深证通信工程技术文档

FDEAPI 用户手册

文档编号： **FDEP-FDEAPI-PUM001**

文档密级： 外部公开

深圳证券通信有限公司
SHENZHEN SECURITIES COMMUNICATION CO.,LTD.

2012 年 8 月

文档信息

文档名称	FDEAPI 用户手册
说明	
所属项目	金融数据交换平台

修订历史			
日期	版本	修改人员	修改说明
20060604	0.9	蒋春风、王宏	完成初稿。
20060620	0.9.1	王宏	适用于 C 语言的接口修改。
20060901	0.9.2	蒋春风	增加了 MrReceive3 和 MrGetVersion 接口函数。
20060927	0.9.3	蒋春风	对 MrSend 函数中的 pMsgPropery 成员 m_szCorrPkgID 的使用进行了约定,并增加了对日志的说明。
20061012	0.9.4	蒋春风	修改了对 STUMsgProperty 结构中成员 m_szCorrPkgID 的使用的约定; 细化了各接收函数中超时参数 iMillSecTimeo 的说明。
20061129	0.9.5	蒋春风	新增加了 AIX 版本的 API，并且将 mrapi.h 头文件中的注释改为 C 语言风格的注释。
20061215	0.9.6	蒋春风	第一版正式发布。
20070313	01.04.20070313	蒋春风	第二版正式发布，版本号为 01.04.20070313,对于 FDEAPI 修改了内部协议，并作了以下修改: 1. 对于浏览或接收消息中的 m_szCorrPkgID 的字段作为输入条件增加了<EMPTY>和<NOEMPTY>特殊处理的说明。 2. 增加了函数错误返回值的定义。 3. MrGetVersion 函数的结果，为了方便版本号的比较，格式由 ”20060901.01.04” 修改为”01.04.20060901”.
20070516	01.04.20070313	蒋春风	修改了关于 AIX 版本的说明; 修改关于日志配置文件的说明; 将示例程序中发送消息包的内容修改为标准的消

			息包格式。
20070606	01.04.20070313	蒋春风	修改了关于 AIX 版本的说明; 并增加了关于接口函数 MrInit2 的说明。
20070629	01.04.20070313	蒋春风	修改了关于 AIX 的 HP-UX 版本的说明; 并增加了关于接口函数 MrInit2 的参数 pOnRecvMsgPropery 的说明。
20071211	01.04.20071102	黄新芳	修改平台名称和缩写。
20090617	02.01.20090324	王晨旭	按照 02.01.20090324 版本修改相关内容
20090803	02.02.20090801	蒋春风	准备发布新版本前的修改
20120730	04.01.20120730	利驿飞	在 FDEP V4 中废弃 MrBrowse 函数功能。

目　录

图 索 引

1 引言

本文是金融数据交换平台客户端应用程序开发接口 **FDEAPI**（Financial Data Exchange Application Programming Interface）的使用手册。主要说明使用 FDEAPI 进行开发的方法、各个 API 函数的功能、输入、输出等，以指导开发人员使用 FDEAPI。

本文的读者为使用 FDEAPI 的开发人员、测试人员、维护人员等。

其他术语英文缩写：

FDEP：Financial Data Exchange Platform，金融数据交换平台。

FDSH：FDEP Switching Hub，金融数据交换平台的交换中枢。

FDSU：FDEP Switching Unit，金融数据交换平台交换单元，BSSH 的构成元素。

FDAP：FDEP Access Point，金融数据交换平台的接入客户端。

FDMR：Financial Message Router，金融消息路由器，FDAP 的构成元素。

FPG：Financial Protocol Gateway，银证协议转换网关。

2 安装及应用发布

2.1 Windows平台

FDEAPI 目前支持 32 位 Windows 2003 和 64 位 Windows 2008 操作系统。Windows 平台的 FDEAPI 由以下几个文件组成：

文件名	说明
mr2api.h	开发用头文件。
mrapi.dll	FDEAPI 动态链接库，用 FDEAPI 开发的应用运行时用到。
mrapi_log.conf	日志配置文件。

使用 FDEAPI 进行开发，建议将以上全部文件拷贝到用户的可执行文件同一个目录下；也可以将 dll 文件拷贝到适当的路径下，将其它文件拷贝到开发环境相应的路径下。

（1）发布 FDEAPI 应用时，请把 mrapi.dll 和 mrapi_log.conf 一起发布。

2.2 其他平台

FDEAPI 目前还支持 64 位 Linux 的 RHEL Server 6.2 操作系统。

FDEAPI 可根据用户环境要求进行定制，以支持其他平台上的开发和使用。

2.3 关于日志的说明

FDEAPI 在使用过程中，可以产生运行日志并记入日志文件。。

日志配置文件 mrapi_log.conf 可以控制 FDEAPI 日志的产生和输出。文件格式如下：

```
[LOG]
Type=1
LockType=1
Level=0
Display=1
LogDir=./log
LogName=mrapi.log
MaxFileCount=20
MaxFileSize=500000000
```

(1) Type 表示日志类型，其取值范围是 1 至 2，缺省值为 1。1 表示循环记录日志；2 表示按日期记录日志。

(2) LockType 表示写日志锁类型，其取值范围是 1 至 2，缺省值为 1。1 表示线程锁；2 表示进程锁。

(3) Level 表示日志的级别，其取值范围是 0 至 10，缺省值为 0。0 级日志信息最少，只报告错误和重要的运行信息，这也是正式运行时设置的级别；10 级日志信息最多，包括所有的错误、警告和信息，一般只在程序调试错误时使用。其它常用的级别还有 1 和 5，其信息量中等。

(4) Display 表示日志输出的方式，其取值范围是 0 至 3，缺省值为 1。0 表示不显示也不记录日志；1 表示只在文件中记录日志；2 表示只在屏幕上显示日志（只对控制台程序时有效）；3 表示在文件中记录同时在屏幕上显示日志。

(5) LogDir 表示日志文件的目录；

(6) LogName 表示日志文件名称的前缀；

(7) MaxFileCount 表示最大文件的数目，默认是 20 个。假如 LogName 是 mrapi.log，MaxFileCount 是 20，则日志文件文件名为 mrapi.log、mrapi_01.log、mrapi_02.log、……、mrapi_19.log。其中 mrapi.log 是最新的日志文件，mrapi19.log 是最旧的日志文件，更早的日志文件将自动被删除；

(8) MaxFileSize 表示其中一个日志文件的大小，默认是 500000000，约 500MB；

在 Windows 环境下 mrapi_log.conf 需要和 mrapi.dll 放置在同一目录下；在 AIX 环境下按照解包后的层次结构放置。如果配置文件不存在，或者里面少配置了某个参数，则相应参数采用缺省值。

该配置文件可以在使用 FDEAPI 的程序运行过程中动态修改，修改后 30 秒内生效。

3 使用概述

3.1 功能

FDEAPI 是一组提供给用户调用的 C 语言的应用程序编程接口，用户可以调用该 API 开发，实现与金融数据交换平台进行数据交换。FDEAPI 可以完成通信的自动连接、发送消息包、接收消息包、加密压缩等功能，应用程序可以用 FDEAPI 与金融数据交换平台的接入客户端实现交互。

