为 8。

（3） 通道线程内数据条数 coloca_cfg.qtcp_max_req_cnt

最小值为 1000，最大值为 3000。

3.3.1.1 TCP 通道

（1） 通道设置

Cfg::ColocaCfg::channel_mode 为 ColocatChannelMode.kTCP

（2） 数据获取范围

只支持订阅接口的数据，不包含订阅范围外的数据。

（3） 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。

3.3.1.2 QTCP 通道

（1） 通道设置

Cfg::ColocaCfg::channel_mode 为 ColocatChannelMode.kQTCP

（2） 数据获取范围

只支持查询接口的数据，不包含查询范围外的数据。

（3） 异常情况说明

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。

3.3.1.3 RTCP 通道

（1） 通道设置

Cfg::ColocaCfg::channel_mode 为 ColocatChannelMode.kRTCP

（2） 数据获取范围

只支持回放接口的数据，不包含回放范围外的数据。

（3） 异常情况说明

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。

3.3.1.4 TCP、QTCP、RTCP 通道同时使用的通道设置

（1） 三个通道 TCP、QTCP、RTCP，可两两结合，也可三个同时使用。

（2） 通过设置 Cfg::ColocaCfg::channel_mod，用“|”分开即可。

（3） 例如：

galaxy::tgw::ColocatChannelMode::kQTCP|galaxy::tgw::ColocatChannelMode::kTCP

3.3.2 互联网模式

（1） 数据接入方式

通过 websocket 方式和上游保持连接，并进行行情数据的推送和查询。

（2） 通道模式 api_mode 设置

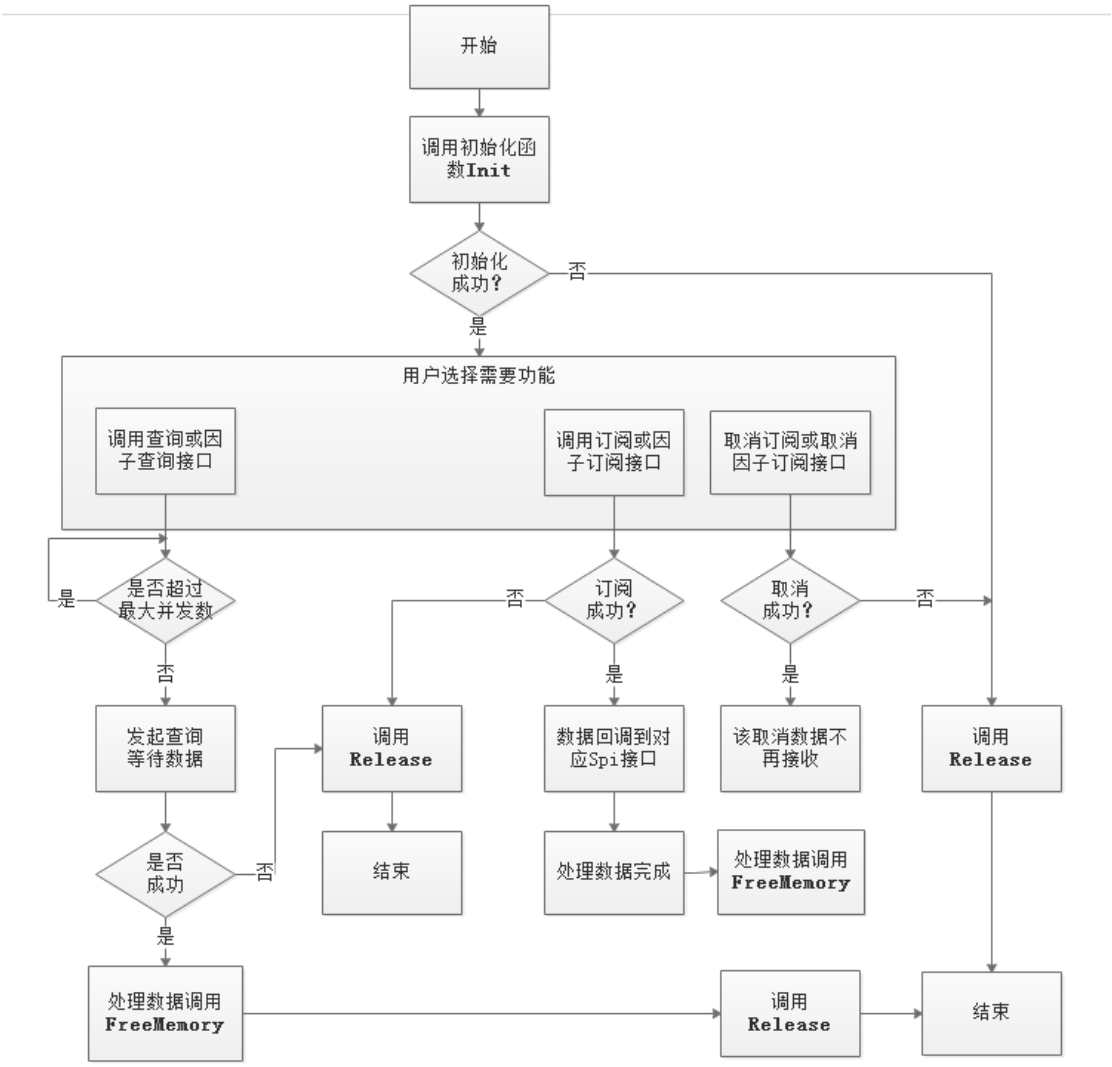
tgw.ApiMode.kInternetMode 为互联网模式

3.4 API 开发步骤

流程描述：

1. tgw 初始化配置
   包含服务器地址、用户账号密码和运行模式配置
2. 基础接口包含初始化、订阅、退订、释放等方法，无需创建实例；
3. 数据接口包含订阅、查询和回放等三类方法；
   （1） 继承所需的 tgw 中的 IGMDSpi 类，在派生类中重载回调函数；
   （2） 派生类实例化；
   （3） 派生类的实例传递给 tgw.IGMDApi 函数，供 tgw 回调使用；
   （4） 派生类的实例生命周期长于 tgw，即派生类实例化必须在 init 初始化之前；

查询和订阅接口使用流程图



3.4.1 初始化 TGW

首先调用初始化函数 IGMDApi::Init，该函数传入用户申请后的账户密码、接入行情网关的 IP 地址和端口，以及注册回调函数类对象。

```
galaxy::tgw::Cfg cfg;
strncpy(cfg.server_vip, "127.0.0.1", sizeof(cfg.server_vip));
cfg.server_port = 9200;
strncpy(cfg.username, "amd", sizeof(cfg.username));
strncpy(cfg.password, "123456", sizeof(cfg.password));
uint16_t api_mode = galaxy::tgw::ApiMode::kColocationMode;   // 设置 api 模式 托管机房模式/互联网模式
cfg.coloca_cfg.channel_mode = galaxy::tgw::ColocatChannelMode::kQTCP |
                              galaxy::tgw::ColocatChannelMode::kTCP|
                              galaxy::tgw::ColocatChannelMode::kRTCP;;
cfg.coloca_cfg.qtcp_channel_thread = 8;
cfg.coloca_cfg.qtcp_max_req_cnt = 1000;


galaxy::tgw::TgwPushSpi spi;
// 初始化 TGW
int32_t ec = galaxy::tgw::ErrorCode::kSuccess;
if ((ec = galaxy::tgw::IGMDApi::Init(&spi, cfg, api_mode))   != galaxy::tgw::ErrorCode::kSuccess)
{
   galaxy::tgw::IGMDApi::Release();
   return -1;
}
```

3.4.2 异常释放资源处理

初始化成功则可执行订阅或者取消订阅操作,初始化失败则调用 IGMDApi::Release 函数释放资源后退出。

3.4.3 数据订阅

订阅时调用接口 IGMDApi::Subscribe，需要填写订阅参数 SubscribeItem 并传入该接口，参数包括订阅的市场类型、数据类型、证券代码。接口可被多次调用多次订阅，如果用户没有被限制订阅数量则可以填写订阅所有市场类型,所有数据类型以及证券代码可提供所有选项，从而可以订阅全市场行情。若有限制订阅数则只能单市场单证券代码多次订阅，此时数据类型不受限制可填写所有类型。如果连续多次调用订阅接口，可参照以下 demo 示例，用数组或者其他方式存储参数进行多次订阅。

```
galaxy::tgw::SubscribeItem sub_cfg[2];
sub_cfg[0].market = galaxy::tgw::MarketType::kSZSE;
sub_cfg[0].flag = galaxy::tgw::SubscribeDataType::kNone;
strncpy(sub_cfg[0].security_code, "000001", sizeof(sub_cfg[0].security_code));
sub_cfg[0].category_type = galaxy::tgw::VarietyCategory::kNone;

sub_cfg[1].market = galaxy::tgw::MarketType::kNone;
sub_cfg[1].flag = galaxy::tgw::SubscribeDataType::kNone;
strncpy(sub_cfg[1].security_code, "000002", sizeof(sub_cfg[1].security_code));
sub_cfg[1].category_type = galaxy::tgw::VarietyCategory::kNone;

galaxy::tgw::IGMDApi::Subscribe(sub_cfg, 2);
```

3.4.4 取消订阅

取消订阅调用接口 IGMDApi::UnSubscribe，参数设置与订阅接口相同，取消订阅后将不再接收该类型的数据。

3.4.5 获取数据

订阅成功后，对应数据将回调到提供的接口，如 OnMDSnapshot()方法将接收到现货快照数据，具体数据类型对应不同接口，用户可选择实现或者不处理，在完成对应处理后，必须调用 FreeMemory()方法释放传入的指针内存，否则将造成内存泄漏。各种数据回调处理函数接口请参看下面的 IGMDSpi 接口详细介绍。

3.5 API 接口详细

3.5.1 参数说明

参数类型

- 入参（in）代表必须输入的参数。
- 出参（out）表示程序输出的参数。

精度说明

- 取值范围：为兼容多市场， API 为对各行情字段的取值范围规则如下：整形以 int64 表示，范围不限定，字符串以各行情字段设定为准，浮点型以 int64 表示，范围不限定。
- 价格精度：价格类型（pre_close_price、open_price、last_prise、high_prise 等）小数位数为 6，实际值需除以 1000000。
- 总金额精度：总金额（total_value_trade）小数位为 5，实际值需除以 100000。
- 总数量精度：总数量（total_volume_trade）小数位为 2，实际值需除以 100。

具体业务字段含义及精度说明如下：

业务字段	精度（不区分沪深）	转换规则
价格 （各种价格、均价）	整型，后 6 位表示小数位	价格、数量、金额字段统一全是 int 型。例如，收到价格后，除以 1000000 表示真正的价格。同样的，收到的量除以 100 得到真实量，收到的金额除以 100000 得到真实金额。
量（各种数量）	整型，后 2 位表示小数位	
金额（各种金额）	整型，后 5 位表示小数位	
个股信息：昨收	整型，后 6 位表示小数位	收到昨收价格后，除以 1000000 表示真正的价格。
汇率	整型，后 8 位表示小数位	收到汇率数据后，除以 100000000 表示真正的汇率值
比例（涨跌幅、市盈率等）	整型，后 6 位表示小数位	涨跌幅一类比例数据需要除以 1000000

单位说明

市场 / 证券品种	上海	深圳

股票	1 股	1 股
债券	10 张	1 张
基金	1 份	1 份
期权	1 张	1 张

常量定义

开发所需数据结构中对数据长度的常量定义请参考：

```
class ConstField
{
public:
    static const uint32_t kIPMaxLen                    = 24;    // IP 地址的最大长度
    static const uint32_t kUMSItemLen                  = 10;    // 服务项信息的最大个数
    static const uint32_t kUsernameLen                 = 32;    // 用户名的最大长度
    static const uint32_t kPasswordLen                 = 64;    // 用户名密码的最大长度
    static const uint32_t kSecurityCodeLen             = 16;    // 证券代码最大长度
    static const uint32_t kHistorySecurityCodeLen      = 32;    // 历史回放和查询证券代码最大长度
    static const uint32_t kPushSecurityCodeLen         = 32;    // 实时推送证券代码最大长度
    static const uint32_t kSecurityNameLen             = 32;    // 证券名称最大长度
    static const uint32_t kTradingPhaseCodeLen         = 8;     // 交易状态标志
    static const uint32_t kExChangeInstIDLen           = 31;    // 合约在交易所代码
    static const uint32_t kPositionLevelLen            = 10;    // 行情档位
    static const uint32_t kMDStreamIDLen               = 6;     // 行情类别
    static const uint32_t kSecurityStatusLen           = 24;    // 证券状态最大长度
    static const uint32_t kSecuritySymbolLen           = 32;    // 证券简称最大长度
    static const uint32_t kSecurityEnglishNameLen      = 128;   // 证券英文简称最大长度
    static const uint32_t kSecurityTypeLen             = 10;    // 证券类别长度
    static const uint32_t kCurrencyLen                 = 4;     // 币种长度
    static const uint32_t kVolumeTradeLen              = 24;    // 成交量长度
    static const uint32_t kValueTradeLen               = 24;    // 交易额长度
    static const uint32_t kFutureSecurityCodeLen       = 32;    // 期货代码最大长度
    static const uint32_t kMDStreamIDMaxLen            = 6;     // 行情类别字段最大长度
    static const uint32_t kDataPermission              = 128;   // 数据权限最大长度
    static const uint32_t kTokenMaxLen                 = 128;   // token 的最大长度
    static const uint32_t kQuerySecurityCodeLen        = 38;    // 查询状态回调代码最大长度
    static const uint32_t kTradingStatusLen            = 8;     // 证券交易状态档位
    static const uint32_t kSymbolLen                   = 128;   // 证券简称最大长度
    static const uint32_t kSecurityAbbreviationLen     = 64;    // 证券简称最大长度
    static const uint32_t kMaxTypesLen                 = 16;    // 证券类型最大长度
    static const uint32_t kTypesLen                    = 8;     //
    static const uint32_t kCodeTableSecurityStatusMaxLen = 16;  // 代码表证券状态字符最大长度
    static const uint32_t RegularShare                 = 9;     // 对应回购标准券长度
};
```

