郑州商品交易所
ZCEAPI 使用手册
（2017）

Version1.1.3.3

郑州商品交易所

2017 年 08 月

目录

0. 修改历史

API 版本	修改时间	修改内容	说明
1.1.3.3	2017-08-01	1） 增加 Linux 平台下库文件说明及其使用说明。 2） 修订支持郑商所域名解析功能 API_ConnectEx 函数的参数说明。 3） 修订使用部分，说明 API 目前只支持多线程模式。 4） 修改 1.5 章节中关于私有流的说明，删掉私有流接收对话流的请求应答包的描述。 5） 修改 1.6 章节中新版 ZCEAPI 做过兼容性测试的平台列表。 6） 删除一些无关冗余文字，精简使用描述。 7） 修改使用环境等时效性较强的说明。 8） 针对 Linux 平台上的使用，新增对 API_SetConnectionOpt 接口的说明。	适应新版修改；完善部分描述，更便于用户理解
1.1.1.9	2014-10-10	1）登录协议版本号目前只支持 2 和 11，不再支持 1。 2）删除日志备份文件 ftdapi.log.bak。日志文件中的时间由原来的秒精确到微秒，日志文件中加入版本号信息。日志内容改为英文。 3）API 回调信息改为全为英文。 4）新增得到 API 版本号接口 ftd_api_getVersion。 5）新增发起与交易所连接时的域名解析服务函数 API_ConnectEx。 6）增加 API 初始化函数 ftd_api_init 必须最先调用的限制。 7）增加业务字段赋值时的类型判断，重新定义赋值函数返回值。 8）修改日期时间的类型定义，名称改为 API_DateTime，微秒类型改为 int 型。 9）修改 API_Now 函数，把微秒由原来的 0 改为实际值。 10）用 string 格式取 double 值，根据存的精度输出。 11）修改 API_GFString 和 API_CFString 函数取时间型数据域时微秒值为 6 位输出。 12）修改 API_Send 函数不能发送登录、登出数据包。 13）完善了连接，登录，未知数据，线程启动	为了适用新版本改动

API 版本	修改时间	修改内容	说明
		和退出等的日志输出。	
1.1.0.1	2007-12-19	1）登录协议版本号，不再是默认 1，必须填写合适的值。 2）增加登录协议版本号 2 和 11。 登录协议版本号=2　广播流使用 TCP 登录协议版本号=11 广播流使用 UDP	为了适用新版本改动

1. 概述

1.1 目的及说明

本文档是郑州商品交易所远程交易 API（应用程序接口）的使用说明，本文档的目标读者为使用郑州商品交易所发布的 API 与交易所交易系统实现通信的交易软件开发者。文档对采用郑商所 API 的开发者具有指导意义，所有使用 API 的开发者都应该认真阅读该文档。

文档描述内容如果与实际情况不符，请及时和郑州商品交易所联系并确认。

文档的最终解释权归郑州商品交易所。

1.2 相关文档

《郑州商品交易所 ZCEAPI 参考手册》（下简称《ZCEAPI 参考手册》），该文档随同本文档一同由郑州商品交易所发布。

ZCEAPI 示例程序展示了 ZCEAPI 多线程模式下的使用，具体情况请参考示例程序中的相应说明，示例程序随本文档一同由郑州商品交易所发布。

1.3 ZCEAPI 简介

ZCEAPI 是郑州商品交易所的远程交易编程接口，是客户应用程序与郑州商品交易所交易系统连接通信的接口。它为程序员提供了一套应用程序接口和一套简单的交易数据报文协议。

ZCEAPI 采用面向对象的思想对 FTD 协议进行了封装。用户在使用 ZCEAPI 向交易所发送数据时只需要参照《ZCEAPI 参考手册》为分配好的 ZCEAPI 数据包填写相应的值，再调用相应的 ZCEAPI 接口函数发送 ZCEAPI 数据包。

心跳机制由 ZCEAPI 自行维护。数据加密、解密，压缩、解压等工作由 ZCEAPI 内部完成，只需用户选择相应的参数。

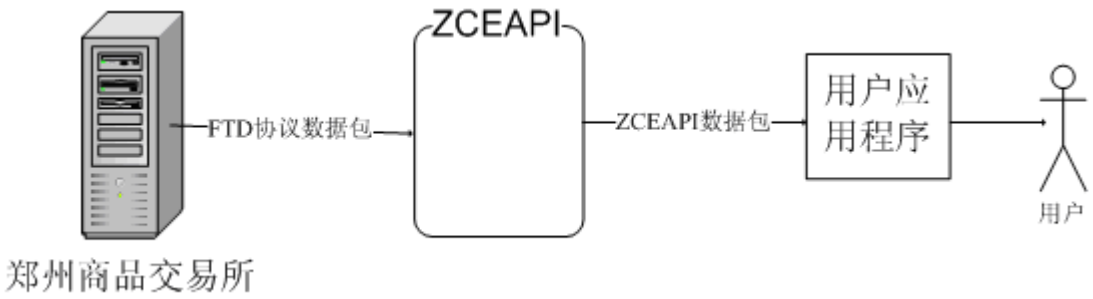
ZCEAPI 使用回调机制，当链路有错误发生、打开链路、链路断开、收到登录应答包、收到退出应答包、接收到普通数据包等事件发生时，ZCEAPI 自动回调用户设置的回调函数进行处理。

总之，使用 ZCEAPI 可以很方便的与郑州商品交易所交易系统实现通信，完成事务操作。

1.4 ZCEAPI 所处位置

ZCEAPI 是在应用程序端，为交易软件开发人员使用，并作为应用程序的一部分而

在交易系统的客户端存在（如下图所示）：



1.5 ZCEAPI 通信简介

ZCEAPI 采用 TCP 可靠连接与交易系统进行通讯，通讯方式采用对话通讯模式、私有通讯模式和广播通讯模式。用户程序通过 ZCEAPI 建立可靠的 TCP 链路连接郑州商品交易所的交易系统，完成交易业务和获取行情信息。

对话模式（对应的数据流为对话流），实现了所有的事务操作和查询操作功能，可以完成“交易系统”与“会员系统”之间数据交换的全部功能。

私有模式（对应的数据流为私有流），是对话模式的补充功能，是可靠的数据连接。主要功能包括：

1) 交易系统定单薄中每一条记录的状态或数量发生变化，交易系统通过私有流发送报单状态确认报文或跨期套利确认、套期保值确认、期权执行确认、报价响应确认报文。

2) 交易系统成交记录发生变化时，交易系统通过私有流发送单边成交回报报文。

广播模式（对应的数据流是广播流），是对话模式的补充功能，交易系统将系统状态、合约信息、交易所公告信息和市场行情信息等公共信息在其发生变化的时刻通过广播流发送给会员系统，减少会员系统查询交易系统的次数，提高信息反馈速度。

当用户登录广播流采用 11 协议时，系统会采用 UDP 广播行情变化等信息。具体参见《郑州商品交易所 ZCEAPI 参考手册》。

1.6 ZCEAPI 使用环境

ZCEAPI 支持的操作系统平台主要有：Windows 和 Linux。并分别提供 32 位和 64 位版本。ZCEAPI 在发布前，在如下平台做过基本的兼容性测试（随着系统平台的持续更新，客户可自行选择后续发行版本，并做好兼容性和功能性测试，如遇到问题需要协助，请及时和郑商所取得联系）：

- Red Hat Enterprise Linux Server release 6.5 (Santiago) X64

内核版本：2.6.32-431.el6.x86_64

gcc 版本： 4.4.7 20120313 (Red Hat 4.4.7-17)

- CentOS release 6.5 (Final) X32

内核版本：2.6.32-431.el6.i686

gcc 版本： 4.4.7 20120313 (Red Hat 4.4.7-17)

- Windows 8.1 专业版 64-bit (9600.130821-1623)

VS 版本：Visual Studio 2013

- Windows 7 旗舰版 64-bit (6.1, Build 7601) Service Pack 1 (7601.130828-1532)

VS 版本：Visual Studio 2013

Windows 平台下它以动态库（.dll）的形式被应用程序调用。在 Linux/Unix 平台下它以动态库（.so）的形式被应用程序调用。相应平台下的动态库文件随本文档发布。

使用该 API 不需要进行特殊的系统配置和安装。用户只需从郑州商品交易所获得相应平台下的动态库文件及相应的辅助文件即可使用。

1.7 ZCEAPI 相关文件说明

发布的 ZCEAPI 相关文件如下：

Windows 平台下：

FTDDLL.dll ---------- ZCEAPI 的动态库。

FTDDLL.lib ---------- ZCEAPI 动态库引导库文件，C++用户开发使用。

Config.h ---------- ZCEAPI 类型和常量声明文件，和 FTDAPI.h 文件配合使用。用于 C++用户开发。

FTDAPI.h ---------- ZCEAPI 的接口定义文件，C++ 用户可以直接使用（结合 Config.h），其他语言用户可以根据文件中的 C++ 形式的接口函数定义，得到相应语言下的接口函数声明及类型定义。

