深圳证券信息有限公司		文档编号			
		名　　称	深证信国证指数接口说明书		
编写	签名:　　日期:	**密级**	内部公开	版本	V1.0.1
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
V1.0.1	record_type 改为 index_type currency_type 改为 currency		2019-09-08

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、上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、上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 CNIndexApi

C 语言版本的 CNIndexApi 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、CentOS 7.2、RedHat 6.7、RedHat 7.2

C 语言版本的 CNIndexApi 由以下几个文件组成：

文件名	说明
cnindex_api.h	CNIndexApi 头文件
cnindex_api.lib	CNIndexApi 库文件
cnindex_api.dll	CNIndexApi 动态链接库文件，Windows 平台
libcnindex_api.so libboost_atomic.so.1.62.0 libboost_chrono.so.1.62.0 libboost_date_time.so.1.62.0 libboost_filesystem.so.1.62.0 libboost_locale.so.1.62.0 libboost_log_setup.so.1.62.0 libboost_log.so.1.62.0	CNIndexApi 动态链接库文件，Linux 平台

libboost_program_options.so.1.62.0 libboost_regex.so.1.62.0 libboost_system.so.1.62.0 libboost_thread.so.1.62.0 libmsg_parser.so	

2.2. Java 语言版本 CNIndexApi

Java 语言版本的 CNIndexApi 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、CentOS 7.2、RedHat 6.7、RedHat 7.2

Java 语言版本的 CNIndexApi 由以下几个文件组成：

文件名	说明
cnindex_api.jar	CNIndexApi jar 包
cnindex_api.dll	CNIndexApi 动态链接库文件，Windows 平台
libcnindex_api.so libboost_atomic.so.1.62.0 libboost_chrono.so.1.62.0 libboost_date_time.so.1.62.0 libboost_filesystem.so.1.62.0 libboost_locale.so.1.62.0 libboost_log_setup.so.1.62.0 libboost_log.so.1.62.0 libboost_program_options.so.1.62.0 libboost_regex.so.1.62.0 libboost_system.so.1.62.0 libboost_thread.so.1.62.0 libmsg_parser.so	CNIndexApi 动态链接库文件，Linux 平台

3. 使用概述

3.1. 功能

CNIndexApi 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，实现深证信互联网国证指数行情的接收，CNIndexApi 可以完成通信的自动连接，接

收数据包等功能，应用程序只需要设置好对应的回调函数就可以接收国证指数二进制行情数据，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

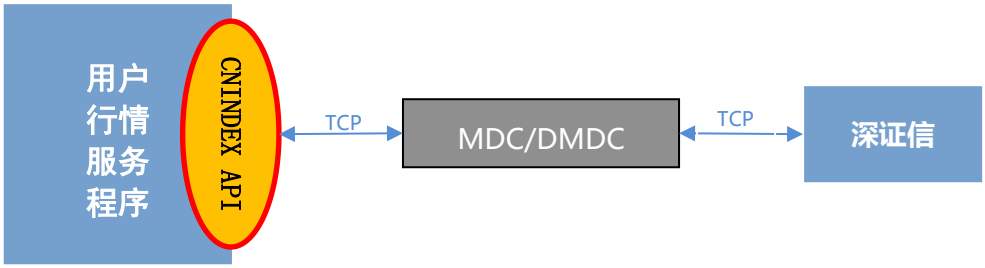

图 1　　CNINDEX API 应用环境

在深交所行情互联网接入服务中，MDC/DMDC 负责接收上游行情服务节点推送的行情数据，对下游提供深交所标准会话协议接口，CNINDEX API 通过直接连接 MDC/DMDC 获取行情数据。

3.3. 线程安全性

TCP

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 CNIndexApi， 在 CNIndexApi 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，CNIndexApi 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
API_MAXLEN_ERRORSTR	256	错误码字符串的最大长度

API_MAXLEN_LOGSTR	512	日志字符串的最大长度

4.1.2. 函数返回值

返回值	含义
0	成功
-1	无效指针
-2	无效参数

4.2. 数据结构说明

4.2.1. CNIndexConnInfo

该结构用来定义国证指数连接参数

结构定义：

```
struct CNIndexConnInfo
{
    char            ip[32];
    unsigned short  port;
    char            sender_comp_id[20];
    char            target_comp_id[20];
    char            password[16];
    unsigned short  heartbeat_interval;
    bool            multi_callback;
};
```