3.5.2 基础接口

TGW 接口操作类，主要用于推送接口。该类不需要创建实例，直接调用类方法即可。如 IGMDApi::Release()。

1) GetVersion 方法

获取 TGW 版本信息，托管机房和互联网模式适用。

函数原型：

```
static const char* GetVersion();
```

返回值：返回版本号信息。

2) Init 方法

初始化 TGW，托管机房和互联网模式适用 。

函数原型：

static int32_t Init(const IGMDSpi* pSpi, const Cfg& cfg, uint16_t api_mode);

参数：

参数	解释
pSpi(in)	IGMDSpi 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)	TGW 内部需要的配置参数
api_mode(in)	模式选择，取值请参照公共数据字典中的 ApiMode

返回值：返回错误码，参照公共数据字典中 ErrorCode 定义。

3) Release 方法

释放 TGW，托管机房和互联网模式适用。

函数原型：

static int32_t Release();

返回值：返回错误码，参照公共数据字典中 ErrorCode 定义。

4) FreeMemory 方法

释放内存， 托管机房和互联网模式适用。

TGW 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。回调除 OnLog 外，其余都需要显式地调用 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

5) GetTaskID 方法

获取唯一的回放任务 id 值， 托管机房和互联网模式适用。

函数原型：

static int64_t GetTaskID();

说明：

返回值为回放任务 id，格式为 MMDDHHmmSS+序列号（1~1000000）。例如：524153030000001（5 月 24 日 15 时 30 分 30 秒，序列号 1）

3.5.3 订阅数据方法

6) Subscribe 方法

订阅行情数据， 托管机房和互联网模式适用。

函数原型：

static int32_t Subscribe(const SubscribeItem*item,uint32_t cnt);

参数：

参数	解释
*item(in)	订阅信息数据结构体(支持一次设置多个订阅信息)
cnt(in)	订阅信息数量

SubscribeItem 结构定义如下：

数据类型	数据名称	说明
uint8_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	各数据类型的集合，为 0 表示订阅所有支持的数据类型,参考公共字典中中的 SubscribeDataType
char	security_code[ConstField::kFutureSecurityCodeLen]	证券代码，为空表示订阅所有代码
uint8_t	category_type	品种类别，参考 VarietyCategory,为 0 表示订阅所有支持的品种，仅该参数仅互联网模式有效

7) UnSubscribe 方法

取消行情数据订阅， 托管机房和互联网模式适用。

函数原型：

static int32_t UnSubscribe(const SubscribeItem* item, uint32_t cnt);

参数：

参数	解释
*item(in)	取消订阅信息数据结构体(一次可取消多个订阅信息)
cnt(in)	取消订阅信息数量

其中，SubscribeItem 结构定义参照 Subscribe 方法：

8) SubFactor 方法

增加加工因子数据订阅，托管机房和互联网模式适用。

函数原型：

static int32_t SubFactor(const SubFactorItem*item,uint32_t cnt);

参数：

参数	解释
*item(in)	订阅信息数据结构体(支持一次设置多个订阅信息)
cnt(in)	订阅信息数量

SubFactorItem 结构定义如下：

数据类型	数据名称	说明
char	factor_type[64]	因子父类型(英文)
char	factor_sub_type[64]	因子子类型(英文)
char	factor_name[64]	因子名称(英文)

全订阅仅支持以下三种方式：

1、全订阅支持 all all all 方式，表示订阅所有权限的组合;

2、支持 xxx all all 方式订阅（指定父类型），表示订阅某一父类型下子类型和因子名称的组合;

3、支持 xxx xxx all 方式订阅（指定父类型、子类型），表示订阅某一父类型和子类型下的全部因子名称;

9) UnSubFactor 方法

取消加工因子数据订阅，托管机房和互联网模式适用。

函数原型：

static int32_t UnSubFactor(const SubFactorItem * item, uint32_t cnt);

参数：

参数	解释
*item(in)	取消订阅信息数据结构体(一次可取消多个订阅信息)
cnt(in)	取消订阅信息数量

其中，SubFactorItem 结构定义参照 SubFactor 方法

取消全订阅仅支持以下三种方式：

1、取消全订阅支持 all all all 方式，表示取消订阅所有权限的组合;

2、支持 xxx all all 方式取消订阅（指定父类型），表示取消订阅某一父类型下子类型和因子名称的组合;

3、支持 xxx xxx all 方式取消订阅（指定父类型、子类型），表示取消订阅某一父类型和子类型下的全部因子名称;

3.5.4 订阅回调方法

TGW 中的接收数据的回调基类。使用 TGW 时需要继承该类，并将派生类的实例传递给 IGMDApi::Init 函数，供 TGW 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 TGW。(接口方法默认支持托管机房和互联网模式，仅单一模式支持时将后缀说明)。

1) OnLog 方法

接收日志数据的回调函数，用户可选择如何处理,。

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，详见公共数据字典 LogLevel
log(out)	日志内容
len(out)	日志内容长度

2) 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（仅托管机房模式适用）。

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，参考公共字典 EventLevel
code(out)	事件代码
event_msg(out)	事件具体信息
len(out)	事件具体信息长度

3) OnIndicator 方法

接收监控数据回调（仅托管机房适用）。

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

4) OnLogon 方法

接收登录成功时的信息，使用者可根据需求对该回调信息做相应的处理。

函数原型：

void OnLogon(LogonResponse* data);

参数：

参数	解释
data(out)	登录成功后的校验信息

LogonResponse 结构定义如下

数据类型	数据名称	说明
uint16_t	api_mode	1：托管机房　2：互联网
uint32_t	logon_msg_len	登录返回信息长度
char*	logon_json	登录返回信息，格式为 json

5) OnMDSnapshot 方法

接收现货快照推送数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDSnapshot(MDSnapshotL1* data, uint32_t cnt);

参数：

参数	解释
data(out)	现货快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	数据个数

6) OnMDIndexSnapshot 方法

接收指数快照推送数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDIndexSnapshot(MDIndexSnapshot* data , uint32_t cnt);

参数：

参数	解释
data(out)	指数快照数据结构体
cnt(out)	数据个数

7) OnMDOptionSnapshot 方法

接收期权快照推送数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDOptionSnapshot(MDOptionSnapshot* data, uint32_t cnt);

参数：

参数	解释
data(out)	期权快照数据结构体
cnt(out)	数据个数

8) OnMDHKTSnapshot 方法

接收港股通快照推送数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDHKTSnapshot(MDHKTSnapshot* data, uint32_t cnt););

参数：

参数	解释
data(out)	港股快照数据结构体
cnt(out)	数据个数

9) OnMDAfterHourFixedPriceSnapshot 方法

接收盘后定价交易快照数据回调（仅互联网模式适用）。

函数原型：

Virtual void OnMDAfterHourFixedPriceSnapshot
(MDAfterHourFixedPriceSnapshot* data, uint32_t cnt);

参数：

参数	解释
data(out)	接收盘后定价交易快照数据结构体
cnt(out)	数据个数

10) OnMDCSIIndexSnapshot 方法

接收中证指数快照数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDCSIIndexSnapshot(MDCSIIndexSnapshot* data, uint32_t cnt);

参数：

参数	解释
data(out)	接收中证指数快照数据结构体
cnt(out)	数据个数

11) OnMDCnIndexSnapshot 方法

接收深交所国证指数快照数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDCnIndexSnapshot(MDCnIndexSnapshot* data, uint32_t cnt);

参数：

参数	解释
data(out)	接收深交所国证指数快照数据结构体
cnt(out)	数据个数

12) OnMDHKTRealtimeLimit 方法

接收港股通实时额度数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDHKTRealtimeLimit(MDHKTRealtimeLimit* data, uint32_t cnt);

参数：

参数	解释
data(out)	港股通实时额度推送数据结构体
cnt(out)	数据个数

13) OnMDHKTProductStatus 方法

接收港股通产品状态数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDHKTProductStatus(MDHKTProductStatus* data, uint32_t cnt);

参数：

参数	解释
data(out)	接收港股通产品状态快照数据结构体
cnt(out)	数据个数

14) OnMDHKTVCM 方法

接收港股 VCM 数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDHKTVCM(MDHKTVCM* data, uint32_t cnt);

参数：

参数	解释
data(out)	接收港股 VCM 推送数据结构体
cnt(out)	数据个数

15) OnMDFutureSnapshot 方法

接收期货快照数据回调（仅互联网模式适用）。

函数原型：

virtual void OnMDFutureSnapshot(MDFutureSnapshot* data, uint32_t cnt);

参数：

参数	解释
data(out)	接收期货快照推送数据结构体
cnt(out)	期货快照数据条数

16) OnKLine 方法

接收历史 K 线数据回调。

函数原型：

virtual void OnKLine(MDKLine * data, uint32_t cnt, uint32_t kline_type);

参数：

参数	解释
data(out)	接收历史 K 线数据推送的数据结构体
cnt(out)	数据个数
kline_type(out)	K 线类型

17) OnSnapshotDerive 方法

接收现货衍生数据的推送回调。

函数原型：

virtual void OnSnapshotDerive(MDSnapshotDerive* data, uint32_t cnt);

参数：

参数	解释
data(out)	接收现货衍生数据结构体
cnt(out)	数据个数

18) OnFactor 方法

接收加工因子数据的推送回调。

函数原型：

virtual void OnFactor(Factor* data);

参数：

参数	解释
data(out)	加工因子数据首指针，需要显式地调用 FreeMemory

Factor 结构定义如下：

数据类型	数据名称	说明
uint32_t	data_size	json 数据的大小
char*	json_buf	json 结构

```
json_buf 结构如下:
{
    "headers":
    {
        "factor_name":,                    // 因子名称
        "factor_type":,                    // 因子父类型
        "factor_sub_type":,                // 因子子类型
        "origin_time":,                    // 数据原始时间
        "key1":                            // 预留字段 1
        "key2":                            // 预留字段 2
        "seq_num":                         // 因子序号
    }
    "body":                                //透传的部分
      {
          ......
      }
}
```

3.5.5 查询数据方法

QueryKline 方法

K 线查询接口，托管机房和互联网模式适用。

函数原型：

static int32_t QueryKline(const IGMDKlineSpi* kline_spi, const ReqKline& req_kline);

参数：

参数	解释
kline_spi(in)	IGMDKlineSpi 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（对应数据回调为 IGMDKlineSpi 下的 OnMDKLine 方法）
req_kline(in)	K 线查询数据项首地址

ReqKline 结构定义如下：

数据类型	数据名称	说明
char	security_code[ConstField::kQuerySecurityCodeLen]	证券代码
uint8_t	market_type	市场类型
uint8_t	cq_flag	除权标志（0:不复权 1:向前复权 2:向后复权默认 0）
uint32_t	cq_date	除权日期（yyyMMdd）（暂不使用）

uint32_t	qj_flag	全价标志（债券）（0：净价，1：全价）（暂不使用）
uint16_t	cyc_type	数据周期（参考 tgw_datatype.h 中的 MDDatatype 描述）
uint32_t	cyc_def	周期数量（暂不使用）
uint8_t	auto_complete	自动补全（0:不补齐，1：补齐默认 1）
uint32_t	begin_date	开始日期（yyyyMMdd）
uint32_t	end_date	结束日期（yyyyMMdd）
uint32_t	begin_time	开始时间（默认：HHmm，支持 HHmmssSSS）
uint32_t	end_time	结束时间（默认：HHmm，支持 HHmmssSSS）

QuerySnapshot 方法

快照（现货、指数、港股、期权、期货）查询接口，托管机房和互联网模式适用。

函数原型：

static int32_t QuerySnapshot(const IGMDSnapshotSpi* snapshot_spi,
const ReqDefault& req_snapshot);

参数：

参数	解释
snapshot_spi(in)	IGMDSnapshotSpi 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（对应数据回调为 IGMDSnapshotSpi 下的现货、指数、期货、期权等方法）
req_snapshot(in)	快照查询信息数据项首地址

ReqDefault 结构定义如下：

数据类型	数据名称	说明
char	security_code[ConstField::kQuerySecurityCodeLen]	证券代码
uint8_t	market_type	市场类型
uint32_t	date	日期（必须为 yyyyMMdd）
uint32_t	begin_time	开始时间（HHmmssSSS）
uint32_t	end_time	结束时间（HHmmssSSS）

QueryOrderQueue 方法

委托队列查询接口（仅托管机房适用）。

函数原型：

static int32_t QueryOrderQueue(const IGMDOrderQueueSpi* order_queue_spi,
const ReqDefault& req_order_queue);

参数：

参数	解释
order_queue_spi(in)	IGMDOrderQueueSpi 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（对应数据回调

	为 IGMDOrderQueueSpi 下的 OnMDOrderQueue 方法）
req_order_queue(in)	委托队列查询数据项首地址

其中，ReqDefault 结构定义参照 QuerySnapshot 方法

QueryTickExecution 方法

逐笔成交查询接口（仅托管机房适用）。

函数原型：

static int32_t QueryTickExecution(const IGMDTickExecutionSpi* tick_exec_spi, const ReqDefault& req_tick_execution);

参数：

参数	解释
tick_exec_spi(in)	IGMDTickExecutionSpi 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（对应数据回调为 IGMDTickExecutionSpi 下的 OnMDTickExecution 方法）
req_order_queue(in)	委托队列查询数据项首地址