3.2 应用环境

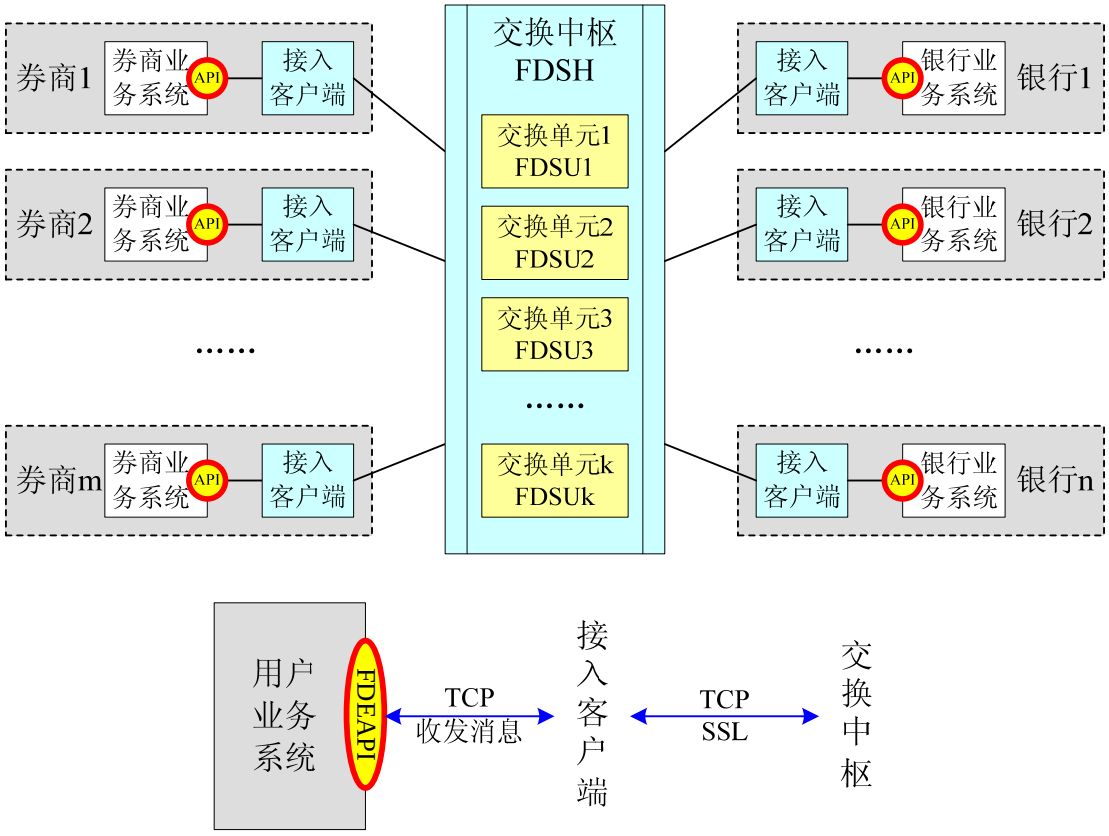

图1　　FDEAPI 应用环境

在金融数据交换平台中，接入客户端负责向交换中枢认证用户的身份，也负责与交换中枢建立安全的 SSL 连接，可靠的传输数据。接入客户端向用户展现的 FDEAPI 不必再提供复杂的身份认证和加密机制。

3.3　地址的标识

金融数据交换平台的基本功能是在用户间传输消息，每个消息都有一个源地址和一个目的地址。

金融数据交换平台的每个用户有一个唯一的用户标识（UserID），用户标识由不超过 63 个字符组成，字符可以是大写字母、小写字母、数字、减号或下划线，首字符必须是字母。事实上，一个用户标识即对应一个接入客户端。

在用户处，一个接入客户端可以支持多个业务应用接入金融数据交换平台。为了区分不同的应用，给每个应用一个唯一的应用标识（AppID），应用标识的命名规则和用户标识相同。

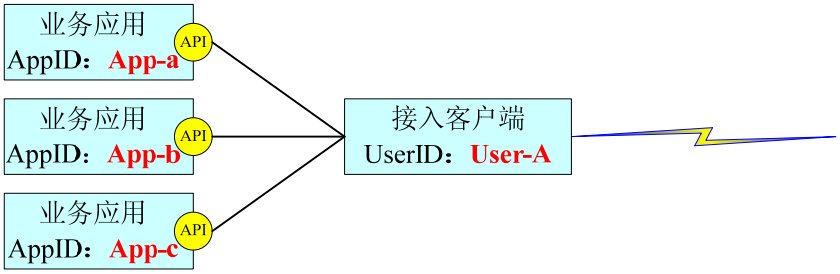

图2　　用户标识与应用标识

消息是由具体的应用发出的，最终也要由具体的应用接收并处理。因此，消息的源地址和目的地址都要用用户标识和应用标识两个要素共同表示。一条消息有四个必需的地址要素：源用户标识、源应用标识、目的用户标识、目的应用标识。

3.4　线程安全性

FDEAPI 中的所有函数都是线程安全的，用户可以根据需要生成多个与接入客户端 FDAP 连接的线程，同时与金融数据交换平台进行数据交换，以提高交换性能。

3.5　应用系统的安全性

用户在其应用软件中调用了 FDEAPI，在 FDEAPI 中不会包含故意攻击用户系统的软件或代码，不会故意影响用户系统的安全运行。用户系统自身的安全性，应该由用户自己进行检查和维护，FDEAPI 无法保证用户系统本身的安全性。

4　编程参考

4.1　常量定义

4.1.1　协议类型常量

名称	定义值	说明
MR_PROTOCOLTYPE_MRSTANDAND	0x01	FDEP 规定的标准业务协议。
MR_PROTOCOLTYPE_SELFCUSTOM	0xFF	用户自定义协议类型。

4.1.2 消息标志位常量

名称	定义值	说明
MR_MSGFLAG_PERSIST	0x01	持久消息标志，用于可靠传输。目前暂不支持。
MR_MSGFLAG_COMPRESS	0x02	压缩标志，需进行压缩传输。

4.1.3 长度常量

名称	定义值	说明
MR_MAXLEN_ADDR	64	用户标识及应用标识的最大长度。
MR_MAXLEN_PKGID	64	消息包标识的最大长度。
MR_MAXLEN_USERDATA	256	用户保留数据的最大长度。
MR_FIXLEN_EXPIREDABSTIME	20	过期绝对时间固定长度。

4.1.4 函数返回错误值

对于 MrSend、MrReceive*的各个函数的返回值，如果返回 0，表示成功，否则表示错误，错误返回值在 mrapi.h 文件中已定义。mrapi.h 文件中函数错误返回值定义如下：

```
#define MR_ERRCODE_OK              0
#define MR_ERRCODE_PARAMERR        -1
#define MR_ERRCODE_CONNERR         -2
#define MR_ERRCODE_TIMEEXPIRED     -3
#define MR_ERRCODE_TIMEOUT         -4
#define MR_ERRCODE_NOMSG           -5
#define MR_ERRCODE_BUFTOOSHORT     -6
#define MR_ERRCODE_BUFTOOBIG       -7
#define MR_ERRCODE_SYSERROR        -8
```

