郑州商品交易所
ZCEAPIV2.2 使用手册

Version2.2

本文档适用于 ZCEAPI V2.2

郑州商品交易所

2025 年 4 月

目录

修改历史

API 版本	修改时间	修改内容	说明
2.0.0	2019 年 12 月	创建使用说明。	在 V1.1.3.7 基础上修改新版本使用说明
2.0.1	2021 年 1 月 20 日	增加 API_Init、API_Stop、API_ConnectEx 三个函数关于应急地址使用的说明。	
2.1.0	2021 年 8 月 08 日	1.删除 API_ConnectEx 接口，增加 API_Connect 接口； 2.API_CreateExchgConnection 函数增加是否自动启动业务驱动线程参数,同时增加接口 API_RunEvent； 3.删除查询流及应急地址相关描述。	
2.2.1	2025 年 4 月 16 日	1. 新增 API_UINT64 类型及 API_GFULong 和 API_SFULong 接口； 2.支持信创海光 x86 平台； 3.配合六期上线，修改相关描述。	

1. 概述

1.1 目的及说明

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

如果本文档描述内容与实际情况不符，请及时和郑商所技术支持联系并确认。

开发者须合理使用各开发接口，不得对交易所系统进行恶意影响或破坏，并对自己开发的系统进行充分测试，保证系统功能正确性。开发者不得采取 API 破解、协议破解等其他变相方式违规开发交易所接入功能。

本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》（《郑商所期货交易数据交换接口规范》）（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(APIDemo)展示了 ZCEAPI 的基本使用方法，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发布。

1.3 ZCEAPI 简介

ZCEAPI 是郑商所的远程交易编程接口，是客户应用程序与郑商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、行情数据报文协议。

ZCEAPI 封装和郑商所交易系统通信的底层连接和通信协议。用户在使用 ZCEAPI 和郑商所进行数据通信时，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。

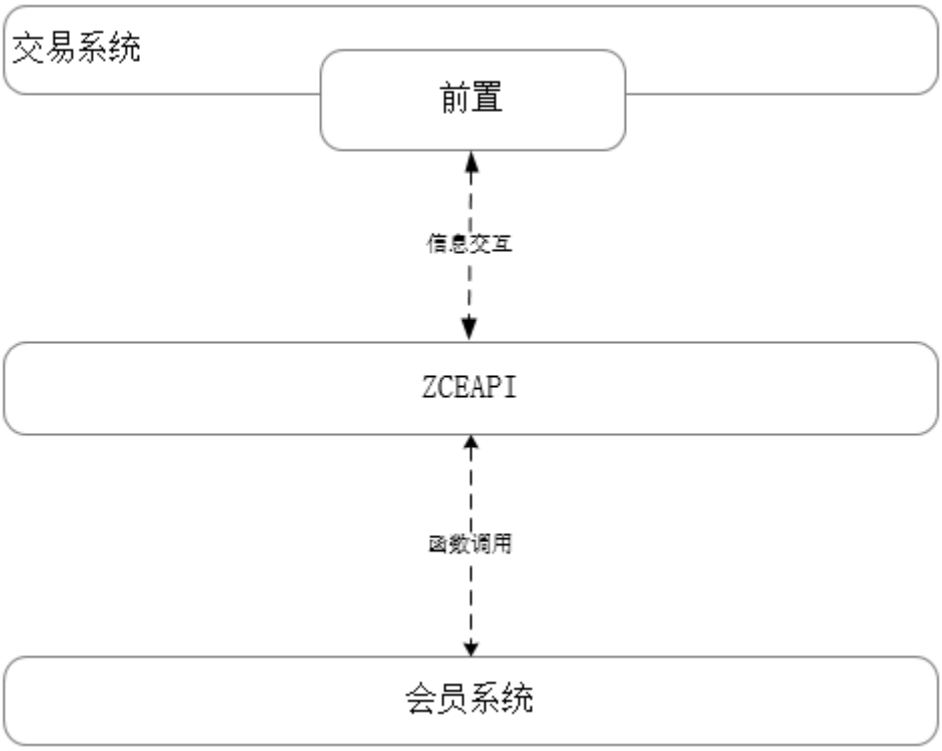
心跳机制由 ZCEAPI 自行维护。数据加密、解密，压缩、解压等工作由 ZCEAPI 内部完成，只需用户选择相应的参数。

