ZCEAPI 使用手册

0．修改历史

修改时间	修改内容	修改人	说明
2007-12-19	1，登陆协议版本号，不再是默认 1，必须填写合适的值。 2，2.1.1 –5)中说明。 3，	张卓亚	为了适用新版本改动

1．数据定义

1.1 基础数据类型

typedef　unsigned short int　**API_UINT16;**

typedef　unsigned char　　**API_BYTE;**

1.2 日期时间类型

```
typedef struct tagDateTime
{
    API_UINT16 year;
    API_BYTE  month,day,hour,minute,second;
    API_UINT16 microsec;
}Date_Time;
```

1.3 API_BOOL

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

1.4 线程模式

```
typedef int THREAD_MODE;
#define SINGLE_THREAD_MODE  0
#define MULTI_THREAD_MODE   1
```

1.5 建立连接时是否阻塞

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

1.6 交易所约定

```
typedef int MARKET_ID;
#define MARKET_ZCE 1
#define MARKET_DCE 2
#define MARKET_SFE 3
```

1.7 数据域类型定义

```
#define FTNULL          0
#define FTCHAR          1
#define FTLONG          2
#define FTSTRING        3
#define FTDOUBLE        4
#define FTDATETIME      5
```

1.8 数据流状态常数

```
#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
```

1.9 数据包句柄

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

1.10 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;
```

1.11 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。

1.12 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。

2. 函数介绍

2.1 ZCEAPI 管理

2.1.1 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，指定监控日志路径和监控端口，打开监控端口。

参数说明： 1)，THREAD_MODE mode 线程模式：0 为单线程，1 为多线程。

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

3），int MonitorPort　　监控端口端口号，建议设置为 0

返回值：　初始化成功返回 API_TRUE （1），否则返回 API_FALSE （0）

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），**不要先初始化为一种线程模式，然后停止 ZCEAPI 服务，随后又初始化为另一种线程模式，并将前一种线程模式下创建的 ZCEAPI 对象用于当前线程模式。**

4），在初始化为单线程模式时，在 Windows 环境下若不能成功初始化，请先调用：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。如果用户向更改这个端口请在初始化前在当前应用程序目录适用文件“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。

2.1.2 停止 ZCEAPI 服务

void　ftd_api_stop();

功能：　停止监控，关闭日志文件

使用提醒：　使用该函数停止 ZCEAPI 服务后可能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。

2.1.3 单线程事件处理驱动

int　ftd_api_event_loop(double wait=0.001);

功能：　在单线程模式下驱动事件处理。

参数说明：　1）double wait　没有事件可处理的最长等待时间。单位是秒。默认是 0.001 秒。如果 wait=0,那么函数直到如“收到数据”、“心跳超时”等事件发生时才返回。

返回值：　有事件发生则返回要处理的事件个数。没有事件需要处理，等待 wait 秒，在 wait 秒内有事件要处理，则进行处理。在 wait 时间内无事件发生，超时返回 0。有错误发生则返回-1。

使用模式：　单线程模式

使用提醒：　调用该函数驱动事件处理，那么该函数调用应该和 ZCEAPI 初始化（调用函数 ftd_api_init（））、登录操作、退出登录操作等在同一个线程中。

2.2 数据包管理

2.2.1 创建数据包对象

MsgPackageHandle API_AllocPackage();

功能：　　创建一个 ZCEAPI 数据包

返回值：　　返回创建的数据包句柄

2.2.2 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掉

参数说明：　ZCEAPI 创建过的一个数据包句柄

使用提醒：　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。

2.3 报文数据操作

2.3.1 取得报文 ID

int API_GetPID(MsgPackageHandle pkg);

功能：　　取得指定报文的 PID

参数说明：　要取 PID 的数据报文句柄

返回值：　　成功返回指定数据包的 PID

使用提醒：　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。

2.3.2 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。

2.3.3 取得某字段的当前数据类型

int API_GetFieldType(MsgPackageHandle pkg,int FieldID);

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 FieldID　　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 FieldID 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”**项中的定义。FTNULL 表示在 pkg 中没有找到编号为 FieldID 的数据域。

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》中规定的 FID。

2.4 字段操作

2.4.1 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）的适合使用该函数。

2.4.2 设置数据包中的字符数据域

void 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　　要取的数据域的数据域编码

3)，char val　　要设置的字符值

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）的适合使用该函数。

2.4.3 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。当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。