4.2 数据结构说明

4.2.1 消息属性STUMsgProperty

该结构用来表示一条消息的各种属性。

结构定义:

```
struct STUMsgProperty
{
    char    m_szSourceUserID[MR_MAXLEN_ADDR];
    char    m_szSourceAppID[MR_MAXLEN_ADDR];
    char    m_szDestUserID[MR_MAXLEN_ADDR];
    char    m_szDestAppID[MR_MAXLEN_ADDR];
    char    m_szPkgID[MR_MAXLEN_PKGID];
    char    m_szCorrPkgID[MR_MAXLEN_PKGID];
    char    m_szUserData1[MR_MAXLEN_USERDATA];
    char    m_szUserData2[MR_MAXLEN_USERDATA];
    char    m_szExpiredAbsTime[MR_FIXLEN_EXPIREDABSTIME];
    unsigned char    m_ucFlag;
    unsigned char    m_ucProtocolType;
};
```

字段说明:

字段	说明
m_szSourceUserID	源用户标识，以“\0”结尾的字符串。
m_szSourceAppID	源应用标识，以“\0”结尾的字符串。
m_szDestUserID	目的用户标识，以“\0”结尾的字符串。
m_szDestAppID	目的应用标识，以“\0”结尾的字符串。
m_szPkgID	消息包的包标识，以“\0”结尾的字符串，或者由用户调用 MrCreatePkgID 函数生成，或者为空（即’\0’）。
m_szCorrPkgID	相关包标识，以“\0”结尾的字符串，供用户自用。
m_szUserData1	用户数据 1，以“\0”结尾的字符串，供用户自用。
m_szUserData2	用户数据 2，以“\0”结尾的字符串，供用户自用。
m_szExpiredAbsTime	该消息的过期时间，以“\0”结尾的字符串，格式为“YYYY-MM-DD HH:MM:SS”。也可以置空，此时如果目的用户不在线，或者目的应用未连接，则消息立即过期。
m_ucFlag	消息标志，由 8 个二进制位组成，各位含义如下： 位 0 --为 1 表示持久消息，需可靠传输，暂不支持； 位 1 --为 1 表示消息需压缩传输。
m_ucProtocolType	协议类型标志，取值可以是下列之一： MR_PROTOCOLTYPE_MRSTANDAND MR_PROTOCOLTYPE_SELFCUSTOM

特殊说明:

m_szCorrPkgID、m_szUserData1、m_szUserData2 三个字段是供用户自己使用的，金融数据交换平台本身不会使用这几个字段，也不会主动的改变它们。

消息包的压缩及解压由金融数据交换平台自动完成，对用户透明。m_ucFlag

的位 1 置 1 是通知金融数据交换平台对消息进行压缩后传输，用户通过 FDEAPI 接触到的消息包永远是非压缩的。

约定:

如果用户调用 MrSend 函数发送的是业务应答包，对于 pMsgPropery 参数中的 m_szCorrPkgID 字段，统一约定填写为与对应请求包中的 m_szPkgID 字段中相同的值，以方便请求方接收到应答包后，根据该字段查找对应的请求包。对于业务请求包或者其它类型的包，该字段填写为空(即'\0')。

4.2.2 连接信息STUConnInfo

该结构用来定义与接入客户端建立连接所需的各种信息。

结构定义:

```
struct STUConnInfo
{
    char                m_szMRIP[16];
    unsigned short      m_usMRPort;
    char                m_szMRIPBak[16];
    unsigned short      m_usMRPortBak;
};
```

字段说明:

字段	说明
m_szMRIP	接入客户端消息路由器的 IP 地址，以“\0”结尾的字符串，格式为“xxx.xxx.xxx.xxx”。
m_usMRPort	接入客户端消息路由器的连接端口。
m_szMRIPBak	备用消息路由器的 IP 地址，不设置备用消息路由器时可以为空。
m_usMRPortBak	备用消息路由器的连接端口，不设置备用消息路由器时可以为 0。

特殊说明:

该结构在调用 MrInit 函数时作为输入参数使用。MrInit 首先尝试与结构中指定的第一个消息路由器连接，如不成功再尝试连接备用消息路由器。接入客户端至少由两个消息路由器组成，可以有更多个。使用 FDEAPI 时，只要指定连接其中一个消息路由器，会自动在接入客户端所有的消息路由器间进行负载均衡。也

就是说，应用程序最终连接的消息路由器可能不是该结构中指定的任何一个。

4.3 函数参考

4.3.1 函数清单

FDEAPI 是一个简明易用的编程接口，它提供了如下 11 个函数：

序号	函数名称	函数功能
1	MrInit	初始化，获取相关资源，并尝试与接入客户端 FDAP 建立连接。
2	MrInit2	初始化，获取相关资源，并尝试与接入客户端 FDAP 建立连接。该函数与 MrInit 相比可以启动多个回调函数的线程，并且可以根据设定接收的条件，比 MrInit 的功能更强。
2	MrIsLinkOK	查看并判断当前与接入客户端 FDAP 的连接是否正常。
3	MrCreatePkgID	生成消息包标识。
4	MrSend	通过 FDAP 向消息中枢发送消息，请求转发。
5	MrReceive1	以方式 1 条件接收消息中枢转发来的消息。
6	MrReceive1_FreeBuf	释放 MrReceive1 函数调用中分配的内存。
7	MrBrowse	查看本用户的下一个可接收数据包。该函数已废弃。
8	MrReceive2	以方式 2 条件接收消息中枢转发来的消息。
9	MrReceive3	以方式 3 条件接收消息中枢转发来的消息。
10	MrDestroy	断开与 FDAP 的连接，释放相关资源。
11	MrGetVersion	取得该 API 的版本号
12	MrRegRecvCondition	注册包下推条件

4.3.2 MrInit

连接 FDAP 时的初始化函数。该函数对 FDEAPI 进行初始化，分配获取相关资源，并尝试与接入客户端建立通信连接。

FDEAPI 在调用此初始化函数时，会尝试使用 TCP 方式连接接入客户端，一般情况下，在该函数返回之前将连接成功。如果接入客户端临时不可用，或者有网络问题，调用此函数返回时，可能还没有连接上接入客户端，由于 FDEAPI 具有断线重连的功能，该 API 内部将自动重试与对方进行连接。

函数原型：

```
void* _stdcall  MrInit(const char* psAppID, const char* psPasswd,
              int (*OnReceive)(const char* psPkg, int iPkgLen,
                               const STUMsgProperty* pMsgPropery,
                               void* pvUserData),
              const STUConnInfo oConnInfo, void* pvUserData);
```

参数说明:

参数	说明
psAppID [in]	本应用的应用标识。
psPasswd [in]	本应用在接入客户端设置的密码，密码必须与预设的匹配才能继续。
OnReceive [in]	接收到消息包时的回调函数。该回调函数不能与下面的 MrReceive1/MrReceive1_FreeBuf 、 MrReceive2 或 MrReceive3/MrReceive1_FreeBuf 同时使用。
oConnInfo [in]	接入客户端连接信息。
pvUserData [in]	供回调函数使用的用户数据。