其中，ReqDefault 结构定义参照 QuerySnapshot 方法

QueryTickOrder 方法

逐笔委托查询接口（仅托管机房适用）。

函数原型：

static int32_t QueryTickOrder(const IGMDTickOrderSpi* tick_order_spi, const ReqDefault& req_tick_order);

参数：

参数	解释
tick_order_spi(in)	IGMDTickOrderSpi 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（对应数据回调为 IGMDTickOrderSpi 下的 OnMDTickOrder 方法）
req_tick_order(in)	逐笔委托查询数据项首地址

其中，ReqDefault 结构定义参照 QuerySnapshot 方法

QueryCodeTable 方法

代码表查询接口（仅托管机房适用）。

函数原型：

static int32_t QueryCodeTable(const IGMDCodeTableSpi* code_table_spi);

参数：

参数	解释
code_table_spi(in)	IGMDCodeTableSpi 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（对应数据回调为

	IGMDCodeTableSpi 下的 OnMDCodeTable 方法）

QuerySecuritiesInfo 方法

股票基础信息查询接口，托管机房和互联网模式适用。

函数原型：

static int32_t QuerySecuritiesInfo (const IGMDSecuritiesInfoSpi* stock_info_spi, const SubCodeTableItem* item, uint32_t cnt);

参数：

参数	解释
stock_info_spi(in)	IGMDSecuritiesInfoSpi*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（对应数据回调为 IGMDSecuritiesInfoSpi*下的 OnMDSecuritiesInfo 方法）
item(in)	查询数组首地址

SubCodeTableItem 定义如下：

数据类型	数据名称	说明
int32_t	market	市场类型,参考 MarketType,为 kNone 表示查询所有支持的市场(代码表目前只支持上交所、深交所与北交所)
char	security_code[ConstField::kFutureSecurityCodeLen]	证券代码,为空表示查询所有代码

QueryExFactorTable 方法

复权因子表查询接口（托管机房和互联网模式适用）。

函数原型：

static int32_t QueryExFactorTable(const IGMDExFactorSpi* ex_factor_spi, const char* code);

参数：

参数	解释
ex_factor_spi(in)	IGMDExFactorSpi 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（对应数据回调为 IGMDExFactorSpi 下的 OnMDExFactor 方法）
code(in)	输入待查询的证券代码（例如：000001）

QueryFactor 方法

加工因子查询接口，托管机房和互联网模式适用。

函数原型：

static int32_t QueryFactor(const IGMDFactorSpi* factor_spi, const ReqFactor& req_factor);

参数：

参数	解释
factor_spi(in)	IGMDFactorSpi 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（对应数据回调为 IGMDFactorSpi 下的 OnFactor 方法）
req_factor(in)	加工因子数据项首地址

ReqFactor 结构中每个字符串的字段名称和类型参考下表：

数据类型	数据名称	说明
char	factor_type[64]	因子父类型(英文)
char	factor_sub_type[64]	因子子类型(英文)
char	factor_name[64]	因子名称(英文)
uint32_t	begin_date	开始日期（yyyyMMdd）
uint32_t	end_date	结束日期（yyyyMMdd）
uint32_t	begin_time	开始时间（HHmmssSSS）
uint32_t	end_time	结束时间（HHmmssSSS）
char	Key1[64]	预留字段 Key1
char	Key2[64]	预留字段 Key2
说明：1、开始日期和结束日期需要相等，目前仅支持某一天加工因子查询；		

SetThirdInfoParam 方法

设置金融资讯数据设置接口，托管机房和互联网模式适用。

函数原型：

static int32_t SetThirdInfoParam(int64_t task_id, const std::string& key, const std::string& value);

参数：

参数	解释
task_id(in)	金融资讯数据请求编号
key(in)	金融资讯数据请求 json 的 key
value(in)	金融资讯数据请求 json 的 value

说明：在进行查询金融资讯数据前，必须先设置 function_id（key）、功能号（value）、任务 id；

function_id： 此 key 为必须输入的值（此字段为指定值，不能为其他值），在进行查询前必须先设置功能号和任务 id 才可以调用 QueryThirdInfo 方法。

QueryThirdInfo 方法

金融资讯数据查询接口，托管机房和互联网模式适用。

函数原型：

static int32_t QueryThirdInfo(const IGMDThirdInfoSpi* third_info_spi, int64_t task_id);

参数：

参数	解释

third_info_spi(in)	IGMDThirdInfoSpi 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（对应数据回调为 IGMDThirdInfoSpi 下的 OnThirdInfo 方法）
task_id(in)	金融资讯数据请求编号

3.5.6 查询回调方法

1) IGMDKlineSpi 接口

接收 K 线数据。

OnMDKLine 方法

接收 K 线数据回调接口（互联网和托管机房模式适用）。

函数原型：

void OnMDKLine(MDKLine* klines, uint32_t cnt, uint16_t kline_type);

参数：

参数	解释
klines(out)	K 线数据首指针，需要显式地调用 FreeMemory
cnt(out)	K 线数据个数

2) IGMDSnapshotSpi 接口

接收快照数据。

OnMDSnapshotL2 方法

L2 现货快照（股票、债券、基金）回调接口（仅托管机房适用）。

函数原型：

void OnMDSnapshotL2(MDSnapshotL2* snapshots, uint32_t cnt);

参数：

参数	解释
snapshots(out)	快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	快照数据个数

OnMDSnapshotL1 方法

L1 现货快照（股票、债券、基金）回调接口（仅互联网使用）。

函数原型：

void OnMDSnapshotL1(MDSnapshotL1* snapshots, uint32_t cnt);

参数：

参数	解释
snapshots(out)	快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	快照数据个数

OnMDIndexSnapshot 方法

指数快照回调接口（互联网和托管机房模式适用）。

函数原型：

void OnMDIndexSnapshot(MDIndexSnapshot* index_snapshots, uint32_t cnt);

参数：

参数	解释
index_snapshots(out)	指数快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	指数快照数据个数

OnMDHKTSnapshot 方法

港股通快照回调接口（互联网和托管机房模式适用）。

函数原型：

void OnMDHKTSnapshot(MDHKTSnapshot* hkt_snapshots, uint32_t cnt);;

参数：

参数	解释
hkt_snapshots(out)	港股通快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	港股通快照数据个数

OnMDOptionSnapshot 方法

期权快照回调接口（互联网和托管机房模式适用）。

函数原型：

void OnMDOptionSnapshot(MDOptionSnapshot* opt_snapshots, uint32_t cnt);

参数：

参数	解释
opt_snapshots(out)	期权快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	期权快照数据个数

OnMDFutureSnapshot 方法

期货快照回调接口（互联网和托管机房模式适用）。

函数原型：

void OnMDFutureSnapshot(MDFutureSnapshot* future_snapshots, uint32_t cnt);

参数：

参数	解释
future_snapshots(out)	期货快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	期货快照数据个数

3) IGMDOrderQueueSpi 接口

接收委托队列数据（仅托管机房模式适用）

OnMDOrderQueue 方法

委托队列回调接口。

函数原型：

void OnMDOrderQueue(MDOrderQueue* order_queues, uint32_t cnt);;

参数：

参数	解释
order_queues(out)	委托队列数据首指针，需要显式地调用 FreeMemory
cnt(out)	委托队列数据个数

4) IGMDTickExecutionSpi 接口

接收逐笔成交数据。

OnMDTickExecution 方法

逐笔成交回调接口（仅托管机房模式适用）。

函数原型：

void OnMDTickExecution(MDTickExecution* tick_execs, uint32_t cnt);

参数：

参数	解释
tick_execs(out)	逐笔成交数据首指针，需要显式地调用 FreeMemory
cnt(out)	逐笔成交数据个数

5) IGMDTickOrderSpi 接口

逐笔委托回调接口（仅托管机房模式适用）。

OnMDTickOrder 方法

逐笔委托回调接口（仅托管机房模式适用）。

函数原型：

void OnMDTickOrder(MDTickOrder* tick_orders, uint32_t cnt);

参数：

参数	解释
tick_orders(out)	逐笔委托数据首指针，需要显式地调用 FreeMemory
cnt(out)	逐笔委托数据个数

6) IGMDCodeTableSpi 接口

接收代码表数据。

OnMDCodeTable 方法

代码表回调接口（仅托管机房模式适用）。

函数原型：

void OnMDCodeTable(MDCodeTable* code_tables, uint32_t cnt)；

参数：

参数	解释
code_tables(out)	代码表数据首指针，需要显式地调用 FreeMemory
cnt(out)	代码表数据个数

7) IGMDSecuritiesInfoSpi 接口

接收个股基本信息数据

OnMDSecuritiesInfo 方法

个股基本信息回调接口（互联网和托管机房模式适用）。

函数原型：

void OnInterMDStockInfo(MDCodeTableRecord* code_tables, uint32_t cnt)；

参数：

参数	解释
code_tables(out)	个股基本信息数据首指针，需要显式地调用 FreeMemory
cnt(out)	个股基本信息数据个数

8) IGMDExFactorSpi 接口

复权因子回调接口（互联网和托管机房模式适用）。

OnMDExFactor 方法

复权因子回调接口（互联网和托管机房模式适用）。

函数原型：

void OnMDExFactor(MDExFactorTable* ex_factor_tables, uint32_t cnt);；

参数：

参数	解释
ex_factor_tables(out)	复权因子数据首指针，需要显式地调用 FreeMemory
cnt(out)	复权因子数据个数

9) IGMDFactorSpi 接口

接收加工因子数据。

OnFactor 方法

加工因子数据回调接口（互联网和托管机房模式适用）。

函数原型：

void OnFactor(Factor* factors, uint32_t cnt)；

参数：

参数	解释
factors(out)	加工因子数据首指针，需要显式地调用 FreeMemory
cnt(out)	加工因子数据个数

10) IGMDThirdInfoSpi 接口

接收金融资讯数据（互联网和托管机房模式适用）。

OnThirdInfo 方法

金融资讯数据回调接口。

函数原型：

void OnThirdInfo(ThirdInfoData* data, uint32_t cnt);

参数：

参数	解释
data(out)	金融资讯数据首指针，需要显式地调用 FreeMemory
cnt(out)	金融资讯数据个数

3.5.7 回放数据方法

ReplayKline 方法

K 线回放接口（支持分钟 K 和日 K 回放）（仅托管机房适用）。

函数原型：

static int32_t ReplayKline(const IGMDHistorySpi* pSpi, const ReqReplayKline& req_kline)

参数：

参数	解释
pSpi(in)	IGMDHistorySpi 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（对应数据回调为 IGMDHistorySpi 下的 OnMDKline 方法）
req_kline(in)	K 线回放请求

ReqReplayKline 结构定义如下：

数据类型	数据名称	说明
uint8_t	cq_flag	除权标志（0:不复权 1:向前复权 2:向后复权，默值认为 0）
uint32_t	cq_date	除权日期（yyyMMdd）（暂不使用）
uint32_t	qj_flag	全价标志（债券）（0：净价，1：全价）（暂不使用）

uint16_t	cyc_type	数据周期，参照 tgw_datatype.h 中的 MDDatatype（不支持周月季年 k）
uint32_t	cyc_def	周期数量（暂不使用）
uint8_t	auto_complete	自动补全（0:不补齐，1：补齐，默认值为 1）
uint32_t	begin_date	开始日期（yyyyMMdd）
uint32_t	end_date	结束日期（yyyyMMdd）
uint32_t	begin_time	开始时间（HHmm）
uint32_t	end_time	结束时间（HHmm）
uint16_t	replay_speed	返回倍速（暂不可用）
int64_t	task_id	任务 id(任务编号，例如:1)调用 GetTaskId 获取
ReqHistoryItem*	req_items	请求数组 item 头指针,不得为空
uint32_t	req_item_cnt	请求回放代码数量

ReqHistoryItem 结构定义如下：

数据类型	数据名称	说明
uint8_t	market	市场类型，参考 MarketType
char	security_code[ConstField::kHistorySecurityCodeLen]	证券代码，代码不能为空

ReplayRequest 方法

快照、逐笔成交回放接口（仅托管机房适用）。

函数原型：

static int32_t ReplayRequest(const IGMDHistorySpi* pSpi, const ReqReplay& req_replay)。

参数：

参数	解释
pSpi(in)	IGMDHistorySpi 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（对应数据回调为 IGMDHistorySpi 下的快照和逐笔成交方法）
req_replay(in)	行情回放请求

ReqReplay 结构定义如下：

数据类型	数据名称	说明
uint16_t	md_data_type	回放数据类型,参照 tgw_datatype.h 中的 MDDatatype
uint32_t	begin_date	开始日期（yyyyMMdd）
uint32_t	end_date	结束日期（yyyyMMdd）
uint32_t	begin_time	开始时间（HHmmssSSS）
uint32_t	end_time	结束时间（HHmmssSSS）
uint16_t	replay_speed	返回倍速（暂不可用）
int64_t	task_id	任务 id(任务编号,例如:1)调用 GetTaskId

		获取
ReqHistoryItem*	req_items	请求数组 item 头指针,不得为空
uint32_t	req_item_cnt	请求回放代码数量

CancelTask 方法

取消指定的回放任务（仅托管机房适用）。

函数原型：

static int32_t CancelTask(const int64_t& task_id)。

参数：

参数	解释
task_id(in)	需取消的回放任务 id

说明：满足快速取消一个回放任务，重新定义另一个回放任务或想快速安全退出程序，重启程序快速开始新的回放任务；默认 0 表示取消所有的任务。

3.5.8 回放回调方法

1) OnMDKline 方法

接收 K 线回放数据回调（仅托管机房适用）。

函数原型：

void OnMDKline(int64_t task_id, MDKLine* klines, uint32_t cnt, uint16_t kline_type);