字段说明：

字段	说明
ip	服务地址
port	服务端口
sender_comp_id	SenderCompID
target_comp_id[	TargetCompID
password	Password
heartbeat_interval;	心跳间隔
multi_callback	多条记录回调标志 true：每次回调可能会有多条记录， false：每次回调一条记录

4.2.2. CNIndexMessage

该结构用来定义国证指数行情数据

结构定义：

```
struct CNIndexMessage
{
    int64_t     orig_time;
    char        index_type[2];
    char        security_id[8];
    char        symbol[40];
    double      pre_close_px;
    double      open_px;
    double      high_px;
    double      low_px;
    double      trade_px;
    double      close_px;
    double      total_volume_trade;
    double      total_value_trade;
    char        currency[3];
    char        trading_phase_code[8];
    double      close_px2;
    double      close_px3;
};
```

字段说明：

字段	说明
orig_time	行情发布时间戳
index_type	指数类型
security_id	指数代码
symbol	指数简称
pre_close_px	昨日收盘
open_px	开盘指数
high_px	最高指数
low_px;	最低指数
trade_px	实时指数
close_px	收盘指数
total_volume_trade;	成交量
total_value_trade	成交金额
currency	币种
trading_phase_code	指数实时阶段及标志
close_px2	收盘指数 2（亚太区）
close_px3	收盘指数 3（全球区）

4.2.3. CNIndexCallParam

该结构用来定义 CNIndexApi 的回调函数设置等

结构定义：

```
struct CNIndexCallParam
{
    void* user_ptr = nullptr;
    void(*OnLog)(void* user_ptr, const LOGLevel& level, const char
log_msg[API_MAXLEN_LOGSTR]) = nullptr;
    void(*OnCNIndexMessage)(void* user_ptr, const CNIndexMessage*
marketdata_snapshot_ptr, const unsigned int& count) = nullptr;
};
```

字段说明：

字段	说明	线程安全性
user_ptr	开发人员根据需要绑定的数据指针	无
OnLog	CnIndexApi 日志回调函数	不保证在同一线程
OnCNIndexMessage	国证指数数据回调函数	不保证在同一线程

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	CNIndexCreate	初始化相关资源	是
2	CNIndexStart	启动 API 的运行	是
3	CNIndexStop	停止 API 的运行	否
4	CNIndexDestroy	释放相关资源	否

4.3.2. CNIndexCreate

CNIndexApi 的初始化函数，在调用其它函数前必须先调用本初始化函数，分配获取相关的资源。

函数原型：

```
CNINDEX_API void* CNIndexCreate(const CNIndexConnInfo& conn_info, const
CNIndexCallParam& call_param, int& err_code, char err_msg[API_MAXLEN_ERRORSTR]);
```

参数说明 ：

参数	说明
conn_info[in]	配置连接信息
call_param[in]	API 的回调函数
err_code[out]	调用失败时的错误码
err_msg[out]	调用失败时的错误描述

返回值说明：

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

4.3.3. CNIndexStart

运行 API，建立连接，启用数据推送。

函数原型：

```
CNINDEX_API int     CNIndexStart(void* handle);
```

参数说明 ：

参数	说明
handle[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

4.3.4. CNIndexStop

停止 API 的运行，断开与上游直接的连接，停止数据推送。

函数原型：

```
CNINDEX_API int     CNIndexStop(void* handle);
```

参数说明 ：

参数	说明
handle[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. CNIndexDestroy

销毁 api 句柄

函数原型：

```
CNINDEX_API void    CNIndexDestroy(void* handle);
```

参数说明 ：

参数	说明
handle[in]	初始化返回的句柄

返回值说明：

返回值	说明
无	无

4.3.6. 调用顺序

按顺序依次调用 CNIndexCreate，CNIndexStart。若要停止 CNIndexApi，则按顺序调用 CNIndexStop，CNIndexDestroy。