2.4.4 设置数据包中的整型数据域

void　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　　要取的数据域的数据域编码

3)，int val　　要设置的整数值

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）的适合使用该函数。

2.4.5 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。当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 LN 的适合使用该函数。

3），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。

2.4.6 设置数据包中的双精度数据域

void 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　　要取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

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 的适合使用该函数。

3），precision 代表小数位数，如果 val 的小数位数大于 precision，那么 val 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。

2.4.7 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）和 DT（日期时间）的适合使用该函数。

3),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。

4)，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。

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,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。

6),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，毫秒 都不为 0，此时取出的格式为：YYYY-MM-DD HH:mm:ss.iii

② ，DT 结构对象中，年，月，日，时，分，秒 都不为 0，毫秒为 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 ，毫秒都不为 0，日期为 0 时，此时取出的格式为：HH:mm:ss.iii

⑤ DT 结构对象中，时，分，秒 都不为 0，日期和毫秒为 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:毫秒　　如：YYYY 表示:4 位数字表示年 ，iii 表示:3 位数字表示毫秒值

2.4.8 设置数据包中的字符串数据域

void 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 要取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

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。

2.4.9 从数据包中取日期时间

API_BOOL API_GFDateTime(MsgPackageHandle pkg,int fid,Date_Time*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)，Date_Time*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。

2.4.10 设置数据包中的日期时间数据域

void API_SFDateTime(MsgPackageHandle pkg,int fid, Date_Time*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　　要取的数据域的数据域编码

3)，Date_Time*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”。

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）的适合使用该函数。

2.4.11 判断某个字段是否为空

API_BOOL　API_FieldIsNull(MsgPackageHandle pkg,int FieldID);

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 FieldID　　要取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

2.4.12 清除某个特定的字段的值

void　API_ClearField(MsgPackageHandle pkg,int FieldID);

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 FieldID　　要取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。

2.4.13 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。

2.4.14 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。

2.5 数据遍历

2.5.1 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。否则返回-1。

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。

2.5.2 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。否则返回-1。

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。

2.5.3 取当前数据域类型

int 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。

2.5.4 取当前数据域的时间

API_BOOL API_CFDT(MsgPackageHandle pkg,Date_Time *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)，Date_Time *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。

2.5.5 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，返回取得的字符型。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。

2.5.6 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。

2.5.7 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。

2.5.8 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“FTSTRING”的适合使用本函数。

3），接收缓存区大小要足够大，否则可能会截断取得的字符串。

4）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。

2.5.9 取当前时间