ZCEAPI 使用回调机制通知链路事件和数据接收，当链路有错误发生、链路断开、接收到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 可以很方便的与郑商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

从业务层面讲，ZCEAPI 和郑商所后台系统之间的通讯方式目前分为对话通讯模式、私有通讯模式和广播通讯模式（后续可能会扩充新的通讯模式）。每一种通信模式用户程序都通过 ZCEAPI 建立可靠的 TCP 链路连接郑商所交易系统，完成交易业务和获取行情数据。

对话模式（对应的数据流为对话流），实现了所有事务操作及部分查询操作，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。

ZCEAPI 底层有选择性的采用 TCP 和 UDP 与交易系统进行通讯。对于每一个通讯模式的数据流（对话流、私有流、广播流）都有一个 TCP 连接与之对应。对于广播流来说，底层通信可能会附加额外的 UDP 通信，具体采用什么方式（单播还是组播）要根据用户登录广播流采用协议。具体信息内容参见《郑商所期货交易数据交换接口规范》。

1.6 ZCEAPI 使用环境

ZCEAPI 支持的操作系统平台主要有：Windows 和 Linux，均提供 64 位版本。Windows 支持 NT10.0 版本（最新版本需要测试兼容性），Linux 平台分别支持普通 x86 和海光 x86 硬件平台。普通 x86 支持内核 3.10 以上，gcc v4.8.5（支持 C++11 标准以上）以上版本，海光 x86 平台支持内核 4.19 以上，gcc v7.3.0（支持 C++11 标准以上）以上版本。

Windows 平台下它以动态库（.dll）的形式被应用程序调用。在 Linux/Unix 和信创海光 x86 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 除标准 C++ 运行时库依赖之外，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：

ZCEFTDAPI.dll ---------- ZCEAPI 的动态库。

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

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

Linux 平台下：

libZCEFTDAPI.so ---- ZCEAPI 的动态库文件。支持普通 x86 平台和信创海光 x86 平台。

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

2. ZCEAPI 使用说明

2.1 使用概述

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 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 请务必从郑商所获得相应平台下的 ZCEAPI 发布包，里面包含使用 ZCEAPI 所必需的文件。

一般情况下使用动态链接库，程序员必须对动态库提供的接口进行声明。在使用 ZCEAPI 时需对接口函数和基本数据定义进行声明。C/C++程序员可以直接使用随动态库发布的 FTDAPI.h 文件，FTDAPI.h 提供了 ZCEAPI 的所有声明。使用其他开发语言的用户，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 进行了相应的编码。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。

注：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的介绍。

使用 ZCEAPI 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 API_Init 函数初始化 API，这是用户必须做成功的第一步；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，所以在每个连接对象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 API_Stop 函数停止 API 服务。

ZCEAPI 的接口不是线程安全的。

ZCEAPI 的详细使用说明请参考 2.3 中的说明。

2.2 线程驱动简介

ZCEAPI 提供了两种驱动模式：API 自动驱动和用户驱动。ZCEAPI 用户可以在连接对象创建的时候选择是否让 API 自身启动线程驱动链路事件（主要是数据接收和基本解析处理），用户也可以选择自己启动线程调用 API 事件驱动函数驱动链路事件。

ZCEAPI 采用回调函数的方式处理事件，用户创建一个连接对象，设置好相应的回调函数，自动启动模式下，ZCEAPI 就会为其启动一个线程驱动这个连接对象上的事件。用户驱动模式下，用户需要自己启动线程调用 API 事件驱动函数驱动链路事件。