返回值说明:

返回值	说明
NULL	初始化失败。
非 NULL	初始化成功，返回一个连接句柄，该句柄将作为其他函数调用的参数。

4.3.3 MrInit2

连接 FDAP 时的初始化函数。该函数对 FDEAPI 进行初始化，分配获取相关资源，并尝试与接入客户端建立通信连接。

FDEAPI 在调用此初始化函数时，会尝试使用 TCP 方式连接接入客户端，一般情况下，在该函数返回之前将连接成功。如果接入客户端临时不可用，或者有网络问题，调用此函数返回时，可能还没有连接上接入客户端，由于 FDEAPI 具有断线重连的功能，该 API 内部将自动重试与对方进行连接。

该函数与 MrInit 函数的区别是：(1)增加了最后一个参数 iThreadCount，表示回调函数 OnReceive 的线程个数(MrInit 函数调用 OnReceive 只有一个线程)；(2)返回的 handle 改为第一个参数，返回双指针；(3)将 STUConnInfo 由对象改成了指针。

函数原型:

```
void   _stdcall MrInit2(void** ppHandle, const char* psAppID,
            const char* psAppPasswd, STUMsgProperty* pOnRecvMsgPropery,
           int (*OnReceive)(const char* psPkg, int iPkgLen,
               const STUMsgProperty* pMsgPropery, void* pvUserData),
             const STUConnInfo* pConnInfo, void* pvUserData,
              int iThreadCount);
```

参数说明:

参数	说明
ppHandle [out]	MrInit2 函数返回的句柄，该句柄将作为其他函数调用的参数。
psAppID [in]	本应用的应用标识。
psPasswd [in]	本应用在接入客户端设置的密码，密码必须与预设的匹配才能继续。
pOnRecvMsgPropery[in]	这是回调函数 OnReceive 的接收条件，如果不需要任何条件，则可以填 NULL
OnReceive [in]	接收到消息包时的回调函数。该回调函数不能与下面的 MrReceive1/MrReceive1_FreeBuf、MrReceive2 或 MrReceive3/MrReceive1_FreeBuf 同时使用。
pConnInfo [in]	接入客户端连接信息。
pvUserData [in]	供回调函数使用的用户数据。
iThreadCount	调用回调函数 OnReceive 的线程数目。

返回值说明:

无

4.3.4 MrIsLinkOK

连接状态判断函数。该函数判断当前与接入客户端的连接是否正常。因为整个金融数据交换平台的消息交换是异步的，而且 FDEAPI 具有自动重连的功能，所以即使当前连接不正常，也并非说明系统不能正常工作，连接自动恢复后，各种任务会继续。

函数原型:

```
int _stdcall  MrIsLinkOK(void* pHandle);
```

参数说明:

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。

返回值说明:

返回值	说明
0	连接不正常。
1	连接正常。

4.3.5 MrCreatePkgID

消息包标识生成函数。该函数生成一个在整个 FDEP 中全局唯一的 UUID，可用作消息包标识。

函数原型:

```
int _stdcall  MrCreatePkgID(void* pHandle,
                    char szPkgID[MR_MAXLEN_PKGID]);
```

参数说明:

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。
szPkgID [out]	生成的消息包标识。

返回值说明:

返回值	说明
0	成功。
其他	失败。

4.3.6 MrSend

消息包发送函数。该函数完成发送消息的功能。

约定:

如果用户调用 MrSend 函数发送的是业务应答包，对于 pMsgPropery 参数中的 m_szCorrPkgID 字段，统一约定填写为与对应请求包中的 m_szPkgID 字段中相同的值，以方便请求方接收到应答包后，根据该字段查找对应的请求包。对于业务请求包或者其它类型的包，该字段填写为空(即'\0')。

函数原型:

```
int _stdcall  MrSend(void* pHandle,
                     const char* psPkg,
                     int iPkgLen,
                     STUMsgProperty* pMsgPropery,
                     int iMillSecTimeo);
```

参数说明:

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。
psPkg [in]	要发送的消息包缓冲区。
iPkgLen [in]	缓冲区中的消息包长度。
pMsgPropery [in/out]	消息包属性。 (1)m_szSourceUserID 的输入被忽略，输出为接入客户端中设置的本用户的用户标识。 (2)m_szSourceAppID 的输入被忽略，输出为 MrInit 中设置的本应用的应用标识。 (3)m_szDestUserID 和 m_szDestAppID 输入为消息包的目的地址——目的用户标识和目的应用标识，输出不变。 (4)m_szPkgID 输入为唯一的消息包标识。该值或者由用户调用 MrCreatePkgID 函数生成的唯一标识符，输出不变；或者输入为空(即‘\0’)，则输出为系统自动分配的唯一消息包标识。对每一个发送的包，该 m_szPkgID 应该在整个金融数据交换平台中唯一，不同的包需要使用不同的 m_szPkgID。虽然 MrCreatePkgID 函数生成的消息标识符是唯一的，但是，由于不同用户的调用方式不同，编程方式也不一样，有一些用户可能不规范调用，有一些用户可能会作特别的用途，添加一些额外的信息，甚至可能出现编程错误等，导致 m_szPkgID 或 m_szCorrPkgID 可能重复，因此，用户不能简单地把 m_szPkgID 或 m_szCorrPkgID 作为全局唯一的标识，FDEP 只负责将该包正确地传送到目的方，而对方收到该消息包后，应该区分不同的情况作出不同的业务处理，比如检查用户流水号，资金帐号、用户名等信息，通过业务信息的检查来确定该包的相关信息。 (5)m_szCorrPkgID、m_szUserData1 和 m_szUserData2 输入为用户自定义的数据，输出不变。 (6)m_szExpiredAbsTime 输入为消息包在整个金融数据交换平

	台中的过期时间，输出不变。绝对时间表示，格式为 YYYY-MM-DD hh:mm:ss (7)m_ucFlag 输入为需要的包处理位标志，输出不变。
iMillSecTimeo [in]	以毫秒为单位的发送最大超时时间。

返回值说明：

返回值	说明
0	成功。
其他	失败。

4.3.7 MrReceive1

消息包接收函数 1。按给定条件从本应用对应的接收队列中收取第一个符合条件的消息，并将消息从队列中移除。该函数接收到的消息包占用的内存由 FDEAPI 内部分配，使用完成后，需要调用 MrReceive1_FreeBuf 函数释放分配的内存。

注意：接收消息包时，或者只能使用函数 MrInit 中的回调函数 OnReceive 进行接收；或者只能使用 MrReceive1/MrReceive1_FreeBuf、MrReceive2 或 MrReceive3/MrReceive1_FreeBuf 进行接收，两者不能同时使用。