FieldID.h ---------- ZCEAPI 数据包的数据域常量定义文件，C++用户可以直接使用。其他语言的用户可以根据其具体情况作出相应的转化。

Linux 平台下：

libZCEFTDAPI.so ---- ZCEAPI 的动态库文件。

Config.h ---------- ZCEAPI 类型和常量声明文件，和 FTDAPI.h 文件配合使用。用于 C++用户开发。

FTDAPI.h ---------- ZCEAPI 的接口定义文件，C++用户可以直接使用（结合 Config.h）。

FieldID.h ---------- ZCEAPI 数据包的数据域常量定义文件，C++用户可以直接使用。

2. ZCEAPI 使用说明

2.1 使用概述

郑州商品交易所应用编程接口（ZCEAPI）具有功能强大使用方便的特点。

ZCEAPI 在系统中以动态库的形式出现，提供的是标准 C 接口函数和符合 C（C++）语法定义的数据类型。这些函数和类型的定义请参见本文档第 4 部分的说明。

用户要使用 ZCEAPI 请务必从郑州商品交易所得到相应平台下的动态库文件（Windows 平台下是 FTDDLL.dll，Linux 平台下是 libZCEFTDAPI.so），这是使用 ZCEAPI 所必需的文件。

一般情况下使用动态链接库，程序员都必须对动态库提供的接口做出声明。在使用

ZCEAPI 时所要做的声明分两部分---接口函数和基本数据定义。C++程序员可以直接使用随动态库发布的 FTDAPI.h 文件和 Config.h 文件（FTDAPI.h 引用了 Config.h），FTDAPI.h 和 Config.h 提供了 ZCEAPI 的所有的声明。其他语言用户，请根据 FTDAPI.h 中关于 ZCEAPI 的接口声明或本文档下面相应部分的说明，声明出相应语言的 ZCEAPI 接口。

另外，因为 ZCEAPI 是一个用来进行数据交换的接口，所以接口提供了自己的数据包格式。用户可以根据相应的业务填写或读取相应的数据域字段。对于业务和数据域 ZCEAPI 也作出了相应的编码。有关编码的对应表请参见《郑州商品交易所 ZCEAPI 参考手册》中的相关说明。用户也可根据《郑州商品交易所 ZCEAPI 参考手册》中的定义结合所使用的语言在程序中给出常量定义。

至于各种平台下的标准动态库的使用方法，请参考相应平台下的相关技术文档或书籍的介绍。

使用 ZCEAPI 实际就是对 ZCEAPI 提供的接口的使用。对 ZCEAPI 的调用有一定的先后顺序：首先，调用 ftd_api_init 函数初始化 API，这是用户必须做成功的一步；其次根据要登录的数据流建立相应的连接对象（**目前，郑州商品交易所交易系统只支持一个连接对象登录一个数据流（1.5 中的说明）**），因为 ZCEAPI 采用回调的方法处理事件，所以在每个连接上请设置相应的事件处理回调函数；之后建立与交易所的连接，建立连接后就可登录相应的数据流并发送和接收交易数据信息；完成所有处理后，请释放创建的连接对象，并调用 ftd_api_stop 函数停止 API 服务。

ZCEAPI 的详细使用说明请参考 2.3 中的说明。

2.2 线程模式简介

ZCEAPI 目前只提供多线程模式给用户使用。ZCEAPI 采用回调函数的方式处理事件，多线程模式下用户创建一个连接对象，ZCEAPI 就会为其启动一个线程驱动这个连接对象上的事件。对用户来说，只要设置好相应的回调函数，所有的事件处理是自动进行的。

2.3 基本过程

多线程模式下 ZCEAPI 会自行启动线程驱动 ZCEAPI 中可能发生的如接收数据、链路出错等事件，用户只需通过回调函数做相应的处理。

下面是 ZCEAPI 在多线程模式下使用的基本过程：

2.3.1 ZCEAPI 初始化

用户调用 ftd_api_init 函数初始化 ZCEAPI（使用模式为多线程模式）。

ZCEAPI 成功初始化是下面一切工作的基础。若 ZCEAPI 不能成功初始化，则不能保证

ZCEAPI 的正常运行。

例如：

```
if(ftd_api_init(MULTI_THREAD_MODE,"d:\\",2222)!=API_TRUE)
{
    cout<<"Fail in Init ZCEAPI ! "<<endl;
    return 0; //ZCEAPI 不能正确初始化则退出
}
else
{
    cout<<"Init ZCEAPI Successfully !"<<endl;
}
```

注：关于初始化函数 ftd_api_init 的使用，请参考本手册第 4 部分对该函数的介绍。

2.3.2 创建连接交易所对象

用户调用 API_CreateExchgConnection 函数创建交易所连接对象。用户要使用 ZCEAPI 连接到交易所，创建连接对象是必须的。创建函数返回的连接对象句柄是与交易所通信的基础。

例如：

```
API_BOOL     Encrypt = API_TRUE;
API_BOOL     Commpress = API_TRUE;
MARKET_ID  Market_ID = 1;
int     heart_beat_interval = 10; //心跳间隔
int     heart_beat_timeout = 100;
ExchgConnectionHandle conn = API_CreateExchgConnection (Encrypt , Commpress , Market_ID, heart_beat_interval, heart_beat_timeout);
 if (conn==NULL)
 {
    cout<<"  Can not create connection to CZCE "<<endl;
    return 0;
 }
```

注：关于创建交易所连接对象函数 API_CreateExchgConnection 的使用，请参考本手册第 4 部分对该函数的介绍。创建连接对象并不等于建立对交易所的连接。

2.3.3 设置链路的回调函数

在成功创建链路后就可设置该链路的回调函数。（建议这时设置相应的回调函数）

回调函数可全部设置，也可有选择地设置。

虽然回调函数的设置不是必须的（如果用户对链路上发生的事情不关心），不过建议用户设置如链路出错，链路断开等回调函数，因为这些事件的发生会对用户程序产生很大的影响。

2.3.3.1 定义回调函数

设置回调函数之前要先定义相应的回调函数。ZCEAPI 中的回调函数分为两类，一类是链路状态回调函数，一类是返回数据包回调函数。

回调函数的类型定义请参考本手册第 3 部分中有关回调函数的定义。

例如：

```
//定义出错的回调函数
void errorsShow(void * CallBackArg,ExchgConnectionHandle,int error_code,const char* error_text)
{
    cout<<"链路出错！ "<<"错误信息为"<<error_text<<" 错误代码为："<<error_code<<endl;
}

//定义登录应答的回调函数，这是收到登录应答报文后的 ZCEAPI 回调
//可以从登录应答 ZCEAPI 数据包中得到想要的信息
int OnAPILogin(void * CallBackArg,ExchgConnectionHandle conn,MsgPackageHandle pkg)
{
    int tempnum;
    char tempbuf1[256],tempbuf2[256];  //接受 ZCEAPI 数据包字段的临时缓存
    tempnum=API_GFString(pkg,FID_ParticipantId,tempbuf1,256);//取对应 ID 的字符串
    tempbuf1[tempnum]='\0';  //设置取得的字符串的结尾标示
    tempnum=API_GFString(pkg,FID_UserId,tempbuf2,256);//取对应 ID 的字符串
    tempbuf2[tempnum]='\0';  //设置取得的字符串的结尾标示
    string str="收到登录应答包！ 交易会员编码为：";
    str+=tempbuf1;
    str+=" 交易员编码：";
    str+=tempbuf2;
    cout<<str<<endl;
    return 0;
}
```

```
//断开链路回调函数，可以通过这个回调函数及时得知链路断开，并做出相应的处理
void OnAPIClose(void * CallBackArg,ExchgConnectionHandle,int error_code,const char* error_text)
{
    cout<<"链路断开！"<<"错误信息为：" <<error_text<<" 错误代码为：" <<error_code<<endl;
}
```

2.3.3.2 设置回调函数

定义回调函数后，就可调用 ZCEAPI 相应的函数进行回调函数的设置。

例如：

```
//设置链路出错回调
API_SetErrorCallBack(conn, errorsShow,NULL);   //第三个参数是一个回调参数
//设置链路断开回调函数
API_SetCloseCallBack(conn, OnAPIClose,NULL);
//设置登录成功回调函数
API_SetLoginCallBack(conn, OnAPILogin,NULL);
```

注：关于设置回调函数接口的具体介绍请参考本手册第 4 部分中关于设置回调函数的函数的介绍。

2.3.4 连接交易所

ZCEAPI 提供两个发起与交易所连接函数：API_Connect 和 API_ConnectEx。

用户调用 API_Connect 函数发起对交易所的连接。连接成功则打开一条到交易所的连接通道（API_Connect 函数的第一个参数为连接对象，连接成功此连接对象被打开。以后就可以利用这个连接对象和交易所通信）。