2.3 基本过程

自动驱动模式下的基本使用过程如下：

2.3.1 ZCEAPI 初始化

用户调用 API_Init 函数初始化 ZCEAPI。

ZCEAPI 成功初始化是正确使用 ZCEAPI 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 (API_Init("./log", "./APIData") == API_TRUE)
{
    std::cout << "Succeed in Init API!!!" << std::endl;
}
else
{
    std::cout << "Fail to Init API!!!" << std::endl;
    return 1;
}
```

注：关于初始化函数 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，必须要创建连接对象。创建连接对象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;                    //是否加密  1:是,0:否
API_BOOL Commpress = API_TRUE;                  //是否压缩  1:是,0:否
MARKET_ID Market_ID = MARKET_ZCE;       //即郑商所  1
int cpunum_for_driver = 1;              //把连接的驱动线程绑到指定的(编号为 1)CPU 核
```

```
上.
    int cpunum_for_udp = -1;            //这里 UDP 接收线程不做 CPU 绑定.(本示例中登录的是对话流,不会启动 UDP 接收线程,所以更没必要为这个参数设置正常 CPU 编号,即便设置也无效)
    //建立对话流连接对象
    ExchgConnectionHandle DialogConn = API_CreateExchgConnection(Encrypt, Commpress, Market_ID, cpunum_for_driver, cpunum_for_udp);
    if (DialogConn == NULL)
    {
        std::cout << "Can not create Dialog connection to ZCE！" << std::endl;
        return 3;
    }
```

注：

1）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）接口 API_CreateExchgConnection 已经取消心跳参数，心跳间隔和心跳超时间隔不再由用户自行配置。

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。也建议创建成功后马上就设置相应的回调函数。

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），但是如果不设置回调那么相应的事件将不会通知给用户，会影响用户业务的实现。

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;
}
```

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,this);   //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);
```

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

在向交易所发起连接之前，建议调用 API_SetConnectionOpt 函数设置链路的 TCP 心跳参数。这样防止 Socket 非优雅的断开时，操作系统检测到链路异常的时间过长。

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句柄，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
char IpAddr[20] = "218.29.68.231";          //交易所的登录地址服务的 IP 地址
int port = 22677;                           //连接交易所的端口
char ErrMsg[128];
if (API_Connect(DialogConn, IpAddr, port, 5000, ErrMsg） !=ERR_SUCCESS)
{//连接不成功
    cout<<" Can not connect with CZCE！"<<endl;
    return 0;
}
else
cout<<" Connect CZCE successfully "<<endl;
```

注：

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

2）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)一段时间后会自动被断开。

3）不同业务流使用独立的端口号，请根据交易所通知进行配置。

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，并通过数据包返回参数获取登录应答包信息（如果该返回值不为空的话）；
6) 调用 API_FreePackage 释放步骤 1 中生成的数据包。

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。具体示例请参考随发布包发布的 APIDemo。

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，此时用户应当调整 wait 的值。

2.3.6 读写数据包

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），可以对该句柄指向的 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 数据包的读写比较简单，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 对该字段进行读写。

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

注：

1) 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。
2) 具体使用实例请参考示例程序中的相关部分。

2.3.7 遍历读取数据包

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，用户必须先通过调用 API_FirstField 得到内部第一个字段，然后调用 API_NextField 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！

API_FirstField 和 API_NextField 结束之间不能对数据包进行增加和删除数据域。

遍历函数不是线程安全的，不要多线程同时遍历数据包。

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。

具体使用实例请参考示例程序中的相关部分。

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（**注意：同一连接对象的 API_Send 不是线程安全的）；**
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 释放步骤 1 中生成的数据包。

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

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**

  通过登录函数的出参。具体参见 API_Login 函数的定义。

- **接收退出登录应答数据包**

  通过登出函数的出参。具体参见 API_Logout 函数的定义。

- **接收交易所一般应答包**

  通过设置接收数据回调函数来处理接收到的数据包。

注： 具体使用实例请参考示例程序中的相关部分。

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系统，在收到交易所的正确的退出登录确认或链路断开后，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;并可以通过登出函数的输出参数得到交易所返回登出应答信息（如果出参不为空的话）；
6) 调用 API_FreePackage 释放刚步骤 1 中生成的数据包。

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。

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。

用户可以直接调用 ZCEAPI 接口函数 API_DisConnect 断开连接。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

在退出调用 ZCEAPI 的应用之前，建议先调用 ZCEAPI 服务停止函数 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;
typedef    unsigned long long API_UINT64;
```