函数原型：

```
int _stdcall  MrReceive1(void* pHandle,
                    char** ppsPkg,
                    int* piOutPkgLen,
                    STUMsgProperty* pMsgPropery,
                    int iMillSecTimeo);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。
ppsPkg [out]	双指针，返回包所指向的内存。该内存在该函数内部分配，用户在使用该内存之后，需要调用 MrReceive1_FreeBuf 函数释放该内存。
piOutPkgLen [out]	接收到的消息包的实际长度。
pMsgPropery	输入将作为接收条件，该结构中有 6 个字段被用作判断条件：

[in/out]	m_szSourceUserID 、 m_szSourceAppID 、 m_szPkgID 、 m_szCorrPkgID、m_szUserData1 和 m_szUserData2。其他字段被忽略。如果这 6 个字段中的某个输入值为空，则该字段也被从判断条件中排除，也就是说，实际的判断条件是这 6 个字段中的非空字段。符合条件的标准是消息属性中所有相应字段与这些非空的条件字段值相同。如果 6 个字段全为空，则收取队列中第一个消息包。 注意，对 m_szCorrPkgID 的输入条件有特殊处理：当 m_szCorrPkgID 字符串的值是空时，则可以匹配消息中 m_szCorrPkgID 字段为任意值的消息；当 m_szCorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为空的消息；当 m_szCorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为非空的消息。 输出是接收到的包的属性。
iMillSecTimeo [in]	以毫秒为单位的接收最大超时时间。该时间是该函数内部为了控制从发送接收包请求到接收到应答或没有应答之间的最大时间间隔，而不是该函数实际执行的时间间隔。例如如果此时没有符合条件的包，则即使该值为 1 分钟，该函数内部只要收到该应答，则马上返回，而不需要等到 1 分钟之后再返回。该值与系统的繁忙程序有关，如果系统繁忙，该值可以适当大一些，如果系统空闲，该值可以适当小一些。一般情况下，建议该值为 2000，即 2 秒。

返回值说明:

返回值	说明
0	成功。
其他	失败。

4.3.8 MrReceive1_FreeBuf

消息包内存释放函数。该函数释放调用 MrReceive1 函数时 FDEAPI 内部自动分配的消息包内存。

函数原型:

```
void _stdcall  MrReceive1_FreeBuf(char* psPkg);
```

参数说明:

参数	说明
psPkg [in]	由 MrReceive1 函数的第二个参数返回的指针。

返回值说明:

返回值	说明
无	

4.3.9 MrBrowse

消息包浏览函数。该函数已废弃。

4.3.10 MrReceive2

消息包接收函数 2。该函数与 MrReceive1 函数功能类似，但用户需要自己为消息接收缓冲区分配和管理内存。用户需预先分配一块足够大的内存，再调用该函数接收消息。

注意：接收消息包时，或者只能使用函数 MrInit 中的回调函数 OnReceive 进行接收；或者只能使用 MrReceive1/MrReceive1_FreeBuf、MrReceive2 或 MrReceive3/MrReceive1_FreeBuf 进行接收，两者不能同时使用。

函数原型:

```
int _stdcall  MrReceive2(void* pHandle,
                    char* psPkg,
                    int* piOutPkgLen,
                    int iBufLenIn,
                    STUMsgProperty* pMsgPropery,
                    int iMillSecTimeo);
```

参数说明:

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。
psPkg [out]	消息包接收缓冲区。
piOutPkgLen [out]	接收到的消息包的实际长度。
iBufLenIn [in]	消息包缓冲区大小。
pMsgPropery	输入将作为接收条件，该结构中有 6 个字段被用作判断条件：

[in/out]	m_szSourceUserID 、 m_szSourceAppID 、 m_szPkgID 、 m_szCorrPkgID、m_szUserData1 和 m_szUserData2。其他字段被忽略。如果这 6 个字段中的某个输入值为空，则该字段也被从判断条件中排除，也就是说，实际的判断条件是这 6 个字段中的非空字段。符合条件的标准是消息属性中所有相应字段与这些非空的条件字段值相同。如果 6 个字段全为空，则接收队列中第一个消息包。 注意，对 m_szCorrPkgID 的输入条件有特殊处理：当 m_szCorrPkgID 字符串的值是空时，则可以匹配消息中 m_szCorrPkgID 字段为任意值的消息；当 m_szCorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为空的消息；当 m_szCorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为非空的消息。 输出是接收到的消息的属性。
iMillSecTimeo [in]	以毫秒为单位的接收最大超时时间。该时间是该函数内部为了控制从发送接收包请求到接收到应答或没有应答之间的最大时间间隔，而不是该函数实际执行的时间间隔。例如如果此时没有符合条件的包，则即使该值为 1 分钟，该函数内部只要收到该应答，则马上返回，而不需要等到 1 分钟之后再返回。该值与系统的繁忙程序有关，如果系统繁忙，该值可以适当大一些，如果系统空闲，该值可以适当小一些。一般情况下，建议该值为 2000，即 2 秒。

返回值说明:

返回值	说明
0	成功。
其他	失败。

4.3.11 MrReceive3

消息包接收函数 3。该函数与 MrReceive1 函数功能类似，但此函数可以接收到系统错误消息，例如当发生目标地址不存在、队列已满、超时过期、系统错误等情况时，该函数的 piErrSXCode 参数的值将为非 0，其值表示错误的原因代码，在 ppsPkg 参数中表示错误的原因字符串；当 piErrSXCode 参数的值为 0 时，表示收到的是正常的消息包。该函数接收到的消息包占用的内存由 FDEAPI 内部分配，使用完成后，需要调用 MrReceive1_FreeBuf 函数释放分配的内存。

注意：接收消息包时，或者只能使用函数 MrInit 中的回调函数 OnReceive 进行接收；或者只能使用 MrReceive1/MrReceive1_FreeBuf、MrReceive2 或

MrReceive3/MrReceive1_FreeBuf 进行接收，两者不能同时使用。

函数原型：

```
int _stdcall MrReceive3(void* pHandle,
                        char** ppsPkg,
                        int* piOutPkgLen,
                        int* piErrSXCode,
                        STUMsgProperty* pMsgPropery,
                        int iMillSecTimeo);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。
ppsPkg [out]	双指针，返回包所指向的内存。该内存在该函数内部分配，用户在使用该内存之后，需要调用 MrReceive1_FreeBuf 函数释放该内存。
piOutPkgLen [out]	接收到的消息包的实际长度。
piErrSXCode[out]	交换错误的原因码，其值为 0 时，表示接收到的是正常的交换消息包；当值将为 1 至 5，分别表示目标不存在、目标错误、队列已满、超时过期、系统错误，此时*ppsPkg 中是错误字符串，pMsgPropery 中是发送原始包时的参数.
pMsgPropery [in/out]	输入将作为接收条件，该结构中有 6 个字段被用作判断条件：m_szSourceUserID、m_szSourceAppID、m_szPkgID、m_szCorrPkgID、m_szUserData1 和 m_szUserData2。其他字段被忽略。如果这 6 个字段中的某个输入值为空，则该字段也被从判断条件中排除，也就是说，实际的判断条件是这 6 个字段中的非空字段。符合条件的标准是消息属性中所有相应字段与这些非空的条件字段值相同。如果 6 个字段全为空，则收取队列中第一个消息包。 注意，对 m_szCorrPkgID 的输入条件有特殊处理：当 m_szCorrPkgID 字符串的值是空时，则可以匹配消息中 m_szCorrPkgID 字段为任意值的消息；当 m_szCorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为空的消息；当 m_szCorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为非空的消息。 输出是接收到的包的属性。
iMillSecTimeo [in]	以毫秒为单位的接收最大超时时间。该时间是该函数内部为了控制从发送接收包请求到接收到应答或没有应答之间的最大时间间隔，而不是该函数实际执行的时间间隔。例如如果此时没有符合条件的包，则即使该值为 1 分钟，该函数内

