深证通信工程技术文档

SSCC-FDEP 消息传输系统

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.

2020 年 4 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
20190408	1.0	研发中心	新建文档
20200117	1.1	研发中心	增加 java 接口
20200403	1.2	研发中心	增加备份解决方案和常见问题

目 录

图 索 引

1 引言

本文是金融数据交换平台消息传输系统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、Window2016 操作系统。Windows 平台的 FDEAPI 由以下几个文件组成：

文件名	说明
mr3api.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。
ini/mrapi.ini	配置文件，此配置文件在 ini 目录下。

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。

发布 FDEAPI 应用时，请把 mrapi.dll 和 ini/mrapi.ini 一起发布。

2.2 其他平台

FDEAPI 目前还支持 64 位 Linux 的 RHEL 6.8 和 RHEL 7.5 的操作系统。

FDEAPI 可根据用户环境要求进行定制，以支持如 AIX 、HP-UNIX 等其他平台上的开发和使用。

2.3 关于配置的说明

配置文件 ini/mrapi.ini 包含 FDEAPI 内部的一些控制参数，以及控制日志的产生和输出。文件格式如下：

```
[CONFIG]
"SendBandwith"="128"                    //发送队列带宽控制参数，单位为 KB/s，此参数也可以理解成带宽，默认值为 128
"RecvBandwith"="128"                    //接收队列带宽控制参数，单位为 KB/s，此参数也可以理解成带宽，默认值为 128
"SendThreadNum"="5"                     //发送传输组件工作线程数目，默认值为 5
"RecvThreadNum"="5"                     //接收传输组件工作线程数目，默认值为 5
"CompressFileSize"="20000000000"        //以字节为单位，大于此值时则压缩文件后再传输，小于等于此值时，不压缩直接传输，此值允许设置的最小值为 5120，最大值不限制
"DeleteAfterDecompress"="0"             //接收文件解压缩后，是否删除对应的.bz2 文件，设置为 0-不删除，其他-删除
"ProcessingThread"="5"                  //预处理线程数，校验和计算、压缩解压缩，默认值为 5
"SendTryTimeInterval"="60"              //发送文件重试事件间隔，默认值为 60，单位为秒
"SendTryTimes"="5"                      //发送文件重试次数，默认值为 5
"CompleteTaskTimeout"="5"               //失败、终止、完成的任务，从队列中删除的时间，默认值为 5，单位为秒
"NotCompressExts"=".bz2.7z.rar.gz.iso.zip"    //不使用压缩的文件后缀名，即便文件大小超过 CompressFileSize 设置的大小，默认值为[.bz2.7z.rar.gz.iso.zip]
"SummaryFileMaxSaveDays"="5"            //任务摘要文件保存天数，建议与[LOG]中日志最长保存天数的设置 MaxSaveDays 保持一致，默认为-1，不启动
```

```
"EncryptType"="0"                                    //0-RC6;2-AES256
"DiffTopicPkgSerial"="10"                            //主题消息包序号的容错值
"EkeyExtId"="1.2.86.100.4.3.2"//ekey 扩展字段

 [IpPortMap]//地址映射
"172.100.1.98:3602"="10.10.218.193:3602"
"172.100.1.99:3602"="10.10.218.64:3602"

[AppRecvPath]//每个 app 对应的接收路径
"app1"="C:\gtest\app1"
"app2"="C:\gtest\app2"
"app3"="C:\gtest\app3"

[LOG]
"Type"="1"  //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
"LockType"="1"  //日志锁的类型，缺省值为 1，1-线程锁，2-进程锁
"Level"="3"      // DEBUG=1，INFO=0，WARN=-1，ERROR=-2，DEBUG 打印日志最多，ERROR 打印最少
"Display"="3" //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
"LogDir"="../log" //日志文件的目录
"LogName"="mrapi.log" //日志文件名称的前缀
"MaxFileCount"="20"    //最大文件的数目，默认是 20 个
"MaxFileSize"="500000000"  //其中一个日志文件的大小，默认是 500000000，约 500MB
"MaxSaveDays"="5"  //日志最长保存天数
```

【说明】其中 CONFIG 段中有隐藏参数"PersistEnable"="0”，PersistEnable 参数控制 FDEAPI 发送数据包时 Flag 标志。当没有该配置项或者 PersistEnable=0 时，Flag 禁用 MR3_MSGFLAG_PERSIST 标志，即发送数据包时，MR3_MSGFLAG_PERSIST 标志会被 FDEAPI 去掉；当 PersistEnable=1 时，Flag 不会禁用 MR3_MSGFLAG_PERSIST。缺省值为"PersistEnable"="0”。配置文件 mrapi.ini 默认隐藏该配置项。

在 Windows 环境下 ini/mrapi.ini 需要和 mrapi.dll 放置在同一目录下；在非 Windows 平台环境下按照解包后的层次结构放置，若配置文件 mrapi.ini 不在 mrapi 动态链接库的同一级目录下，则需要设置环境变量“MRAPI_LOGCONF_PATH”为配置文件 mrapi.ini 的存放路径。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。