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 市场约定

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

3.5 数据域类型定义

```
enum API_FIELDTYPE
{
      FTNULL = 0,
      FTCHAR = 1,
      FTLONG = 2,
      FTSTRING = 3,
      FTDOUBLE = 4,
      FTDATETIME = 5,
      FTINT64 = 6
};
```

3.6 数据流标示

```
typedef API_BYTE API_DFFLAG;
#define DFF_DIALOG              0     //对话流
#define DFF_PRIVATE       1     //私有流
#define DFF_BROADCAST       2     //广播流
```

3.7 数据流状态常数

```
//数据流状态常数
enum API_DFSTATUS
{
      DFS_CLOSED = 0,           //连接断开
      DFS_OPENED = 1,           //刚刚建立连接
      DFS_NEGOTIATEKEY = 2, //刚刚协商密钥，还没有登录。
      DFS_LOGIN_OK = 3    //已经登录成功
```

```
};
```

3.8 对象句柄

连接对象和数据包对象句柄，C++中声明为类对象指针，C 语言声明为 void 指针。

```
#ifdef __cplusplus
class MsgPackage;
class ExchangeConnection;
/*数据包句柄*/
typedef MsgPackage* MsgPackageHandle;
/*交易所连接对象句柄*/
typedef ExchangeConnection* ExchgConnectionHandle;
#else
/*数据包句柄*/
typedef void* MsgPackageHandle;
/*交易所连接对象句柄*/
typedef void* ExchgConnectionHandle;
#endif
```

3.9 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。

3.10 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 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 接收版本号字符串的缓冲区指针；

2）unsigned int buflen 接收版本号字符串缓冲区大小。

返回值： 成功返回版本字符串长度；

缓冲区不够返回-1，此时 buflen 返回需要的最小缓冲区长度。

使用提醒： 字符串正常格式为 x.x.x.x（x 为整数），如：2.10.16.35，原则上不会超过 16 个字节。

4.1.2 ZCEAPI 初始化

```
API_BOOL API_Init(const char* logFilePath, const char* APIDataPath);
```

功能： 初始化 ZCEAPI，指定诊断日志路径和诊断端口。

参数说明： 1）logFilePath API 日志文件路径，存在并且用户有写权限；

2）APIDataPath API 私有数据保存目录，存在并且用户有写权限。

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，会导致 ZCEAPI 初始化失败。

2）APIDataPath 私有数据保存目录，暂时未用到，其路径不存在不影响初始化。

3）在一个应用程序中只初始化一次。

4）ZCEAPI 初始化成功是后续一切工作的基础。若 ZCEAPI 不能成功地初始化，则后续操作不保证能成功执行。

4.1.3 停止 ZCEAPI 服务

```
void API_Stop();
```

功能： 停止 API 库的使用。

使用提醒： 调用该函数后,就不要再使用 ZCEAPI 的任何函数，除非重新初始化（尽量不要来回初始化和停止 API）。

建议在准备退出应用前调用该函数。

4.1.4 取当前时间

```
int API_Now(API_DateTime*dt);
```

功能： 取得当前本地系统的时间。

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

返回值： 成功获取时间返回 0，失败返回-1。

使用提醒： 通过该函数取出的当前系统本地时间值（localtime）保存到 dt 中，和本地系统时区设置有关。

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，否则返回 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： MsgPackageHandle pkg ZCEAPI 创建的一个数据包句柄。

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句柄，否则会出现异常。

2）回收数据包对象之后，不能再使用该对象。

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： MsgPackageHandle pkg 要取 PID 的数据包句柄。

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（参看《ZCEAPI 参考手册》）。

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

```
enum 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)API_UINT16 fid 要取类型的数据域的编号(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“**数据域类型定义**”章节。

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。

4.4 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）API_UINT16 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。C++语言默认为空格。

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,API_UINT16 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）API_UINT16 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 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,API_UINT16 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）API_UINT16 fid　　要取的数据域的数据域编码；

3）int def　　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-4（4 字节无符号整数）的字段适合使用该函数。

3）若数据域类型为 UN-4，那么返回值需要用户强制转换为无符号整型。

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,API_UINT16 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）API_UINT16 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-4（4 字节无符号整数）的字段适合使用该函数。

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 从数据包中取无符号长整数

