深圳证券信息有限公司		文档编号				
		名　　称	深交所行情互联网接入服务接口说明书			
编写	签名：　日期：		**密级**	内部公开	版本	V1.0.0
审核	签名：　日期：		**批准**			

修订记录：

版本编号	编写/修订内容	修订人	修订日期

深圳证券信息有限公司

目 录

1. 引言

深圳证券信息有限公司负责深圳证券交易所的基础行情、增强行情的牌照授权工作，目前所有从本公司获得合法行情牌照授权的客户，仅能通过深证通、上证信息、其他信息商等途径获取深交所行情数据，存在着接入成本高、服务不稳定、响应不及时、行情数据延时等种种问题。为保障已授权客户的权益，本公司建立深交所基础行情系统，为所有已授权基础行情客户提供深交所基础行情数据接入服务。

深交所行情互联网接入服务，旨在为各类已授权基础行情客户提供深交所的基础行情数据接入服务，系统建设目标是通过多种服务方式覆盖各类客户群体的行情数据接入需求，提供低成本、低门槛、稳定、高效的行情接入方案。

2. 安装及应用发布

2.1. C 语言版本 SzsimdApi

C 语言版本的 SzsimdApi 支持的操作系统列表如下：

操作系统	说明
Windows 7	Windows7 或以上，仅支持 64 位操作系统
Linux	CentOS 6.7、RedHat 6.7

C 语言版本的 SzsimdApi 由以下几个文件组成：

文件名	说明
szsimdapi.h	SzsimdApi 头文件
szsimdapi.lib	SzsimdApi 库文件
szsimdapi.dll	SzsimdApi 动态链接库文件，Windows 平台
szsimdapi.ini	SzsimdApi 配置文件
libszsimdapi.so	SzsimdApi 动态链接库文件，Linux 平台

2.2. Java 语言版本 SzsimdApi

Java 语言版本的 SzsimdApi 支持的操作系统列表如下：

操作系统	说明

Windows	Windows7 或以上，仅支持 64 位操作系统
Linux	CentOS 6.7、RedHat 6.7

Java 语言版本的 SzsimdApi 由以下几个文件组成：

文件名	说明
szsimdapi.jar	SzsimdApi jar 包
szsimdapi.dll	SzsimdApi 动态链接库文件，Windows 平台
szsimdapi.ini	SzsimdApi 配置文件
libszsimdapi.so	SzsimdApi 动态链接库文件，Linux 平台

2.3. 关于配置的说明

SzsimdApi 在使用过程中，可以产生运行日志并计入日志文件中。配置文件 szsimdapi.ini 包含了 SzsimdApi 日志的产生和输出。文件格式如下：

```
[Glob]
// 监听的端口，同一个端口只能初始化一次
ListenPidPort=1999
// 连接用户网关的设置，其中用户名密码需要用户网关配置文件配置才能正常登陆使用
ToAmdUserGw=127.0.0.1:5016:5017:user1:password1

[SzsimdApiLog]
Type="2"                    //1-循环日志; 2-按照日期每天写一个日志.
Level="0"                   //日志级别，运行时一般取 0.
Display="3"                 //显示在哪里:1-文件, 2-屏幕, 3-文件和屏幕.
LogDir="log"                //日志目录.
LogName="szsimdapi.log"         //日志文件名称.
MaxFileCount="99"           //最大文件数目，对循环日志有效.
MaxFileSize="5000000000"    //最大日志文件大小 Byte.
```

3. 使用概述

3.1. 功能

SzsimdApi 是一组提供给用户调用的 C/Java 语言应用程序接口，用户可以调用该 API 开发，实现深证信互联网行情的接收，SzsimdApi 可以完成通信的自动连接，接收数据包，数据压缩解压等功能，应用程序只需要设置好对应的回调函数就可以接收深交所二进制行情数据，API 处于最底层，由应用程序调用，通过 TCP 连接与上层节点通信。将接收到的数

据包处理后，通过回调接口通知给调用的应用程序。

3.2. 应用环境

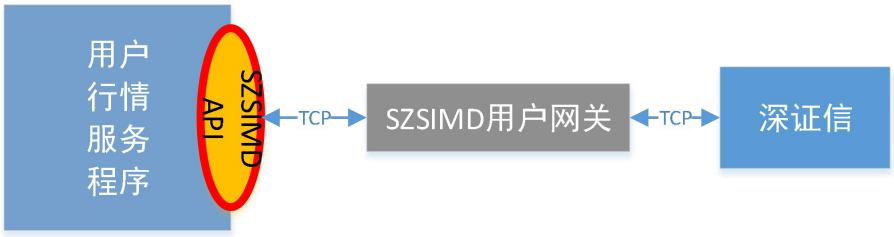

图 1　　SZSIMD API 应用环境

在深交所行情互联网接入服务中，SZSIMD 用户网关负责接收 SZSIMD 行情转发节点推送的行情数据，SZSIMD API 通过直接连接 SZSIMD 用户网关获取行情数据。

3.3. 线程安全性

参看下面 4.3 节。

3.4. 应用系统的安全性

用户在应用软件中调用了 SzsimdApi，在 SzsimdApi 中不会包含故意攻击用户系统的软件或代码，不会故意影响用户系统的安全运行。用户系统自身的安全性，应该由用户自己进行检查和维护，SzsimdApi 无法保证用户系统本身的安全性。

4. 编程参考

4.1. 常量定义

4.1.1. 长度常量

名称	定义值	说明
G_SZSIMDAPI_MAXLEN_ERRORSTR	256	错误码字符串的最大长度

4.1.2. 函数返回值

返回值	含义
0	成功

-1	失败

4.2. 数据结构说明

4.2.1. 字段说明

类型	说明
const char*	以'\0'字符结尾的字符串
(U)INT(8/16/32/64)	(无符号)整数(8/16/32/64 位)

4.2.2. STUSzsimdApiSnapshot_XianHuo

该结构用来定义现货（股票，基金，债券等）集中竞价交易快照行情数据

结构定义：

```
struct STUSzsimdApiSnapshot_XianHuo
{
    const char*         m_psMDEntryType;
    INT64               m_i64MDEntryPx;
    INT64               m_i64MDEntrySize;
    UINT16              m_ui16MDPriceLevel;
    INT64               m_i64NumberOfOrders;
    UINT32              m_ui32NoOrders;
    INT64*              m_pArrayOrderQty;
};
```

字段说明：

字段	说明
m_psMDEntryType	行情条目类别
m_i64MDEntryPx	价格
m_i64MDEntrySize	数量
m_ui16MDPriceLevel	买卖盘档位
m_i64NumberOfOrders	价位总委托笔数，为 0 表示不揭示
m_ui32NoOrders	价位揭示委托笔试，为 0 表示不揭示
m_pArrayOrderQty	委托数量数组