例如：

```
//发起对交易所的连接
if ( API_Connect(conn,IpAddr,22677,5)!=API_TRUE)   //连接不成功
{
    cout<<"  Can not connect with CZCE！ "<<endl;
    return 0;
}
else  cout<<"  Connect CZCE successfully "<<endl;
```

用户调用 API_ConnectEx 函数发起对交易所的连接，不同于 API_Connect 函数的直连交

易所，API_ConnectEx 函数首先连接域名解析服务器，域名解析服务根据交易所各前置机的负载状况随机分配可连接的交易所前置地址，通过该地址，用户即可实现与交易所的连接。使用 API_ConnectEx 函数需要指定域名解析服务 IP 地址和端口、席位号、前置版本等信息。连接成功则打开一条到交易所的连接通道（API_ConnectEx 函数的第一个参数为连接对象，连接成功此连接对象被打开，以后就可以利用这个连接对象和交易所通信）。

例如：

```
//发起对交易所的连接
if(API_ConnectEx(conn, DNSIp, DNSPort, 5, UserID, gwVersion, '0', gwIP, 22677, errMsg)!=0）//连接不成功
{
    cout<<" Can not connect with CZCE！"<<endl;
    cout<<"Error Message："<<errMsg<<endl;
    return 0;
}
else cout<<" Connect CZCE successfully！"<<endl;
```

注：1），关于连接函数 API_Connect 和 API_ConnectEx 请参考本手册第 4 部分中对该函数的介绍。

2），交易所一般都有加密和非加密两个端口，连接时根据连接对象创建时选择是否加密连接相应的端口。如：在 2.3.2 中创建了加密(加密必压缩)的连接对象 conn（严格的说是 conn 指向的对象）。在测试环境，交易所提供两个端口，一个是加密端口，端口号为 22677，另一个是非加密端口，端口号为 22577。但在正式交易环境交易所只提供加密端口，端口号为 22677。（具体端口号以交易所通知为准）

3），用户在建立和交易所的连接后，应尽快在此连接上登录，否则一个空连接(不登录任何数据流的连接)一段时间后会自动断开。

2.3.5 登录交易所

在建立与交易所的连接后就可以在该连接对象上登录。**用户在一个连接对象上只能登录对话流、私有流、广播流中的一个流。**

登录一个流的具体过程如下：

① 调用 API_AllocPackage 函数生成一个 ZCEAPI 数据包

② 调用 API_SetPID 函数设置数据包 PID（登录报文的 PID 为：0x00016）

③ 调用相应的字段操作函数填写数据包字段（如：API_SFInt (APIpkg, FID_SequenceNo, 1)）

④ 调用 API_Login 函数登录交易所

⑤ 根据 API_Login 函数的返回值判断是否登录成功

⑥ 调用 API_FreePackage 释放刚才用的那个数据包

例如：

```
bool testLogin(ExchgConnectionHandle conn, short int DF)
{
    //先定义要使用的变量
    MsgPackageHandle    APIpkg;      //a pointer to a APIpkg;
    short int DataFlowFlag =DF;      //数据流号 , 0->对话流,1->私有流,2->广播流
    char ParticipantId[8]="0018";    //交易会员编码
    char UserId[15]="00180090";      //交易员编码
    char Password[20]="88888";       //口令
    char ProtocolVersion[10]="2";    //使用 FTD 版本号  目前=2 或 11
    char IpAddr[20]="192.168.0.8";   //登录者的 IP 地址  如 192.168.99.100
    char INIT_PASSWORD[20]="12345678";                    //协商初始密钥
    char AUTH_SERIAL_NO[20]="12345678-12345678" ;    //序列号
    char AUTH_CODE[20]="12345678-12345678" ;             //授权码
    int FRspSeqNO =0;                                    //请求序列号
    int LoginTimeout=20 ;

    APIpkg = API_AllocPackage();
    //下面填写登录数据包然后再将他们发送出去
    API_SetPID(APIpkg,0x00016);    //设置登录 PID 为十六进制的 00016
    API_SFInt(APIpkg ,FID_DataFlowFlag ,DataFlowFlag);
    API_SFString(APIpkg,FID_ParticipantId ,ParticipantId,strlen(ParticipantId));
    API_SFString(APIpkg ,FID_UserId ,UserId,strlen(UserId));
    API_SFString(APIpkg,FID_Password ,Password,strlen(Password));
    API_SFString(APIpkg ,FID_IpAddr ,IpAddr,strlen(IpAddr));
    API_SFString(APIpkg ,FID_AUTH_SERIAL_NO,AUTH_SERIAL_NO,
                    strlen(AUTH_SERIAL_NO));
    API_SFString(APIpkg ,FID_AUTH_CODE,AUTH_CODE,strlen(AUTH_CODE));
    API_SFString(APIpkg ,FID_INIT_PASSWORD ,INIT_PASSWORD,
                    strlen(INIT_PASSWORD));
    API_SFInt(APIpkg ,FID_SequenceNo ,FRspSeqNO);
    //登录包设置后发送出去
    if ( API_Login (conn,APIpkg,LoginTimeout)== API_TRUE )
```

```
    {
        cout<<"  Login successfully！"<<endl;
        return ture;
    }
    else
    {
        cout<<“Fail in login！"<<endl;
        return false;
    }
    //释放掉我们分配的 ZCEAPI 数据包对象
    API_FreePackage(APIpkg);
}
```

注：1），具体函数的使用请参考本手册第 4 部分对其具体的介绍。

2），分配的数据包用过之后不一定要释放，可以再次使用，但是需要根据实际情况修改数据包中的内容，或者清空该数据包。

3），用户可能会因为 API_Login 函数中登录超时值设置的过小或网络状况不佳，产生登录超时的情况。超时后，用户还可以通过设置登录应答回调函数得到登录应答包。理论上讲，用户可以通过登录应答包而得到具体的登录状态，但是不建议这么做。用户应当调整 wait 的值。

4），因为登录过程中可能会有错误发生，用户若要将 API_Login 函数中的 wait 设为合适的值，然后忽略其返回值，只以是否收到正确的成功登录应答包为登录成功与否的标志，则务必设置错误回调函数。否则，有可能会因为错误的发生而始终得不到登录应答报文。

5），对话流、私有流和广播流 3 个流必须使用相同的登录协议版本号。

即使使用登录协议版本号 11，采用 UDP 方式传送广播流数据，广播流也必须登录。

ProtocolVersion=2, 广播流使用 TCP 方式

ProtocolVersion=11, 广播流使用 UDP 方式

2.3.6 读写数据包

像在 2.3.3.1 和 2.3.5 的例子中看到的，可以在得到一个合法的 ZCEAPI 数据包句柄后（可能是通过 ZCEAPI 接口函数生成或通过回调函数参数传入），对该句柄指向的 ZCEAPI 数据包进行读写操作。数据包的读写是用户通过 ZCEAPI 得到数据和发送数据的主要途径。

ZCEAPI 数据包的读写相当简单，用户只需根据要读写的数据域的类型（具体类型请参考《ZCEAPI 参考手册》中关于数据包字段的类型的规定）选择相应的 ZCEAPI 函数进行读写

即可。

例如：

通过《ZCEAPI 参考手册》可以查到 FID_BuyPrice 是 LN-4 型的。也就是精度为 4 的双精度浮点数。于是可以通过 ZCEAPI 提供的对双精度浮点数读写函数 API_GFDouble 和 API_SFDouble 进行对该字段的读写。

```
double d = API_GFDouble(APIpkg, FID_BuyPrice);              //读取
API_SFDouble(APIpkg， FID_BuyPrice, 1555.552， 4);      //赋值
```

注：用户在读写数据包时一定要根据要读写的数据包字段的具体类型选择相应的 ZCEAPI 读写数据包函数。具体函数的用法和适用类型请参考本手册第 4 部分中相应的介绍。

具体使用实例请参考示例程序中的相关部分。

2.3.7 遍历数据包

数据包的遍历操作是对数据包读写操作的一个补充。用户在得到一个合法的数据包后就可对该数据包进行遍历操作。数据包的遍历只能读取数据包字段，不能改写。

在对数据包遍历时，建议用户先通过调用 API_FirstField 将内部指针移到开始的位置，然后进行遍历。使用相应的遍历函数读取当前字段的值时务必判断当前字段的类型并选择合适的函数读取。