void API_Now(Date_Time*dt);

功能： 取得当前本地系统的时间

参数说明： 1)，Date_Time*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 中表示毫秒的字段为 0。

2.6 连接管理

2.6.1 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。

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“**交易所约定**”

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)，创建的连接与初始化的线程模式有关。在单线程模式下创建的连接对象应在单线程模式下使用，多线程模式下创建的连接对象应在多线程模式下使用。交叉使用可能会导致用错连接对象接收不到数据，对错误事件不做出反应等问题。

2)，heart_beat_interval 应该小于 heart_beat_timeout 的 1/2。

3)，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。

2.6.2 发起与交易所连接

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

功能： 连接交易所，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。

2.6.3 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

2.6.4 登录到交易所

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

功能：　登录到交易所，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），在单线程模式下该操作应该和初始化、事件驱动在同一个线程。

7),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。

2.6.5 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认证、流标志、流编号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)，在单线程模式下该操作应该和初始化、事件驱动在同一个线程。

2.6.6 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》。

2.6.7 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。

2.6.8 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。

2.6.9 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。执行该函数后连接句柄 conn 将不能再被使用。

2.6.10 取得数据流状态

int　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**”。

2.7 设置回调函数

2.7.1 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 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。请参看一中“**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。

2.7.2 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。请参看一中“**链路状态回调函数**”。

2.7.3 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。请参看一中“**链路状态回调函数**”。

2.7.4 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。请参看一中“**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。

2.7.5 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。

2.7.6 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。

3，接口使用介绍

3.1 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 提供两种使用模式：多线程模式和单线程模式。

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。用户可以完全摆脱复杂的 FTD 协议。

心跳由 ZCEAPI 自行维护。加密、解密，压缩、解压等也只需用户选择相应的参数，具体工作由 ZCEAPI 自行完成。

ZCEAPI 使用回调机制，当链路有错误发生、打开链路、链路断开、收到用户在该链路的登录应答包、收到用户在该链路的退出登录应答包、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。

ZCEAPI 还具有链路监控的功能，交易所和其他用户可以连接到 ZCEAPI 的监控端口，对 ZCEAPI 的工作状态和与交易所的连接状态等信息进行监控，有助于发现和解决链路出现的问题。

3.2 单线程模式和多线程模式的区别

ZCEAPI 提供两种模式给用户使用：单线程模式和多线程模式。用户在 ZCEAPI 初始化时

指定使用哪种线程模式。

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。

单线程模式下用户创建一个连接对象时，ZCEAPI 并不会为其启动一个线程驱动该连接对象上发生的事件。所以在单线程模式下用户必须自行调用 ZCEAPI 提供的事件驱动函数来驱动所有可能发生的事件。并且用到的 ZCEAPI 的函数调用应在一个线程内完成。

具体使用时，除了以上所述的事件驱动方式不一样外，其余的处理两种线程模式基本相同。

对于监控系统，无论是单线程模式下还是多线程模式下 ZCEAPI 都会进行处理。用户不必关心监控系统的工作。

这里的多线程模式和单线程模式主要是指用户使用 ZCEAPI 处理“协议转换相关事件”的线程模式。也可以说这里的多线程模式是 ZCEAPI 自动处理的模式；单线程模式是用户在使用时自主驱动的模式。

3.3 多线程模式下的使用

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，不必自己再驱动事件处理。

下面是 ZCEAPI 在多线程模式下使用的基本过程：

3.3.1 ZCEAPI 初始化

用户调用 ftd_api_init 函数初始化 ZCEAPI（初始化参数选择多线程模式）。

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，ZCEAPI 不能保证正常的运行。系统初始化成功后，监控功能即可使用。

例如：

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

注：关于初始化函数 ftd_api_init 的使用，请参考本手册第二部分对该函数的介绍。

3.3.2 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 =1;
API_BOOL      Commpress =1;
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<<"创建交易所连接失败！"<<endl;
      return 0;
 }
```