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.3. STUSzsimdApiSnapshot_PanHou

该结构用来定义盘后定价交易业务行情快照数据

结构定义：

```
struct STUSzsimdApiSnapshot_PanHou
{
    const char*         m_psMDEntryType;
    INT64               m_i64MDEntryPx;
    INT64               m_i64MDEntrySize;
};
```

字段说明：

字段	说明
m_psMDEntryType	行情条目类别，0=买入，1=卖出
m_i64MDEntryPx	价格
m_i64MDEntrySize	数量

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.4. STUSzsimdApiSnapshot_ZhiShu

该结构用来定义指数行情快照数据

结构定义：

```
struct STUSzsimdApiSnapshot_ZhiShu
{
    const char*         m_psMDEntryType;
    INT64               m_i64MDEntryPx;
};
```

字段说明：

字段	说明
m_psMDEntryType	行情条目类别
m_i64MDEntryPx	指数点位

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.5. STUSzsimdApiSnapshot_HKStk_MDEntry

该结构用来定义港股行情快照扩展字段的行情条目

结构定义：

```
struct STUSzsimdApiSnapshot_HKStk_MDEntry
{
```

```
    const char*         m_psMDEntryType;
    INT64               m_i64MDEntryPx;
    INT64               m_i64MDEntrySize;
    UINT16              m_ui16MDPriceLevel;
};
```

字段说明：

字段	说明
m_psMDEntryType	行情条目类别
m_i64MDEntryPx	价格
m_i64MDEntrySize	数量
m_ui16MDPriceLevel	买卖盘档位

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.6. STUSzsimdApiSnapshot_HKStk_ComplexEvent

该结构用来定义港股行情快照扩展字段的冷静期数据

结构定义：

```
struct STUSzsimdApiSnapshot_HKStk_ComplexEvent
{
    INT64               m_i64ComplexEventStartTime;
    INT64               m_i64ComplexEventEndTime;
};
```

字段说明：

字段	说明
m_i64ComplexEventStartTime	冷静期开始时间
m_i64ComplexEventEndTime	冷静期结束时间

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。：

4.2.7. STUSzsimdApiSnapshot_HKStkExt

该结构用来定义港股行情快照扩展字段数据

结构定义：

```
struct STUSzsimdApiSnapshot_HKStkExt
{
    STUSzsimdApiSnapshot_HKStk_MDEntry*     m_pArrayHKStkMDEntry;
```

```
    UINT32 m_ui32NoComplexEventTimes;
    STUSzsimdApiSnapshot_HKStk_ComplexEvent*    m_pArrayHKStkComplexEvent;
};
```

字段说明：

字段	说明
m_pArrayHKStkMDEntry	STUSzsimdApiSnapshot_HKStk_MDEntry 数组
m_ui32NoComplexEventTimes	VCM 冷静期个数，0 或 1
m_pArrayHKStkComplexEvent	STUSzsimdApiSnapshot_HKStk_ComplexEvent 数组

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.8. STUSzsimdApiSnapshotMD

该结构用来定义现货、盘后、指数和统计等快照数据

结构定义：

```
struct STUSzsimdApiSnapshotMD
{
    UINT32                          m_ui32MsgType;
    INT64                           m_i64OrigTime;
    UINT16                          m_ui16ChannelNo;
    const char*                     m_psMDStreamID;
    const char*                     m_psSecurityID;
    const char*                     m_psSecurityIDSource;
    const char*                     m_psTradingPhaseCode;
    INT64                           m_i64PrevClosePx;
    INT64                           m_i64NumTrades;
    INT64                           m_i64TotalVolumeTrade;
    INT64                           m_i64TotalValueTrade;

    //XianHuoPart
    UINT32                                  m_ui32NoMDEntries;
    STUSzsimdApiSnapshot_XianHuo*       m_pArrayXianHuoPart;

    //PanHou
    //UINT32                                m_ui32NoMDEntries;
    STUSzsimdApiSnapshot_PanHou*        m_pArrayPanHouPart;

    //ZhiShu
    //UINT32                                m_ui32NoMDEntries;
    STUSzsimdApiSnapshot_ZhiShu*            m_pArrayZhiShuPart;
```

```
    STUSzsimdApiSnapshot_HKStkExt        m_oHKStkExt;

    //Tongji
    UINT32                                    m_ui32_Tongji_StockNum;
};
```

字段说明：

字段	说明
m_ui32MsgType	消息头，MsgType=306311
m_i64OrigTime	数据生成时间
m_ui16ChannelNo	频道代码
m_psMDStreamID	行情类别源
m_psSecurityID	证券代码
m_psSecurityIDSource	证券代码源
m_psTradingPhaseCode	产品所处的交易阶段代码
m_i64PrevClosePx	昨收价
m_i64NumTrades	成交笔数
m_i64TotalVolumeTrade	成交总量
m_i64TotalValueTrade	成交总金额
m_ui32NoMDEntries	行情条目个数
m_pArrayXianHuoPart	STUSzsimdApiSnapshot_XianHuo 数组
m_pArrayPanHouPart	STUSzsimdApiSnapshot_PanHou 数组
m_pArrayZhiShuPart	STUSzsimdApiSnapshot_ZhiShu 数组
m_oHKStkExt	STUSzsimdApiSnapshot_HKStkExt 数组
m_ui32_Tongji_StockNum	统计量指标样本个数

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.9. STUSzsimdApiOneByOneWeiTuo

该结构用来定义逐笔委托行情数据

结构定义：

```
struct STUSzsimdApiOneByOneWeiTuo
{
    UINT32          m_ui32MsgType;
    UINT16          m_ui16ChannelNo;
    INT64           m_i64ApplSeqNum;
    const char*      m_psMDStreamID;
    const char*      m_psSecurityID;
    const char*      m_psSecurityIDSource;
```

```
    INT64               m_i64Price;
    INT64               m_i64OrderQty;
    const char*          m_psSide;
    INT64               m_i64TransactTime;

    const char*         m_psOrdType;

    const char*         m_psConfirmID;
    const char*         m_psContactor;
    const char*         m_psContactInfo;

    UINT16             m_ui16ExpirationDays;
    UINT8              m_ui8ExpirationType;
};
```

字段说明：

字段	说明
m_ui32MsgType	消息头，MsgType=30xx92
m_ui16ChannelNo	频道代码
m_i64ApplSeqNum	消息记录号，从 1 开始计数
m_psSecurityID	证券代码
m_psSecurityIDSource	证券代码源
m_i64Price	委托价格
m_i64OrderQty	委托数量
m_psSide	买卖方向，1=买，2=卖，G=借入，F=出错
m_i64TransactTime	委托时间
m_psOrdType	订单类别，1=市价，2=现价，U=本方最优
m_psConfirmID	定价行情约定号
m_psContactor	联系人
m_psContactInfo	联系方式
m_ui16ExpirationDays	期限，单位为天数
m_ui8ExpirationType	期限类型，1=固定起点