参数：

参数	解释
task_id(out)	回放任务 id
kline(out)	K 线数据首指针，需要显式地调用 FreeMemory
cnt(out)	K 线个数
kline_type(out)	取值参考公共字典 MDDatatype

2) OnMDSnapshot 方法

接收 K 线回放数据回调（仅托管机房适用）。

函数原型：

void OnMDSnapshot(int64_t task_id, MDSnapshotL2* snapshots, uint32_t cnt);

参数：

参数	解释
task_id(out)	回放任务 id
snapshots(out)	快照数据首指针，需要显式地调用 FreeMemory
cnt(out)	快照数据个数

3) OnMDTickExecution 方法

接收逐笔成交回放数据回调（仅托管机房适用）。

函数原型：

void OnMDTickExecution(int64_t task_id, MDTickExecution* ticks, uint32_t cnt);

参数：

参数	解释
task_id(out)	回放任务 id
ticks(out)	逐笔成交数据首指针，需要显式地调用 FreeMemory
cnt(out)	逐笔成交数据个数

4) OnRspTaskStatus 方法

接收回放任务状态（仅托管机房使用）。

函数原型：

void OnRspTaskStatus(int64_t task_id, RspTaskStatus* task_status);

参数：

参数	解释
task_id(out)	回放任务 id
task_status(out)	回放任务状态
task_id(out)	回放任务 id

RspTaskStatus 结构定义如下：

数据类型	数据名称	说明
int64_t	task_id	任务 id(任务编号，例如:1)
int16_t	status	状态值(详细参考 HistoryTaskStatus)
int16_t	process_rate	进度（暂不支持）
int16_t	error_code	状态为失败的错误码(详细参考 ErrorCode)
int16_t	error_msg_len	错误消息长度
char*	error_msg	错误消息首地址

3.6 开发 Demo 示例

SDK 安装包中提供 tgw_demo 开发示例可供参考：

```
#include <string.h>
#include <unistd.h>
#include "tgw_push_spi.h"
#include "tgw_query_spi.h"
#include "tgw_replay_spi.h"
#include <tgw_datatype.h>

// ------------------------------ 推送请求示例 ----------------------------------
void GeneralSubscribeReq()
{
    // 设置订阅参数，以两次订阅为例
```

```
    galaxy::tgw::SubscribeItem sub_cfg[2];

    // 第一次订阅的市场类型设置
    sub_cfg[0].market = galaxy::tgw::MarketType::kSZSE;
    // 第一次订阅的数据类型设置
    sub_cfg[0].flag = galaxy::tgw::SubscribeDataType::kNone;
    // 第一次订阅的证券代码设置
    strncpy(sub_cfg[0].security_code, "000001", sizeof(sub_cfg[0].security_code));
    // 第一次订阅的品种类别设置
    sub_cfg[0].category_type = galaxy::tgw::VarietyCategory::kNone;

    // 第二次订阅的市场类型设置
    sub_cfg[1].market = galaxy::tgw::MarketType::kNone;
    // 第二次订阅的数据类型设置
    sub_cfg[1].flag = galaxy::tgw::SubscribeDataType::kNone;
    // 第二次订阅的证券代码设置，为空代表全部证券代码
    strncpy(sub_cfg[1].security_code, "", sizeof(sub_cfg[1].security_code));
    // 第二次订阅的品种类别设置
    sub_cfg[1].category_type = galaxy::tgw::VarietyCategory::kStock;

    //  发送多个订阅请求
    galaxy::tgw::IGMDApi::Subscribe(sub_cfg, 2);

    //  取消订阅
    // galaxy::tgw::IGMDApi::UnSubscribe(sub_cfg, 2);
}

void FactorSubscribeReq()
{
    /**
     * @note 父类型不支持单独全订阅及取消订阅，即 all xxx xxx
     *       因子子类型不支持单独全订阅及取消订阅,即 xxx all xxx
     *       不支持父类型和子类型全订阅及取消订阅，但是因子名称为指定因子，即 all all xxx
     *       不支持因子父类型和因子名称全订阅及取消订阅，但是因子子类型为指定因子，即 all xxx all
     */
    galaxy::tgw::SubFactorItem item[4];
    // 订阅全因子类型设置 all all all
    strncpy(item[0].factor_type, "all", sizeof(item[0].factor_type));
    strncpy(item[0].factor_sub_type, "all", sizeof(item[0].factor_sub_type));
    strncpy(item[0].factor_name, "all", sizeof(item[0].factor_name));

    galaxy::tgw::IGMDApi::SubFactor(&item[0], 1);

    // 订阅指定父类型下全因子类型设置 xxx all all
    strncpy(item[1].factor_type, "Financial_Factor", sizeof(item[1].factor_type));
    strncpy(item[1].factor_sub_type, "all", sizeof(item[1].factor_sub_type));
    strncpy(item[1].factor_name, "all", sizeof(item[1].factor_name));

    galaxy::tgw::IGMDApi::SubFactor(&item[1], 1);

    // 订阅指定父类型+子类型下全因子类型设置 xxx xxx all
    strncpy(item[2].factor_type, "Financial_Factor", sizeof(item[2].factor_type));
    strncpy(item[2].factor_sub_type, "PE", sizeof(item[2].factor_sub_type));
    strncpy(item[2].factor_name, "all", sizeof(item[2].factor_name));

    galaxy::tgw::IGMDApi::SubFactor(&item[2], 1);

    // 订阅指定父类型+子类型+因子名称类型设置 xxx xxx xxx
    strncpy(item[3].factor_type, "Financial_Factor", sizeof(item[3].factor_type));
    strncpy(item[3].factor_sub_type, "PE", sizeof(item[3].factor_sub_type));
    strncpy(item[3].factor_name, "PE_TTL", sizeof(item[3].factor_name));

    galaxy::tgw::IGMDApi::SubFactor(&item[3], 1);
    // 取消因子订阅
    // galaxy::tgw::IGMDApi::UnSubFactor(&item[3], 1);   入参同上
}

// ------------------------------ 查询请求示例 ------------------------------------
void QueryKlineReq(galaxy::tgw::QueryKlineSpi* spi)
{
    galaxy::tgw::ReqKline req_kline;
    memset(&req_kline, 0, sizeof(req_kline));
    strncpy(req_kline.security_code, "000001", sizeof(req_kline.security_code));
    req_kline.market_type = 102;
    req_kline.cq_flag = 0;
    req_kline.auto_complete = 1;
    req_kline.cyc_type = galaxy::tgw::MDDatatype::k1KLine; // 1 分钟 k
```

```
	req_kline.begin_date = 20210611;
	req_kline.end_date = 20210611;
	req_kline.begin_time = 930;
	req_kline.end_time = 1700;

	int32_t ec;
	if ((ec = galaxy::tgw::IGMDApi::QueryKline(spi, req_kline)) != galaxy::tgw::ErrorCode::kSuccess)
		std::cout << "QueryKline faild, ErrorCode: " << ec << std::endl;
}

void QuerySnapshotReq(galaxy::tgw::QuerySnapshotSpi* spi)
{
	galaxy::tgw::ReqDefault req_snapshot;
	memset(&req_snapshot, 0, sizeof(req_snapshot));
	strncpy(req_snapshot.security_code, "000001", sizeof(req_snapshot.security_code));
	req_snapshot.market_type = 102;
	req_snapshot.date = 20210611;
	req_snapshot.begin_time = 93000000;
	req_snapshot.end_time = 170000000;

	int32_t ec;
	if ((ec = galaxy::tgw::IGMDApi::QuerySnapshot(spi, req_snapshot)) != galaxy::tgw::ErrorCode::kSuccess)
		std::cout << "QuerySnapshot faild, ErrorCode: " << ec << std::endl;
}

void QueryTickExecReq(galaxy::tgw::QueryTickExecutionSpi* spi)
{
	galaxy::tgw::ReqDefault req;
	memset(&req, 0, sizeof(req));
	strncpy(req.security_code, "000001", sizeof(req.security_code));
	req.market_type = 102;
	req.date = 20210611;
	req.begin_time = 93000000;
	req.end_time = 170000000;

	int32_t ec;
	if ((ec = galaxy::tgw::IGMDApi::QueryTickExecution(spi, req)) != galaxy::tgw::ErrorCode::kSuccess)
		std::cout << "QueryTickExec faild, ErrorCode: " << ec << std::endl;
}

void QueryTickOrderReq(galaxy::tgw::QueryTickOrderSpi* spi)
{
	galaxy::tgw::ReqDefault req;
	memset(&req, 0, sizeof(req));
	strncpy(req.security_code, "000001", sizeof(req.security_code));
	req.market_type = 102;
	req.date = 20210611;
	req.begin_time = 93000000;
	req.end_time = 170000000;

	int32_t ec;
	if ((ec = galaxy::tgw::IGMDApi::QueryTickOrder(spi, req)) != galaxy::tgw::ErrorCode::kSuccess)
		std::cout << "QueryTickOrder faild, ErrorCode: " << ec << std::endl;
}

void QueryOrderQueueReq(galaxy::tgw::QueryOrderQueueSpi* spi)
{
	galaxy::tgw::ReqDefault req;
	memset(&req, 0, sizeof(req));
	strncpy(req.security_code, "000001", sizeof(req.security_code));
	req.market_type = 102;
	req.date = 20210611;
	req.begin_time = 93000000;
	req.end_time = 170000000;

	int32_t ec;
	if ((ec = galaxy::tgw::IGMDApi::QueryOrderQueue(spi, req)) != galaxy::tgw::ErrorCode::kSuccess)
		std::cout << "QueryOrderQueue faild, ErrorCode: " << ec << std::endl;
}

void QueryCodeTableReq(galaxy::tgw::QueryCodeTableSpi* spi)
{
	int32_t ec;
	if ((ec = galaxy::tgw::IGMDApi::QueryCodeTable(spi)) != galaxy::tgw::ErrorCode::kSuccess)
		std::cout << "QueryCodeTable faild, ErrorCode: " << ec << std::endl;
}

void QuerySecuritiesInfoReq(galaxy::tgw::QuerySecuritiesInfoSpi* spi)
{
	galaxy::tgw::SubCodeTableItem req_codelist;
```

```
    memset(&req_codelist, 0, sizeof(req_codelist));
    strncpy(req_codelist.security_code, "000001", sizeof(req_codelist.security_code));
    req_codelist.market = 102;

    int32_t ec;
    if ((ec = galaxy::tgw::IGMDApi::QuerySecuritiesInfo(spi, &req_codelist, 1)) != galaxy::tgw::ErrorCode::kSuccess)
        std::cout << "QuerySecuritiesInfo faild, ErrorCode: " << ec << std::endl;
}

void QueryExFactorReq(galaxy::tgw::QueryExFactorSpi* spi)
{
    int32_t ec;
    if ((ec = galaxy::tgw::IGMDApi::QueryExFactorTable(spi, "000001")) != galaxy::tgw::ErrorCode::kSuccess)
        std::cout << "QueryExFactorTable faild, ErrorCode: " << ec << std::endl;
}

void QueryFactorReq(galaxy::tgw::QueryFactorSpi* spi)
{
    galaxy::tgw::ReqFactor req_factor;
    memset(&req_factor, 0, sizeof(req_factor));
    strncpy(req_factor.factor_type, "factor_type", sizeof(req_factor.factor_type));
    strncpy(req_factor.factor_sub_type, "factor_sub_type", sizeof(req_factor.factor_sub_type));
    strncpy(req_factor.factor_name, "factor_name", sizeof(req_factor.factor_name));
    req_factor.begin_date = 20210611;
    req_factor.end_date = 20210611;
    req_factor.begin_time = 93000000;
    req_factor.end_time = 170000000;

    int32_t ec;
    if ((ec = galaxy::tgw::IGMDApi::QueryFactor(spi, req_factor)) != galaxy::tgw::ErrorCode::kSuccess)
        std::cout << "QueryFactor faild, ErrorCode: " << ec << std::endl;
}

void QueryThirdInfoReq(galaxy::tgw::QueryThirdInfoSpi* spi)
{
    auto task_id = galaxy::tgw::IGMDApi::GetTaskID();
    galaxy::tgw::IGMDApi::SetThirdInfoParam(task_id, "function_id", "01");
    galaxy::tgw::IGMDApi::SetThirdInfoParam(task_id, "WD_CODE", "0000001");
    galaxy::tgw::IGMDApi::SetThirdInfoParam(task_id, "trade_data", "20210101");

    int32_t ec;
    if ((ec = galaxy::tgw::IGMDApi::QueryThirdInfo(spi, task_id)) != galaxy::tgw::ErrorCode::kSuccess)
        std::cout << "QueryThirdInfo faild, ErrorCode: " << ec << std::endl;
}

void ReplaySnapshotReq(galaxy::tgw::ReplayHistorySpi* spi)
{
    sleep(3);    // 延迟 3s 至登录成功
    galaxy::tgw::ReqHistoryItem history_item[2];
    history_item[0].market = 102;
    strcpy(history_item[0].security_code, "000001");
    history_item[1].market = 102;
    strcpy(history_item[1].security_code, "000002");

    galaxy::tgw::ReqReplay req_replay;
    req_replay.begin_date = 20211228;
    req_replay.end_date = 20211228;
    req_replay.begin_time = 91500000;
    req_replay.end_time = 103100000;
    req_replay.req_item_cnt = 2;
    req_replay.req_items = history_item;
    req_replay.md_data_type = galaxy::tgw::MDDatatype::kSnapshot;

    int32_t ec;
    if ((ec = galaxy::tgw::IGMDApi::ReplayRequest(spi, req_replay)) != galaxy::tgw::ErrorCode::kSuccess)
        std::cout << "ReplaySnapshot faild, ErrorCode: " << ec << std::endl;
}

void ReplayTickExecReq(galaxy::tgw::ReplayHistorySpi* spi)
{
    sleep(3);    // 延迟 3s 至登录成功
    galaxy::tgw::ReqHistoryItem history_item[2];
    history_item[0].market = 102;
    strcpy(history_item[0].security_code, "000001");
    history_item[1].market = 102;
    strcpy(history_item[1].security_code, "000002");

    galaxy::tgw::ReqReplay req_replay;
    req_replay.begin_date = 20211228;
    req_replay.end_date = 20211228;
```

```
    req_replay.begin_time = 91500000;
    req_replay.end_time = 103100000;
    req_replay.req_item_cnt = 2;
    req_replay.req_items = history_item;

    req_replay.md_data_type = galaxy::tgw::MDDatatype::kTickExecution;

    int32_t ec;
    if ((ec = galaxy::tgw::IGMDApi::ReplayRequest(spi, req_replay)) != galaxy::tgw::ErrorCode::kSuccess)
        std::cout << "ReplayTickExec faild, ErrorCode: " << ec << std::endl;
}

void ReplayKlineReq(galaxy::tgw::ReplayHistorySpi* spi)
{
    sleep(3);    // 延迟 3s 至登录成功
    galaxy::tgw::ReqHistoryItem history_item[2];
    history_item[0].market = 102;
    strcpy(history_item[0].security_code, "000001");
    history_item[1].market = 102;
    strcpy(history_item[1].security_code, "000002");

    galaxy::tgw::ReqReplayKline req_k;
    memset(&req_k, 0, sizeof(req_k));
    req_k.begin_date = 20211228;
    req_k.begin_time = 930;
    req_k.cq_flag = 0;
    req_k.cyc_type = galaxy::tgw::MDDatatype::k1KLine; // 1 分钟 k
    req_k.auto_complete = 1;
    req_k.end_date = 20211228;
    req_k.end_time = 1032;
    req_k.task_id = galaxy::tgw::IGMDApi::GetTaskID();
    req_k.req_item_cnt = 2;
    req_k.req_items = history_item;

    int32_t ec;
    if ((ec = galaxy::tgw::IGMDApi::ReplayKline(spi, req_k)) != galaxy::tgw::ErrorCode::kSuccess)
        std::cout << "ReplayKline faild, ErrorCode: " << ec << std::endl;
}


int main()
{
    // 准备配置
    galaxy::tgw::Cfg cfg;
    strncpy(cfg.server_vip, "127.0.0.1", sizeof(cfg.server_vip));
    cfg.server_port = 9200;
    strncpy(cfg.username, "amd", sizeof(cfg.username));
    strncpy(cfg.password, "123456", sizeof(cfg.password));

    uint16_t api_mode = galaxy::tgw::ApiMode::kColocationMode;   // 设置 api 模式 托管机房模式/互联网模式
    if (api_mode == galaxy::tgw::ApiMode::kColocationMode)
    {
        cfg.coloca_cfg.channel_mode = galaxy::tgw::ColocatChannelMode::kQTCP;
        cfg.coloca_cfg.qtcp_channel_thread = 8;
        cfg.coloca_cfg.qtcp_max_req_cnt = 1000;
    }

    galaxy::tgw::TgwPushSpi spi;
    bool is_runing = true;
    // 初始化 TGW
    int32_t ec = galaxy::tgw::ErrorCode::kSuccess;
    if ((ec = galaxy::tgw::IGMDApi::Init(&spi, cfg, api_mode))
        != galaxy::tgw::ErrorCode::kSuccess)
    {
        std::cout << "Init TGW faild, ErrorCode: " << ec << std::endl;
        galaxy::tgw::IGMDApi::Release();
        return -1;
    }
    // 修改密码
    // galaxy::tgw::UpdatePassWordReq req;
    // memset(&req, 0, sizeof(req));
    // strncpy(req.username, "amd", sizeof(req.username));
    // strncpy(req.old_password, "123456", sizeof(req.old_password));
    // strncpy(req.new_password, "123", sizeof(req.new_password));
    // ec = galaxy::tgw::IGMDApi::UpdatePassWord(req);
    // if (ec != galaxy::tgw::ErrorCode::kSuccess)
    //     std::cout << "UpdatePassWord faild, ErrorCode: " << ec << std::endl;
    // else
    //     std::cout << "UpdatePassWord success" << std::endl;
```

```
/*====================================推送示例====================================*/
// //1、常规订阅示例
// GeneralSubscribeReq();
// //2、加工因子订阅示例
// FactorSubscribeReq();

/*====================================查询示例====================================*/
// //1、查询 K 线
// galaxy::tgw::QueryKlineSpi* kline_spi = new galaxy::tgw::QueryKlineSpi();
// QueryKlineReq(kline_spi);

// //2、查询快照
// galaxy::tgw::QuerySnapshotSpi* snapshot_spi = new galaxy::tgw::QuerySnapshotSpi();
// QuerySnapshotReq(snapshot_spi);

// //3、查询逐笔成交
// galaxy::tgw::QueryTickExecutionSpi* tick_exec_spi = new galaxy::tgw::QueryTickExecutionSpi();
// QueryTickExecReq(tick_exec_spi);

// //4、查询逐笔委托
// galaxy::tgw::QueryTickOrderSpi* tick_order_spi = new galaxy::tgw::QueryTickOrderSpi();
// QueryTickOrderReq(tick_order_spi);

// //5、查询委托队列
// galaxy::tgw::QueryOrderQueueSpi* order_queue_spi = new galaxy::tgw::QueryOrderQueueSpi();
// QueryOrderQueueReq(order_queue_spi);

// //6、查询代码表
// galaxy::tgw::QueryCodeTableSpi* code_table_spi = new galaxy::tgw::QueryCodeTableSpi();
// QueryCodeTableReq(code_table_spi);

// //7、查询证券代码基本信息
// galaxy::tgw::QuerySecuritiesInfoSpi* secur_info_spi = new galaxy::tgw::QuerySecuritiesInfoSpi();
// QuerySecuritiesInfoReq(secur_info_spi);

// //8、查询复权因子
// galaxy::tgw::QueryExFactorSpi* ex_fator_spi = new galaxy::tgw::QueryExFactorSpi();
// QueryExFactorReq(ex_fator_spi);

// //9、查询加工因子
// galaxy::tgw::QueryFactorSpi* factor_spi = new galaxy::tgw::QueryFactorSpi();
// QueryFactorReq(factor_spi);

// //10、查询金融资讯数据
// galaxy::tgw::QueryThirdInfoSpi* thirdinfo_spi = new galaxy::tgw::QueryThirdInfoSpi();
// QueryThirdInfoReq(thirdinfo_spi);

/*====================================回放示例====================================*/
// //1、回放快照
// galaxy::tgw::ReplayHistorySpi* hSpi = new galaxy::tgw::ReplayHistorySpi();
// ReplaySnapshotReq(hSpi);

// //2、回放逐笔成交
// ReplayTickExecReq(hSpi);

// //3、回放 k 线
// ReplayKlineReq(hSpi);

// 这里必须保证用户的 spi 实例的生命周期长于 TGW 的 API
while (is_runing)
{
    sleep(3);
    std::cout << "TGW is running ..." <<std::endl;
}

// 释放申请的对象（需保证在数据全部查询/回放完毕后，才可释放）
{
    // delete kline_spi;
    // delete snapshot_spi;
    // delete tick_exec_spi;
    // delete tick_order_spi;
    // delete order_queue_spi;
    // delete code_table_spi;
    // delete internet_stock_info_spi;
    // delete coloc_stock_info_spi;
    // delete ex_fator_spi;
    // delete factor_spi;
    // delete thirdinfo_spi;
```

```
        // delete hSpi;
    }
    // 释放资源
    galaxy::tgw::IGMDApi::Release();
    return 0;
}
```