注：关于初始化函数 API_CreateExchgConnection 的使用，请参考本手册第二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。

3.3.3 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）

回调函数可设置多个，也可有选择的设置。

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产生很大的影响。

3.3.3.1 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。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一部分中的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;
}
```

3.3.3.2 设置回调函数

定义过回调函数后就可调用 ZCEAPI 相应的设置回调函数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：关于设置回调函数的函数的具体介绍请参考本手册第二部分中关于设置回调函数的函数的介绍。

3.3.4 连接交易所

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，连接成功即此连接对象被打开。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<<"连接交易所失败!"<<endl;
    return 0;
}
else  cout<<"成功连接到交易所！"<<endl;
```

注：1) ,关于连接函数 API_Connect 请参考本手册第二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，连接时要根据连接对象创建时选择是否加密而连接相应的端口。如：在 3.3.2 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），如果交易所提供的加密端口为 22677，非加密端口为 22577，那么这里就要选择 22677 进行连接。

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。

3.3.5 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。用户可以在一个连接对象上登录对话流、私有流、广播流中的任意的一个流，或任意两个流，或者同时登录三个流。但登录一次只能发送一个登录包，也就是说只能登录到一个流，不能一次登录两个或三个流。

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)
{
    //先定义用到的变量
    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]="1";          /*使用 FTD 版本号 目前=1*/
    char IpAddr[20]="192.168.0.8";   /*登录者的 IP 地址 如 192.168.99.100*/
    char INIT_PASSWORD[20]="77GgGwG7"; //联络初始密钥
    char AUTH_SERIAL_NO[20]="WZab77Z7-7n979w7w" ;   //序列号：
    char AUTH_CODE[20]="77GgGwG7-ZU9b9ww6" ;//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);//创建时得到的
    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_PASS
            WORD));
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<<"登录交易所成功！"<<endl;
    return ture;
}
else
{
    cout<<“登录交易所失败！"<<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二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掉，可以再次使用，不过再次使用前必须清空该数据包。

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，用户可能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。

3.3.6 读写数据包

像在 3.3.3.1 和 3.3.5 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《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
API_SFDouble(APIpkg，FID_ BuyPrice,1555.552，4);  //写
```

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二部分中相应的介绍。

具体使用实例请参考示例程序中的相关部分。

3.3.7 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和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二部分相关的介绍。

有关数据包遍历的例子请参考示例程序中的函数“APIpkgTofile”。

3.3.8 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 );
Date_Time 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二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，不过再次使用前必须清空该数据包。

3.3.9 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 3.3.3.1 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：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。

3.3.10 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。

用户可以有选择的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（登录为：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_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二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掉，可以再次使用，不过再次使用前必须清空该数据包。

c, 用户也可以调用 API_Send 函数发送退出登录请求包，不过用户要通过退出登录回调函数来判断是否退出登录成功。但是不建议这样做。

3.3.11 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，然后再释放掉连接对象。

断开连接用户可以直接调用 ZCEAPI 接口函数 API_DisConnect。成功执行后，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二部分对对其具体的介绍。

3.3.12 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 的监控功能。ZCEAPI 服务停止后监控客户将无法再监控到 ZCEAPI 的工作信息。

建议用户不要随便停止 ZCEAPI 的服务，这样不但会不能再监控 ZCEAPI 的工作信息，也可能会对 ZCEAPI 的工作造成不利的影响。

3.4 单线程模式下的使用

3.4.1 使用概述

如果用户要使用单线程模式，需要在 ZCEAPI 初始化时将线程模式指定为“单线程模式”。

用户在使用单线程模式时，所有的 ZCEAPI 调用应该在一个线程中。否则请考虑使用多线程模式。

通过 3.2 中的介绍，可以了解到单线程模式和多线程模式使用时的主要不同在于单线程模式下用户要自行调用 ZCEAPI 接口函数 ftd_api_event_loop 来驱动 ZCEAPI 的事件。也就是说如果用户以合适的方式驱动 ZCEAPI 事件处理，其他操作和多线程没有任何区别。

因为其他操作和多线程模式下相同，本部分不再介绍单线程模式下其他操作的使用。具体请参考本手册中 3.3 部分的介绍。下面重点介绍单线程模式下 ZCEAPI 的事件驱动。

3.4.2 使用示例

应用程序使用 ZCEAPI 的单线程模式，典型的用法有：循环的调用驱动函数驱动 ZCEAPI 事件的处理，设置一个定时器定时地驱动 ZCEAPI 事件的处理 等。

下面我们对以上两种方法给出示例：

- 设置一个定时器定时的调用函数 ftd_api_event_loop 驱动事件处理

  在 Delphi 中使用 ZCEAPI 的单线程模式，设置一个定时器，在其 Timer 事件中调用驱动函数实现 ZCEAPI 事件驱动。

如下：

```
procedure TFormMain.LoopTimerTimer(Sender: TObject);
var i:integer;
begin
        i:=20;
         while i>0 do
         begin
           if    ftd_api_event_loop(0.01)<=0 then exit ;   //驱动事件驱动器一次,
                                                  //直到没有可做的事或有错误发生
             dec(i);
         end;
end;
```

- 循环的调用驱动函数驱动 ZCEAPI 事件的处理。

  用户可以在一个循环中调用 ZCEAPI 单线程事件驱动函数 ftd_api_event_loop，驱动 ZCEAPI 事件处理。

如下：