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.10.STUSzsimdApiOneByOneChengJiao

该结构用来定义逐笔成交快照行情

结构定义：

```
struct STUSzsimdApiOneByOneChengJiao
{
```

```
    UINT32          m_ui32MsgType;
    UINT16          m_ui16ChannelNo;
    INT64           m_i64ApplSeqNum;
    const char*     m_psMDStreamID;
    INT64           m_i64BidApplSeqNum;
    INT64           m_i64OfferApplSeqNum;
    const char*     m_psSecurityID;
    const char*     m_psSecurityIDSource;
    INT64           m_i64LastPx;
    INT64           m_i64LastQty;
    const char*     m_psExecType;
    INT64           m_i64TransactTime;
};
```

字段说明：

字段	说明
m_ui32MsgType	消息头，MsgType=30xx91
m_ui16ChannelNo	频道代码
m_i64ApplSeqNum	消息记录号，从 1 开始计数
m_psMDStreamID	行情类别
m_i64BidApplSeqNum	买方委托索引，从 1 开始计数，0 表示无对应委托
m_i64OfferApplSeqNum	卖方委托索引，从 1 开始计数，0 表示无对应委托
m_psSecurityID	证券代码
m_psSecurityIDSource	证券代码源
m_i64LastPx	委托价格
m_i64LastQty	委托数量
m_psExecType	成交类别，4=撤销，F=成交
m_i64TransactTime	委托时间

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.11.STUSzsimdApiChannelStatisticsPart

该结构用来定义频道统计行情数据扩展字段

结构定义：

```
struct STUSzsimdApiChannelStatisticsPart
{
    const char*     m_psMDStreamID;
    UINT32          m_ui32StockNum;
    const char*     m_psTradingPhaseCode;
};
```

字段说明：

字段	说明
m_psMDStreamID	行情类别个数
m_ui32StockNum	证券只数
m_psTradingPhaseCode	闭市状态，第 0 位：T=连续竞价（全部证券尚未闭市），E=已闭市（全部证券已闭市）

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.12.STUSzsimdApiChannelStatisticsMD

该结构用来定义快照行情频道统计信息

结构定义：

```
struct STUSzsimdApiChannelStatisticsMD
{
    UINT32    m_ui32MsgType;
    INT64     m_i64OrigTime;
    UINT16    m_ui16ChannelNo;
    UINT32    m_ui32NoMDStreamID;
    //need to be free by user.
    STUSzsimdApiChannelStatisticsPart* m_pChannelStatisticsPart;
};
```

字段说明：

字段	说明
m_ui32MsgType	消息头，MsgType=390090
m_i64OrigTime	数据生成时间
m_ui16ChannelNo	频道代码
m_ui32NoMDStreamID	行情类别个数
m_pChannelStatisticsPart	STUSzsimdApiChannelStatisticsPart 数组

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.13.STUSzsimdApiStkRtStatusPart

该结构用来定义证券实时状态行情开关明细

结构定义：

```
struct STUSzsimdApiStkRtStatusPart
{
    UINT16        m_ui16SecuritySwitchType;
```

```
    UINT16          m_ui16SecuritySwitchStatus;
};
```

字段说明：

字段	说明
m_ui16SecuritySwitchType	开关类别
m_ui16SecuritySwitchStatus	开关状态

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.14.STUSzsimdApiStkRtStatusMD

该结构用来定义证券实时状态行情

结构定义：

```
struct STUSzsimdApiStkRtStatusMD
{
    UINT32          m_ui32MsgType;
    INT64           m_i64OrigTime;
    UINT16          m_ui16ChannelNo;
    const char*     m_psSecurityID;
    const char* m_psSecurityIDSource;
    const char*     m_psFinancialStatus;
    UINT32                                  m_ui32NoSwitch;
   //need to be free by user
   STUSzsimdApiStkRtStatusPart*      m_pStkRtStatusPart;
};
```

字段说明：

字段	说明
m_ui32MsgType	消息头，MsgType=390013
m_i64OrigTime	数据生成时间
m_ui16ChannelNo	频道代码
m_psSecurityID	证券代码
m_psSecurityIDSource	证券代码源
m_psFinancialStatus	证券状态
m_ui32NoSwitch	开关个数
m_pStkRtStatusPart	STUSzsimdApiStkRtStatusPart 数组

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.15.STUSzsimdApiMktRtStatusMD

该结构用来定义市场实施状态

结构定义：

```
struct STUSzsimdApiMktRtStatusMD
{
    UINT32          m_ui32MsgType;
    INT64           m_i64OrigTime;
    UINT16          m_ui16ChannelNo;
    const char*     m_psMarketID;
    const char*     m_psMarketSegmentID;
    const char*     m_psTradingSessionID;
    const char*     m_psTradingSessionSubID;
    UINT16          m_ui16TradSesStatus;
    INT64           m_i64TradSesStartTime;
    INT64           m_i64TradSesEndTime;
    INT64           m_i64ThresholdAmount;
    INT64           m_i64PosAmt;
    const char*     m_psAmountStatus;
};
```

字段说明：

字段	说明
m_ui32MsgType	消息头，MsgType=390019
m_i64OrigTime	数据生成时间
m_ui16ChannelNo	频道代码
m_psMarketID	市场代码
m_psMarketSegmentID	市场板块代码，预留
m_psTradingSessionID	交易会话 ID
m_psTradingSessionSubID	交易会话子 ID
m_ui16TradSesStatus	交易会话状态，预留
m_i64TradSesStartTime	交易会话起始时间，预留
m_i64TradSesEndTime	交易会话结束时间，预留
m_i64ThresholdAmount	每日初始额度
m_i64PosAmt	日中剩余额度，额度不可用时，发布固定值 0.0000
m_psAmountStatus	额度状态

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.16.STUSzsimdApiNotice

该结构用来定义公告消息行情

结构定义：

```
struct STUSzsimdApiNotice
{
    UINT32          m_ui32MsgType;
     INT64          m_i64OrigTime;
     UINT16         m_ui16ChannelNo;

     const char*    m_psNewsID;
     const char*  m_psHeadline;
     const char*  m_psRawDataFormat;
     UINT32         m_ui32RawDataLength;
     const char*  m_psRawData;
};
```

字段说明：

字段	说明
m_ui32MsgType	消息头，MsgType=390012
m_i64OrigTime	公告时间
m_ui16ChannelNo	频道代码
m_psHeadline	公告标题
m_psRawDataFormat	二进制数据格式
m_ui32RawDataLength	二进制数据长度
m_psRawData	二进制数据