```
API_UINT64 API_GFULong(MsgPackageHandle pkg,API_UINT16 fid,API_UINT64 def=0);
```

功能： 从指定数据包的指定数据域中按无符号长整数型得到无符号长整型数据。

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

2）API_UINT16 fid 要取的数据域的数据域编码；

3）API_UINT64 def 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 的数据域的无符号长整型数，取无符号长整数失败返回 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 UN-8（8 字节无符号整数）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）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 设置数据包中的无符号长整型数据域

```
int API_SFULong(MsgPackageHandle pkg,API_UINT16 fid,API_UINT64 val);
```

功能： 设置指定数据包中指定数据域的无符号长整数类型值。

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

2）API_UINT16 fid 要设置的数据域的数据域编码；

3）API_UINT64 val 要设置的无符号长整数值。

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 UN-8（8 字节无符号长整数）的字段适合使用该函数。

3）使用该函数之前必须保证 ZCEAPI 成功初始化，否则赋值失败。

4.4.7 从数据包中取双精度小数

```
double API_GFDouble(MsgPackageHandle pkg,API_UINT16 fid,double def=0.0);
```

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）API_UINT16 fid 要取的数据域的数据域编码；

3）double def=0.0 取数据失败时默认返回值，默认值为 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），如果填写之后 pkg 没有传输到后台系统或 copy，那么用户填写什么值，还能通过该函数取出该值，此时填写的精度不起作用。

5）使用该函数之前必须保证 ZCEAPI 成功初始化，否则取值失败。

4.4.8 设置数据包中的双精度数据域

int API_SFDouble(MsgPackageHandle pkg,API_UINT16 fid,double val,unsigned 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）API_UINT16 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）unsigned int precision 设置的小数的精度，默认为 4。最小 1 位精度。

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；

-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）的字段适合使用该函数。

3）precision 代表小数位数，如果 val 的小数位数大于 precision，那么 val 在发送到交易所时只保留前 precision 位小数有效。不传输或 copy 时，用 API_GFDouble（）取相应字段的值精度不会损失。

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）郑商所交易系统目前支持的最大值为 12 位整数 2 位精度 或者 8 位整数 4 位精度（具体参见参考手册）。如果填写超出范围浮点数，将不能被有效传递。

6）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 从数据包中取字符串

```
int API_GFString(MsgPackageHandle pkg,API_UINT16 fid, char* buf,unsigned 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）API_UINT16 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）unsigned int bufsize 接收缓冲区大小。

返回值： 返回值>0: 当返回值<=bufsize,返回获取字符串的长度；当返回值>bufsize 表示给定的缓冲区太小,只是把 bufsize 长度的数据复制到了 buf 中，返回值为取完整数据需要的缓冲区长度。

返回值==0，表示取不到该字段的值，或给的参数就不可能取到(比如 buf 为空,bufsize 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）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）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，从而只取字段的一部分而非完整字段。

5）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。

6）函数中的字符串**不一定是可见字符串**。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。

7）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）

8）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 设置数据包中的字符串数据域

```
int API_SFString(MsgPackageHandle pkg,API_UINT16 fid,const char* buf,unsigned 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）API_UINT16 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）unsigned 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。

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。

4.4.11 从数据包中取日期时间

```
API_BOOL API_GFDateTime(MsgPackageHandle pkg,API_UINT16 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）API_UINT16 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（按北京时间（UTC+8）解析）。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）的字段适合使用该函数。

3）使用该函数之前必须保证 ZCEAPI 成功初始化，否则取值失败。

4）API 时间是按照北京时间（UTC+8）解析，也就是获取的时间都是按北京时间解析。比如：API_DateTime(2019,11,23,9,45,08,234208),表示的是北京时间 2019 年 11 月 23 日 09 点 45 分 08 秒 234208 微秒。