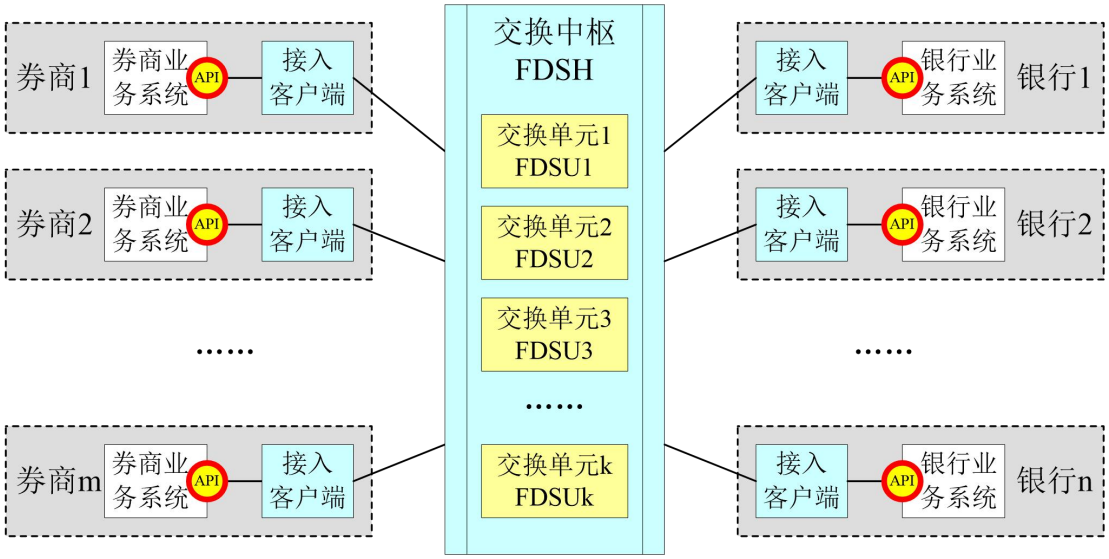
FDEAPI 在使用过程中，可以产生运行日志并记入日志文件，在配置文件的[LOG]关于日志的参数选项可以在使用 FDEAPI 的程序运行过程中动态修改，修改后 30 秒内生效。[CONFIG]的参数选项只在 FDEAPI 初始化时才有效。

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消息传输系统进行数据交换。FDEAPI 可以完成通信的自动连接、发送/接收消息包、发送/订阅主题、发送/接收文件、加密压缩等功能，应用程序可以用 FDEAPI 与金融数据交换平台消息传输系统的接入客户端实现交互。

3.2 应用环境



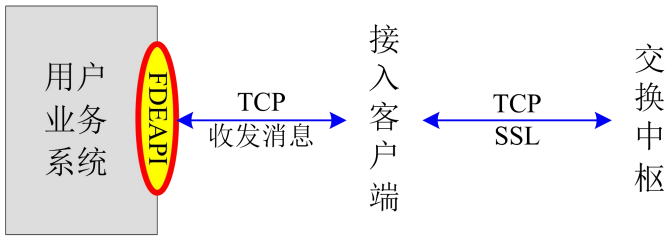


图 1　　FDEAPI 应用环境

在金融数据交换平台消息传输系统中，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

金融数据交换平台消息传输系统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。

金融数据交换平台消息传输系统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消息传输系统。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。

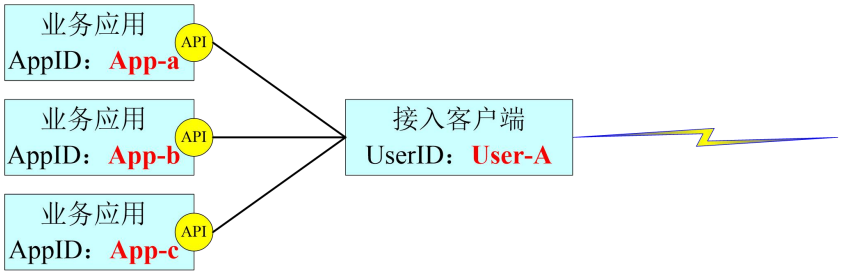


图 2　　用户标识与应用标识

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，同时与消息传输系统进行数据交换，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 消息标志位常量

名称	定义值	说明
MR3_MSGFLAG_PERSIST	0x01	持久消息标志，用于可靠传输。目前暂不支持。
MR3_MSGFLAG_COMPRESS	0x02	压缩标志，需进行压缩传输。
MR3_MSGFLAG_REPLYTOME	0x04	应答包自动推送到发送方标志。

4.1.2 长度常量

名称	定义值	说明
MR3_MAXLEN_ADDR	32	用户标识及应用标识的最大长度。
MR3_MAXLEN_PKGID	64	消息包标识的最大长度。
MR3_MAXLEN_USERDATA	256	用户保留数据的最大长度。
MR3_MAXLEN_MSGTYPE	8	消息类型标识的最大长度。
MR3_MAXLEN_IP	16	IP 地址长度。
MR3_MAXLEN_TOPICNAME	17	主题名的最大长度，包含结束符。

4.1.3 函数返回错误值

对于 Mr3Send、Mr3Receive*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 mr3api.h 文件中已定义。mr3api.h 文件中函数错误返回值定义如下：

```
#define MR3_ERRCODE_OK                  0
#define MR3_ERRCODE_PARAMERR           -1
#define MR3_ERRCODE_CONNERR             -2
#define MR3_ERRCODE_TIMEEXPIRED         -3
#define MR3_ERRCODE_TIMEOUT            -4
#define MR3_ERRCODE_NOMSG               -5
#define MR3_ERRCODE_BUFTOOSHORT         -6
#define MR3_ERRCODE_BUFTOOBIG             -7
#define MR3_ERRCODE_SYSERROR              -8
#define MR3_ERRCODE_COMMU_NOTALLOW      -9
#define MR3_ERRCODE_DEST_NOTONLINE      -10
#define MR3_ERRCODE_DEST_FULL            -11
#define MR3_ERRCODE_PUBTOPIC_NOTALLOW   -12
```