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.2.17.STUSzsimdApiCallParam

该结构用来定义 SZSIMDApi 的回调函数设置等

结构定义：

```
struct STUSzsimdApiCallParam
{
    void*     m_pUserPtr;
    void      (*OnSnapshotMD)(void* pUserPtr, const STUSzsimdApiSnapshotMD
*pSnapshotMD);
    void      (*OnOneByOneWeiTuo)(void* pUserPtr, const STUSzsimdApiOneByOneWeiTuo
```

```
*pOneByOneWeiTuo);
        void        (*OnOneByOneChengJiao)(void* pUserPtr, const
STUSzsimdApiOneByOneChengJiao *pByOneChengJiao);
        void        (*OnChannelStatisticsMD)(void* pUserPtr, const
STUSzsimdApiChannelStatisticsMD *pChannelStatisticsMD);
        void        (*OnStkRtStatusMD)(void* pUserPtr, const STUSzsimdApiStkRtStatusMD
*pStkRtStatusMD);
        void        (*OnMktRtStatusMD)(void* pUserPtr, const STUSzsimdApiMktRtStatusMD
*pMktRtStatusMD);
        void        (*OnNotice)(void* pUserPtr, const STUSzsimdApiNotice *pNotice);
};
```

字段说明：

字段	说明	线程安全性
m_pUserPtr	开发人员根据需要绑定的数据指针	无
OnSnapshotMD	行情快照数据回调函数	不保证在同一线程
OnOneByOneWeiTuo	逐笔委托数据回调函数	不保证在同一线程
OnOneByOneChengJiao	逐笔成交数据回调函数	不保证在同一线程
OnStkRtStatusMD	证券实时状态状态回调函数	不保证在同一线程
OnMktRtStatusMD	市场实时状态回调函数	不保证在同一线程
OnNotice	公告消息回调函数	调用 SzsimdApi_GetNoticeList 所在线程

字段具体含义和格式定义等明细参见《深圳证券交易所 Binary 行情数据接口规规范》。

4.3. C 函数接口

4.3.1. 函数清单

SZSIMD API 是一个简明易用的编程接口，它提供了如下 4 个函数：

序号	函数名称	函数功能	线程安全性
1	SzsimdApi_Init	初始化，获取相关资源，并尝试与 SZSIMD 用户网关联立链接	否
2	SzsimdApi_Run	运行 API	否
3	SzsimdApi_Stop	停止 API 的运行	是
4	SzsimdApi_Destroy	释放 API 句柄	是

4.3.2. SzsimdApi_Init

SzsimdApi 的初始化函数，该函数对 API 进行初始化，分配获取相关的资源。

函数原型：

```
void* _stdcall SzsimdApi_Init(const char* psApiIniFilePath, STUSzsimdApiCallParam*
pApiCallParam, int* piErrCode, char szErrString[G_SZSIMDAPI_MAXLEN_ERRORSTR]);
```

参数说明 ：

参数	说明
psApiIniFilePath[in]	配置文件路径
pApiCallParam [in]	设置用户收到消息后的回调处理
piErrCode [out]	初始化失败时的错误码
szErrString [out]	初始化失败时的错误描述

返回值说明：

返回值	说明
NULL	初始化失败
非 NULL	初始化成功，返回一个链接句柄

4.3.3. SzsimdApi_Run

运行 API

函数原型：

```
int _stdcall SzsimdApi_Run(void* pHandle, int bIsBlock)
```

参数说明 ：

参数	说明
pHandle [in]	初始化返回的句柄
bIsBlock [in]	是否阻塞，0 表示不阻塞，函数立即返回。1 表示阻塞，函数不返回，需要调用 SzsimdApi_Stop 停止 API 运行，此时函数才会返回

返回值说明：

返回值	说明
0	成功返回
非 0	运行失败

4.3.4. SzsimdApi_Stop

停止 API 的运行

函数原型：

```
int _stdcall SzsimdApi_Stop(void* pHandle);
```

参数说明 ：

参数	说明
pHandle [in]	初始化返回的句柄

返回值说明：

返回值	说明
0	成功返回
非 0	运行失败

4.3.5. SzsimdApi_Destroy

销毁 api 句柄

函数原型：

```
void _stdcall SzsimdApi_Destroy(void* pHandle)
```

参数说明 ：

参数	说明
pHandle [in]	初始化返回的句柄

返回值说明：

返回值	说明
无	无

4.3.1. SzsimdApi_GetNoticeList

获取公告列表，函数调用后，如果有公告数据，则会触发 OnNotice 回调函数。特殊说明：如果调用了该函数，则会接收到用户网关上的所有公告和公告概要。如果不调用该函数，则只会收到用户网关新接收到的公告。

函数原型：

```
int _stdcall SzsimdApi_GetNoticeList(void* pHandle)
```

参数说明 ：

参数	说明
pHandle [in]	初始化返回的句柄

返回值说明：

返回值	说明

0	成功返回
非 0	运行失败

4.3.2. 调用顺序

按顺序依次调用 SzsimdApi_Init，SzsimdApi_Run。若要停止 SzsimdApi，则按顺序调用 SzsimdApi_Stop，SzsimdApi_Destroy。

注意事项：如果是多线程运行 SzsimdApi，并且使用了 SzsimdApi_Run 阻塞版本，那么，调用 SzsimdApi_Stop 后应等待调用 SzsimdApi_Run 的线程退出后，才能调用 SzsimdApi_Destroy 释放资源。

5. 配置文件

SzsimdApi 需要配置文件(szsimdapi.ini)才能正常初始化和运行，配置文件信息如下：

```
[Glob]
ListenPidPort=499
ToSzsiUserGw=127.0.0.1:5016:5017:user1:password1

//连接用户网关的信息，gwip:实时端口:重传端口:用户名::密码:

// 用户名和密码字段在用户网关的配置文件[ToAPI]段设置

[SzsimdApiLog]
Type="2"                    //1-循环日志; 2-按照日期每天写一个日志.
Level="0"                   //日志级别，运行时一般取 0.
Display="3"                 //显示在哪里:1-文件, 2-屏幕, 3-文件和屏幕.
LogDir="log"                //日志目录.
LogName="szsimdapi.log"     //日志文件名称.
MaxFileCount="99"           //最大文件数目，对循环日志有效.
MaxFileSize="5000000000"    //最大日志文件大小 Byte.
HostIP=
```

6. 日志文件

根据 szsimd.ini 配置文件的设置，会在可执行程序的当前目录下创建名为 log 的目录，存放每天的运行日志文件。日志文件内容按照级别(level)分为[WARN][ERROR][INFO][DEBUG][FATAL]，数值分别为日常运行过程中 level 设置为 0 即可，0 代表只输出 INFO 级别及以下的日志信息，