```
..... //原来的处理
//发起对交易所的连接
if ( API_Connect(conn,IpAddr,port,5)!=API_TRUE)   //连接不成功
{
      coutprint("Can not connected with CZCE！ ");
      return 0;
}
else
```

```
{
    coutprint ("Connect CZCE successfully！ ");
}
//连接成功后登录交易所
if (!ZCELogin(conn,0)) {coutprint("登录对话流失败！ ");return 0;}   //不成功登陆
if (!ZCELogin(conn,1)) coutprint("登录私有流失败！ ");   //不成功登陆
if (!ZCELogin(conn,2)) coutprint("登录广播流失败！ ");   //不成功登陆

//登录成功后循环的接收命令，处理命令
while( API_Connected(conn)==API_TRUE)   //链路断开之前
{
    bool quit=false;
    coutprint("Wait for command: ");
    std::cin>>command;
    switch (command)
    {
    case 'Q':  {quit =true;break;}//输入Q退出
    case 'S':  {marketQrysend(conn);break;} //发出市场状态查询请求
    case 'M': {marketStatusQrysend(conn);break;} //发出市场状态查询请求
    default: ;
    }
    command=' ';   //将 command 还原
    if(quit) break;   //退出
    while (ftd_api_event_loop(1)){} //驱动，直到无事可做
}

. . . . . .    //继续进行其他处理
```

注：A，建议在发起对交易所的连接后适时开始驱动事件，因为系统的心跳机制是要靠事件驱动才能完成的。

B，可以根据 ftd_api_event_loop 的返回值判断本次驱动是否处理了事件。

C，对于函数 ftd_api_event_loop 的具体介绍请参考本手册第二部分对其的介绍。

D，用户也可以另外启动一个线程进行相关处理，但另外启动一个线程去处理 ZCEAPI 操作（如驱动事件处理）时，一定要注意 ZCEAPI 初始化、事件驱动、建立连接、登录和退出登录等操作要保持在一个线程中。即使如此，也

不建议这样使用，用户可以考虑使用多线程模式。

4. 监控接口介绍

4.1 监控的使用

ZCEAPI 初始化成功后，监控用户可以使用 telnet（或其他监控软件）登录到 ZCEAPI 初始化时指定的监控端口。

例如：用户（使用 ZCEAPI 的用户）的 IP 地址是 192.168.0.6，ZCEAPI 初始化时指定的监控端口为：2222。那么监控用户就可以用如下命令登录到 ZCEAPI 的监控服务程序：telnet 192.168.0.6 2222

登录之后，输入具体的监控命令可以查看具体的监控信息。

4.2 监控命令

ZCEAPI 监控系统支持的具体命令如下：

'x'：	开始实时监控
'z'：	停止实时监控
't'：	察看连接状态
'l'：	查看所有日志信息
'm'：	查看最近日志信息
'='：	移动到日志信息的尾部，并向上翻一行
'a'，'w'，'-'：	日志信息向上翻一行
's'，' '：	日志信息向下翻一行
'0x0d','0x0a','d'：	日志信息向下翻一页

如果信息一页不能显示完，会显示 "<more...>$: "，等待用户输入下翻（或上翻）命令查看后面（或前面）日志信息。

注：

1)，'0x0d','0x0a'为回车换行符。也就是说，输入"Enter"会下翻两页。

2)，日志信息和链路信息是在用户输入相应的请求时才发送给监控用户的。

3)，日志信息在 ZCEAPI 内部按"行页"组织。如"移动到日志信息的尾部，并向上翻一行 "的意思是将最后一行信息发给用户。

4)，当日志信息指针移到开始位置，再执行上翻命令，监控用户会得到："reach the first line"

5)，当日志信息指针移到结束位置，再执行下翻命令，监控用户会得到：“reach the last line”

6)，所谓“实时监控”是指如果有监控日志信息，马上发送给监控客户端。

7)，如果用户输入的命令不是上面的命令之一，那么监控系统将断开与用户的连接。