|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **深圳证券信息有限公司** | | 文档编号 |  | | | | |
| 名 称 | 深证信互联网行情转发系统API接口说明书 | | | | |
| **编写** | 签名： 日期： | | **密级** | 外部公开 | 版本 | V1.0.0 |
| **审核** | 签名： 日期： | | **批准** |  | | |

top_logo

**版权所有 不得复制**

**目 录**

[1. 引言 4](#_Toc498700807)

[2. 安装及应用发布 4](#_Toc498700808)

[2.1. C语言版本SzsimdApi 4](#_Toc498700809)

[2.2. Java语言版本SzsimdApi 4](#_Toc498700810)

[2.3. 关于配置的说明 5](#_Toc498700811)

[3. 使用概述 5](#_Toc498700812)

[3.1. 功能 5](#_Toc498700813)

[3.2. 应用环境 6](#_Toc498700814)

[3.3. 线程安全性 6](#_Toc498700815)

[3.4. 应用系统的安全性 6](#_Toc498700816)

[4. 编程参考 6](#_Toc498700817)

[4.1. 常量定义 6](#_Toc498700818)

[4.1.1. 长度常量 6](#_Toc498700819)

[4.1.2. 函数返回值 7](#_Toc498700820)

[4.2. 数据结构说明 7](#_Toc498700821)

[4.2.1. 字段说明 7](#_Toc498700822)

[4.2.2. STUSzsimdApiSnapshot\_XianHuo 7](#_Toc498700823)

[4.2.3. STUSzsimdApiSnapshot\_PanHou 8](#_Toc498700824)

[4.2.4. STUSzsimdApiSnapshot\_ZhiShu 8](#_Toc498700825)

[4.2.5. STUSzsimdApiSnapshot\_HKStk\_MDEntry 9](#_Toc498700826)

[4.2.6. STUSzsimdApiSnapshot\_HKStk\_ComplexEvent 9](#_Toc498700827)

[4.2.7. STUSzsimdApiSnapshot\_HKStkExt 10](#_Toc498700828)

[4.2.8. STUSzsimdApiSnapshotMD 10](#_Toc498700829)

[4.2.9. STUSzsimdApiOneByOneWeiTuo 12](#_Toc498700830)

[4.2.10. STUSzsimdApiOneByOneChengJiao 13](#_Toc498700831)

[4.2.11. STUSzsimdApiChannelStatisticsPart 14](#_Toc498700832)

[4.2.12. STUSzsimdApiChannelStatisticsMD 14](#_Toc498700833)

[4.2.13. STUSzsimdApiStkRtStatusPart 15](#_Toc498700834)

[4.2.14. STUSzsimdApiStkRtStatusMD 15](#_Toc498700835)

[4.2.15. STUSzsimdApiMktRtStatusMD 16](#_Toc498700836)

[4.2.16. STUSzsimdApiNotice 17](#_Toc498700837)

[4.2.17. STUSzsimdApiCallParam 18](#_Toc498700838)

[4.3. C函数接口 19](#_Toc498700839)

[4.3.1. 函数清单 19](#_Toc498700840)

[4.3.2. SzsimdApi\_Init 19](#_Toc498700841)

[4.3.3. SzsimdApi\_Run 19](#_Toc498700842)

[4.3.4. SzsimdApi\_Stop 20](#_Toc498700843)

[4.3.5. SzsimdApi\_Destroy 20](#_Toc498700844)

[4.3.1. SzsimdApi\_GetNoticeList 21](#_Toc498700845)

[4.3.2. 调用顺序 21](#_Toc498700846)

[5. 配置文件 21](#_Toc498700847)

[6. 日志文件 22](#_Toc498700848)

[7. C++编程示例 22](#_Toc498700849)

[8. Java接口 31](#_Toc498700850)

[8.1. Java接口编程示例 31](#_Toc498700851)

[9. 注意事项 36](#_Toc498700852)

[9.1. C语言SzsimdApi线程相关问题 36](#_Toc498700853)

[9.2. Java语言SzsimdApi线程安全问题 36](#_Toc498700854)

# 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，本公司拟建立深交所基础行情转发平台，为所有已授权基础行情客户提供深交所基础行情数据转发服务。

深证信互联网行情转发系统，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转发服务，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。

# 安装及应用发布

## 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 |

## 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 |

## 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. |

# 使用概述

## 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。

## 应用环境



1. SZSIMD API应用环境

在深证信互联网行情转发系统中，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。

## 线程安全性

参看下面4.3节。

## 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。

# 编程参考

## 常量定义

### 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 |

### 函数返回值

|  |  |
| --- | --- |
| **返回值** | **含义** |
| 0 | 成功 |
| -1 | 失败 |

## 数据结构说明

### 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) |

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。：

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。

### 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》。

## C函数接口

### 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 |

### 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 |

### 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 |

### 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 |

### 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 |

### 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 |

### 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。

# 配置文件

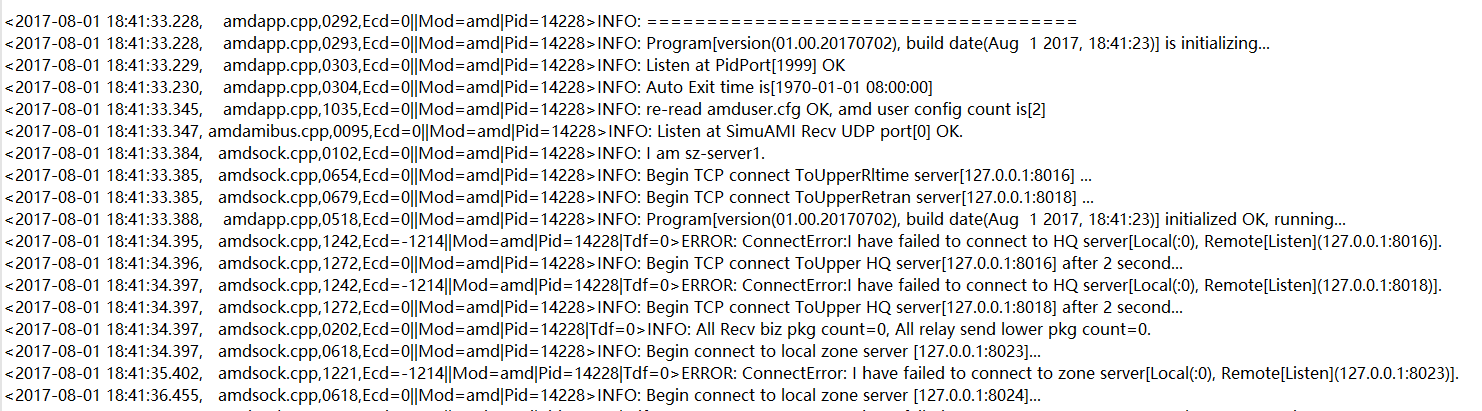
SzsimdApi需要配置文件(szsimdapi.ini)才能正常初始化和运行，配置文件信息如下：

|  |
| --- |
| [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= |

# 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。



1. 日志文件示例

# 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;  } |

# 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。

## 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;  }  } |

# 注意事项

## 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。

## 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。