4.2 数据结构说明

4.2.1 消息属性 STUMsgProperty3 和 STUMultiDestMsgProperty3

STUMsgProperty3 结构用来表示一条单发消息的各种属性，STUMultiDestMsgProperty3 结构用来表示一条群发消息的各种属性。

结构定义：

```
struct STUMsgProperty3
{
    char                m_szSourceUserID[MR3_MAXLEN_ADDR];
    char                m_szSourceAppID[MR3_MAXLEN_ADDR];
    unsigned short      m_usSourceInstID;
    char                m_szDestUserID[MR3_MAXLEN_ADDR];
    char                m_szDestAppID[MR3_MAXLEN_ADDR];
    unsigned short      m_usDestInstID;
    char                m_szPkgID[MR3_MAXLEN_PKGID];
    char                m_szCorrPkgID[MR3_MAXLEN_PKGID];
```

```
    char                m_szUserData1[MR3_MAXLEN_USERDATA];
    char                m_szUserData2[MR3_MAXLEN_USERDATA];
    unsigned char       m_ucFlag;
    unsigned char       m_ucBizType;
    unsigned char       m_ucPriority;
    unsigned char       m_ucSensitiveLevel;
    char                m_szMsgType[MR3_MAXLEN_MSGTYPE];
    char                m_szTopicName[MR3_MAXLEN_TOPICNAME];
} ;
```

```
struct STUUserAddr3
{
    char m_szUserID[MR3_MAXLEN_ADDR];
    char m_szAppID[MR3_MAXLEN_ADDR];
} ;

struct STUMultiDestMsgProperty3
{
    char                m_szSourceUserID[MR3_MAXLEN_ADDR];
    char                m_szSourceAppID[MR3_MAXLEN_ADDR];
    int                 m_iDestUserCount;
    STUUserAddr3*       m_pArrDestUserAddr;
    char                m_szPkgID[MR3_MAXLEN_PKGID];
    char                m_szCorrPkgID[MR3_MAXLEN_PKGID];
    char                m_szUserData1[MR3_MAXLEN_USERDATA];
    char                m_szUserData2[MR3_MAXLEN_USERDATA];

    unsigned char       m_ucFlag;
    unsigned char       m_ucBizType;
    unsigned char       m_ucPriority;
    unsigned char       m_ucSensitiveLevel;
    char                m_szMsgType[MR3_MAXLEN_MSGTYPE];
} ;
```

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_usSourceInstID	源实例标识，大于 0，小于 65535。
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_usDestInstID	目的实例标识，大于 0，小于 65535。
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_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； 位 2 --为 1 表示应答包会自动推送到发送方。
m_ucBizType	业务类型标志：0～255。
m_ucPriority	优先级别标志：3～5，5 为最低，3 为最高。
m_ucSensitiveLevel	敏感级别标志：0～255，0 为最低，255 为最高。
m_szMsgType	消息类型标识，首字符‘M’为消息，‘F’为文件。
m_iDestUserCount	目的地址个数。
m_pArrDestUserAddr	包含 m_iDestUserCount 个目的地址的数组指针。
m_szTopicName	主题名。

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 0 置 1 是通知消息传输系统对消息进行可靠传输；m_ucFlag 的位 1 置 1 是通知消息传输系统对消息压缩后传输，用户通过 FDEAPI 接触到的消息包永远是非压缩的；m_ucFlag 的位 2 置 1 是通知消息传输系统对应答包自动推送到发送方。

为了统一属性，实例只在文件传输过程中才起作用，其它接口按照要求填写即可。

约定：

如果用户调用 Mr3Send 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 STUConnInfo3

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 STUConnInfo3
{
    char                m_szMRIP[MR3_MAXLEN_IP];
    unsigned short      m_usMRPort;
};
```

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。

特殊说明：

该结构在调用 Mr3Init 函数时作为输入参数使用，用户需传入该结构的数组指针和数组元素个数。Mr3Init 首先尝试与结构数组中指定的第一个消息路由器连接，如不成功再尝试连接数组的下一个消息路由器。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。

4.2.3 App 实例状态信息 STUAppStatus

STUAppStatus 结构用来表示 app 单个实例的状态信息。

结构定义：

```
STUAppStatus
{
    char                m_szAppID[MR3_MAXLEN_ADDR];                /*源应用标识，必须是'\0'结尾的字符串 */
    char                m_szLinkID[MR3_MAXLEN_ADDR];               /*应用连接标识，以“\0”结尾的字符串 */
    unsigned short      m_usInstanceID;                            /*应用实例标识，以“\0”结尾的字符串 */
```