注：ZCEAPI 数据包中字段的顺序与向数据包中写入字段的先后顺序没有直接的关系！

有关数据包遍历的函数请参考本手册第 4 部分相关的介绍。

有关数据包遍历的例子请参考示例程序中的函数“APIpkgTofile”。

2.3.8 发送数据包

用户登录对话流成功后，就可以调用 API_Send 函数向交易所发送各种请求数据包。

发送数据包的过程和登录的过程基本一致：

① 调用 API_AllocPackage 函数生成一个 ZCEAPI 数据包

② 调用 API_SetPID 函数设置数据包 PID

③ 调用相应的字段操作函数填写数据包字段

④ 调用 API_Send 函数将以上所填数据包发送出去

⑤ 根据 API_Send 的返回值判断数据包是否发送成功

⑥ 调用 API_FreePackage 释放刚才用的那个数据包

例如：

```
bool Order(ExchgConnectionHandle conn) //发送一个下单数据包
{
    MsgPackageHandle    APIpkg;
```

```
bool successful = true;
//创建一个 ZCEAPI 数据包
APIpkg = API_AllocPackage();
API_SetPID(APIpkg,0x00003);    //设置 PID
API_SFInt(APIpkg ,FID_SequenceNo ,ReqSeqNo);    //设置数据包的请求序列号
API_SFString(APIpkg ,FID_OrderSysId ,"0",1);
API_SFString(APIpkg ,FID_OrderLocalId,"2006031900000043",strlen("200603190000
0043"));
API_SFString(APIpkg ,FID_ClientId,"00090008",strlen("00090008"));
API_SFString(APIpkg ,FID_InstrumentId,"CF605",strlen("CF605"));
API_SFInt(APIpkg ,FID_Direction ,1 );
API_SFInt(APIpkg ,FID_OffsetFlag ,0 );
API_SFInt(APIpkg ,FID_HedgeFlag,0 );
API_SFDouble(APIpkg ,FID_StopPrice ,0 ,4);
API_SFDouble(APIpkg ,FID_LimitPrice ,15500.5,4);
API_SFInt(APIpkg ,FID_VolumeTotalOrginal,2 );
API_SFInt(APIpkg ,FID_OrderType ,0 );
API_SFInt(APIpkg ,FID_MatchCondition ,3 );
API_DateTime dt;
dt.year =2005;
dt.month =4;
dt.day =6;
API_SFDateTime(APIpkg ,FID_ValidThrough ,&dt);
API_SFInt(APIpkg ,FID_MinimalVolume ,0 );
API_SFInt(APIpkg ,FID_AutoSuspend ,1 );
API_SFInt(APIpkg ,FID_CmbType ,32 );
API_SFString(APIpkg ,FID_SecondLeg," ",1);
API_SFString(APIpkg ,FID_ParticipantId ,"0018",strlen("0018"));
API_SFString(APIpkg ,FID_UserId,"00180090",strlen("00180090"));
if (API_Send(conn,APIpkg)==API_FALSE)
{
    std::cout<<"Fail in send order request"<<std::endl;
    successful=false;
}
else
```

```
    {
        ReqSeqNo ++;
        std::cout<<"Send order request successfully！"<<std::endl;
    }
    //释放数据包
    API_FreePackage(APIpkg);
    return successful ;
}
```

注：具体函数的使用请参考本手册第 4 部分对其具体的介绍。分配的数据包用过之后不一定要释放掉，可以再次使用，但是需要根据实际情况修改数据包中的内容，或者清空该数据包。

2.3.9 数据的接收

在多线程模式下 ZCEAPI 会自动地接收交易所发来的数据将其转化为 ZCEAPI 数据包。用户可以通过以下几种方式接收 ZCEAPI 数据包：

- **接收登录应答数据包**

  登录应答数据包要通过设置登录应答回调函数来接收。（如 2.3.3.1 中的例子）

  注：登录应答回调函数的设置请参考本手册有关设置回调函数的介绍。

- **接收退出登录应答数据包**

  退出登录应答数据包要通过设置退出登录应答回调函数来接收。（与接收登录应答数据包相似）

  注：退出登录应答回调函数的设置请参考本手册有关设置回调函数的介绍。

- **接收交易所一般应答包**

  接收交易所发来的一般应答数据包有两种方式：

  1），通过设置接收数据回调函数来处理接收到的数据包。

  2），若用户没有设置数据处理的回调函数或回调函数处理后返回 **0**，ZCEAPI 将接收的数据转化为 ZCEAPI 数据包后将其放入 ZCEAPI 自身的数据包队列。这时用户需调用 API_Recv 函数得到 ZCEAPI 数据包队列中的数据包。然后进行相应的处理。

注：关于数据包的接收，请参考有关数据包接收的示例程序。

2.3.10 退出登录

用户在登录交易所完成工作后，可以退出登录。退出登录其实质是发送一个退出登录的数据包给交易所。在收到交易所的正确的退出登录确认后，退出交易系统。

用户可以有选择地对其登录上的任意一个流进行退出操作。但用户一次只能从一个流上

退出。

退出登录与登录的过程基本相似：

① 调用 API_AllocPackage 函数生成一个 ZCEAPI 数据包

② 调用 API_SetPID 函数设置数据包 PID（退出登录为：0x00017）

③ 调用相应的字段操作函数填写数据包字段（如：API_SFInt (APIpkg , FID_SequenceNo, 1);）

④ 调用 API_Logout 函数将以上所填数据包发送出去

⑤ 根据 API_Logout 函数的返回值判断是否退出登录成功

⑥ 调用 API_FreePackage 释放刚才用的那个数据包

例如：

```
bool logout(ExchgConnectionHandle conn)   //退出登录请求
{
 MsgPackageHandle    APIpkg;
 bool successful = true;
 //创建一个 ZCEAPI 数据包
  APIpkg = API_AllocPackage();
  API_SetPID(APIpkg ,0x00017);  //设置 PID
  API_SFInt(APIpkg ,FID_SequenceNo ,ReqSeqNo);   //设置数据包的请求序列号
 API_SFInt(APIpkg ,FID_DataFlowFlag ,0);              //设置数据包所在的流
 API_SFString(APIpkg ,FID_ParticipantId ,"0018",strlen("0018"));
 API_SFString(APIpkg ,FID_UserId,"00180090",strlen("00180090"));
 if (API_Logout (conn,APIpkg,10)==API_FALSE)
 {
       std::cout<<"Fail in logout request"<<std::endl;
       successful=false;
 }
 else
 {
       std::cout<<"Send logout request successfully！"<<std::endl;
 }
 //释放数据包
 API_FreePackage(APIpkg);
 return successful ;
}
```

注：a, 具体函数的使用请参考本手册第 4 部分对其具体的介绍。

b, 分配的数据包用过之后不一定要释放，可以再次使用，但是需要根据实际情况修改数据包中的内容，或者清空该数据包。

2.3.11 释放链路连接

用户退出登录后，在应用程序退出前，用户应该先断开与交易所的连接，然后再释放连接对象。

断开连接用户可以直接调用 ZCEAPI 接口函数 API_DisConnect。成功执行后，与交易所的链路会断开。但这时与交易所的连接对象还依然存在，用户可以选择调用 API_Connect 重新连接到交易所，也可以调用 API_FreeExchgConnection 直接断开并释放连接对象。

注：连接对象释放后就不能再使用！

具体函数的使用请参考本手册第 4 部分对其具体的介绍。

2.3.12 停止 ZCEAPI 服务

在退出 ZCEAPI 之前，建议先调用 ZCEAPI 服务停止函数 ftd_api_stop 停止 ZCEAPI 的服务。这里停止 ZCEAPI 的服务主要是停止 ZCEAPI 的诊断功能。

建议用户不要随便停止 ZCEAPI 的服务，这样可能会对 ZCEAPI 的工作造成不利的影响。

3 数据定义

3.1 基础数据类型

typedef unsigned short int **API_UINT16;**

typedef unsigned char **API_BYTE;**

typedef unsigned int **API_UINT32;**

3.2 日期时间类型

```
//结构按 1 字节对齐
typedef struct tagDateTime
{
    API_UINT16  year;
    API_BYTE    month,day,hour,minute,second;
    API_UINT32  microsec;
```

```
}API_DateTime;
```

3.3 API_BOOL

```
typedef int API_BOOL;
#define API_TRUE    1
#define API_FALSE   0
```

3.4 线程模式

```
typedef int THREAD_MODE;
#define MULTI_THREAD_MODE 1
```

3.5 建立连接时是否阻塞

```
#define CONNECT_WAIT    1
#define CONNECT_NO_WAIT 0
```

3.6 市场约定

```
typedef int MARKET_ID;
#define MARKET_ZCE  1
```

3.7 数据域类型定义

```
typedef int API_FIELDTYPE;
#define FTNULL          0
#define FTCHAR          1
#define FTLONG          2
#define FTSTRING        3
#define FTDOUBLE        4
#define FTDATETIME      5
```

3.8 数据流标示

```
typedef int API_DFFLAG;
```

```
#define DFF_DIALOG      0   //对话流
#define DFF_PRIVATE     1   //私有流
#define DFF_PUBLISH     2   //广播流
```

3.9 数据流状态常数

```
typedef int API_DFSTATUS;
#define DFS_CLOSED          0   //连接断开
#define DFS_OPENED          1   //刚刚建立连接
#define DFS_NEGOTIATEKEY    2   //刚刚协商密钥，还没有登录
#define DFS_LOGIN_OK        3   //已经登录成功
```

3.10 数据包句柄

```
class MsgPackage;   //数据包类声明
/*数据包句柄定义*/
Typedef  MsgPackage*  MsgPackageHandle;
```