5. 日志输出

CNIndexApi 的日志输出是以回调函数的方式返回给用户的，回调函数请参考 4.2.3 章节。常见的日志有程序的启动信息，网络建立和断开信息、登录成功或失败信息等。

6. C++编程示例

```
#include <string>
#include <iostream>
#include <cnindexapi/cnindex_api.h>


void OnLog(void* user_ptr, const LOGLevel& level, const char* log)
{
    std::cout << "LEVEL: " << level << "||" << log << std::endl;
}
```

```
void OnCNIndexMessage(void* user_ptr, const CNIndexMessage& message)
{
	std::cout << "OnCNIndexMessagebegin" << std::endl;
	std::cout << "orig_time: " << message.orig_time << std::endl;
	std::cout << "index_type[2]:" << message.index_type << std::endl;
	std::cout << "security_id[8]:" << message.security_id << std::endl;
	std::cout << "symbol[40]:" << message.symbol << std::endl;
	std::cout << "pre_close_px:" << message.pre_close_px << std::endl;
	std::cout << "open_px:" << message.open_px << std::endl;
	std::cout << "high_px:" << message.high_px << std::endl;
	std::cout << "low_px:" << message.low_px << std::endl;
	std::cout << "trade_px:" << message.trade_px << std::endl;
	std::cout << "close_px:" << message.close_px << std::endl;
	std::cout << "total_volume_trade:" << message.total_volume_trade << std::endl;
	std::cout << "total_value_trade:" << message.total_value_trade << std::endl;
	std::cout << "currency[3]:" << message.currency << std::endl;
	std::cout << "trading_phase_code[8]:" << message.trading_phase_code << std::endl;
	std::cout << "close_px2:" << message.close_px2 << std::endl;
	std::cout << "close_px3:" << message.close_px3 << std::endl;
	std::cout << "OnCNIndexMessageend" << std::endl;
}

int main(int argc, char* argv[])
{
	CNIndexConnInfo conn_info;
	memset(&conn_info, 0, sizeof(conn_info));
	strncpy_s(conn_info.ip, "192.168.102.48", sizeof(conn_info.ip));
	strncpy_s(conn_info.sender_comp_id, "VSS", sizeof(conn_info.sender_comp_id));
	strncpy_s(conn_info.target_comp_id, "VDE", sizeof(conn_info.target_comp_id));
	conn_info.heartbeat_interval = 10;
	conn_info.port = 9133;
	CNIndexCallParam callback;
	callback.user_ptr = nullptr;
	callback.OnLog = OnLog;
	callback.OnCNIndexMessage = OnCNIndexMessage;
	int32_t err_code;
	char err_msg[API_MAXLEN_ERRORSTR];
	void* handle;
	if ((handle = CNIndexCreate(conn_info, callback, err_code, err_msg)) != nullptr)
	{
		std::cout << "CNIndexCreate success" << std::endl;
```

```
    }
    else
    {
        std::cout << "CNIndexCreate failed, err_code: " << err_code << ", err_msg: " << err_msg
<< std::endl;
        return -1;
    }

    if (CNIndexStart(handle) == 0)
    {
        std::cout << "CNIndexStart success" << std::endl;
    }
    else
    {
        std::cout << "CNIndexStart failed" << std::endl;
        return -1;
    }
    getchar();
    if (CNIndexStop(handle) == 0)
    {
        std::cout << "CNIndexStop success" << std::endl;
    }
    else
    {
        std::cout << "CNIndexStop failed" << std::endl;
        return -1;
    }

    CNIndexDestroy(handle);
    getchar();
    return 0;
}
```