3.7 编译运行步骤

- Linux 系统下 TGW 使用前需要运行根目录下 tgw_install.sh 脚本设置环境变量
- Linux 系统下 TGW 已经提供了 Makefile 编译文件，用户可选择如下方案编译运行

  方案 1：(linux)

  a) 进入 c++/demo 所在目录，直接运行 make 命令。
  b) 再次运行 make install 命令，将 tgw_demo 可执行程序安装到 bin 目录下。
  c) 再次执行 make clean 命令，可直接清除编译过程中产生的.o 文件和其他不需要的文件。
  d) 接下来可以运行，直接执行如下命令：
     LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./lib ./bin/tgw _demo

  方案 2：(linux)

  a) 已经提供了 make.sh 脚本，其中包含了方案 1 的所有操作步骤。
  b) 直接运行./make.sh 命令即可完成编译运行所有步骤。
- Windows 系统下 TGW 使用前需要运行根目录下 tgw_install.bat 脚本设置环境变量
- Windows 系统下 TGW 已经提供了 VC++2017 工程文件，用户可在 visual studio 开发环境中打开对应解决方案文件并编译运行。

  编译步骤：(windows)

  a) 进入到 c++目录：cd AMD_tgw_msvc-14.0_Windows_*/c++/。
  b) 进入 project 目录(visual studio 工程)。
  c) 打开工程即可,demo 程序源文件需要根据实际情况填写自行修改。
  d) 修改完成后即可按键盘 F5 直接编译运行。

3.8 开发注意事项

1) 初始化成功则可执行订阅或者取消订阅操作或进行查询，初始化失败则调用 Release 函数释放资源后退出。
2) 订阅时调用接口 Subscribe 或 SubFactor，若账号不限制订阅数则可以填写成订阅所有市场、类型、证券代码。若有限制则只能单市场，单证券代码，数据类型不受限制)，需要连续订阅多只，则可参照 demo 示例，使用数组入参的方式。
3) 订阅成功后，对应数据将回调到提供的接口，如 OnMDSnapshot()方法将接收到现货快照数据，具体数据类型对应不同接口，用户可选择实现或者不处理，在完成对应处理后，必须调用 FreeMemory()方法释放传入的指针内存，否则将造成内存泄漏。
4) 取消订阅的流程、规则，与订阅流程、规则类似，取消后将不再接收该类型的数据，取消即时生效但数据为异步回调，取消前已收到的数据仍然会接收。

5) 查询接口使用时请先判断返回错误码，若不为成功则出参无数据，若成功则根据 QueryData 中数据类型进行指定处理（即将指针转成对应的结构体进行解析处理，查询和推送结构相同），使用后需要对 QueryData 中的 msg 指针调用 FreeMemory() 方法释放内存，以避免内存泄漏
6) 调用查询接口时有最大并发量限制，超过最大并发量则等待部分任务完成，未超过则发起查询请求等待结果返回。
7) 登录和初始化不冲突，初始化成功并不一定登陆成功，登录失败将有对应错误日志提示，用户可参照 tgw_datatype.h 文件中的错误码，排查对应原因。
8) 编译运行当前 demo 没有问题，但是如果用户移动了默认的目录层级或者自行引入了开发所需的其他三方库，可能造成编译运行的问题，用户需自行解决。
9) 如果运行报证书认证失败，请运行根目录下设置环境变量脚本 tgw_install。

3.9 演示示例（获取和保存数据）

SDK 安装包分为 Windows 和 Linux 两种版本，请分别下载。SDK 提供 tgw_test 标准演示程序，该程序支持动态订阅和取消订阅操作：在终端与 tgw_test 监听地址连接后，终端按照指定格式完成输入订阅或退订命令，从而获取实时行情数据，并将数据通过所配置的路径，落地成 csv 文件供查看，以下是程序的命令参数说明和示例。

tgw_test 程序用于指引用户了解 API 工作流程和使用步骤，也供开发人员以及测试人员进行功能验证和测试分析。

3.9.1 修改参数

etc/tgw.json 关键参数更改

```
"ApiMode":1,
"CsvFileDir": "./",
"CommandMode": 1,
"DynamicSubscribeIp": "127.0.0.1",
"DynamicSubscribePort":8017,
"ReqFile":"./req_file.txt",
"ServerVip": "127.0.0.1",
"ServerPort": 9200,
"UserName" : "amd",
"Password" : "amd@2022",
"ColocChannelMode": ["TCP"],
"ColocQTcpReqTimeOut": 1,
"ColocQTcpChannelThread": 2,
"ColocQTcpMaxReqCnt": 100,
"MinLogLevel": 2
```

- ApiMode 功能模式，1 托管机房模式，2 互联网模式
- CsvFileDir 为落地 csv 文件路径。
- CommandMode 为命令输入方式，1 动态输入命令，2 文件读取命令
- DynamicSubscribeIp 动态订阅时的 ip 地址，仅可为本机 ip
- DynamicSubscribePort 动态订阅的端口，默认 8019，可改动
- ReqFile 为查询命令文件路径及文件，格式遵循 nc 动态查询的格式
- ServerVip 连接的服务地址
- ServerPort 连接的服务端口，默认 8600。
- UserName 用户名
- Password 密码
- ColocChannelMode 托管机房模式下的通道模式
- ColocQTcpReqTimeOut 托管机房模式查询通道请求超时时间
- ColocQTcpChannelThread 托管机房模式下通道线程数
- ColocQTcpMaxReqCnt 托管机房模式查询通道查询任务最大请求数
- MinLogLevel 输出日志等级，默认 info=2