```
    char                m_szIP[MR3_MAXLEN_IP];                  /*应用
实例程序服务器 IP 地址，以“\0”结尾的字符串 */
    unsigned short      m_usPort;                               /*应用实例程
序服务器端口号 */
    char                m_szMrName[MR3_MAXLEN_ADDR];            /*应
用实例登录的 MR 名 */
    char                m_szLoginTime[MR3_MAXLEN_USERDATA];     /*应
用实例登录上 MR 的时间点 */
    long long           m_i64QueueLength;                       /*应用标识
的接收队列长度 */
    long long           m_i64TopicQueueLength;                  /*应用标识
的接收主题消息队列长度 */
    long long           m_i64FileQueueLength;                   /*应用标识的
接收文件消息队列长度 */
    long long           m_i64SendPkgCountTotal;                 /*应用实例发
送报文总数（本次登录） */
    long long           m_i64RecvPkgCountTotal;                 /*应用实例接
收报文总数（本次登录） */
};
```

4.3 C 函数接口

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 18 个函数：

序号	函数名称	函数功能
1	Mr3Init	初始化，获取相关资源，并尝试与接入客户端 FDAP 建立连接。
2	Mr3IsLinkOK	查看并判断当前与接入客户端 FDAP 的连接是否正常。
3	Mr3CreatePkgID	生成消息包标识。
4	Mr3Send	通过 FDAP 向消息中枢单发消息，请求转发。
5	Mr3MultiDestSend	通过 FDAP 向消息中枢群发消息，将在中枢端分拆成多个独立的消息分别处理并转发。
6	Mr3Receive1	以方式 1 条件接收消息中枢转发来的消息。
7	Mr3Receive1_FreeBuf	释放 Mr3Receive1 函数调用中分配的内存。
8	Mr3Receive2	以方式 2 条件接收消息中枢转发来的消息。
9	Mr3Receive3	以方式 3 条件接收消息中枢转发来的消息。
10	Mr3SendTopicMsg	通过 FDAP 向消息中枢单发主题消息，请求转发。
11	Mr3ReceiveTopicMsg	订阅主题后，接收中枢转发来的主题消息。

12	Mr3ReceiveTopicMsg_FreeBuf	释放 Mr3ReceiveTopicMsg 函数调用中分配的内存。
13	Mr3Destroy	断开与 FDAP 的连接，释放相关资源。
14	Mr3GetVersion	取得该 API 的版本号。
15	Mr3RegRecvCondition	注册包下推条件，一次推送所有条件。
16	Mr3GetPeerUserStat	获取通信对端用户状态。
17	Mr3GetAppStatus	获取 app 实例的状态信息。
18	Mr3GetAppStatus_FreeBuf	释放 Mr3GetAppStatus 函数调用中分配的内存。

注意：MRAPI 动态链接库中也包含有 MR/MR2 开头的函数，是为了兼容旧版的接口而保留的；MR3 开头的函数是新接口，新旧接口函数不可混用。

4.3.2 Mr3Init

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 Mr3Init(const char* psAppID, unsigned short usInstID, const char* psAppPasswd, OnReceiveCallBack3 onReceive, const STUConnInfo3* pArrConnInfo3, int iArrConnInfoCount, void* pvUserData);
```

参数说明：

参数	说明
psAppID [in]	本应用的应用标识。
usInstID[in]	实例 ID 标识。
psAppPasswd[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 Mr3Receive1/Mr3Receive1_FreeBuf、Mr3Receive2 或 Mr3Receive3/Mr3Receive1_FreeBuf 同时使用。
pArrConnInfo3[in]	接入客户端连接信息数组。
iArrConnInfoCount [in]	接入客户端连接信息数组元素个数。
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 Mr3IsLinkOK

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。

注意：Mr3Init 返回可用句柄之后，并不代表网络已连接成功，有可能正在进行网络连接，所以在 Mr3Init 函数调用后，应该等待数秒后再进行 Mr3IsLinkOK 判断，以免出现误判。

函数原型：

```
int _stdcall    Mr3IsLinkOK(void* pHandle);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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。

4.3.4 Mr3CreatePkgID

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 Mr3CreatePkgID(void* pHandle, char szPkgID[MR3_MAXLEN_PKGID]);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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。

4.3.5 Mr3Send

单发消息包函数。该函数完成发送消息给单个目的用户的功能。

约定：

如果用户调用 Mr3Send 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 Mr3Send(void* pHandle, const char* psPkg, int iPkgLen, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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 Mr3Init 中设置的本应用的应用标识。 (3)m_usSourceInstID 的输入为初始化的实例名。

	(4)m_szDestUserID、m_szDestAppID 和 m_usDestInstID 输入为消息包的目的地址——目的用户标识、目的应用标识和实例标识，输出不变。 (5)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 Mr3CreatePkgID 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。 (6)m_szCorrPkgID、m_szUserData1 和 m_szUserData2 输入为用户自定义的数据，输出不变。 (7)m_ucFlag 输入为需要的包处理位标志，输出不变。 (8)m_ucBizType 输入为数据的业务类型。 (9)m_ucPriority 输入为消息发送的优先级别。 (10)m_ucSensitiveLevel 输入为消息内容的敏感级别。 (11)m_szMsgType 输入为消息的类型。 (12)m_szTopicName 输入空。
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。

4.3.6 Mr3MultiDestSend

群发消息包函数。该函数完成发送消息给多个目的用户的功能。群发消息不支持发送可靠消息。

函数原型：

```
int _stdcall Mr3MultiDestSend(void* pHandle, const char* psPkg, int iPkgLen,
```

STUMultiDestMsgProperty3* pMsgPropery);

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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_iDestUserCount 输入为待发送的目的地址个数。 (2)m_pArrDestUserAddr 输入为指向数组的指针，数组包含多个目的地址结构体。 其它字段同 Mr3Send 的 pMsgPropery 的字段。

返回值说明：

返回值	说明
0	成功。
其他	失败。

4.3.7 Mr3Receive1

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 Mr3Receive1_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 Mr3Init 中的回调函数 OnReceive 进行接收；或者只能使用 Mr3Receive1/Mr3Receive1_FreeBuf、Mr3Receive2 或 Mr3Receive3/Mr3Receive1_FreeBuf 进行接收，两者不能同时使用。

函数原型：

int _stdcall Mr3Receive1(void* pHandle, char** ppsPkg, int* piOutPkgLen, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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 Mr3Receive1_FreeBuf 函数释放该内存。
piOutPkgLen [out]	接收到的消息包的实际长度。
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。

4.3.8 Mr3Receive1_FreeBuf

消息包内存释放函数。该函数释放调用 Mr3Receive1 或 Mr3Receive3 函数时 FDEAPI 内部自动分配的消息包内存。

函数原型：