日志以每行为单位，分别输出 日期，日志所在文件名，模块信息，进程号，级别，以及实际的日志信息。常见的有程序的启动信息，网络建立和断开信息、登录成功或失败信息等。

```
<2017-08-01 18:41:33.228,    amdapp.cpp,0292,Ecd=0||Mod=amd|Pid=14228>INFO: ====================================
<2017-08-01 18:41:33.228,    amdapp.cpp,0293,Ecd=0||Mod=amd|Pid=14228>INFO: Program[version(01.00.20170702), build date(Aug  1 2017, 18:41:23)] is initializing...
<2017-08-01 18:41:33.229,    amdapp.cpp,0303,Ecd=0||Mod=amd|Pid=14228>INFO: Listen at PidPort[1999] OK
<2017-08-01 18:41:33.230,    amdapp.cpp,0304,Ecd=0||Mod=amd|Pid=14228>INFO: Auto Exit time is[1970-01-01 08:00:00]
<2017-08-01 18:41:33.345,    amdapp.cpp,1035,Ecd=0||Mod=amd|Pid=14228>INFO: re-read amduser.cfg OK, amd user config count is[2]
<2017-08-01 18:41:33.347, amdamibus.cpp,0095,Ecd=0||Mod=amd|Pid=14228>INFO: Listen at SimuAMI Recv UDP port[0] OK.
<2017-08-01 18:41:33.384,   amdsock.cpp,0102,Ecd=0||Mod=amd|Pid=14228>INFO: I am sz-server1.
<2017-08-01 18:41:33.385,   amdsock.cpp,0654,Ecd=0||Mod=amd|Pid=14228>INFO: Begin TCP connect ToUpperRltime server[127.0.0.1:8016] ...
<2017-08-01 18:41:33.385,   amdsock.cpp,0679,Ecd=0||Mod=amd|Pid=14228>INFO: Begin TCP connect ToUpperRetran server[127.0.0.1:8018] ...
<2017-08-01 18:41:33.388,    amdapp.cpp,0518,Ecd=0||Mod=amd|Pid=14228>INFO: Program[version(01.00.20170702), build date(Aug  1 2017, 18:41:23)] initialized OK, running...
<2017-08-01 18:41:34.395,   amdsock.cpp,1242,Ecd=-1214||Mod=amd|Pid=14228|Tdf=0>ERROR: ConnectError:I have failed to connect to HQ server[Local(:0), Remote[Listen](127.0.0.1:8016)].
<2017-08-01 18:41:34.396,   amdsock.cpp,1272,Ecd=0||Mod=amd|Pid=14228>INFO: Begin TCP connect ToUpper HQ server[127.0.0.1:8016] after 2 second...
<2017-08-01 18:41:34.397,   amdsock.cpp,1242,Ecd=-1214||Mod=amd|Pid=14228|Tdf=0>ERROR: ConnectError:I have failed to connect to HQ server[Local(:0), Remote[Listen](127.0.0.1:8018)].
<2017-08-01 18:41:34.397,   amdsock.cpp,1272,Ecd=0||Mod=amd|Pid=14228>INFO: Begin TCP connect ToUpper HQ server[127.0.0.1:8018] after 2 second...
<2017-08-01 18:41:34.397,   amdsock.cpp,0202,Ecd=0||Mod=amd|Pid=14228|Tdf=0>INFO: All Recv biz pkg count=0, All relay send lower pkg count=0.
<2017-08-01 18:41:34.397,   amdsock.cpp,0618,Ecd=0||Mod=amd|Pid=14228>INFO: Begin connect to local zone server [127.0.0.1:8023]...
<2017-08-01 18:41:35.402,   amdsock.cpp,1221,Ecd=-1214||Mod=amd|Pid=14228|Tdf=0>ERROR: ConnectError: I have failed to connect to zone server[Local(:0), Remote[Listen](127.0.0.1:8023)].
<2017-08-01 18:41:36.455,   amdsock.cpp,0618,Ecd=0||Mod=amd|Pid=14228>INFO: Begin connect to local zone server [127.0.0.1:8024]...
```

图 2　　日志文件示例

7. C++编程示例

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
#if defined(_WIN32) || defined(_WIN64)
#else
#include<unistd.h>
#endif

#include "szsimdapi.h"

template<typename T>
void print(char* psDesp,T t)
{
	std::cout<<psDesp<<"="<<t<<"\n";//std::endl;
}
template<>
void print(char* psDesp,std::string str)
{
	std::cout<<psDesp<<"="<<str.c_str()<<"\n";//std::endl;
}

#define PRT(v) print(#v,v)
```

```
void        OnSnapshotMD(void* pUserPtr, const STUSzsimdApiSnapshotMD *pSnapshotMD)
{

    std::cout<<"OnSnapshotMD
------------------------------------------------------------------------------\n";
    PRT(pSnapshotMD->m_ui32MsgType);
    PRT(pSnapshotMD->m_i64OrigTime);
    PRT(pSnapshotMD->m_ui16ChannelNo);
    PRT(pSnapshotMD->m_psMDStreamID);
    PRT(pSnapshotMD->m_psSecurityID);
    PRT(pSnapshotMD->m_psSecurityIDSource);
    PRT(pSnapshotMD->m_psTradingPhaseCode);
    PRT(pSnapshotMD->m_i64PrevClosePx);
    PRT(pSnapshotMD->m_i64NumTrades);
    PRT(pSnapshotMD->m_i64TotalVolumeTrade);
    PRT(pSnapshotMD->m_i64TotalValueTrade);

    switch (pSnapshotMD->m_ui32MsgType)
    {
    case 300111:
        {
            PRT(pSnapshotMD->m_ui32NoMDEntries);
            for (int i=0;i<pSnapshotMD->m_ui32NoMDEntries;i++)
            {

                std::cout<<"i="<<i<<"\n";
                PRT(pSnapshotMD->m_pArrayXianHuoPart[i].m_psMDEntryType);
                PRT(pSnapshotMD->m_pArrayXianHuoPart[i].m_i64MDEntryPx);
                PRT(pSnapshotMD->m_pArrayXianHuoPart[i].m_i64MDEntrySize);
                PRT(pSnapshotMD->m_pArrayXianHuoPart[i].m_ui16MDPriceLevel);
                PRT(pSnapshotMD->m_pArrayXianHuoPart[i].m_i64NumberOfOrders);
                PRT(pSnapshotMD->m_pArrayXianHuoPart[i].m_ui32NoOrders);
                for (int j=0;j<pSnapshotMD->m_pArrayXianHuoPart[i].m_ui32NoOrders;j++)
                {
                    PRT(pSnapshotMD->m_pArrayXianHuoPart[i].m_pArrayOrderQty[j]);
                }
            }
            break;
        }
    case 300611:
        {
            PRT(pSnapshotMD->m_ui32NoMDEntries);
```