	部只要收到该应答，则马上返回，而不需要等到 1 分钟之后再返回。该值与系统的繁忙程序有关，如果系统繁忙，该值可以适当大一些，如果系统空闲，该值可以适当小一些。一般情况下，建议该值为 2000，即 2 秒。

返回值说明：

返回值	说明
0	成功。
其他	失败。

4.3.12 MrReceive4

消息包接收函数。该函数与 MrReceive3 函数功能类似，其与 MrReceive3 唯一的区别在于其可以根据相关包 ID 接收系统错误包。

函数原型：

```
int _stdcall MrReceive4(void* pHandle,
                        char** ppsPkg,
                        int* piOutPkgLen,
                        int* piErrSXCode,
                        STUMsgProperty* pMsgPropery,
                        int iMillSecTimeo);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。
ppsPkg [out]	双指针，返回包所指向的内存。该内存在该函数内部分配，用户在使用该内存之后，需要调用 MrReceive1_FreeBuf 函数释放该内存。
piOutPkgLen [out]	接收到的消息包的实际长度。
piErrSXCode[out]	交换错误的原因码，其值为 0 时，表示接收到的是正常的交换消息包；当值将为 1 至 5，分别表示目标不存在、目标错误、队列已满、超时过期、系统错误，此时*ppsPkg 中是错误字符串，pMsgPropery 中是发送原始包时的参数.
pMsgPropery [in/out]	输入将作为接收条件，该结构中有 6 个字段被用作判断条件：m_szSourceUserID、m_szSourceAppID、m_szPkgID、m_szCorrPkgID、m_szUserData1 和 m_szUserData2。其他字

	段被忽略。如果这 6 个字段中的某个输入值为空，则该字段也被从判断条件中排除，也就是说，实际的判断条件是这 6 个字段中的非空字段。符合条件的标准是消息属性中所有相应字段与这些非空的条件字段值相同。如果 6 个字段全为空，则收取队列中第一个消息包。 注意，对 m_szCorrPkgID 的输入条件有特殊处理：当 m_szCorrPkgID 字符串的值是空时，则可以匹配消息中 m_szCorrPkgID 字段为任意值的消息；当 m_szCorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为空的消息；当 m_szCorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为非空的消息。 输出是接收到的包的属性。
iMillSecTimeo [in]	以毫秒为单位的接收最大超时时间。该时间是该函数内部为了控制从发送接收包请求到接收到应答或没有应答之间的最大时间间隔，而不是该函数实际执行的时间间隔。例如如果此时没有符合条件的包，则即使该值为 1 分钟，该函数内部只要收到该应答，则马上返回，而不需要等到 1 分钟之后再返回。该值与系统的繁忙程序有关，如果系统繁忙，该值可以适当大一些，如果系统空闲，该值可以适当小一些。一般情况下，建议该值为 2000，即 2 秒。

返回值说明:

返回值	说明
0	成功。
其他	失败。

4.3.13 MrDestroy

资源释放函数。在 FDEAPI 使用完毕后，释放所有资源。该函数通常是最后一个被调用的 FDEAPI 函数。

函数原型:

```
void _stdcall  MrDestroy(void* pHandle);
```

参数说明:

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。

返回值说明:

返回值	说明
无	

4.3.14 MrGetVersion

取得该 API 的版本号。该函数可以在装载该动态库后的任何时间调用。

函数原型:

```
void _stdcall MrGetVersion(char* psBufVersion, int iBufLen);
```

参数说明:

参数	说明
psBufVersion [out]	返回的该 API 版本号的字符串。该版本号的格式为“01.04.20060901”，其中 01 表示大版本号， 04 表示小版本号，20060901 表示该版本发布的日期。
iBufLen[in]	表示 psBufVersion 缓存区的长度，该长度应该大于或等于 15。

返回值说明:

返回值	说明
无	

4.3.15 MrRegRecvCondition

注册下推条件函数。所谓注册下推条件是指，按照正常的函数调用规则，调用接收函数是从 FDEAPI 发送一个接收条件到 FDMR，再由 FDMR 按照条件送回一个包给 FDEAPI。如果调用了注册下推函数，则 FDMR 会主动将满足条件的包依次下推到FDEAPI,这样的会带来FDMR 到FDEAPI 单链接传输速率的提高，代价是牺牲一定灵活性。此函数只在对单链接流量有较高要求时调用。

函数原型:

```
int _stdcall MrRegRecvCondition(void* pHandle, STUMsgProperty* pMsgPropery, int iType);
```

参数说明:

参数	说明
pHandle [in]	连接句柄，调用 MrInit 时返回的值。
pMsgPropery [in/out]	输入将作为 M 下推包的选择条件，该结构中有 6 个字段被用作判断条件：m_szSourceUserID、m_szSourceAppID、m_szPkgID、m_szCorrPkgID、m_szUserData1 和 m_szUserData2。其他字段被忽略。如果这 6 个字段中的某个输入值为空，则该字段也被从判断条件中排除，也就是说，实际的判断条件是这 6 个字段中的非空字段。符合条件的标准是消息属性中所有相应字段与这些非空的条件字段值相同。如果 6 个字段全为空，则收取队列中第一个消息包。 注意，对 m_szCorrPkgID 的输入条件有特殊处理：当 m_szCorrPkgID 字符串的值是空时，则可以匹配消息中 m_szCorrPkgID 字段为任意值的消息；当 m_szCorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为空的消息；当 m_szCorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 m_szCorrPkgID 字段为非空的消息。 输出是接收到的包的属性。
iType [in]	0 增加一个条件， 1 删除一个条件， 2 清空所有条件。

返回值说明:

返回值	说明
0	成功。
其他	失败。

4.3.16 调用顺序

下图不是流程图，而是示意 FDEAPI 中主要函数的调用顺序。使用 FDEAPI，必须先调用函数 MrInit，最后调用 MrDestroy，其间是消息发送和接收的各种调用。

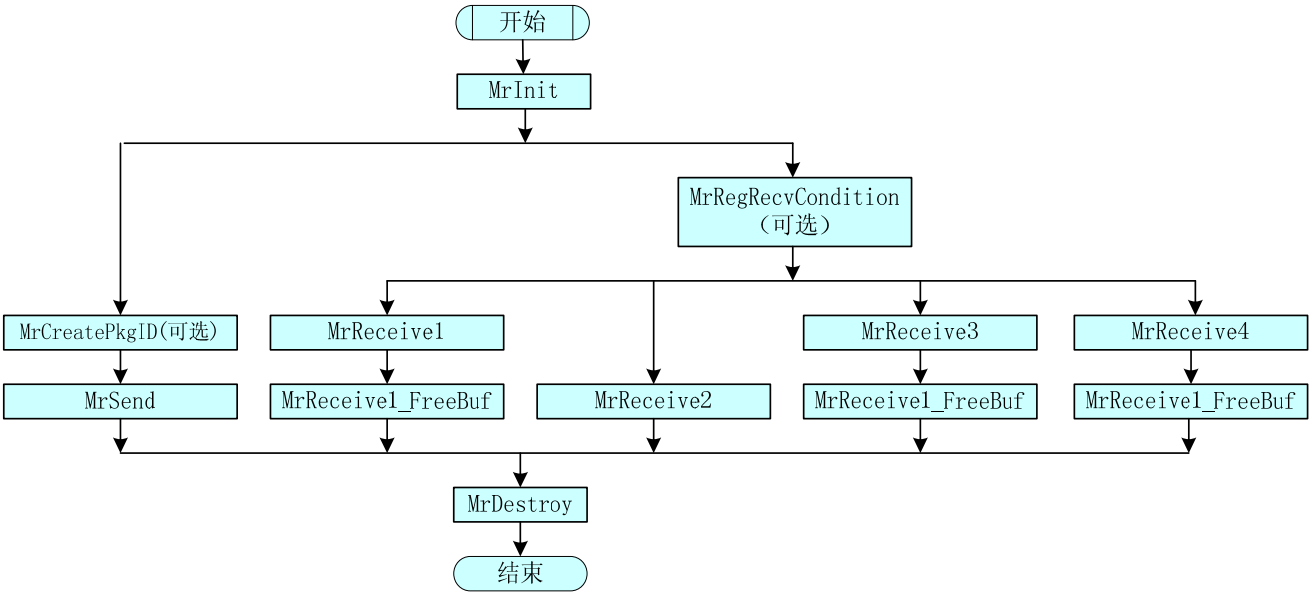

图3　　FDEAPI 主要函数调用顺序

注意：接收消息包时，或者只能使用函数 MrInit 中的回调函数 OnReceive 进行接收；或者只能使用 MrReceive1/MrReceive1_FreeBuf、MrReceive2 或 MrReceive3(或 MrReceive4)/MrReceive1_FreeBuf 进行接收，两者不能同时使用。

5　编程示例

```
#include <stdio.h>
#include <string>
#include <windows.h>
#include <winbase.h>
#include <process.h>

#include <time.h>
#include "mrapi.h"

using namespace std;


void MySleep(int iMillSecond)
{
    Sleep(iMillSecond);
```

```
}


void*   g_pHandle = NULL;
time_t  g_ltLastTimeRecv = 0;
time_t  g_ltBeginTimeRecv = 0;
int     g_iRecvCount = 0;


int OnReceive(const char* psPkg, int iPkgLen, const STUMsgProperty* pMsgPropery,
void* pvUserData)
{
    g_ltLastTimeRecv = time(NULL);
    ++g_iRecvCount;
    if(g_iRecvCount%100==0)
    {
        printf("recv ok: PkgContent[%s]\n, pkgID[%s], src[%s.%s], dest[%s.%s],
pkgLen[%d],   CorrPkgID[%s],   UserData1[%s],   UserData2[%s]   \n",   psPkg,
pMsgPropery->m_szPkgID,                          pMsgPropery->m_szSourceUserID,
pMsgPropery->m_szSourceAppID,                      pMsgPropery->m_szDestUserID,
pMsgPropery->m_szDestAppID,        iPkgLen,        pMsgPropery->m_szCorrPkgID,
pMsgPropery->m_szUserData1, pMsgPropery->m_szUserData2);
        printf("client recv count[%d]\n", g_iRecvCount);
    }

#ifndef MR_CLIENT
    STUMsgProperty oMsgProperySend;
    memset(&oMsgProperySend, '\0', sizeof(STUMsgProperty));
    strcpy(oMsgProperySend.m_szDestUserID, pMsgPropery->m_szSourceUserID);
    strcpy(oMsgProperySend.m_szDestAppID, pMsgPropery->m_szSourceAppID);
    strcpy(oMsgProperySend.m_szCorrPkgID, pMsgPropery->m_szPkgID);
    strcpy(oMsgProperySend.m_szUserData1, pMsgPropery->m_szUserData1);
    strcpy(oMsgProperySend.m_szUserData2, pMsgPropery->m_szUserData2);
    oMsgProperySend.m_ucFlag = 0;
    oMsgProperySend.m_ucProtocolType = MR_PROTOCOLTYPE_MRSTANDAND;
    int iRet = MrSend(g_pHandle, psPkg, iPkgLen, &oMsgProperySend, 2000);
    //if(iRet!=0) MySleep(1000);
    //printf("server send echo[iRet=%d]\n", iRet);
#endif

    return 0;
}
```

```
unsigned __stdcall RecvThrd(void *pvParam)
{
    int iLastGetOk = 0;
    for(;;)
    {
        STUMsgProperty oMsgPropery;
        memset(&oMsgPropery, '\0', sizeof(STUMsgProperty));

        int iRecvPkgLen = 0;
        char* psPkg = NULL;
        if(MrReceive1(g_pHandle, &psPkg, &iRecvPkgLen, &oMsgPropery, 1000)==0)
        {
            g_ltLastTimeRecv = time(NULL);
            ++g_iRecvCount;
            if(g_iRecvCount%100==0)
            {
                printf("recv ok: PkgContent[%s]\n, pkgID[%s], src[%s.%s],
dest[%s.%s], pkgLen[%d], CorrPkgID[%s], UserData1[%s], UserData2[%s] \n", psPkg,
oMsgPropery.m_szPkgID,                                oMsgPropery.m_szSourceUserID,
oMsgPropery.m_szSourceAppID,                          oMsgPropery.m_szDestUserID,
oMsgPropery.m_szDestAppID,          iPkgLen,          oMsgPropery.m_szCorrPkgID,
oMsgPropery.m_szUserData1, oMsgPropery.m_szUserData2);
                printf("client recv count[%d]\n", g_iRecvCount);
            }


#ifndef MR_CLIENT
            STUMsgProperty oMsgProperySend;
            memset(&oMsgProperySend, '\0', sizeof(STUMsgProperty));
            strcpy(oMsgProperySend.m_szDestUserID,
oMsgPropery.m_szSourceUserID);
            strcpy(oMsgProperySend.m_szDestAppID, oMsgPropery.m_szSourceAppID);
            strcpy(oMsgProperySend.m_szCorrPkgID, oMsgPropery.m_szPkgID);
            strcpy(oMsgProperySend.m_szUserData1, oMsgPropery.m_szUserData1);
            strcpy(oMsgProperySend.m_szUserData2, oMsgPropery.m_szUserData2);
            oMsgProperySend.m_ucFlag = 0;
            oMsgProperySend.m_ucProtocolType = MR_PROTOCOLTYPE_MRSTANDAND;
            int iRet = MrSend(g_pHandle, psPkg, iRecvPkgLen, &oMsgProperySend,
2000);
            //if(iRet!=0) MySleep(1000);
            //printf("server send echo[iRet=%d]\n", iRet);
#endif
            MrReceive1_FreeBuf(psPkg);
        }
```