```
void _stdcall    Mr3Receive1_FreeBuf(char* psPkg);
```

参数说明：

参数	说明
psPkg [in]	由 Mr3Receive1 或 Mr3Receive3 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 Mr3Receive2

消息包接收函数 2。该函数与 Mr3Receive1 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 Mr3Init 中的回调函数 OnReceive 进行接收；或者只能使用 Mr3Receive1/Mr3Receive1_FreeBuf、Mr3Receive2 或 Mr3Receive3/Mr3Receive1_FreeBuf 进行接收，两者不能同时使用。

函数原型：

```
int _stdcall    Mr3Receive2(void* pHandle, char* psPkg, int* piOutPkgLen, int iBufLenIn, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr2Init 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 [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。

4.3.10 Mr3Receive3

消息包接收函数 3。该函数与 Mr3Receive1 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 Mr3Receive1_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 Mr3Init 中的回调函数 OnReceive 进行接收；或者只能使用 Mr3Receive1/Mr3Receive1_FreeBuf、Mr3Receive2 或 Mr3Receive3/Mr3Receive1_FreeBuf 进行接收，两者不能同时使用。

函数原型：

```
int _stdcall Mr3Receive3(void* pHandle, char** ppsPkg, int* piOutPkgLen, int*
```

piErrSXCode, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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 Mr3Receive1_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。

4.3.11 Mr3SendTopicMsg

发送主题消息包函数。该函数完成发送主题消息的功能。

函数原型：

```
int _stdcall Mr3SendTopicMsg(void* pHandle, const char* psPkg, int iPkgLen, STUMsgProperty3 *pMsgProperty);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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]	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 Mr3Init 中设置的本应用的应用标识。 (3)m_szDestUserID 和 m_szDestAppID 输入为空。 (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 Mr3CreatePkgID 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_ucFlag 输入为需要的包处理位标志，输出不变。 (7)m_ucBizType 输入为数据的业务类型。 (8)m_ucPriority 输入为消息发送的优先级别。

	(9)m_ucSensitiveLevel 输入为消息内容的敏感级别。 (10)m_szMsgType 输入为消息的类型。 (11)m_szTopicName 输入为主题名。

返回值说明：

返回值	说明
0	成功。
其他	失败。

4.3.12 Mr3ReceiveTopicMsg

主题消息包接收函数。该函数与 Mr3Receive1 函数功能类似，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 Mr3ReceiveTopicMsg_FreeBuf 函数释放分配的内存。

主题消息采用下推模式，由中枢分配，无须提交注册条件。

函数原型：

```
int _stdcall Mr3ReceiveTopicMsg(void* pHandle, char** ppsPkg, int* piOutPkgLen, STUMsgProperty3 *pMsgProperty)
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 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 Mr3ReceiveTopicMsg_FreeBuf 函数释放该内存。
piOutPkgLen [out]	接收到的消息包的实际长度。
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。 m_szTopicName 可以输入需要接收主题的消息。如果为空，则接收所有订阅的主题消息。

返回值说明：

返回值	说明
0	成功。
其他	失败。

4.3.13 Mr3ReceiveTopicMsg_FreeBuf

消息包内存释放函数。该函数释放调用 Mr3ReceiveTopicMsg 函数时 FDEAPI 内部自动分配的消息包内存。

函数原型：

```
void _stdcall Mr3ReceiveTopicMsg_FreeBuf(char* psPkg);
```

参数说明：

参数	说明
psPkg [in]	由 Mr3ReceiveTopicMsg 函数的第二个参数返回的指针。

返回值说明：

返回值	说明
无	

4.3.14 Mr3Destroy

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　Mr3Destroy(void* pHandle);

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 时返回的值。

返回值说明：

返回值	说明
无	

4.3.15　Mr3GetVersion

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 Mr3GetVersion(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“01.04.20190630”，其中 01 表示大版本号， 04 表示小版本号，20190630 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	

4.3.16　Mr3RegRecvCondition

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 发送多个接收条件到 FDMR，再由 FDMR 发送满足条件的包给 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，此函数调用多次时，最后一次调用的下推条件才有效，之前的条件都将被清除。

函数原型：

```
int _stdcall Mr3RegRecvCondition(void* pHandle, STUMsgProperty3* pArrMsgPropery, int iArrayCount);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 时返回的值。
pArrMsgPropery [in/out]	指向消息属性结构体数组的指针，输入将作为 MR 下推包的选择条件，消息属性结构体中有 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。
iArrayCount [in]	接收条件个数，即消息属性结构体数组中元素个数。

返回值说明：

返回值	说明
0	成功。
其他	失败。

4.3.17 Mr3GetPeerUserStat

获取通信对端用户状态。调用了 Mr3Init 初始化成功后一段时间，使用该函

数获取对端用户状态。

注意：获取的对端用户状态信息并不是实时的，可能存在一定的延时。

函数原型：

```
int _stdcall Mr3GetPeerUserStat(void* pHandle, const char* psPeerUserID, int* piOutStat);
```

参数说明：

参数	说明
pHandle [in]	连接句柄，调用 Mr3Init 时返回的值。
pPeerUserID [in]	通信对端用户标识。
piOutStat [out]	返回通信对端用户当前状态。返回值：1-在线，0-不在线。

返回值说明：

返回值	说明
0	成功
其他	失败，详细参考 4.1.4 节。

4.3.18 Mr3GetAppStatus

获取 app 实例的状态信息。

函数原型：