```
                for (int i=0;i<pSnapshotMD->m_ui32NoMDEntries;i++)
                {

                    std::cout<<"i="<<i<<"\n";
                    PRT(pSnapshotMD->m_pArrayPanHouPart[i].m_psMDEntryType);
                    PRT(pSnapshotMD->m_pArrayPanHouPart[i].m_i64MDEntryPx);
                    PRT(pSnapshotMD->m_pArrayPanHouPart[i].m_i64MDEntrySize);
                }
                break;
            }
        case   306311:
            {
                PRT(pSnapshotMD->m_ui32NoMDEntries);
                for (int i=0;i<pSnapshotMD->m_ui32NoMDEntries;i++)
                {

                    std::cout<<"i="<<i<<"\n";

        PRT(pSnapshotMD->m_oHKStkExt.m_pArrayHKStkMDEntry[i].m_psMDEntryType);

        PRT(pSnapshotMD->m_oHKStkExt.m_pArrayHKStkMDEntry[i].m_i64MDEntryPx);

        PRT(pSnapshotMD->m_oHKStkExt.m_pArrayHKStkMDEntry[i].m_i64MDEntrySize);

        PRT(pSnapshotMD->m_oHKStkExt.m_pArrayHKStkMDEntry[i].m_ui16MDPriceLevel);
                }

                PRT(pSnapshotMD->m_oHKStkExt.m_ui32NoComplexEventTimes);
                for (int i=0;i<pSnapshotMD->m_oHKStkExt.m_ui32NoComplexEventTimes;i++)
                {

                    std::cout<<"i="<<i<<"\n";

        PRT(pSnapshotMD->m_oHKStkExt.m_pArrayHKStkComplexEvent[i].m_i64ComplexEventStartTime);

        PRT(pSnapshotMD->m_oHKStkExt.m_pArrayHKStkComplexEvent[i].m_i64ComplexEventEndTime);
                }

                break;
            }
```

```
	case	309011:
		{
			PRT(pSnapshotMD->m_ui32NoMDEntries);
			for (int i=0;i<pSnapshotMD->m_ui32NoMDEntries;i++)
			{

				std::cout<<"i="<<i<<"\n";
				PRT(pSnapshotMD->m_pArrayZhiShuPart[i].m_psMDEntryType);
				PRT(pSnapshotMD->m_pArrayZhiShuPart[i].m_i64MDEntryPx);
			}
			break;
		}
	case	309111:
		{
			PRT(pSnapshotMD->m_ui32_Tongji_StockNum);
			break;
		}
	default:
		std::cout<<"UnKnown MsgType:"<<pSnapshotMD->m_ui32MsgType<<std::endl;
	}


	std::cout<<"OnSnapshotMD END
------------------------------------------------------------------------------\n";


}
void	OnOneByOneWeiTuo(void* pUserPtr, const STUSzsimdApiOneByOneWeiTuo
*pOneByOneWeiTuo)
{
	std::cout<<"OnOneByOneWeiTuo
------------------------------------------------------------------------------\n";

	PRT(pOneByOneWeiTuo->m_ui32MsgType);
	PRT(pOneByOneWeiTuo->m_ui16ChannelNo);
	PRT(pOneByOneWeiTuo->m_i64ApplSeqNum);
	PRT(pOneByOneWeiTuo->m_psMDStreamID);
	PRT(pOneByOneWeiTuo->m_psSecurityID);
	PRT(pOneByOneWeiTuo->m_psSecurityIDSource);
	PRT(pOneByOneWeiTuo->m_i64Price);
	PRT(pOneByOneWeiTuo->m_i64OrderQty);
	PRT(pOneByOneWeiTuo->m_psSide);
```

```
	PRT(pOneByOneWeiTuo->m_i64TransactTime);
	switch (pOneByOneWeiTuo->m_ui32MsgType)
	{
	case 300192:
		{
			PRT(pOneByOneWeiTuo->m_psOrdType);
			break;
		}
	case 300592:
		{
			PRT(pOneByOneWeiTuo->m_psConfirmID);
			PRT(pOneByOneWeiTuo->m_psContactor);
			//std::string strValue=Utf8ToGBK(pOneByOneWeiTuo->m_psContactor);
			//PRT(strValue);
			//print("pOneByOneWeiTuo->m_psContactor",strValue);
			PRT(pOneByOneWeiTuo->m_psContactInfo);
			//std::string strValue2=Utf8ToGBK(pOneByOneWeiTuo->m_psContactInfo);
			//print("pOneByOneWeiTuo->m_psContactInfo",strValue2);
			break;
		}
	case  300792:
		{
			PRT(pOneByOneWeiTuo->m_ui16ExpirationDays);
			PRT(pOneByOneWeiTuo->m_ui8ExpirationType);
			break;
		}
	default:
		std::cout<<"UnKnown
MsgType:"<<pOneByOneWeiTuo->m_ui32MsgType<<std::endl;
	}
	std::cout<<"OnOneByOneWeiTuo END
----------------------------------------------------------------------------\n";

}
void	OnOneByOneChengJiao(void* pUserPtr, const STUSzsimdApiOneByOneChengJiao
*pByOneChengJiao)
{
	std::cout<<"OnOneByOneChengJiao
----------------------------------------------------------------------------\n";
	PRT(pByOneChengJiao->m_ui32MsgType);
	PRT(pByOneChengJiao->m_ui16ChannelNo);
	PRT(pByOneChengJiao->m_i64ApplSeqNum);
```

```
	PRT(pByOneChengJiao->m_psMDStreamID);
	PRT(pByOneChengJiao->m_i64BidApplSeqNum);
	PRT(pByOneChengJiao->m_psSecurityID);
	PRT(pByOneChengJiao->m_psSecurityIDSource);
	PRT(pByOneChengJiao->m_i64LastPx);
	PRT(pByOneChengJiao->m_i64LastQty);
	PRT(pByOneChengJiao->m_psExecType);
	PRT(pByOneChengJiao->m_i64TransactTime);
	std::cout<<"OnOneByOneChengJiao
END----------------------------------------------------------------------------\n";

}
void	OnChannelStatisticsMD(void* pUserPtr, const STUSzsimdApiChannelStatisticsMD
*pChannelStatisticsMD)
{
	std::cout<<"OnChannelStatisticsMD
--------------------------------------------------------------------------------\n";
	PRT(pChannelStatisticsMD->m_ui32MsgType);
	PRT(pChannelStatisticsMD->m_i64OrigTime);
	PRT(pChannelStatisticsMD->m_ui16ChannelNo);
	PRT(pChannelStatisticsMD->m_ui32NoMDStreamID);
	for (int i=0;i<pChannelStatisticsMD->m_ui32NoMDStreamID;i++)
	{
		std::cout<<"i="<<i<<"\n";
		PRT(pChannelStatisticsMD->m_pChannelStatisticsPart[i].m_psMDStreamID);
		PRT(pChannelStatisticsMD->m_pChannelStatisticsPart[i].m_ui32StockNum);
		PRT(pChannelStatisticsMD->m_pChannelStatisticsPart[i].m_psTradingPhaseCode);
	}

	std::cout<<"OnChannelStatisticsMD
END----------------------------------------------------------------------------\n";