3.9.2 运行启动脚本

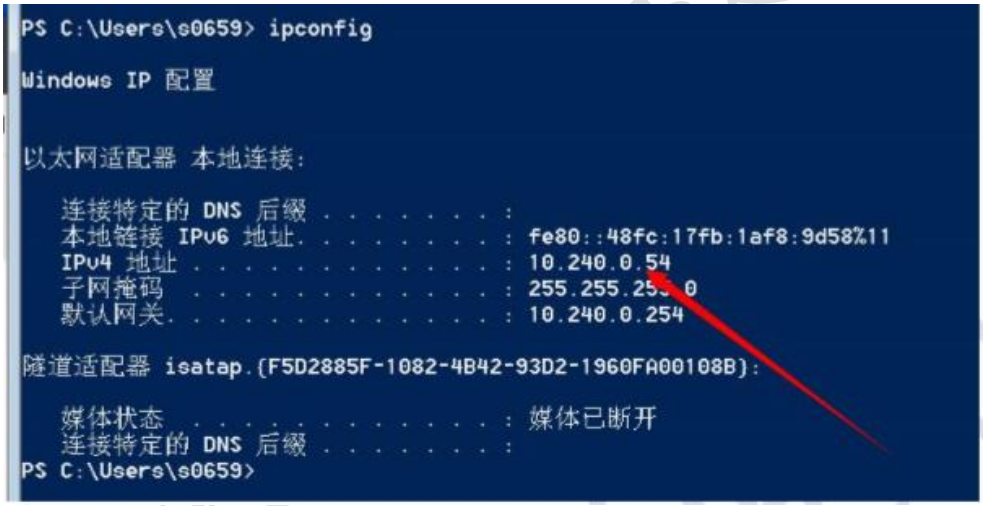
- Linux 版本 SDK 运行前需要先运行根目录下 tgw_install.sh 脚本设置环境变量；之后对应启动位于 SDK 根目录 c++/test_tool 下的 run.sh 脚本。终端中输入./run.sh 启动运行。
- Windows 版本 SDK 运行前需要先运行根目录下 tgw_install.bat 脚本设置环境变量；之后对应启位于 SDK 根目录 c++/test_tool 下的 run.bat 脚本。cmd 命令框中输入 run.bat 启动运行。

3.9.3 开启订阅命令输入终端

Linux 版本下，另开一个终端，通过 nc 连接，然后，输入对应的测试示例，逐条输入，回车进入下一条命令即可完成对应操作。

```
(base) [root@localhost bin]# nc 127.0.0.1 8039
Sub All All All
```

Windows 版本下，另开一个 cmd 命令输入框，输入 ipconfig，找到 IPv4 地址。



通过 telnet 连接，回车即可进行连接。然后，输入对应的测试示例，逐条输入，回车进入下一条命令即可完成对应操作（此时命令输入不可见， 最好由其他地方提前准备，之后右键复制进去），

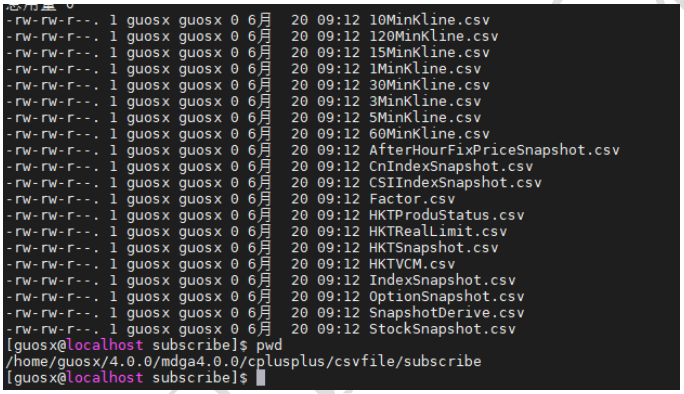
如果提示 telnet 无法识别，

则进入计算机找到该选项进入，

选择下方两个选项，点击确定即可启用。

3.9.4 查看订阅数据 csv 文件

在设置的 csv 路径下自动生成“subscribe”、“query/查询日期”两个子目录用于存放订阅和查询接收到的行情数据文件，tgw_test 按照数据类型生成单独的 csv 文件。



3.9.5 tgw_test 程序订阅命令说明

1) 查询命令示例：

- QuerySnapshot ${市场} ${证券代码} ${日期} ${开始时间} ${结束时间}（查询快照数据）

  示例：QuerySnapshot SSE 000002 20200924 90000000 93500000

  说明：

  20200924：表示 2020 年 9 月 24 日

  93500000：表示 9 点 35 分 00 秒 000 毫秒

- QueryTickOrder ${市场} ${证券代码} ${日期} ${开始时间} ${结束时间}（查询逐笔委托数据）

  示例：QueryTickOrder SZSE 000002 20211229 90000000 95000000

- QueryTickExecution ${市场} ${证券代码} ${日期} ${开始时间} ${结束时间}（查询逐笔成交数据）

示例：QueryTickExecution SZSE 000002 20211229 90000000 95000000

- QueryKline ${市场} ${k 线类型} ${证券代码} ${复权类型} ${自动补全} ${开始日期} ${结束日期} ${开始时间} ${结束时间}（查询 K 线数据）

  k 线类型：MD1MinKline、MD3MinKline、MD5MinKline、MD10MinKline、MD15MinKline 、 MD30MinKline 、 MD60MinKline 、 MD120MinKline 、 MDDayKline 、MDWeekKline、MDMonthKline、MDSeasonKline、MDYearKline

  复权：0 1 2 (0:不复权;1:向前复权;2:向后复权)

  自动补全：0:不补齐，1：补齐 默 1

  示例 1： QueryKline SSE MD1MinKline 600004 0 1 20211229 20211229 900 1500

- QueryOrderQueue ${市场} ${证券代码} ${日期} ${开始时间} ${结束时间}（查询委托队列数据）

  示例：QueryOrderQueue SZSE 000002 20211229 90000000 95000000

- QuerySecurInfo ${市场类型} ${证券代码} （查询证券个股基本信息数据）

  示例：QuerySecurInfo SZSE 000002

- QueryCodeTable codeTable（查询证券代码表数据）

  示例：QueryCodeTable codeTable

- QueryExFactor ${证券代码} （查询复权因子表信息数据）

  示例： QueryExFactor 000002

- QueryFactor ${因子类型} ${因子子类型} ${因子名称} ${开始日期} ${结束日期} ${开始时间} ${结束时间}（查询因子数据）

  示例：QueryFactor A B C 20211229 20211229 90000000 95000000 备注：开始日期和结束日期需要相等，目前不支持多天查询。

2) 订阅命令类型：Sub(订阅)
   UnSub(取消订阅)
   SubFactor(订阅因子类型)
   UnSubFactor(取消因子类型订阅)

格式： Sub ${市场类型} ${数据类型} ${品种类别} ${证券代码}
UnSub ${市场类型} ${数据类型} ${品种类别} ${证券代码}

- 市场类型：All（全市场），NEEQ（北京所），SHFE（上期所），CFFEX（中金所），DCE（大商所），CZCE（郑商所），INE（上海国际能源交易中心），SSE（上交所），SZSE（深交所）。
- 数据类型：（多个数据类型之间用逗号“,”分隔）All 包含所有数据类型，类型有 MDSnapshot，MD1MinKline，MDCnIndexSnapshot 等。
- 品种类别： All 包含所有类别，类别有 Stock，Fund，Bond，Option，Index，HKT，FutureOption 等。
- 证券代码：支持已有的任意代码
- 订阅命令示例：（多个数据类型之间用逗号“,”分隔）

  示例：Sub SZSE MDSnapshot Stock 000002

  Sub SZSE MD1MinKline All 000002

  Sub All All All All

- 取消订阅命令示例: （多个数据类型之间用逗号“,”分隔）

  示例： UnSub SZSE MDSnapshot Bond 000002

  UnSub All All All All

- 因子类型订阅示例：

SubFactor all all all

SubFactor Quantitative_Factor MA MA_S

说明：

Quantitative_Factor：因子父类型 MA：因子子类型 MA_S：因子名称

支持的全订阅方式为以下三种：

- *全订阅支持 all all all 方式;*
- *支持 xxx all all 方式订阅（指定父类型）*
- *支持 xxx xxx all 方式订阅（指定父类型、子类型）;*

不支持以下的方式：

- *父类型不支持单独全订阅，即 all xxx xxx;*
- *因子子类型不支持单独全订阅,即 xxx all xxx*
- *不支持父类型和子类型全订阅，因子名称为指定因子，即 all lall xxx;*
- *不支持父类型和因子名称全订阅，子类型为指定类型，即 all xxx all;*

- 取消因子类型订阅示例:

示例： UnSubFactor all all all

UnSubFactor Quantitative_Factor MA MA_S

说明：

Quantitative_Factor：因子父类型 MA：因子子类型 MA_S：因子名称

支持的取消因子全订阅方式为以下三种：

- *取消全订阅支持 allall all 方式;*
- *支持 xxx all all 方式取消订阅（指定父类型）;*
- *支持 xxx xxx all 方式取消订阅（指定父类型、子类型）;*

不支持以下的方式：

- *父类型不支持单独取消全订阅，即 all xxxx xx;*
- *因子子类型不支持单独取消全订阅,即 xxx all xxx;*
- *不支持父类型和子类型取消全订阅，因子名称为指定因子，即 all all xxx;*
- *不支持父类型和因子名称取消全订阅，子类型为指定类型，即 all xxx all;*

3) 回放命令类型：${命令类型} ${证券代码.市场,证券代码.市场...} ${开始日期} ${结束日期}${开始日期} ${结束日期}

- ReplayTickExecution${证券代码.市场}${订阅日期}${开始日期}${结束日期}${开始时间}${结束时间}**（逐笔成交回放获取命令）**

示例：ReplayTickExecution 000001.SZSE,600381.SSE 20200924 20200924 093000000 093500000

说明：

20200924： 表示 2020 年 9 月 24 日

093500000：表示 9 点 35 分 00 秒 000 毫秒

备注：支持多天回放

- ReplaySnapshot${证券代码.市场,证券代码.市场..}${开始日期}${结束日期}${开始 时间}${结束时间}**（历史快照回放获取命令）**

示 例 ： ReplaySnapshot000001.SZSE,600381.SSE 20211228 20211228 093000000103600000

备注：支持多天回放。

- ReplayKLine ${k 线类型} ${证券代码.市场,证券代码.市场...} ${复权类型} ${自动补全} ${开始日期} ${结束日期} ${开始时间} ${结束时间}**（历史 K 线回放获取命令）**

示例 1： 分钟 k： ReplayKLine MD1MinKline 000001.SZSE,000002.SZSE 0 1 20201118 20201118 0930 11300

k 线类型：可参考查询命令

复权：0 1 2 (0:不复权;1:向前复权;2:向后复权)

自动补全：0:不补齐，1：补齐 默 1

4. 公共数据字典

所有公共数据字典的的定义都在 C++/include/tgw_datatype.h 下面定义。

4.1 事件级别(EventLevel)

事件级别	数值	说明
kInfo	枚举(=1)	普通事件
kWarn	枚举(=2)	告警事件
kError	枚举(=3)	错误事件，比较严重,需要介入处理

4.2 错误码(ErrorCode)

错误码	数值	说明
kFailure	枚举(-100)	失败
kUnInited	枚举(-99)	未初始化
kNullSpi	枚举(-98)	空指针
kParamIllegal	枚举(-97)	参数非法
kNetError	枚举(-96)	网络异常
kPermissionError	枚举(-95)	数据无权限
kLogonFailed	枚举(-94)	未登录
kAllocateMemoryFailed	枚举(-93)	分配内存失败
kChannelError	枚举(-92)	通道错误
kOverLoad	枚举(-91)	hqs 任务队列溢出
kLogoned	枚举(-90)	账号已登录
kHqsError	枚举(-89)	HQS 系统错误
kNonQueryTimePeriod	枚举(-88)	非查询时间段(非查询时间段不支持查询)
kDbAndCodeTableNoCode	枚举(-87)	数据库和代码表中没有指定的代码
kIllegalMode	枚举(-86)	api 模式非法
kThreadBusy	枚举(-85)	超过最大可用线程资源
kParseDataError	枚举(-84)	数据解析出错
kTimeout	枚举(-83)	获取数据超时
kFlowOverLimit	枚举(-82)	周流量耗尽
kCodeTableCacheNotAvailable	枚举(-81)	代码表缓存不可用
kOverMaxSubLimit	枚举(-80)	超过最大订阅限制

kLostConnection	枚举(-79)	丢失连接
kOverMaxQueryLimit	枚举(-78)	超过最大查询数（含代码表）
kFunctionIdNull	枚举(-77)	金融资讯数据查询未设置功能号
kDataEmpty	枚举(-76)	数据为空
kUserNotExist	枚举(-75)	用户不存在
kVerifyFailure	枚举(-74)	账号/密码错误
kApiInterfaceUsing	枚举(-73)	api 接口不能同时多次调用
kSuccess	枚举(0)	成功

4.3 数据定义长度(ConstField)

数据类型	数据名称	数值	说明
static const uint32_t	kIPMaxLen	24	IP 地址的最大长度
static const uint32_t	kUMSItemLen	10	服务项信息的最大个数
static const uint32_t	kUsernameLen	32	用户名的最大长度
static const uint32_t	kPasswordLen	64	用户名密码的最大长度
static const uint32_t	kSecurityCodeLen	16	证券代码最大长度
static const uint32_t	kHistorySecurityCodeLen	32	历史回放和查询证券代码最大长度
static const uint32_t	kPushSecurityCodeLen	32	实时推送证券代码最大长度
static const uint32_t	kSecurityNameLen	32	证券名称最大长度
static const uint32_t	kTradingPhaseCodeLen	8	交易状态标志
static const uint32_t	kExChangeInstIDLen	31	合约在交易所代码
static const uint32_t	kPositionLevelLen	10	行情档位
static const uint32_t	kMDStreamIDLen	6	行情类别
static const uint32_t	kSecurityStatusLen	24	证券状态最大长度
static const uint32_t	kSecuritySymbolLen	32	证券简称最大长度
static const uint32_t	kSecurityEnglishNameLen	128	证券英文简称最大长度
static const uint32_t	kSecurityTypeLen	10	证券类别长度
static const uint32_t	kCurrencyLen	4	币种长度
static const uint32_t	kVolumeTradeLen	24	成交量长度
static const uint32_t	kValueTradeLen	24	交易额长度
static const uint32_t	kFutureSecurityCodeLen	32	期货代码最大长度
static const uint32_t	kMDStreamIDMaxLen	6	行情类别字段最大长度
static const uint32_t	kDataPermission	128	数据权限最大长度
static const uint32_t	kTokenMaxLen	128	token 的最大长度
static const uint32_t	kQuerySecurityCodeLen	38	查询状态回调代码最大长度
static const uint32_t	kTradingStatusLen	8	证券交易状态档位
static const uint32_t	kSymbolLen	128	证券简称最大长度（包括中文简称）
static const uint32_t	kSecurityAbbreviationLen	64	证券简称最大长度

static const uint32_t	kMaxTypesLen	16	证券类型最大长度
static const uint32_t	kTypesLen	8	
static const uint32_t	kCodeTableSecurityStatusMaxLen	16	代码表证券状态字符最大长度
static const uint32_t	RegularShare	9	对应回购标准券长度