```
int _stdcall Mr3GetAppStatus(void* pHandle, const char* psAppID, STUAppStatus** ppOutAppStatus, int* piOutArrLen);
```

参数说明：

参数	说明
pHandle [in]	Mr3Init 函数返回的句柄，该句柄作为其他函数调用的参数。
psAppID[in]	若填写 psAppID，则查询的是单个 app 的状态信息；若不填写，查询所有全部 app。
ppOutAppStatus[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 Mr3GetAppStatus_FreeBuf 函数释放该内存。
piOutArrLen[out]	iArrLen 返回状态信息数组长度。

返回值说明：

返回值	说明
0	成功。
其他	失败。

4.3.19 Mr3GetAppStatus_FreeBuf

消息包内存释放函数。该函数释放调用 Mr3GetAppStatus 函数时 FDEAPI 内部自动分配的消息包内存。

函数原型：

```
void _stdcall Mr3GetAppStatus_FreeBuf(STUAppStatus* pAppStatus);
```

参数说明：

参数	说明
pAppStatus[in]	由 Mr3GetAppStatus 函数的第三个参数返回的指针。

返回值说明：

返回值	说明
无	

4.3.20 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 Mr3Init，最后调用 Mr3Destroy，其间是消息/主题/文件发送和接收的各种调用。

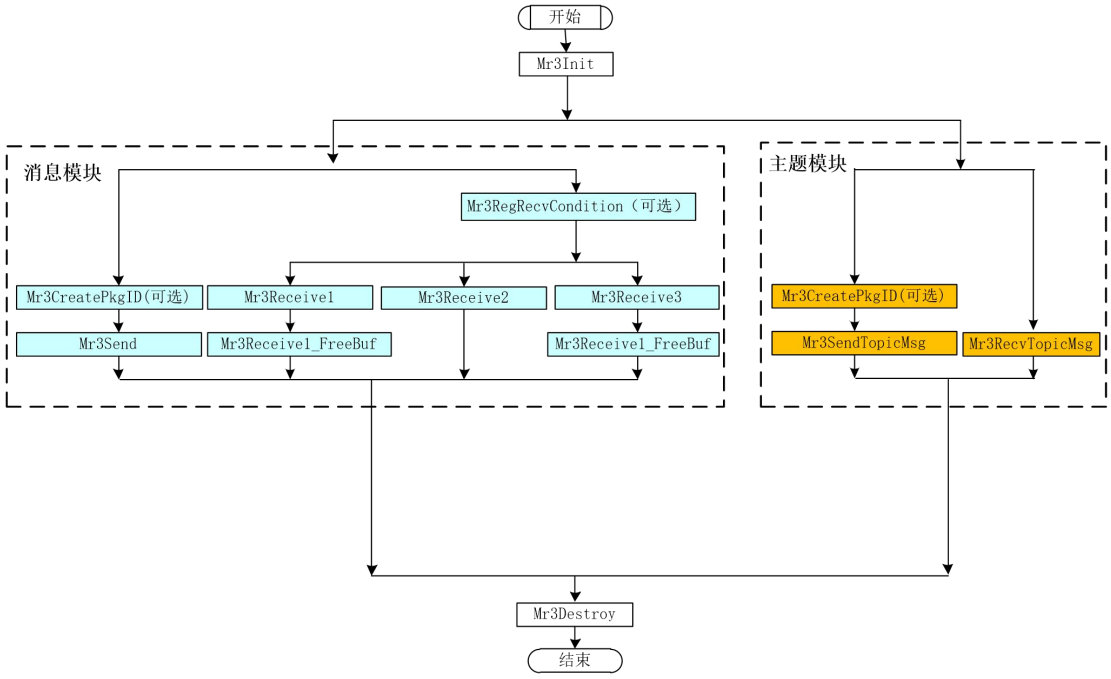


图 3　　FDEAPI 主要函数调用顺序

图 3 中分为三个模块，实现普通消息、主题消息和文件的发送与接收，使用时只需要一次初始化和一次退出，即可实现三种功能。

注意：接收消息包时，或者只能使用函数 Mr3Init 中的回调函数 OnReceive 进行接收；或者只能使用 Mr3Receive1/Mr3Receive1_FreeBuf、Mr3Receive2 或 Mr3Receive3/Mr3Receive1_FreeBuf 进行接收，两者不能同时使用。

5　C 编程示例

建立控制台应用程序工程，添加发布包 rar 中 platform.h,inifile.h,inifile.cpp,loadmrapi.h,loadmrapi.cpp,mrapi.h,mr3api.h,demo.cpp 文件到工程，编译即可。以下只附上 demo.cpp 代码。

```
#include "loadmrapi.h"
#include "inifile.h"
#include <stdio.h>
#include <string>
#include <string.h>
#if (G_PLATFORM_OS==G_PLATFORM_OS_WINDOWS)
#include <Windows.h>
#include <WinBase.h>
```

```
#include <process.h>
#else
#include <pthread.h>
#endif
#include <time.h>

using namespace std;

#if (G_PLATFORM_OS==G_PLATFORM_OS_WINDOWS)
#define LIB_MRAPI_FILE       "./mrapi.dll"
#define THREAD_RTN unsigned __stdcall
#else
#define LIB_MRAPI_FILE       "./libmrapi.so"
#define THREAD_RTN void*
#define    Sleep(waitTime) usleep((waitTime)*1000)
#endif

typedef struct _tagCfgParam
{
      string m_ssAppID;                    //该应用使用的AppID。
      string m_ssAppPasswd;                //App登陆到MR的密码。
     STUConnInfo3 m_oIpAddress[2];         //App连接MR的IP地址，若有两个MR，则填写两个MR
的IP。
      STUMsgProperty3 m_oSendMsgProperty;  //发送消息包的属性。
} STUCfgParam;