}
void	OnStkRtStatusMD(void* pUserPtr, const STUSzsimdApiStkRtStatusMD
*pStkRtStatusMD)
{
	std::cout<<"OnStkRtStatusMD
--------------------------------------------------------------------------------\n";
	PRT(pStkRtStatusMD->m_ui32MsgType);
	PRT(pStkRtStatusMD->m_i64OrigTime);
	PRT(pStkRtStatusMD->m_ui16ChannelNo);
	PRT(pStkRtStatusMD->m_psSecurityID);
```

```
	PRT(pStkRtStatusMD->m_psSecurityIDSource);
	PRT(pStkRtStatusMD->m_psFinancialStatus);
	PRT(pStkRtStatusMD->m_ui32NoSwitch);
	for (int i=0;i<pStkRtStatusMD->m_ui32NoSwitch;i++)
	{
		std::cout<<"i="<<i<<"\n";
		PRT(pStkRtStatusMD->m_pStkRtStatusPart[i].m_ui16SecuritySwitchType);
		PRT(pStkRtStatusMD->m_pStkRtStatusPart[i].m_ui16SecuritySwitchStatus);
	}

	std::cout<<"OnStkRtStatusMD
END----------------------------------------------------------------------------\n";

}
void	OnMktRtStatusMD(void* pUserPtr, const STUSzsimdApiMktRtStatusMD
*pMktRtStatusMD)
{
	std::cout<<"OnMktRtStatusMD
------------------------------------------------------------------------------\n";
	PRT(pMktRtStatusMD->m_i64OrigTime);
	PRT(pMktRtStatusMD->m_ui16ChannelNo);
	PRT(pMktRtStatusMD->m_psMarketID);
	PRT(pMktRtStatusMD->m_psMarketSegmentID);
	PRT(pMktRtStatusMD->m_psTradingSessionID);
	PRT(pMktRtStatusMD->m_psTradingSessionSubID);
	PRT(pMktRtStatusMD->m_ui16TradSesStatus);
	PRT(pMktRtStatusMD->m_i64TradSesStartTime);
	PRT(pMktRtStatusMD->m_i64TradSesEndTime);
	PRT(pMktRtStatusMD->m_i64ThresholdAmount);
	PRT(pMktRtStatusMD->m_i64PosAmt);
	PRT(pMktRtStatusMD->m_psAmountStatus);
	std::cout<<"OnMktRtStatusMD
END----------------------------------------------------------------------------\n";

}
void	OnNotice(void* pUserPtr, const STUSzsimdApiNotice *pNotice)
//SzsimdApi_GetNoticeList 才会触发
{
	std::cout<<"OnNotice
------------------------------------------------------------------------------\n";
	PRT(pNotice->m_ui32MsgType);
	PRT(pNotice->m_i64OrigTime);
```

```
	PRT(pNotice->m_ui16ChannelNo);
	PRT(pNotice->m_psNewsID);
	PRT(pNotice->m_psHeadline);
	PRT(pNotice->m_psRawDataFormat);
	PRT(pNotice->m_ui32RawDataLength);
	PRT(pNotice->m_psRawData);
	std::cout<<"OnNotice
END----------------------------------------------------------------------------\n";
	/
}

bool g_running=true;

int main(int argc, char* argv[])
{
	std::ios::sync_with_stdio(false);

	int iErrCode = 0;
	char szErrString[G_SZSIMDAPI_MAXLEN_ERRORSTR];
	memset(szErrString, 0, sizeof(szErrString));

	STUSzsimdApiCallParam oCallParam;
	memset(&oCallParam, 0, sizeof(STUSzsimdApiCallParam));

	oCallParam.OnSnapshotMD=OnSnapshotMD;
	oCallParam.OnStkRtStatusMD=OnStkRtStatusMD;
	oCallParam.OnOneByOneWeiTuo=OnOneByOneWeiTuo;
	oCallParam.OnOneByOneChengJiao=OnOneByOneChengJiao;
	oCallParam.OnMktRtStatusMD=OnMktRtStatusMD;
	oCallParam.OnChannelStatisticsMD=OnChannelStatisticsMD;
	oCallParam.OnNotice=OnNotice;

	void* pHandle = SzsimdApi_Init("szsimdapi.ini", &oCallParam, &iErrCode, szErrString);
	if(pHandle==NULL)
	{
		printf("Init failed: %s.\n", szErrString);
		return -1;
	}

	if(SzsimdApi_Run(pHandle, 0)!=0)
	{
		printf("SzsimdApi_Run Error\n");
```

```
            return -1;
        }

        while(g_running)
        {
            SzsimdApi_GetNoticeList(pHandle);

#if defined(_WIN32) || defined(_WIN64)
            Sleep(10000);
#else
            sleep(10);
#endif

        }

        SzsimdApi_Stop(pHandle);
        SzsimdApi_Destroy(pHandle);
        return 0;
}
```

8. Java 接口

Java 接口相对于 C 语言接口，在使用上类似。使用方式为，通过继承接口类，实现接口方法，即可响应底层线程调用的回调函数。

8.1. Java 接口编程示例

```
package cn.com.szsi.szsimd;

import cn.com.szsi.szsimdapi.*;

public class Main {

    static {
        try {
            System.loadLibrary("szsimdapi");
        } catch (UnsatisfiedLinkError e) {
            System.err.println("Native code library failed to load. " + e);
            System.exit(1);
        }
    }
```

```
static class SzsimdApiCallback extends SzsimdApiInterface {
    public SzsimdApiCallback() {
        super();
    }

    //@Override
    public void OnSnapshotMD(STUSzsimdApiSnapshotMD snapshot) {
        snapshot.getM_i64NumTrades();
        snapshot.getM_i64OrigTime();
        snapshot.getM_i64PrevClosePx();
        snapshot.getM_i64TotalValueTrade();
        snapshot.getM_i64TotalVolumeTrade();
        snapshot.getM_psMDStreamID();
        snapshot.getM_i64NumTrades();
        if (snapshot.getM_ui32MsgType() == 300111) {
            for (int i = 0; i < snapshot.getM_ui32NoMDEntries(); i++) {
                STUSzsimdApiSnapshot_XianHuo part = snapshot.getXianHuoPart(i);
                part.getM_i64MDEntryPx();
                part.getM_i64MDEntrySize();
                part.getM_i64NumberOfOrders();
                part.getM_psMDEntryType();
                part.getM_ui16MDPriceLevel();
                for (int j = 0; j < part.getM_ui32NoOrders(); j++) {
                    part.getOrderQty(j);
                }
            }
        } else if (snapshot.getM_ui32MsgType() == 300611) {
            for (int i = 0; i < snapshot.getM_ui32NoMDEntries(); i++) {
                STUSzsimdApiSnapshot_PanHou part = snapshot.getPanHouPart(i);
                part.getM_i64MDEntryPx();
                part.getM_i64MDEntrySize();
                part.getM_psMDEntryType();
            }
        } else if (snapshot.getM_ui32MsgType() == 309011) {
            for (int i = 0; i < snapshot.getM_ui32NoMDEntries(); i++) {
                STUSzsimdApiSnapshot_ZhiShu part = snapshot.getZhiShuPart(i);
                part.getM_i64MDEntryPx();
                part.getM_psMDEntryType();
            }
        } else if (snapshot.getM_ui32MsgType() == 309111) {
            snapshot.getM_ui32_Tongji_StockNum();
```