3.11 连接对象句柄

```
class ExchangeConnection;   //连接对象声明
/*交易所连接对象句柄定义*/
Typedef  ExchangeConnection*  ExchgConnectionHandle;
```

3.12 返回数据包回调函数

```
typedef int (*ExchgPackageCallBack)(void * CallBackArg, ExchgConnectionHandle,
MsgPackageHandle);
```

注：ExchgConnectionHandle 为 ZCEAPI 提供的连接对象句柄。

MsgPackageHandle 为 ZCEAPI 提供的数据包对象句柄。

3.13 链路状态回调函数

```
typedef void (*ExchgConnectionCallBack)(void * CallBackArg, ExchgConnectionHandle,
int error_code, const char* error_text);
```

注：ExchgConnectionHandle 为 ZCEAPI 提供的连接对象句柄。

4. 函数介绍

4.1 ZCEAPI 管理

4.1.1 获取 ZCEAPI 版本号

int ftd_api_getVersion(char * versionbuf,unsigned int buflen);

功能： 得到 ZCEAPI 的版本号字符串

参数说明： 1），char * versionbuf 接受版本号字符串的缓冲区指针

2），unsigned int buflen 接受版本号字符串缓冲区大小

返回值： 成功返回取得版本字符串长度，缓冲区不够返回-1

使用提醒： 给出的 buflen 要足够大，否则若缓冲区大小小于 ZCEAPI 版本号长度，则获取版本号失败。

4.1.2 ZCEAPI 初始化

API_BOOL ftd_api_init(THREAD_MODE mode,const char *LogFilePath,int MonitorPort);

功能： 初始化 ZCEAPI，设置线程模式，指定诊断日志路径和诊断端口。

参数说明： 1），THREAD_MODE mode 线程模式，目前只能取MULTI_THREAD_MODE

2），const char *LogFilePath 诊断日志文件路径，路径字符串指针

3），int MonitorPort 诊断端口的端口号，目前未用。

返回值： 初始化成功返回 API_TRUE，否则返回 API_FALSE

使用提醒： 1），**LogFilePath 路径必须已经存在**，如果路径不存在，ZCEAPI 不会自动创建相应的路径，进而 ZCEAPI 初始化会失败。

2），建议在一个应用程序中只初始化一次。

3），**目前 ZCEAPI 只能被初始化为多线程模式。**

4），在 Windows 环境下若不能成功初始化，可尝试先调用：WSAStartup。

5），如果用户要采用协议版本号 11 登录交易系统，接收行情信息 UDP 默认端口为 22677。如果用户想更改这个端口请在初始化前在当前应用程序目录使用文件“APIConfig.dat”来指定更改的端口号。如：在文件中写入“22899”，则 API 将来会根据这个端口号打开 UDP 端口。

6），ZCEAPI 成功的初始化是下面一切工作的基础。若 ZCEAPI 不能成功的初始化，则后续操作不能成功执行。

4.1.3 停止 ZCEAPI 服务

void ftd_api_stop();

功能：　　停止诊断，关闭日志文件

使用提醒：　使用该函数停止 ZCEAPI 服务后，还可以进行建立连接、登录等事件的处理。但是建议不要这样做。

4.2 数据包管理

4.2.1 创建数据包对象

MsgPackageHandle API_AllocPackage();

功能：　　创建一个 ZCEAPI 数据包

返回值：　　成功返回创建的数据包句柄，不然返回 NULL

使用提醒：　创建数据包对象之前必须保证 ZCEAPI 成功初始化,否则创建数据包对象失败。

4.2.2 回收数据包对象

void API_FreePackage(MsgPackageHandle pkg);

功能：　　将现有的一个数据包释放

参数说明：　1），MsgPackageHandle pkg　ZCEAPI 创建过的一个数据包句柄

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），回收数据包对象之后，不能再用该对象进行其他操作。

4.3 报文数据操作

4.3.1 取得报文 ID

int API_GetPID(MsgPackageHandle pkg);

功能：　　取得指定报文的 PID

参数说明：　1），MsgPackageHandle pkg　要取 PID 的数据报文句柄

返回值：　　成功返回指定数据包的 PID，不然返回 0xFFFFF

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的还未释放的数据包，否则会出现异常。

2），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.3.2 设置报文 ID

void API_SetPID(MsgPackageHandle pkg,int pid);

功能：　　设置指定数据包的 PID

参数说明：　1），MsgPackageHandle pkg　要设置 PID 的数据包

2），int pid　　要设置的 PID 的值

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应当填写有意义的 PID。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.3.3 取得某字段的当前数据类型

API_FIELDTYPE API_GetFieldType(MsgPackageHandle pkg,int fid);

功能：　　取得指定数据包中某字段的当前数据类型

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

2），int fid　　要取类型的数据域的域编号。参看《ZCEAPI 参考手册》

返回值：　成功返回 pkg 所指向的数据包中数据域编号为 fid 的数据类型编码，没有该数据域返回 FTNULL。具体类型请参考本手册第一部分的“**数据域类型定义**”项中的定义。

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），填写的 fid 应该是《ZCEAPI 参考手册》中规定的 FID。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.4 字段操作

4.4.1 从数据包中取字符

char API_GFChar(MsgPackageHandle pkg,int fid,char def=' ');

功能：　　按字符格式取出指定数据包中指定数据域的字符

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

2），int fid　　要取的数据域的数据域编码

3），char def=’’　　默认值。当取数据失败时默认返回值。默认为空格。

返回值：　成功返回 pkg 所指向的 ZCEAPI 数据包中的编号为 fid 的数据域的字符。取字符失败返回 def

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 C（字符型）的字段适合使用该函数。

3），使用该函数取非 C（字符型）字段，函数会根据具体类型做转化，但不保证取值正确。

4），使用该函数之前必须保证 ZCEAPI 成功初始化，否则取值失败。

4.4.2 设置数据包中的字符数据域

int API_SFChar(MsgPackageHandle pkg,int fid,char val);

功能：　设置指定数据包中指定数据域的字符类型值

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

2），int fid　要设置的数据域的数据域编码

3），char val　要设置的字符值

返回值：　0：代表赋值成功

-1：代表该数据域不能用该类型赋值

-99：代表数据包非法

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 代表的数据域类型为 C（字符型）适合使用该函数。当 Fid 不在《ZCEAPI 参考手册》数据域定义范围内时，不对数据域类型做判断，直接可以赋值成功。

3），使用该函数之前必须保证 ZCEAPI 成功初始化，否则赋值失败。

4.4.3 从数据包中取整数

int API_GFInt(MsgPackageHandle pkg,int fid,int def=0);

功能：　从指定数据包的指定数据域中按整数型得到整型数据。

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

2），int fid　要取的数据域的数据域编码

3），int def=0　默认值，取数据失败时默认返回值，默认为 0。

返回值： 成功返回 pkg 所指向的 ZCEAPI 数据包中编号为 fid 的数据域的整型数。取整数失败返回 def

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 SI（短整型），N（整型）和 UN（无符号整数）的字段适合使用该函数。

3），若数据域类型为 UN，那么返回值可以看作是无符号整型。

4），使用该函数取非匹配类型字段，函数会根据具体类型做转化，但不保证取值正确。

5），使用该函数之前必须保证 ZCEAPI 成功初始化，否则取值失败。

4.4.4 设置数据包中的整型数据域

int API_SFInt(MsgPackageHandle pkg,int fid,int val);

功能： 设置指定数据包中指定数据域的整数类型值

参数说明： 1），MsgPackageHandle pkg 指定的 ZCEAPI 数据包句柄

2），int fid 要设置的数据域的数据域编码

3），int val 要设置的整数值

返回值： 0：代表赋值成功

-1：代表该数据域不能用该类型赋值

-99：代表数据包非法

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 SI（短整型）、N（整型）、UN（无符号整数）适合使用该函数。当 Fid 不在《ZCEAPI 参考手册》数据域定义范围内时，不对数据域类型做判断，直接可以赋值成功。

3），使用该函数之前必须保证 ZCEAPI 成功初始化，否则赋值失败。

4.4.5 从数据包中取双精度小数

double API_GFDouble(MsgPackageHandle pkg,int fid,double def=0);

功能： 按双精度小数格式取出指定数据包的指定数据域的数值

参数说明： 1），MsgPackageHandle pkg 指定的 ZCEAPI 数据包句柄

2），int fid 要取的数据域的数据域编码

3），double def=0　　默认值。取数据失败时默认返回值。默认为 0。

返回值：　成功返回 pkg 所指向的 ZCEAPI 数据包中的编号为 fid 的数据域的双精度小数。取双精度小数失败返回 def

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 LN（双精度型）的字段适合使用该函数。

3），使用该函数取非匹配类型字段，函数会根据具体类型做转化，但不保证取值正确。

4），若要取的字段是用户自己填写的（用下面介绍的 API_SFDouble 函数），那么用户填写什么值，还能通过该函数取出该值。使用该函数时填写的精度是不起作用的。