```
        else
        {
            MySleep(500);
        }

    }
    return 0;
}


int main(int argc, char* argv[])
{
    string ssName1 = "BANK_A";
    string ssName2 = "APP_a";
    string ssName3 = "SECURITIES_B";
    string ssName4 = "APP_b";
#ifdef MR_CLIENT
    string  ssSourceUserID = ssName1;
    string  ssSourceAppID = ssName2;
    string  ssDestUserID = ssName3;
    string  ssDestAppID = ssName4;
#else
    string  ssSourceUserID = ssName3;
    string  ssSourceAppID = ssName4;
    string  ssDestUserID = ssName1;
    string  ssDestAppID = ssName2;
#endif

    g_ltLastTimeRecv = time(NULL);

    string ssMyPasswd = "111111";
    string *pUserData = new string("hello");

    STUConnInfo  oConnInfo;
    memset(&oConnInfo, 0, sizeof(STUConnInfo));
    if(argc<2) strcpy(oConnInfo.m_szMRIP, "127.0.0.1");
    else strcpy(oConnInfo.m_szMRIP, argv[1]);
    oConnInfo.m_usMRPort = 51231;
    strcpy(oConnInfo.m_szMRIPBak, "");
    oConnInfo.m_usMRPortBak = 51233;

    g_pHandle   =   MrInit(ssSourceAppID.c_str(),   ssMyPasswd.c_str(),   NULL,
oConnInfo, pUserData);
     if(g_pHandle==NULL) return -1;
```

```
/*
    STUMsgProperty oOnRecvMsgPropery;
    memset(&oOnRecvMsgPropery, 0, sizeof(STUMsgProperty));
    strcpy(oOnRecvMsgPropery.m_szCorrPkgID, "<EMPTY>");
    MrInit2Func(&g_pHandle, ssMyAppID.c_str(), ssSrcAppPwd.c_str(),
&oOnRecvMsgPropery, OnReceive, &oConnInfo, NULL, 1);
    if(g_pHandle==NULL) return -1;

*/
    if(_beginthreadex(NULL, 0,  RecvThrd, NULL, 0, NULL)==0)
    {
        printf("create thread failed\n");
        return -1;
    }

    printf("\n");
    MySleep(10000);

    g_ltLastTimeRecv = time(NULL);


#ifdef MR_CLIENT
    STUMsgProperty oMsgPropery;
    int i=0;
    for(;;)
    {
        memset(&oMsgPropery, '\0', sizeof(STUMsgProperty));
        strcpy(oMsgPropery.m_szDestUserID, ssDestUserID.c_str());
        strcpy(oMsgPropery.m_szDestAppID, ssDestAppID.c_str());
        oMsgPropery.m_ucFlag = 0;

        ++i;
        char * pSendStrNew = new char[65536];
        strcpy(pSendStrNew, " <IFTS Len=" 1205" DataVer ="1.0.0.1" SeqNo=""
Type="B" Dup="N" CheckSum="">\n");
        strcat(pSendStrNew, "  <MsgText> \n");
        strcat(pSendStrNew, "     <Acmt.001.01> \n");
        strcat(pSendStrNew, "       <MsgHdr> \n");
        strcat(pSendStrNew, "          <Ver>1.0</Ver> \n");
        strcat(pSendStrNew, "          <SysType>0</SysType> \n");
        strcat(pSendStrNew, "          <InstrCd>11002</InstrCd> \n");
        strcat(pSendStrNew, "          <Sender> \n");
        strcat(pSendStrNew, "            <InstType>S</InstType> \n");
        strcat(pSendStrNew, "             <BrchId>1</BrchId> \n");
```

```
        strcat(pSendStrNew, "          </Sender> \n");
        strcat(pSendStrNew, "          <Recver> \n");
        strcat(pSendStrNew, "             <InstType>B</InstType> \n");
        strcat(pSendStrNew, "             <BrchId>2</BrchId> \n");
        strcat(pSendStrNew, "          </Recver> \n");
        strcat(pSendStrNew, "          <Date>20061216</Date> \n");
        strcat(pSendStrNew, "          <Time>114340</Time> \n");
        strcat(pSendStrNew, "          <Ref> \n");
        strcat(pSendStrNew, "           <IssrType>S</IssrType> \n");
        strcat(pSendStrNew, "           <Ref>9</Ref> \n");
        strcat(pSendStrNew, "          </Ref> \n");
        strcat(pSendStrNew, "        </MsgHdr> \n");
        strcat(pSendStrNew, "        <Cust> \n");
        strcat(pSendStrNew, "          <Name>郭达</Name> \n");
        strcat(pSendStrNew, "          <CertType>10</CertType> \n");
        strcat(pSendStrNew, "          <CertId>42218901</CertId> \n");
        strcat(pSendStrNew, "          <Type>INVE</Type> \n");
        strcat(pSendStrNew, "          <Sex>M</Sex> \n");
        strcat(pSendStrNew, "          <Ntnl>CHN</Ntnl> \n");
        strcat(pSendStrNew, "          <Addr>八卦四路 22 号</Addr>\n");
        strcat(pSendStrNew, "          <PstCd>518029</PstCd> \n");
        strcat(pSendStrNew, "          <Email></Email> \n");
        strcat(pSendStrNew, "          <Fax></Fax> \n");
        strcat(pSendStrNew, "          <Mobile>13843453</Mobile> \n");
        strcat(pSendStrNew, "          <Tel></Tel> \n");
        strcat(pSendStrNew, "         </Cust> \n");
        strcat(pSendStrNew, "         <Agt> </Agt>\n");
        strcat(pSendStrNew, "         < BkAcct> </ BkAcct>\n");
        strcat(pSendStrNew, "         < ScAcct> </ ScAcct>\n");
        strcat(pSendStrNew, "         < ScAcct> </ ScAcct>\n");
        strcat(pSendStrNew, "      </Acmt.001.01>\n");
        strcat(pSendStrNew, "   </MsgText>\n");
        strcat(pSendStrNew, "</IFTS>\n");
        string ssPkg = pSendStrNew;
        delete []pSendStrNew;
        sprintf(oMsgPropery.m_szCorrPkgID, "%d", i);
        sprintf(oMsgPropery.m_szUserData1, "%d", i+1);
        sprintf(oMsgPropery.m_szUserData2, "%d", i+2);
        int  iRet  =  MrSend(g_pHandle,  ssPkg.c_str(),  (int)ssPkg.length(),
&oMsgPropery, 2000);
        if(i%100==0)
            printf("client send[%d-%d]\n", i, iRet);

        MySleep(10);
```

```
    }
    printf("send finished\n");

#else

    for(;;MySleep(1000))
    {
    }
#endif

    MrDestroy(g_pHandle);
    printf("End\n");

    return 0;
}
```

深圳证券通信有限公司

2012 年 8 月