4.4 回放请求任务状态(HistoryTaskStatus)

数据类型	数据名称	数值	说明
static const uint8_t	kSuccess	0	任务成功完成
static const uint8_t	kFailed	1	任务执行失败
static const uint8_t	kTaskCancel	2	任务取消
static const uint8_t	kTaskWaiting	3	任务执行中
static const uint8_t	kTaskTimeOut	4	回放任务超时

4.5 数据类型(MDDatatype)

类型名	数值	说明
k1KLine	枚举(=10000)	1 分钟 K 线
k3KLine	枚举(=10001)	3 分钟 K 线
k5KLine	枚举(=10002)	5 分钟 K 线
k10KLine	枚举(=10003)	10 分钟 K 线
k15KLine	枚举(=10004)	15 分钟 K 线
k30KLine	枚举(=10005)	30 分钟 K 线
k60KLine	枚举(=10006)	60 分钟 K 线
k120KLine	枚举(=10007)	120 分钟 K 线
kDayKline	枚举(=10008)	日 K 线
kWeekKline	枚举(=10009)	周 K 线
kMonthKline	枚举(=10010)	月 K 线
kSeasonKline	枚举(=10011)	季 K 线
kYearKline	枚举(=10012)	年 K 线
kTickExecution	枚举(=10013)	逐笔成交
kSnapshot	枚举(=10014)	快照

4.6 日志输出级别(LogLevel)

日志输出级别	数值	说明
kTrace	枚举(=0)	跟踪级别日志
kDebug	枚举(=1)	调试级别日志

kInfo	枚举(=2)	普通级别日志
kWarn	枚举(=3)	警告级别日志
kError	枚举(=4)	错误级别日志
kFatal	枚举(=5)	致命级别日志

4.7 市场类型定义(MarketType)

市场类型	数值	说明
kNone	枚举(=0)	表示全市场
kNEEQ	枚举(=2)	北交所
kSHFE	枚举(=3)	上期所
kCFFEX	枚举(=4)	中金所
kDCE	枚举(=5)	大商所
kCZCE	枚举(=6)	郑商所
kINE	枚举(=7)	上海国际能源交易中心
kSSE	枚举(=101)	上交所
kSZSE	枚举(=102)	深交所
kHKEx	枚举(=103)	港交所
kBK	枚举(=201)	板块
kMax	枚举(=255)	市场类型最大值

4.8 API 通道模式(ApiMode)

数据类型	数据名称	数值	说明
static const uint16_t	kColocationMode	1	托管机房模式
static const uint16_t	kInternetMode	2	互联网模式

4.9 托管机房模式通道(ColocatChannelMode)

数据类型	数据名称	数值	说明
static const uint64_t	kTCP	0x0000000200000000	TCP 推送获取数据
static const uint64_t	kQTCP	0x0000000800000000	TCP 查询获取数据
static const uint64_t	kRTCP	0x0000001000000000	TCP 回放获取数据

4.10订阅数据类型(SubscribeDataType)

数据类型	数据名称	数值	说明	适用模式

static const uint32_t	kNone	0	订阅全部数据	托管机房模式 互联网模式
static const uint32_t	k1MinKline	1	订阅 1 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	k3MinKline	2	订阅 3 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	k5MinKline	3	订阅 5 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	k10MinKline	4	订阅 10 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	k15MinKline	5	订阅 15 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	k30MinKline	6	订阅 30 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	k60MinKline	7	订阅 60 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	k120MinKline	8	订阅 120 分钟 k 线数据	托管机房模式 互联网模式
static const uint32_t	kSnapshotDerive	9	订阅快照衍生数据	托管机房模式 互联网模式
static const uint32_t	kSnapshot	10	订阅现货快照数据	互联网模式
static const uint32_t	kOptionSnapshot	11	订阅期权快照数据	互联网模式
static const uint32_t	kHKTSnapshot	12	订阅港股快照数据	互联网模式
static const uint32_t	kIndexSnapshot	13	订阅指数快照数据	互联网模式
static const uint32_t	kAfterHourFixedPriceSnapshot	14	订阅盘后快照数据	互联网模式
static const uint32_t	kCSIIndexSnapshot	15	订阅中证指数数据	互联网模式
static const uint32_t	kCnIndexSnapshot	16	订阅国证指数快照数据	互联网模式
static const uint32_t	kHKTRealtimeLimit	17	订阅港股通实时额度数据	互联网模式
static const uint32_t	kHKTProductStatus	18	订阅港股通产品状态数据	互联网模式
static const uint32_t	kHKTVCM	19	订阅港股 VCM 数据	互联网模式
static const uint32_t	kFutureSnapshot	20	订阅期货数据	互联网模式

4.11品种类型定义(VarietyCategory)

数据类型	数据名称	数值	说明
static const uint8_t	kNone	0	None
static const uint8_t	kStock	1	股票
static const uint8_t	kFund	2	基金
static const uint8_t	kBond	3	债券
static const uint8_t	kOption	4	期权
static const uint8_t	kIndex	5	指数

static const uint8_t	kHKT	6	港股通
static const uint8_t	kFutureOption	7	期货期权
static const uint8_t	kCFETSRMB	8	银行间本币交易产品
static const uint8_t	kHKEx	9	港股
static const uint8_t	kOthers	255	其他

5. 行情数据字典

所有行情数据字典的的定义都在 **C++/include/tgw_struct.h** 下面定义。

5.1 K 线(MDKLine)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kPushSecurityCodeLen]	证券代码
int64_t	orig_time	时间（YYYYMMDDHHMMSSsss)
int64_t	kline_time	k 线时间
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	close_price	最新价，实际值需除以 1000000
int64_t	volume_trade	成交量，无倍数放大
int64_t	value_trade	成交金额，无倍数放大

5.2 现货衍生(MDSnapshotDerive)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	证券代码
int64_t	orig_time	时间（为 YYYYMMDDHHMMSSsss）
uint64_t	average_price	均价，实际值需除以 1000000
uint64_t	circulation_value	流通市值，实际值需除以 1000000
uint64_t	total_value	总市值，实际值需除以 1000000
uint64_t	initiative_buy_volume	外盘，实际值需除以 100
uint64_t	initiative_sell_volume	内盘，实际值需除以 100
uint32_t	turnover_rate	换手率，实际值需除以 100000
int32_t	volume_ratio	量比，实际值需除以 100000
int32_t	ask_bid_ratio	委比，实际值需除以 100000
uint32_t	amplitude	振幅，实际值需除以 100000
int32_t	PE_static	静态市盈率，实际值需除以 100
int32_t	PE_dynamic	动态市盈率，实际值需除以 100
int32_t	PE_TTM	最近 4 季度滚动市盈率，实际值需除以 100

int32_t	PB	市净率，实际值需除以 100
int64_t	entrustment_diff	委差
char	initiative_flag	内外盘标记(B/S)

5.3 加工因子(Factor)

数据类型	字段名称	说明
uint32_t	data_size	json 数据的大小
char*	json_buf	json 结构

5.4 L1 现货快照(MDSnapshotL1)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	证券代码
uint8_t	variety_category	品种类别（参照描述）
int64_t	orig_time	交易所行情数据时间（YYYYMMDDHHMMSSsss）
char	trading_phase_code[ConstField::kTradingPhaseCodeLen]	交易阶段代码
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（北交所为 0）
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	IOPV	IOPV 净值估产（仅基金品种有效），实际值需除以 1000000，北交所为 0
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

5.5 指数快照(MDIndexSnapshot)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	证券代码
int64_t	orig_time	交易所行情数据时间（YYYYMMDDHHMMSSsss)
char	trading_phase_code[ConstField::kTradingPhaseCodeLen]	产品实时阶段及标志（仅深圳有效）
int64_t	pre_close_index	前收盘指数 N18(6)，实际值需除以 1000000
int64_t	open_index	今开盘指数 N18(6)，实际值需除以 1000000
int64_t	high_index	最高指数 N18(6)，实际值需除以 1000000
int64_t	low_index	最低指数 N18(6)，实际值需除以 1000000
int64_t	last_index	最新指数 N18(6)，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 N18(2)，实际值需除以 100（上交所:手，深交所:张）
int64_t	total_value_trade	参与计算相应指数的成交金额 N18(2)，实际值需除以 100000

5.6 期权快照(MDOptionSnapshot)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	期权代码
int64_t	orig_time	交易所行情数据时间（YYYYMMDDHHMMSSsss)
char	trading_phase_code[ConstField::kTradingPhaseCodeLen]	产品实时阶段及标志
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
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，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，仅上海有效），实际值需除以 1000000
int64_t	auction_volume	虚拟匹配数量（仅上海有效），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	今收盘价，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]	申卖量，实际值需除以 100
int64_t	settle_price	今日结算价（仅上海有效），实际值需除以 1000000
int64_t	ref_price	参考价（仅深圳有效），实际值需除以 1000000（此字段仅互联网模式有效）
char	contract_type	合约类别（此字段仅互联网模式有效）
int32_t	expire_date	到期日（此字段仅互联网模式有效）
char	underlying_security_code[ConstField::kSecurityCodeLen]	标的代码（此字段仅互联网模式有效）
int64_t	exercise_price	行权价，实际值需除以 1000000（此字段仅互联网模式有效）

5.7 港股通快照(MDHKTSnapshot)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	港股代码
int64_t	orig_time	交易所行情数据时间（YYYYMMDDHHMMSSsss)
char	trading_phase_code[ConstField::kTradingPhaseCodeLen]	产品实时阶段及标志
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
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,(正常情况下为一档，集中竞价阶段可能有两档)实际值需除以 1000000
int64_t	bid_volume[5]	申买量,(正常情况下为一档，集中竞价阶段可能有两档)实际值需除以 100
int64_t	offer_price[5]	申卖价,(正常情况下为一档，集中竞价阶段可能有两档)实际值需除以 1000000
int64_t	offer_volume[5]	申买量，(正常情况下为一档，集中竞价阶段可能有两档)实际值需除以 100
int64_t	ref_price	参考价（此字段仅互联网模式有效）
int64_t	high_limited	冷静期价格上限（此字段仅互联网模式有效）
int64_t	low_limited	冷静期价格下限（此字段仅互联网模式有效）
int64_t	bid_price_limit_up	买盘上限价，实际值需除以 1000000（仅深圳有效）（此字段仅互联网模式有效）
int64_t	bid_price_limit_down	买盘下限价，实际值需除以 1000000（仅深圳有效）（此字段仅互联网模式有效）
int64_t	offer_price_limit_up	卖盘上限价，实际值需除以 1000000（仅深圳有效）（此字段仅互联网模式有效）
int64_t	offer_price_limit_down	卖盘下限价，实际值需除以 1000000（仅深圳有效）（此字段仅互联网模式有效）

5.8 期货快照(MDFutureSnapshot)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kFutureSecurityCodeLen]	合约代码
int64_t	orig_time	交易日 YYYYMMDDHHMMSSsss (ActionDay+UpdateTime+UpdateMillisec)
int32_t	action_day	业务日期
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	总成交金额，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（暂不可用）
int64_t	latest_volume_trade	最新成交量，实际值需除以 100（此字段仅托管机房模式有效）
int64_t	init_volume_trade	初始持仓量，实际值需除以 100（此字段仅托管机房模式有效）
int64_t	change_volume_trade	持仓量变化，实际值需除以 100（此字段仅托管机房模式有效）
int64_t	bid_imply_volume	申买推导量，实际值需除以 100（此字段仅托管机房模式有效）
int64_t	offer_imply_volume	申卖推导量，实际值需除以 100（此字段仅托管机房模式有效）
int64_t	total_bid_volume_trade	总买入量，实际值需除以 100（此字段仅托管机房模式有效）
int64_t	total_ask_volume_trade	总卖出量，实际值需除以 100（此字段仅托管机房模式有效）
char	exchange_inst_id[ConstField::kExChangeInstIDLen]	合约在交易所的代码（此字段仅互联网模式有效）

5.9 盘后定价交易快照(MDAfterHourFixedPriceSnapshot)

数据类型	字段名称	说明
uint8_t	market_type	市场类型

char	security_code[ConstField::kSecurityCodeLen]	证券代码
uint8_t	variety_category	品种类别
int64_t	orig_time	交易所行情数据时间（为YYYYMMDDHHMMSSsss）
char	trading_phase_code[ConstField::kTradingPhaseCodeLen]	交易阶段代码
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，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，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