5），使用该函数之前必须保证 ZCEAPI 成功初始化，否则取值失败。

4.4.6 设置数据包中的双精度数据域

int API_SFDouble(MsgPackageHandle pkg,int fid,double val,int precision=4);

功能：　设置指定数据包中指定数据域的双精度小数类型值

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

2），int fid　　要设置的数据域的数据域编码

3），double val　　要设置的双精度小数值

4), int precision=4　　设置的小数的精度，默认为 4

返回值：　0：代表赋值成功

-1：代表该数据域不能用该类型赋值

-99：数据包非法

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 LN（双精度型）适合使用该函数。当 Fid 不在《ZCEAPI 参考手册》数据域定义范围内时，不对数据域类型做判断，直接可以赋值成功。

3），precision 代表小数位数，如果 val 的小数位数大于 precision，那么 val 将只保留前 precision 位小数有效。即在发送到交易所时只保留前 precision 位小数有效。而在不发送时，用 API_GFDouble（）取相应字段的值返回值仍是 val。

4），用户设置的小数精度应该参考《ZCEAPI 参考手册》中数据域规定的小数精度，如果用户设置的小数精度和《ZCEAPI 参考手册》中数据域规定的精度

不一致，在发送到交易所时按数据域规定的精度进行处理。

5），使用该函数之前必须保证 ZCEAPI 成功初始化，否则赋值失败。

4.4.7 从数据包中取字符串

int API_GFString(MsgPackageHandle pkg,int fid, char* buf,int bufsize);

功能： 按字符串的格式取出指定数据包的指定数据域的数值

参数说明： 1），MsgPackageHandle pkg 指定的 ZCEAPI 数据包句柄

2），int fid 要取的数据域的数据域编码

3），char* buf 接收字符串的缓冲区指针

4），int bufsize 接收缓冲区大小

返回值： 成功返回所取的字符串的长度。如果格式不对，或者有错误发生返回值为 0

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 S（字符串）、DT（日期时间）、LN（双精度小数）适合使用该函数。

3）使用该函数取非匹配类型字段，函数会根据具体类型做转化，但不保证取值正确。

4），使用该函数取 LN（双精度小数）时，取到的数据域字段值保存了小数精度。

5）,给出的 bufsize 要足够大,否则若缓冲区小于字段字符长度,只将 bufsize 长度的字符复制到 buf 中，从而只取字段的一段而非完整字段。

6），bufsize 的大小不要大于 buf 指向的缓冲区的实际大小，否则可能会出现内存错误。

7）,成功时返回的字符串长度是字符串中包含的字符的长度。缓冲区中接收的字符串是没有结束符的,用户要将 buf 当作字符串处理时需要自己添加字符串结尾符。

8）,当用该函数取 DT（日期时间）类型的数据域时，有以下五种情况：

① DT 结构对象中，年，月，日，时，分，秒，微秒都不为 0，此时取出的格式为：YYYY-MM-DD HH:mm:ss.iiiiii

② DT 结构对象中，年，月，日，时，分，秒都不为 0，微秒为 0 时，此时取出的格式为：YYYY-MM-DD HH:mm:ss

③ DT 结构对象中，年，月，日都不为 0，其余为 0 时（只有日期时），此时取出的格式为：YYYY-MM-DD

④ DT 结构对象中，时，分，秒，微秒都不为 0，日期为 0 时，此时取出的格

式为：HH:mm:ss.iiiiii

⑤ DT 结构对象中，时，分，秒都不为 0，日期和微秒为 0 时（只有时间时），此时取出的格式为：HH:mm:ss

注：以上格式符号意义如下：Y:年、M:月、D:日、H:小时、m:分钟、s:秒、i:微秒（如：YYYY 表示:4 位数字表示年，iiiiii 表示:6 位数字表示微秒值）

9），使用该函数之前必须保证 ZCEAPI 成功初始化，否则取值失败。

4.4.8 设置数据包中的字符串数据域

int API_SFString(MsgPackageHandle pkg,int fid,const char* buf,int bufsize);

功能：　以字符串格式设置指定数据包的指定数据域值

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

2），int fid　要设置的数据域的数据域编码

3），char* buf　字符串的头指针

4），int bufsize　字符串长度

返回值：　0：代表赋值成功

-1：代表该数据域不能用该类型赋值

-2：代表 bufsize 为 0，并且 pkg 中 fid 字段并不存在

-99：代表数据包非法

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 S（字符串）适合使用该函数。当 Fid 不在《ZCEAPI 参考手册》数据域定义范围内时，不对数据域类型做判断，直接可以赋值成功。

3），bufsize 如果是字符串的大小，不能小于字符长度，否则只将前 bufsize 个字节的字符填入。

4),使用该函数之前必须保证 ZCEAPI 成功初始化，否则赋值失败。

4.4.9 从数据包中取日期时间

API_BOOL API_GFDateTime(MsgPackageHandle pkg,int fid,API_DateTime*val);

功能：　以日期时间格式读取指定数据包指定字段的值

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

2），int fid　要取的数据域的数据域编码

3），API_DateTime*val　ZCEAPI 日期类型指针，指向存放取得的日期的日期变量。ZCEAPI 日期时间类型参看本手

册中“日期时间类型”。

返回值： 取日期成功返回 API_TRUE，否则返回 API_FALSE。

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 D（日期型）DT（日期时间）的适合使用该函数。

3），使用该函数取非匹配类型字段，函数会根据具体类型做转化，但不保证取值正确。

4），使用该函数之前必须保证 ZCEAPI 成功初始化，否则取值失败。

4.4.10 设置数据包中的日期时间数据域

```
int API_SFDateTime(MsgPackageHandle pkg,int fid,API_DateTime*val);
```

功能： 以日期时间格式设置指定数据包指定字段的值

参数说明： 1），MsgPackageHandle pkg 指定的 ZCEAPI 数据包句柄

2），int fid 要设置的数据域的数据域编码

3），API_DateTime*val ZCEAPI 日期类型指针，指向存放要设置的日期的日期变量。ZCEAPI 日期时间类型参看本手册中“日期时间类型”。

返回值： 0:代表赋值成功

-1:代表该数据域不能用该类型赋值

-99：代表数据包非法

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），用户应该参考《ZCEAPI 参考手册》中定义的数据域和数据域类型。Fid 对应的数据域类型为 D（日期型）和 DT（日期时间）适合使用该函数。当 Fid 不在《ZCEAPI 参考手册》数据域定义范围内时，不对数据域类型做判断，直接可以赋值成功。

3），使用该函数之前必须保证 ZCEAPI 成功初始化，否则赋值失败。

4.4.11 判断某个字段是否为空

```
API_BOOL API_FieldIsNull(MsgPackageHandle pkg,int fid);
```

功能： 判断指定数据包中指定字段是否为空

参数说明： 1），MsgPackageHandle pkg 指定的 ZCEAPI 数据包句柄

2），int fid 要判断的数据域的数据域编码

返回值：　指定字符为空返回 API_TRUE，否则返回 API_FALSE。

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），填写的 FieldID 应该是《ZCEAPI 参考手册》中所规定的 FID

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.4.12 清除某个特定的字段的值

void API_ClearField(MsgPackageHandle pkg,int fid);

功能：　将指定字段从指定的数据包中清除

参数说明：　1），MsgPackageHandle pkg　　指定的 ZCEAPI 数据包句柄

2），int fid　　要清除的数据域的数据域编码

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），填写的 FieldID 应该是《ZCEAPI 参考手册》中所规定的 FID。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.4.13 清除数据包里的所有字段

void API_ClearAll(MsgPackageHandle pkg);

功能：　清除数据包里的所有字段。

参数说明：　1），MsgPackageHandle pkg　指定的 ZCEAPI 数据包句柄

使用提醒：　1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

4.4.14 数据包复制

void API_Copy(MsgPackageHandle pkg,MsgPackageHandle source);

功能：　把 source 指定的数据全部复制到 pkg 指定的数据包

参数说明：　1），MsgPackageHandle pkg　　指定的目标 ZCEAPI 数据包句柄

2），MsgPackageHandle source　　指定的源 ZCEAPI 数据包句柄

使用提醒：　1），所填写的参数 pkg 和 source 必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），复制时将数据包的 PID 也进行了复制。

4.5 数据遍历

4.5.1 数据包的数据域指针移到第一个数据域

int API_FirstField(MsgPackageHandle pkg);

功能： 移动指定数据包的内部数据域指针到第一个数据域

参数说明： 1），MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

返回值： 数据包中有数据则返回第一个数据域的数据域编号 FID。

-1:代表数据包为空包

-99：代表数据包非法

使用提醒： 1），所填写的参数 pkg 必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），数据域的顺序不是按照给数据包赋值的先后顺序。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.2 下一个数据域

int API_NextField(MsgPackageHandle pkg);