4.4.12 设置数据包中的日期时间数据域

```
int API_SFDateTime(MsgPackageHandle pkg,API_UINT16 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）API_UINT16 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（按北京时间（UTC+8）解析）。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；
-2：	不可传输的时间（本地保存，不可传输和拷贝）；
-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）的字段适合使用该函数。

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）API 时间是按照北京时间（UTC+8）解析，也就是设置的时间都是按北京时间解析。比如：API_DateTime(2019,11,23,9,45,08,234208),表示的是北京时间 2019 年 11 月 23 日 09 点 45 分 08 秒 234208 微秒。

5）设置的时间大致应该在 1970 年 1 月 1 日 08:00:00 ~ 3000 年 12 月 31 日 23:59：59（UTC 时间）之间，用户可以根据函数返回值判断。

4.4.13 判断某个字段是否为空

```
API_BOOL API_FieldIsNull(MsgPackageHandle pkg,API_UINT16 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）API_UINT16 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）该函数等效于:API_GetFieldType() == API_FIELDTYPE::FTNULL

3）使用该函数之前必须保证 ZCEAPI 成功初始化。

4.4.14 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）API_UINT16 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）使用该函数之前必须保证 ZCEAPI 成功初始化。

4.4.15 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： 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。

2）清除数据包所有字段，不包括 PID 的清理。

4.4.16 数据包复制

```
int 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。

返回值：　复制成功返回 0，否则返回非零错误码。

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。

3）该函数可能造成设置的双精度浮点数只保留设置的精度。

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：　找到要遍历数据包的第一个数据域。

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

返回值：　0：　遍历时第一个数据域的 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）在遍历数据包时必须先调用该函数。

5）数据包遍历结束之前不能对 pkg 进行增加和删除数据域。

6）函数不是线程安全的，不要多线程同时遍历数据包。

4.5.2 下一个数据域

```
int API_NextField(MsgPackageHandle pkg);
```

功能：　得到遍历数据包的下一个数据域的编号。

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

返回值：　>0：　遍历数据包时下一个数据域的 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）数据包遍历结束之前不能对 pkg 进行增加和删除数据域。

5）函数不是线程安全的,不要多线程同时遍历数据包。

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 thread_bind_cpu, int thread_bind_cpu_udp, API_BOOL drive_auto = API_TRUE);
```

功能： 创建一个交易所连接对象。

参数说明：

1）API_BOOL Encrypt 是否加密。API_TRUE 加密，API_FALSE 不加密，目前只支持加密；

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）thread_bind_cpu 连接的业务驱动线程绑定的 CPU 编号；

5）thread_bind_cpu_udp 广播流采用 UDP 接收时驱动线程绑定的 CPU 编号；

6）drive_auto 是否自动启动业务驱动线程。

返回值： 成功返回交易所连接对象句柄，失败返回空指针。

使用提醒：

1）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。

2）CPU 的编号(从 0 开始)，参数值为负数或超出本机 CPU 个数，则不绑定。如果绑定，驱动线程会一直占用该 CPU 核心。

3）对不采用 UDP 协议收取广播流的连接，thread_bind_cpu_udp 参数设置不会生效。因为该驱动线程不会启动。

4）如果 drive_auto 为 API_TRUE 或不设置该参数时，则该连接对象会自动启动业务驱动线程；drive_auto 为 API_FALSE，API 不再为该连接对象启动驱动线程，用户必须自行启动线程并调用该连接的事件驱动函数 API_RunEvent。

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。Linux 平台单位为秒，Windows 平台单位为毫秒；

3）int keepInterval 连接检测间隔。Linux 平台单位为秒，Windows 平台单位为毫秒；

4）int keepCount 检测次数。

返回值： 成功设置返回 0，否则返回错误码。

使用提醒： 该函数在发起连接前调用设置才有效。

4.6.3 事件驱动

```
int FTDAPI_CALL API_RunEvent(ExchgConnectionHandle conn);
```

功能： 当用户调用 API_CreateExchgConnection 创建连接对象，指定 drive_auto 为 API_FALSE，用户线程替代自动线程驱动连接业务时，用户线程需要调用的事件驱动函数。

参数说明： ExchgConnectionHandle conn 交易所连接对象句柄。

返回值: 0： 事件正常处理；

3000： 参数有误；

3029： 非法调用；

其他，说明事件处理过程中发生错误，链路中断，事件处理结束。

使用提醒： 1）如果创建连接对象时，采用的是自动启动业务驱动线程（drive_auto = API_TRUE），请不要调用该函数。

2）该函数不是线程安全，不要多线程同时调用。

4.6.4 发起与交易所的连接

```
int API_Connect(ExchgConnectionHandle conn, const char* IP, unsigned short port, int wait, char* errMsg);
```