7. 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。

7.1. Java 接口编程示例

```
import com.archforce.amd.cnindexapi.*;

import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Arrays;


public class Test {
    static String dataPath = "./data.csv";

    static {
        try {
            String libPath = System.getProperty("user.dir") + "/lib/cnindex_api.dll";
            System.out.println(libPath);
            System.load(libPath);
            ArrayList<ArrayList<String>> csvData = new ArrayList<ArrayList<String>>();
            csvData.add(new ArrayList<String>(Arrays.asList("发布时间", "记录类型", "指数代码", "
指数简称", "昨日收盘", "开盘指数", "最高指数",
                    "最低指数", "实时指数", "收盘指数", "成交量", "成交金额", "币种", "指数实时阶段
及标志", "收盘指数 2（亚太区）", "收盘指数 3（全球区）")));
            ArrayToCSV(csvData, dataPath, false);
        } catch (Exception e) {
            System.err.println("Native code library failed to load. " + e);
            System.exit(1);
        }
    }

    public static void ArrayToCSV(ArrayList<ArrayList<String>> data, String path, Boolean
append){
        try{
            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(path, append), "UTF-8"));
            for (int i = 0; i < data.size(); ++i){
                ArrayList<String> row = data.get(i);
                for(int j = 0; j < row.size(); ++j){
                    out.write(DelQuota(row.get(j)));
                    out.write(",");
                }
                out.newLine();
            }
```

```
                out.flush();
                out.close();
            }catch(Exception e){
                e.printStackTrace();
            }
        }

        public static String DelQuota(String str){
            String result = str;
            String[] strQuota = {"~", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "`", ";", "'", ",", "/", ":",
"/,", "<", ">", "?"};
            for (int i = 0; i < strQuota.length; ++i){
                result = result.replace(strQuota[i], "");
            }
            return result;
        }

        static class CNIndexSpiImpl extends CNIndexSpiInterface {
            public CNIndexSpiImpl() {
                super();
            }

            @Override
            public void OnLog(LOGLevel logLevel, String s) {
                super.OnLog(logLevel, s);
                System.out.println(s);
            }

            @Override
            public void OnCNIndexMessage(CNIndexMessage cnIndexMessage, long l) {
                super.OnCNIndexMessage(cnIndexMessage, l);
                System.out.printf("Receive CNIndexMessage message, count: %d", l);
                ArrayList<ArrayList<String>> csvData = new ArrayList<ArrayList<String>>();
                for(long i = 0; i < l; ++i){
                    csvData.add(new
ArrayList<String>(Arrays.asList(cnIndexMessage.getOrig_time()+"",cnIndexMessage.getIndex_ty
pe(), cnIndexMessage.getSecurity_id(), cnIndexMessage.getSymbol(),
cnIndexMessage.getPre_close_px()+"",
                            cnIndexMessage.getOpen_px()+"", cnIndexMessage.getHigh_px()+"",
cnIndexMessage.getLow_px()+"", cnIndexMessage.getTrade_px()+"",
cnIndexMessage.getClose_px()+"", cnIndexMessage.getTotal_volume_trade()+"",
                            cnIndexMessage.getTotal_value_trade()+"",
```

```
cnIndexMessage.getCurrency()+"", cnIndexMessage.getTrading_phase_code()+"",
cnIndexMessage.getClose_px2()+"", cnIndexMessage.getClose_px3()+"")));
                }
                ArrayToCSV(csvData, dataPath, true);
            }
        }


        public static void    main(String[] args) {
            CNIndexSpiInterface spi = new CNIndexSpiImpl();
            CNIndexApiInterface api = CNIndexApiInterface.CreateCNIndexApi(spi);
            if(null != api){
                CNIndexConnInfo connInfo = new CNIndexConnInfo();
                connInfo.setIp("192.168.102.48");
                connInfo.setPort(9133);
                connInfo.setSender_comp_id("VSS");
                connInfo.setTarget_comp_id("VDE");
                connInfo.setHeartbeat_interval(10);
                connInfo.setPassword("");

                if(api.Init(connInfo) == 0){
                    System.out.println("Init success");
                }
                else{
                    System.out.println("Init failed");
                    return;
                }
                if(api.Start() == 0){
                    System.out.println("Start success");
                }
                else{
                    System.out.println("Start failed");
                    return;
                }
                while(true){
                    try {
                        Thread.sleep(100000000);
                        break;
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
```

```
                if(api.Stop() == 0){
                    System.out.println("Stop success");
                }
                else{
                    System.out.println("Stop failed");
                    return;
                }
            }
        }
}
```

8. 注意事项

8.1. C 语言 CNIndexApi 线程相关问题

CNIndexApi 的函数线程安全性可参见 4.3.1 章节。回调函数由 CNIndexApi 底层线程负责执行，为了保证 CNIndexApi 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。

8.2. Java 语言 CNIndexApi 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。