功能： 将指定数据包的内部数据域指针下移一位

参数说明： MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

返回值： 数据包中存在下一个数据域则返回下一个数据域的数据域编号 FID。

-1：代表已到数据包结尾

-99：代表数据包非法

使用提醒： 1），所填写的参数 pkg 必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），数据域的顺序不是按照给数据包赋值的顺序。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.3 取当前数据域类型

API_FIELDTYPE API_CFType(MsgPackageHandle pkg);

功能： 取出指定数据包的当前数据域的数据类型

参数说明： MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

返回值： 成功返回 pkg 所指向的数据包中当前数据域的数据类型编码。具体类型请参考本手册第一部分的“**数据域类型定义**”中的定义。

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），指定数据包为空，返回 FTNULL。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.4 取当前数据域的时间

API_BOOL API_CFDT(MsgPackageHandle pkg,API_DateTime *dt);

功能： 以日期格式取出当前数据域的值

参数说明： 1），MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

2），API_DateTime *dt 存放取得的时间的日期变量指针

返回值： 成功返回 API_TRUE，否则返回 API_FALSE

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.5 取当前数据域的字符

char API_CFChar(MsgPackageHandle pkg,char def=' ');

功能： 以字符格式取出当前数据域的值

参数说明： 1），MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

2），char def=' ' 默认值。取字符失败时返回的值。默认为空格。

返回值： 成功，返回取得的字符值。失败，返回默认值 def

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），应先判断当前数据域类型，对于类型为“FTCHAR”的适合使用本函数。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.6 取当前数据域的整数

int API_CFInt(MsgPackageHandle pkg,int def=0);

功能： 以整型读取当前数据域的值

参数说明： 1），MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

2），int def=0 默认值。取整数失败时的返回值。默认为 0。

返回值： 成功，返回取得的整数值，不然返回 def

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数

据包，否则会出现异常。

2），应先判断当前数据域类型，对于类型为“FTLONG”的适合使用本函数。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.7 取当前数据域的浮点数

double API_CFDouble(MsgPackageHandle pkg,int *prec=0,double def=0);

功能： 以浮点数类型读取当前数据域中的值

参数说明： 1），MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

2），int *prec=0 传回浮点数的精度。默认为 0，即不取精度。

3），double def=0 默认值。取浮点数失败返回该值。默认为 0。

返回值： 成功，返回取得的浮点数值，通过 prec 得到浮点数精度。不然返回 def。

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），应先判断当前数据域类型，对于类型为“FTDOUBLE”的适合使用本函数。

3），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.8 取得当前数据域的字符串

int API_CFString(MsgPackageHandle pkg,char* buf,int bufsize);

功能： 以字符串类型来读取当前数据域中的值

参数说明： 1），MsgPackageHandle pkg 指定的目标 ZCEAPI 数据包句柄

2），char* buf 接收缓冲区的头指针

3），int bufsize 接收缓冲区的大小

返回值： 成功，返回取得的字符串的长度。否则返回为 0

使用提醒： 1），所填写的参数必须是通过 API_AllocPackage 分配得到的但还未释放的数据包，否则会出现异常。

2），应先判断当前数据域类型，对于类型为 S（字符串）、DT（日期时间）和 LN（双精度小数）的适合使用该函数。

3），使用该函数取 LN（双精度小数）时，取到的数据域字段值保存了小数精度。

4），接收缓存区大小要足够大，否则可能会截断取得的字符串。

5），成功时返回的字符串长度是字符串中包含的字符的长度。缓冲区中接收的字符串是没有结束符的。

6），使用该函数之前必须保证 ZCEAPI 成功初始化。

4.5.9 取当前时间

void API_Now(API_DateTime*dt);

功能： 取得当前本地系统的时间

参数说明： 1），API_DateTime*dt 存放当前时间的 APIDateTime 型数据的指针

使用提醒： 通过该函数取出的当前系统时间值保存到 dt 中，注意，这时 dt 中微秒能取到实际值。

4.6 连接管理

4.6.1 创建交易所连接对象

ExchgConnectionHandle API_CreateExchgConnection(API_BOOL Encrypt, API_BOOL Commpress, MARKET_ID Market ,int heart_beat_interval, int heart_beat_timeout);

功能： 创建一个与交易所的连接对象

参数说明： 1），API_BOOL Encrypt 是否加密。API_TRUE 加密，API_FALSE 不加密。目前支持加密链路。

2），API_BOOL Commpress 是否压缩。API_TRUE 压缩，API_FALSE 不压缩。

3），MARKET_ID Market 连接的交易市场。参看本手册的“**市场约定**”

4），int heart_beat_interval 心跳间隔。单位秒

5），int heart_beat_timeout 心跳最大超时。单位秒

返回值： 成功则返回交易所连接对象句柄，失败返回 NULL

使用提醒： 1），heart_beat_interval 应该小于 heart_beat_timeout 的 1/2。

2），heart_beat_timeout 不能小于交易所交易系统的心跳间隔，不然 ZCEAPI 会常因心跳超时而错误的断开与交易所系统的连接，导致系统无法工作。具体心跳间隔的设置请参考交易所系统的心跳间隔。

3），创建交易所连接对象之前必须保证 ZCEAPI 成功初始化，否则创建交易所连接对象失败。

4.6.2 设置连接属性参数

int API_SetConnectionOpt(ExchgConnectionHandle conn,int keepIdle,int keepInterval, int keepCount);

功能： 设置连接的属性参数

参数说明： 1），ExchgConnectionHandle conn 交易所连接对象句柄

2），int keepIdle　　连接空闲时间。单位秒

3），int keepInterval　　连接检测间隔。单位秒

4），int keepCount　　检测次数

返回值：　成功设置返回 0，否则返回错误码-1。

使用提醒：　1），该函数目前只对 Linux 平台起作用。

2），在参数 Conn 为空时函数返回 1。

4.6.3 发起与交易所连接

API_BOOL API_Connect(ExchgConnectionHandle conn,const char* HostIpAddr, int Port,double Wait);

功能：　连接交易系统，建立与交易所的连接

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

2），const char* HostIpAddr　　代表交易所 IP 地址的字符串指针

3），int Port　　交易所系统端口号

4），double Wait　　超时等待时间，单位秒。

返回值：　连接建立成功返回 API_TRUE,否则返回 API_FALSE。

使用提醒：　1），参数 1 必须是 ZCEAPI 创建的连接对象的句柄。

2），Port 端口号，要根据 conn 指向的连接对象是否为加密（加密必压缩）来决定。即：加密的连接对象要连接到加密的端口，非加密的连接对象要连接到非加密的端口。

3），用户在建立和交易所的连接后，应尽快在此连接上登录，否则连接会自动断开。

4），该函数不推荐使用，建议使用 API_ConnectEx.

4.6.4 通过域名服务器发起与交易所的连接

int API_ConnectEx(ExchgConnectionHandle conn,const char* DNSIp,int DNSPort,double Wait,const char * UserID,const char * gwVersion,char ConnMode,const char* gwIP,int gwPort,char * errMsg);

功能：　由域名服务指定的交易所前置 IP 和端口或者由域名服务自动分配的交易所前置 IP 和端口连接交易系统，建立与交易所的连接

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

2），const char* DNSIp　　域名解析服务 IP 地址

3），int DNSPort　　域名解析服务端口

4），double Wait　　超时等待时间

5），const char * UserID　　席位号

6），const char * gwVersion　　前置的版本信息

7），char ConnMode　　申请方式，‘0’：自动分配，‘1’：指定 IP

8），const char* gwIP　　在申请方式为 1 的情况下指定的前置 IP

9），int gwPort　　在申请方式为 1 的情况下指定的前置端口

10），char * errMsg　　输出参数。返回错误信息的内存空间，由用户分配，最少 51 个字节

返回值：　0：代表连接成功

-1：代表发送 DNS 请求失败

-2：代表接收 DNS 应答失败

-3：代表接收 DNS 数据错误

-4：代表没有得到分配的前置 IP

-5：代表连接 DNS 服务失败

-6：代表连接交易所前置失败

使用提醒：　1），参数 1 必须是 ZCEAPI 创建的连接对象的句柄。

2），由于交易所协议只让自动获取 IP，当 ConnMode 参数在设置为自动分配时，也要指定交易所的前置端口 gwPort。

3），用户在建立和交易所的连接后，应尽快在此连接上登录，否则连接会自动断开。

4），在参数 Conn 为空时函数返回 1。

4.6.5 是否已经连接

API_BOOL API_Connected(ExchgConnectionHandle conn);

功能：　判断交易所连接对象是否已经连接到交易所

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

返回值：　若连接对象连接到了交易所，返回 API_TRUE,否则返回 API_FALSE

4.6.6 登录到交易所

API_BOOL API_Login(ExchgConnectionHandle conn,MsgPackageHandle reqPkg, double Wait);