//加载配置参数
bool LoadCfg(STUCfgParam* pCfgParam)
{
      //配置文件为"demo.ini"
      string ssCfgFile = "./demo.ini";

      char szAppPasswd[10];
      memset(szAppPasswd, 0, 10);
      memset(&pCfgParam->m_oIpAddress, 0, 2*sizeof(pCfgParam->m_oIpAddress));
      memset(&pCfgParam->m_oSendMsgProperty, 0, sizeof(pCfgParam->m_oSendMsgProperty));

      read_profile_string("Config", "AppID", pCfgParam->m_oSendMsgProperty.m_szSourceAppID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szSourceAppID), "app1", ssCfgFile.c_str());
      read_profile_string("Config", "AppPasswd", szAppPasswd, sizeof(szAppPasswd), "1",
ssCfgFile.c_str());
      read_profile_string("Config", "MrIP1", pCfgParam->m_oIpAddress[0].m_szMRIP,
sizeof(pCfgParam->m_oIpAddress[0].m_szMRIP), "127.0.0.1",ssCfgFile.c_str());
      pCfgParam->m_oIpAddress[0].m_usMRPort = (unsigned short)read_profile_int("Config", "MrPort1",
```

```
0, ssCfgFile.c_str());
	read_profile_string("Config", "MrIP1", pCfgParam->m_oIpAddress[1].m_szMRIP,
sizeof(pCfgParam->m_oIpAddress[1].m_szMRIP), "127.0.0.1", ssCfgFile.c_str());
	pCfgParam->m_oIpAddress[1].m_usMRPort = (unsigned short)read_profile_int("Config", "MrPort2",
0, ssCfgFile.c_str());

	pCfgParam->m_ssAppID = pCfgParam->m_oSendMsgProperty.m_szSourceAppID;
	pCfgParam->m_ssAppPasswd = szAppPasswd;

	read_profile_string("Config", "SourceUserID",
pCfgParam->m_oSendMsgProperty.m_szSourceUserID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szSourceUserID), "", ssCfgFile.c_str());
	read_profile_string("Config", "DestUserID", pCfgParam->m_oSendMsgProperty.m_szDestUserID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szDestUserID), "",ssCfgFile.c_str());
	read_profile_string("Config", "DestAppID", pCfgParam->m_oSendMsgProperty.m_szDestAppID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szDestAppID), "", ssCfgFile.c_str());
	read_profile_string("Config", "UserData1", pCfgParam->m_oSendMsgProperty.m_szUserData1,
sizeof(pCfgParam->m_oSendMsgProperty.m_szUserData1), "",ssCfgFile.c_str());
	read_profile_string("Config", "UserData2", pCfgParam->m_oSendMsgProperty.m_szUserData2,
sizeof(pCfgParam->m_oSendMsgProperty.m_szUserData2), "",ssCfgFile.c_str());
	pCfgParam->m_oSendMsgProperty.m_ucFlag = (unsigned char)read_profile_int("Config", "Flag", 0,
ssCfgFile.c_str());
	pCfgParam->m_oSendMsgProperty.m_ucBizType = (unsigned char)read_profile_int("Config",
"BizType", 0, ssCfgFile.c_str());
	pCfgParam->m_oSendMsgProperty.m_ucPriority = (unsigned char)read_profile_int("Config",
"Priority", 0, ssCfgFile.c_str());
	pCfgParam->m_oSendMsgProperty.m_ucSensitiveLevel = (unsigned char)read_profile_int("Config",
"SensitiveLevel", 0, ssCfgFile.c_str());

	return true;
}

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


void*	g_pHandle = NULL; //调用Mr3Init初始化函数返回的句柄。


//回调函数OnReceive，可作为Mr3Init第3个参数传入，该App内部会调用Mr3Recveive函数接收消息包，
并将消息包通过该回调函数返给用户。
//注意：若已使用OnReceive回调方式接收消息包，则不要使用Mr3Receive函数进行接收。
```

```
int OnReceive(const char* psPkg, int iPkgLen, const STUMsgProperty3* pMsgPropery, void* pvUserData)
{
	//接收到一个数据包。
	static int s_iRecvCount;
	++s_iRecvCount;

	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", s_iRecvCount);

	if (pMsgPropery->m_szCorrPkgID[0]=='\0')
	{
		//一般情况：若m_szCorrPkgID为空，则说明该消息包是请求包；若不为空，则说明该消息
包是应答包。
		//处理请求包:建议以下处理将消息包抛到业务处理线程进行处理。
		//...