5.10 中证指数行情(MDCSIIndexSnapshot)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
uint8_t	index_market	指数市场
char	security_code[ConstField::kSecurityCodeLen]	证券代码
int64_t	orig_time	交易所行情数据时间（YYYYMMDDHHMMSSsss)
int64_t	last_index	最新指数 N11(4)，实际值需除以 1000000
int64_t	open_index	今开盘指数 N11(4)，实际值需除以 1000000
int64_t	high_index	最高指数 N11(4)，实际值需除以 1000000
int64_t	low_index	最低指数 N11(4)，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 N11(4)，实际值需除以 1000000
int64_t	change	涨跌 N11(4)，实际值需除以 1000000
int64_t	ratio_of_change	涨跌幅 N11(4)，实际值需除以 1000000
int64_t	total_volume_trade	成交量 N11(4)，实际值需除以 100
int64_t	total_value_trade	成交金额 N16(5)，实际值需除以 100000(单位为万元)
int64_t	exchange_rate	汇率 N12(8)，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）

5.11国证指数快照(MDCnIndexSnapshot)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	证券代码
int64_t	orig_time	交易所行情数据时间(YYYYMMDDHHMMSSsss)
char	trading_phase_code[ConstField::kTradingPhaseCodeLen]	产品实时阶段及标志
int64_t	pre_close_index	前收盘指数 N18(6)，实际值需除以 1000000
int64_t	open_index	今开盘指数 N18(6)，实际值需除以 1000000
int64_t	high_index	最高指数 N18(6)，实际值需除以 1000000
int64_t	low_index	最低指数 N18(6)，实际值需除以 1000000
int64_t	last_index	最新指数 N18(6)，实际值需除以 1000000
int64_t	close_index	收盘指数，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 N18(2)，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 N18(2)，实际值需除以 100000

5.12港股通实时额度(MDHKTRealtimeLimit)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
int64_t	orig_time	交易所行情数据时间（YYYYMMDDHHMMSSsss)
int64_t	threshold_amount	每日初始额度，单位人民币元，实际值需除以 100000
int64_t	pos_amt	日中剩余额度，单位人民币元，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)
char	mkt_status[ConstField::kTradingStatusLen]	上交所港股通市场状态(上交所独有，来源于上交所文件行情)

5.13港股通产品状态(MDHKTProductStatus)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	证券代码
int64_t	orig_time	交易所行情数据时间(YYYYMMDDHHMMSSsss)
char	trading_status1[ConstField::kTradingStatusLen]	证券交易状态（整手订单）
char	trading_status2[ConstField::kTradingStatusLen]	证券交易状态（零股订单）

5.14港股 VCM(MDHKTVCM)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	港股代码
int64_t	orig_time	交易所行情数据时间(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，实际值需除以 1000000
int64_t	low_price	市调机制最低价格，实际值需除以 1000000
int64_t	high_price	市调机制最高价格，实际值需除以 1000000

5.15L2 现货快照(MDSnapshotL2)

数据类型	字段名称	说明
uint8_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
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（仅上海有效），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

5.16现货逐笔成交(MDTickExecution)

数据类型	字段名称	说明
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	成交价格(类型:价格)
int64_t	exec_volume	成交数量(类型:数量)
int64_t	value_trade	成交金额(类型:金额)

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(深圳:4-撤销 F-成交,上海:F-成交)
char	md_stream_id[ConstField::kMDStreamIDMaxLen]	行情类别(仅深圳有效)
int64_t	biz_index	业务序号
uint8_t	variety_category	品种类别(此字段暂不可用)

5.17逐笔委托(MDTickOrder)

数据类型	字段名称	说明
uint8_t	market_type	市场类型
char	security_code[ConstField::kSecurityCodeLen]	证券代码
int64_t	appl_seq_num	消息记录号
int32_t	channel_no	频道编号
int64_t	order_time	委托时间（YYYYMMDDHHMMSSsss）
int64_t	order_price	委托价格
int64_t	order_volume	委托数量
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
char	md_stream_id[ConstField::kMDStreamIDLen]	行情类别
char	product_status[ConstField::kTradingPhaseCodeLen]	产品状态(仅上海有效)
int64_t	orig_order_no	原始订单号
int64_t	biz_index	业务序号

5.18委托队列(MDOrderQueue)

数据类型	字段名称	说明
uint8_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

int64_t	order_volume	委托数量
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	频道代码
char	md_stream_id[ConstField::kMDStreamIDLen]	行情类别

5.19代码表(MDCodeTable)

数据类型	字段名称	说明
char	security_code[ConstField::kSecurityCodeLen]	交易所证券代码
char	symbol[ConstField::kSecuritySymbolLen]	证券简称
char	english_name[ConstField::kSecurityEnglishNameLen]	英文简称
uint8_t	market_type	市场类型
char	security_type[ConstField::kSecurityTypeLen]	证券类别
char	currency[ConstField::kCurrencyLen]	币种

说明：此数据结构为 IGMDCodeTableSpi 中 OnMDCodeTable 数据回调接口所用。

5.20复权因子表(MDExFactorTable)

数据类型	字段名称	说明
char	inner_code[ConstField::kSecurityCodeLen]	证券内部代码
char	security_code[ConstField::kSecurityCodeLen]	证券代码
uint32_t	ex_date	除权除息日（为 yyyyMMdd）
double	ex_factor	复权因子 N38(18)
double	cum_factor	累计复权因子 N38(18)

5.21个股基础信息(MDStockInfo)

数据类型	字段名称	说明
char	security_code[ConstField::kSec	交易所证券代码

	urityCodeLen]	
char	symbol[ConstField::kSecuritySymbolLen]	证券简称
uint8_t	market_type	市场类型
char	security_type[ConstField::kSecurityTypeLen]	证券类别
char	currency[ConstField::kCurrencyLen]	币种
char	security_status[ConstField::kSecurityStatusLen]	证券状态 kSecurityStatusLen=24
uint64_t	pre_close_price	昨收价
uint64_t	total_shares	总股本
uint64_t	flow_shares	流通股本
uint8_t	noprofit	是否盈利 Y：是，未盈利 N：否，已盈利
uint8_t	weighted_voting_rights	是否存在投票权差异 Y.是 N.否
uint8_t	registration_flag	是否注册制 Y.是 N.否
double	eps	每股收益 N18(6)
double	eps_cell	预计每股收益 N18(6)
double	net_profit_ttm	净利润（TTM）N18(6)
double	net_profit	净利润 N18(6)
double	net_asset	净资产 N18(6)
double	net_profit_recent_annual	净利润 N18(6)
double	net_profit_first_quarter	净资产 N18(6)

5.22代码表(MDCodeTableRecord)

数据类型	字段名称	说明
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]	证券子类别
char	currency[ConstField::kTypesLen]	币种(CNY:人民币,HKD:港币,USD:美元,AUD:澳币,CAD:加币,JPY:日圆,SGD:新加坡币,GBP:英镑,EUR:欧元,TWD:新台币,Other:其他)
uint8_t	variety_category	证券类别

int64_t	pre_close_price	昨收价(类型:价格)
char	underlying_security_id[ConstField::kSecurityCodeLen]	标的代码(仅期权/权证/期货期权有效)
char	contract_type[ConstField::kMaxTypesLen]	合约类别(仅期权/期货期权有效)
int64_t	exercise_price	行权价(仅期权/期货期权有效,类型:价格)
uint32_t	expire_date	到期日(仅期权/期货期权有效)
int64_t	high_limited	涨停价(类型:价格)
int64_t	low_limited	跌停价(类型:价格)
char	security_status[ConstField::kCodeTableSecurityStatusMaxLen]	产品状态标志
int64_t	price_tick	最小价格变动单位(类型:价格)
int64_t	buy_qty_unit	限价买数量单位(类型:数量)
int64_t	sell_qty_unit	限价卖数量单位(类型:数量)
int64_t	market_buy_qty_unit	市价买数量单位(类型:数量)
int64_t	market_sell_qty_unit	市价卖数量单位(类型:数量)
int64_t	buy_qty_lower_limit	限价买数量下限(类型:数量)
int64_t	buy_qty_upper_limit	限价买数量上限(类型:数量)
int64_t	sell_qty_lower_limit	限价卖数量下限(类型:数量)
int64_t	sell_qty_upper_limit	限价卖数量上限(类型:数量)
int64_t	market_buy_qty_lower_limit	市价买数量下限(类型:数量)
int64_t	market_buy_qty_upper_limit	市价买数量上限(类型:数量)
int64_t	market_sell_qty_lower_limit	市价卖数量下限(类型:数量)
int64_t	market_sell_qty_upper_limit	市价卖数量上限(类型:数量)
uint32_t	list_day	上市日期
int64_t	par_value	面值(类型:价格)
int64_t	outstanding_share	总发行量(上交所不支持,类型:数量)
int64_t	public_float_share_quantity	流通股数(上交所不支持,类型:数量)
int64_t	contract_multiplier	对回购标准券折算率(类型:比例)
char	regular_share[ConstField::RegularShare]	对应回购标准券(仅深交所)
int64_t	interest	应计利息(类型:汇率)
int64_t	coupon_rate	票面年利率(类型:比例)
char	product_code[ConstField::kFutureSecurityCodeLen]	期货品种产品代码(仅期货期权有效)
uint32_t	delivery_year	交割年份(仅期货期权有效)
uint32_t	delivery_month	交割月份(仅期货期权有效)
uint32_t	create_date	创建日期(仅期货期权有效)
uint32_t	start_deliv_date	开始交割日(仅期货期权有效)
uint32_t	end_deliv_date	结束交割日(仅期货期权有效)
uint32_t	position_type	持仓类型(仅期货期权有效)

说明：此数据结构为 IGMDStockInfoSpi 中 OnMDStockInfo 数据回调接口所用。

5.23金融资讯数据(ThirdInfoData)

数据类型	字段名称	说明
uint64_t	task_id	任务 id
uint64_t	data_size	数据大小
char*	json_data	数据 json 串

6. 补充字段取值说明

6.1 证券代码表 security_type

深交所:
1 主板 A 股
2:中小板股票
3:创业板股票
4:主板 B 股
5:国债（含地方债）
6:企业债
7:公司债
8:可转债
9:私募债
10:可交换私募债
11:证券公司次级债
12:质押式回购
13:资产支持证券
14:本市场股票 ETF
15:跨市场股票 ETF
16:跨境 ETF
17:本市场实物债券 ETF
18:现金债券 ETF
19:黄金 ETF
20:货币 ETF
21:杠杆 ETF （预留）
22:商品期货 ETF
23:标准 LOF
24:分级子基金
25:封闭式基金
26:仅申赎基金
28:权证
29:个股期权
30:ETF 期权
33:优先股
34:证券公司短期债
35:可交换公司债
36:主板、中小板存托凭证
37:创业板存托凭证

港股:
EQTY:股本
TRST:信托
WRNT:权证
BOND:债券
BWRT:一篮子权证

上交所：
GBF:国债
GBZ:无息国债
DST:国债分销（仅用于分销阶段）
DVP:公司债（地方债）分销
CBF:企业债券
CCF:可转换企业债券
CPF:公司债券（或地方债券）
FBF:金融机构发行债券
CRP:质押式国债回购
BRP:质押式企债回购
ORP:买断式债券回购
CBD:分离式可转债
OBD:其它债券
CEF:封闭式基金
OEF:开放式基金
EBS:交易所交易基金（买卖）
OFN:其它基金
ASH:以人民币交易的股票
BSH:以美元交易的股票
CSH:国际版股票
OEQ:其它股票
CIW:企业发行权证
COV:备兑权证
FEQ:个股期货
FBD:债券期货
OFT:其它期货
AMP:集合资产管理计划
WIT:国债预发行
LOF:LOF 基金
OPS:公开发行优先股
PPS:非公开发行优先股
QRP:报价回购
CMD:控制指令（中登身份认证密码服务产品复用 CMD 证券子类别）

6.2 证券代码表的 currency 取值

CNY: 人民币
HKD: 港币
USD: 美元
AUD: 澳币
CAD: 加币
JPY: 日圆
SGD: 新加坡币
GBP: 英 镑
EUR: 欧元

6.3 交易阶段代码取值

**********上海现货快照交易状态**********
该字段为 8 位字符数组,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’表示开盘集合竞价时段,‘T’表示连续交易时段,‘E’表示闭市时段,‘P’表示产品停牌。
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。交易阶段代码

6.4 Security_status

1:停牌
2:除权
3:除息
4:ST
5:*ST
6:上市首日
7:公司再融资
8:恢复上市首日
9:网络投票
10:退市整理期
12:增发股份上市
13:合约调整
14:暂停上市后协议转让
15:实施双转单调整
16:特定债券转让
17:上市初期

上交所，参考参考 IS101_上海证券交易所竞价撮合平台市场参与者接口规格说明书 1.47 版_20200703 .docx
该字段为 20 位字符串，每位表示含义如下，无定义则填空格。
该字段为 20 位字符串，每位表示含义如下，无定义则填空格。
第 1 位对应：‘N’表示首日上市。
第 2 位对应：‘D’表示除权。
第 3 位对应：‘R’表示除息。
第 4 位对应：‘D’表示国内主板正常交易产品，’S’表示股票风险警示产品，’P’表示退市整理产品，’T’表示退市转让产品，‘U’表示优先股产品。
第 5 位不启用。
第 6 位对应：‘L’表示债券投资者适当性要求类，’M’表示债券机构投资者适当性要求类。
第 7 位对应：‘F’表示 15:00 闭市的产品，对应 mktdt00.txt 行情文件中市场行情状态第 4 位闭市标志，‘S’表示 15:30 闭市的产品，对应 mktdt00.txt 行情文件中市场行情状态第 5 位闭市标志，为空表示非竞价撮合平台挂牌产品，无意义。

6.5 K 线算法说明

前推算法举例：9:31 的 1 分钟 K 线，计算的是 9:30:00.000~9:30:59.999 期间的 K 线。 9:40 的 5 分钟 K 线，计算的是 9:35:00.000~9:39:59.999 期间的 K 线。