功能：　登录到交易系统三个流中的任一个

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

2），MsgPackageHandle reqPkg　　包含登录信息的 ZCEAPI 数据包句柄

3），double Wait　　登录超时等待时间。单位秒。

返回值： 登录交易所成功返回 API_TRUE,不成功返回 API_FALSE

使用提醒： 1），conn 必须是已经连接到交易所的连接对象的句柄。

2），需要在 reqPkg 指向的数据包中填写认证、流标志、流编号等正确的登录信息。具体参考《ZCEAPI 参考手册》。

3），当参数 wait 的值不足够大时，该函数会因超时而返回 API_FALSE，这时其返回值不能准确地反映出登录状态。应当以收到的登录应答包为准。

4），对于加密信道，参数 wait 的值不能太小，更不能等于零。

5），建议用户将 wait 值设置的足够大（视网速而定），根据该函数返回值以确定登录成功与否。

6),对于加密信道因为链路密钥协商只需进行一次,在第一次登录时进行协商。

4.6.7 退出登录

API_BOOL API_Logout(ExchgConnectionHandle conn,MsgPackageHandle reqPkg, double Wait);

功能： 退出登录，三个数据流中的任一个

参数说明： 1），ExchgConnectionHandle conn 交易所连接对象句柄

2），MsgPackageHandle reqPkg 包含退出登录信息的 ZCEAPI 数据包句柄

3），double Wait 退出登录超时等待时间。单位秒。

返回值： 退出登录成功返回 API_TRUE,不成功返回 API_FALSE

使用提醒： 1），conn 必须是已经连接到交易所的连接对象的句柄。

2），需要在 reqPkg 指向的数据包中填写流标志等正确的退出登录信息。具体参考《ZCEAPI 参考手册》。

3），当参数 wait 的值不足够大时，该函数会因超时而返回 API_FALSE，这时其返回值不能准确地反映出是否退出登录。应当以收到的退出登录应答包为准。

4.6.8 发送一个数据包

API_BOOL API_Send(ExchgConnectionHandle conn,MsgPackageHandle pkg);

功能： 发送一个 ZCEAPI 数据包

参数说明： 1），ExchgConnectionHandle conn 交易所连接对象句柄

2），MsgPackageHandle reqPkg 要发送的 ZCEAPI 数据包句柄

返回值： 发送数据包成功返回 API_TRUE,不成功返回 API_FALSE

使用提醒： 1），conn 必须是已经连接到交易所的连接对象的句柄。

2），根据业务需要在 Pkg 指向的数据包中填写必要的正确的信息。具体参考

《ZCEAPI 参考手册》。

3），该函数不能发送登录、退出数据包。

4.6.9 接收一个数据包

MsgPackageHandle API_Recv(ExchgConnectionHandle conn,double wait=0);

功能：　　从 ZCEAPI 的数据包队列中接收一个数据包

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

2），double wait=0　　ZCEAPI 数据包队列没有数据时等待秒数，默认 wait=0:没有数据立即返回

返回值：　　有数据包则返回取得的 ZCEAPI 数据包句柄。没有数据包则返回 NULL

使用提醒：　1），conn 必须是已经连接到交易所的连接对象的句柄。

2），该函数是在用户没有设置处理 ZCEAPI 数据包的回调函数或回调函数的返回值为 0 的情况下得到数据包的。

3），使用该函数得不到登录应答数据包和退出登录应答数据包。

4），用户在处理完通过该函数接收到的数据包后，需要调用 API_FreePackage() 去释放接收到的数据包。

4.6.10 断开与交易所的连接

void API_DisConnect(ExchgConnectionHandle conn);

功能：　　断开指定交易所连接对象与交易所的连接

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

使用提醒：　执行完该函数后，连接对象还没被释放。只是与交易所的连接中断。

4.6.11 关闭并释放交易所连接对象

void API_FreeExchgConnection(ExchgConnectionHandle conn);

功能：　　断开并释放指定的交易所连接对象

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

使用提醒：　执行完该函数后，与交易所的连接先被断开，然后连接对象被释放。执行该函数后连接对象句柄 conn 将不能再被使用。

4.6.12 取得数据流状态

API_DFSTATUS API_GetDataFlowStatus(ExchgConnectionHandle conn ,int DataFlowFlag);

功能：　　取得通过指定的交易所连接对象登录的数据流状态

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

2），int DataFlowFlag　　数据流标示 0-〉对话流，1-〉私有流，2-〉广播流

返回值：　　成功返回通过交易所连接对象 conn 登录的 DataFlowFlag 指定的数据流状态。参见本手册中的“数据流状态常数”。

4.7 设置回调函数

4.7.1 设置连接打开通知

void API_SetOpenCallBack(ExchgConnectionHandle conn, ExchgConnectionCallBack callback,void * CallBackArg);

功能：　　设置指定的交易所连接对象打开链路时的回调函数

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

2），ExchgConnectionCallBack callbac　　要设置的链路状态回调函数

3），void * CallBackArg　　要设置的链路状态回调函数的参数，它将最终传给链路状态回调函数。请参看 3 中“**链路状态回调函数**”。

使用提醒：　设置该回调函数后，不论链路打开成功与否，都会回调该函数设置的回调函数，回调函数可以根据回传的参数值判断链路打开成功与否。

4.7.2 设置连接断开通知

void API_SetCloseCallBack(ExchgConnectionHandle conn, ExchgConnectionCallBack callback,void * CallBackArg);

功能：　　设置指定的交易所连接对象链路断开时的回调函数

参数说明：　1），ExchgConnectionHandle conn　　交易所连接对象句柄

2），ExchgConnectionCallBack callback　　要设置的链路状态回调函数

3），void * CallBackArg　　要设置的链路状态回调函数的参数，它将最终传给链路状态回调函数。请参看 3 中“**链路状态回调函数**”。

4.7.3 设置出错通知

void API_SetErrorCallBack(ExchgConnectionHandle conn, ExchgConnectionCallBack callback,void * CallBackArg);

功能： 设置指定的连接交易所对象的链路出错时的回调函数

参数说明： 1），ExchgConnectionHandle conn 交易所连接对象句柄

2），ExchgConnectionCallBack callback 要设置的链路状态回调函数

3），void * CallBackArg 要设置的链路状态回调函数的参数，它将最终传给链路状态回调函数。请参看 3 中“**链路状态回调函数**”。

4.7.4 设置接收数据通知

void API_SetRecvCallBack(ExchgConnectionHandle conn, ExchgPackageCallBack callback,void * CallBackArg);

功能： 设置有数据处理时的回调函数

参数说明： 1），ExchgConnectionHandle conn 交易所连接对象句柄

2），ExchgPackageCallBack callback 要设置的返回数据包回调函数

3），void * CallBackArg 要设置的返回数据包回调函数的参数，它将最终传给返回数据包回调函数。请参看 3 中“**返回数据包回调函数**”。

使用提醒： 1），并不是所有的应答数据包都能通过该函数设置相应的回调函数而得到处理。登录应答数据包是要设置登录通知回调才可得到，而退出登录应答数据包要设置退出登录通知回调才可得到。

2），如果设置的回调函数返回值为 0，那么回调函数处理后的数据包还将被放入 ZCEAPI 的数据包队列中。

3），若设置的回调函数返回值为 0，那么用户不能调用 API_FreePackage()去释放该数据包，否则会将一个悬空指针放入 ZCEAPI 数据包队列，而引起错误。

4），设置的回调函数返回值不为 0，用户不要调用 API_FreePackage()去释放该数据包，因为 ZCEAPI 会自行释放该数据包。

4.7.5 设置登录应答通知

void API_SetLoginCallBack(ExchgConnectionHandle conn, ExchgPackageCallBack

callback,void * CallBackArg);

功能： 设置收到登录应答时的回调函数

参数说明： 1），ExchgConnectionHandle conn 交易所连接对象句柄

2），ExchgPackageCallBack callback 要设置的返回数据包回调函数

3），void * CallBackArg 要设置的返回数据包回调函数的参数,它将最终传给返回数据包回调函数。请参看本手册中的“**返回数据包回调函数**”。

使用提醒： 1），可以通过此函数的设置得到登录应答数据包，并进行相应的处理。

2），用户不要调用 API_FreePackage 去释放接收到的数据包，因为 ZCEAPI 会自行释放该数据包。

4.7.6 设置退出登录应答通知

void API_SetLogoutCallBack(ExchgConnectionHandle conn, ExchgPackageCallBack callback,void * CallBackArg);

功能： 设置收到退出登录应答通知时的回调函数

参数说明： 1），ExchgConnectionHandle conn 交易所连接对象句柄

2），ExchgPackageCallBack callback 要设置的返回数据包回调函数

3），void * CallBackArg 要设置的返回数据包回调函数的参数,它将最终传给返回数据包回调函数。请参看本手册中“**返回数据包回调函数**”。

使用提醒： 1），可以通过此函数的设置得到退出登录应答数据包，并进行相应的处理。

2），用户不要调用 API_FreePackage 去释放接收到的数据包，因为 ZCEAPI 会自行释放该数据包。