```
            } else if (snapshot.getM_ui32MsgType() == 306311) {
                STUSzsimdApiSnapshot_HKStkExt hkStkExt = snapshot.getM_oHKStkExt();
                for (int i = 0; i < snapshot.getM_ui32NoMDEntries(); i++) {
                    STUSzsimdApiSnapshot_HKStk_MDEntry entry =
hkStkExt.getHKStkMDEntry(i);
                    entry.getM_i64MDEntryPx();
                    entry.getM_i64MDEntrySize();
                    entry.getM_psMDEntryType();
                    entry.getM_ui16MDPriceLevel();
                }
                for (int i = 0; i < hkStkExt.getM_ui32NoComplexEventTimes(); i++) {
                    STUSzsimdApiSnapshot_HKStk_ComplexEvent event =
hkStkExt.getHKStkComplexEvent(i);
                    event.getM_i64ComplexEventEndTime();
                    event.getM_i64ComplexEventStartTime();
                }
            }

        }

        //@Override
        public void OnOneByOneWeiTuo(STUSzsimdApiOneByOneWeiTuo tickOrder) {
            tickOrder.getM_i64ApplSeqNum();
            tickOrder.getM_i64OrderQty();
            tickOrder.getM_i64Price();
            tickOrder.getM_i64TransactTime();
            tickOrder.getM_psConfirmID();
            tickOrder.getM_psContactInfo();
            tickOrder.getM_psContactor();
            tickOrder.getM_psMDStreamID();
            tickOrder.getM_psOrdType();
            tickOrder.getM_psSecurityID();
            tickOrder.getM_psSecurityIDSource();
            tickOrder.getM_psSide();
            tickOrder.getM_ui8ExpirationType();
            tickOrder.getM_ui16ChannelNo();
            tickOrder.getM_ui16ExpirationDays();
            tickOrder.getM_ui32MsgType();
        }

        //@Override
        public void OnOneByOneChengJiao(STUSzsimdApiOneByOneChengJiao tickTrade) {
```

```
        tickTrade.getM_i64ApplSeqNum();
        tickTrade.getM_i64BidApplSeqNum();
        tickTrade.getM_i64LastPx();
        tickTrade.getM_i64LastQty();
        tickTrade.getM_i64OfferApplSeqNum();
        tickTrade.getM_i64TransactTime();
        tickTrade.getM_i64ApplSeqNum();
        tickTrade.getM_psExecType();
        tickTrade.getM_psMDStreamID();
        tickTrade.getM_psSecurityID();
        tickTrade.getM_psSecurityIDSource();
        tickTrade.getM_ui16ChannelNo();
        tickTrade.getM_ui32MsgType();
    }

    //@Override
    public void OnChannelStatisticsMD(STUSzsimdApiChannelStatisticsMD stat) {
        stat.getM_i64OrigTime();
        stat.getM_ui16ChannelNo();
        stat.getM_ui16ChannelNo();
        stat.getM_ui32MsgType();
        for (int i = 0; i < stat.getM_ui32NoMDStreamID(); i++) {
            STUSzsimdApiChannelStatisticsPart part = stat.getChannelStatisticsPart(i);
            part.getM_psMDStreamID();
            part.getM_psTradingPhaseCode();
            part.getM_ui32StockNum();
        }
    }

    //@Override
    public void OnStkRtStatusMD(STUSzsimdApiStkRtStatusMD status) {
        status.getM_i64OrigTime();
        status.getM_psFinancialStatus();
        status.getM_psSecurityID();
        status.getM_psSecurityIDSource();
        status.getM_ui16ChannelNo();
        status.getM_ui32MsgType();
        for (int i = 0; i < status.getM_ui32NoSwitch(); i++) {
            STUSzsimdApiStkRtStatusPart part = status.getStkRtStatusPart(i);
            part.getM_ui16SecuritySwitchStatus();
            part.getM_ui16SecuritySwitchType();
        }
```

```
        }

        //@Override
        public void OnMktRtStatusMD(STUSzsimdApiMktRtStatusMD status) {
            status.getM_i64OrigTime();
            status.getM_i64PosAmt();
            status.getM_i64ThresholdAmount();
            status.getM_i64TradSesEndTime();
            status.getM_i64TradSesStartTime();
            status.getM_psAmountStatus();
            status.getM_psMarketID();
            status.getM_i64OrigTime();
            status.getM_psMarketSegmentID();
            status.getM_psTradingSessionID();
            status.getM_psTradingSessionSubID();
            status.getM_ui16ChannelNo();
            status.getM_ui32MsgType();
        }

        //@Override
        public void OnNotice(STUSzsimdApiNotice notice) {
            notice.getM_i64OrigTime();
            notice.getM_psHeadline();
            notice.getM_psNewsID();
            notice.getM_psRawData();
            notice.getM_psRawDataFormat();
            notice.getM_ui16ChannelNo();
            notice.getM_ui32MsgType();
            notice.getM_ui32RawDataLength();
        }
    }

    public static void main(String[] args) {
        SzsimdApiInterface callback = new SzsimdApiCallback();
        SzsimdApiCaller caller = SzsimdApiCaller.createSzsimdApiCaller();
        if (caller != null) {
            caller.setInterface(callback);
            final int init = caller.Init("C:\\szsimdapi.ini");
            if (init == 0) {
                caller.Run(0);
                while (true) {
                    try {
```

```
                            Thread.sleep(1000);
                            // caller.GetNoticeList();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                } else {
                    System.out.println(init);
                }
            }
            return;
        }
}
```

9. 注意事项

9.1. C 语言 SzsimdApi 线程相关问题

SzsimdApi 的函数线程安全性可参见 4.2.11 小节和 4.3.1 小节。除公告消息回调函数外，其余回调函数由 SzsimdApi 底层线程负责执行，为了保证 SzsimdApi 的处理性能，不建议在回调函数中执行较耗时的函数，以免阻塞底层工作线程。

9.2. Java 语言 SzsimdApi 线程安全问题

Java 语言版本的线程安全性与 C 语言版本一致，在使用上可参考上一小节。