功能： 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* IP 交易前置 IP 地址；

3）unsigned short port 交易前置服务端口；

4）int wait 超时等待时间，毫秒数；

5）char * errMsg 输出参数。返回错误信息的内存空间，由用户分配，最少 64 个字节。

返回值： 0： 连接成功；

3000： 参数有误；

3027： 回调中不能调用；

3035： 连接交易前置失败。

使用提醒： 1）参数 conn 必须是 ZCEAPI 创建的连接对象的句柄。

2）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。

4.6.5 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： 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

```
int API_Login(ExchgConnectionHandle conn,MsgPackageHandle reqPkg, double Wait , MsgPackageHandle* rspPkg);
```

功能： 登录三个数据流中的任一个到交易系统。

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；
4）MsgPackageHandle * rspPkg	输出参数，保存登录应答包句柄的指针。如果没有应答为空指针。

返回值：

0： 登录成功；

3000： 参数错误；

3001： 连接还未建立；

3002： 链路协商数据发送错误；

3003： 链路协商超时；

3004： 登录数据发送错误；

3005： 登录超时；

4001： 登录协议版本号错误；

3018： 数据流标示错误；

3019： 不能重复登录；

3020： 登录没有指定席位；

3022： 协商密钥验证失败；

3027： 驱动线程不能调用；

3051： 协商密钥生成错误。

后台错误码：

2007： 数据流登录模式不正确；

610： 非法交易员代码；

262： 交易员已被挂起；

673： 非法登录请求；

674： 非法口令；

675： 你不能从该工作站登录系统；

2005： 登录协议版本错误；

40： 不能从该工作站登录系统,请求私有流数量过多；

7001： 重复登录。

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交易员、流编号、密码等正确的登录信息，具体参考《ZCEAPI 参考手册》。

3）该函数为阻塞函数，当参数 wait 的值不足够大时，该函数会因超时而返回 API_FALSE。参数 wait 的值不能太小，更不能等于零。

4）rspPkg 指向的 MsgPackage 只读。

5）该函数为同步函数，不要在 API 的回调函数中调用。

4.6.7 退出登录

```
int API_Logout(ExchgConnectionHandle conn,MsgPackageHandle reqPkg, double Wait, MsgPackageHandle* rspPkg);
```

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；

4）MsgPackageHandle *rspPkg 输出参数，保存登出应答包句柄的指针。如果没有应答为空指针。

返回值： 0： 登出成功；

3000： 参数错误；

3006： 没有登录无法登出；

3018： 数据流标示错误。

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。

4）rspPkg 指向的 MsgPackage 只读。

5）该函数为同步函数，不要在 API 的回调函数中调用。

6）链路断开链路登录状态失效，相当于自动登出。

4.6.8 发送一个数据包

```
int 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： 0： 发送成功；

3000： 参数错误；

3009： 没有登录不能发送；

3010： 数据包格式错误；

3015： 数据包不能被识别，可能是 API 版本过低；

3016： 数据发送错误；

3018： 数据流标示错误；

3024： 数据包 PID 错误；

3025： 数据包为空；

3028： 太过频繁的查询。

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）函数不是线程安全的，不要多线程同时在一个连接对象上发送数据。

5）API 会限制用户发起并发查询的数量，超过限制时发送查询命令会失败，错误码为 3028（太过频繁的查询）。

4.6.9 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： 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。

4.6.10 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： 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。

4.6.11 取得数据流状态

```
enum 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。

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**”。

4.7.2 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**”。

使用提醒：对于采用 UDP 接收广播流数据的链路，会有两个线程并行回调设置的回调函数可能性，设置的回调函数需要做好线程同步处理。

4.7.3 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）登录登出应答数据包通过登录登出函数中的输出参数返回给用户。
2）用户不能调用 API_FreePackage()去释放回调的数据包。
3）对于采用 UDP 接收广播流数据的链路，会有两个线程并行回调设置的回调函数，设置的回调函数需要做好线程同步处理。