		//返回应答包：将应答包属性m_szCorrPkgID的值赋值为对应请求包属性m_szPkgID的值。
		STUMsgProperty3 oMsgProperySend;
		memset(&oMsgProperySend, '\0', sizeof(STUMsgProperty3));
		strcpy(oMsgProperySend.m_szDestUserID, pMsgPropery->m_szSourceUserID);
		strcpy(oMsgProperySend.m_szDestAppID, pMsgPropery->m_szSourceAppID);
		strcpy(oMsgProperySend.m_szCorrPkgID, pMsgPropery->m_szPkgID); //set response's
CorrPkgID = PkgID of request pkg.
		strcpy(oMsgProperySend.m_szUserData1, pMsgPropery->m_szUserData1);
		strcpy(oMsgProperySend.m_szUserData2, pMsgPropery->m_szUserData2);
		oMsgProperySend.m_ucFlag = 0;

		//构造应答包数据，并调用Mr2Send函数发送应答包。
		const char* psResponsePkg = "Response pkg.";
		int iResponsePkgLen = strlen(psResponsePkg);
		Mr3SendFunc(g_pHandle, psResponsePkg, iResponsePkgLen, &oMsgProperySend, 2000);
	}
	else
	{
		//一般情况：若m_szCorrPkgID为空，则说明该消息包是请求包；若不为空，则说明该消息
包是应答包。
		//处理应答包:建议以下处理将消息包抛到业务处理线程进行处理。
		//...
	}
```

```
    return 0;
}

//接收线程：调用Mr3Receive函数接收消息包和处理消息，并返回应答数据。
//注意：若已使用Mr3Receive函数接收消息包，则不要使用OnReceive回调方式接收消息包。
THREAD_RTN RecvThrd(void *pvParam)
{
    //循环调用Mr3Receive接收消息包。
    while (1)
    {
        STUMsgProperty3 oMsgPropery;
        memset(&oMsgPropery, '\0', sizeof(STUMsgProperty3));

        int iRecvPkgLen = 0;
        char* psPkg = NULL;
        int iRecvRet = Mr3Receive1Func(g_pHandle, &psPkg, &iRecvPkgLen, &oMsgPropery, 1000);
        if(iRecvRet==MR3_ERRCODE_OK)
        {
            //返回值为MR3_ERRCODE_OK：接收到一个数据包。
            static int s_iRecvCount;
            ++s_iRecvCount;
            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, iRecvPkgLen, oMsgPropery.m_szCorrPkgID,
oMsgPropery.m_szUserData1, oMsgPropery.m_szUserData2);
            printf("client recv count[%d]\n", s_iRecvCount);

            if (oMsgPropery.m_szCorrPkgID[0]=='\0')
            {
                //一般情况：若m_szCorrPkgID为空，则说明该消息包是请求包；若不为空，则说
明该消息包是应答包。
                //处理请求包:建议以下处理将消息包抛到业务处理线程进行处理。
                //...

                //返回应答包：将应答包属性m_szCorrPkgID的值赋值为对应请求包属性
m_szPkgID的值。
                STUMsgProperty3 oMsgProperySend;
                memset(&oMsgProperySend, '\0', sizeof(STUMsgProperty3));
                strcpy(oMsgProperySend.m_szDestUserID, oMsgPropery.m_szSourceUserID);
                strcpy(oMsgProperySend.m_szDestAppID, oMsgPropery.m_szSourceAppID);
                strcpy(oMsgProperySend.m_szCorrPkgID, oMsgPropery.m_szPkgID); //set response's
CorrPkgID = PkgID of request pkg.
                strcpy(oMsgProperySend.m_szUserData1, oMsgPropery.m_szUserData1);
```

```
                    strcpy(oMsgProperySend.m_szUserData2, oMsgPropery.m_szUserData2);
                    oMsgProperySend.m_ucFlag = 0;

                    //构造应答包数据，并调用Mr3Send函数发送应答包。
                    const char* psResponsePkg = "Response pkg.";
                    int iResponsePkgLen = strlen(psResponsePkg);
                    Mr3SendFunc(g_pHandle, psResponsePkg, iResponsePkgLen, &oMsgProperySend,
2000);
                }
                else
                {
                    //一般情况：若m_szCorrPkgID为空，则说明该消息包是请求包；若不为空，则说
明该消息包是应答包。
                    //处理应答包:建议以下处理将消息包抛到业务处理线程进行处理。
                    //...
                }

                //调用Mr3Receive1_FreeBuf释放资源
                Mr3Receive1_FreeBufFunc(psPkg);
            }
            else
            {
                //返回值为其它值（参考mr3api.h错误码定义）。
                MySleep(20);
            }
        }

    return 0;
}

//发送线程：构造发送数据，调用Mr3Send函数发送消息包给接收方。
THREAD_RTN SendThrd(void *pvParam)
{
    //构造发送数据。
    string ssPkg;
    ssPkg +=" <IFTS Len=\" 1205\" DataVer =\"1.0.0.1\" SeqNo=\"\" Type=\"B\" Dup=\"N\"
CheckSum=\"\">\n";
    ssPkg += "   <MsgText> \n";
    ssPkg += "       <Acmt.001.01> \n";
    ssPkg += "           <MsgHdr> \n";
    ssPkg += "               <Ver>1.0</Ver> \n";
    ssPkg += "               <SysType>0</SysType> \n";
    ssPkg += "               <InstrCd>11002</InstrCd> \n";
    ssPkg += "               <Sender> \n";
```

```
ssPkg += "                  <InstType>S</InstType> \n";
ssPkg += "                    <BrchId>1</BrchId> \n";
ssPkg += "                </Sender> \n";
ssPkg += "                <Recver> \n";
ssPkg += "                    <InstType>B</InstType> \n";
ssPkg += "                    <BrchId>2</BrchId> \n";
ssPkg += "                </Recver> \n";
ssPkg += "                <Date>20061216</Date> \n";
ssPkg += "                <Time>114340</Time> \n";
ssPkg += "                <Ref> \n";
ssPkg += "                  <IssrType>S</IssrType> \n";
ssPkg += "                  <Ref>9</Ref> \n";
ssPkg += "                </Ref> \n";
ssPkg += "              </MsgHdr> \n";
ssPkg += "              <Cust> \n";
ssPkg += "                <Name>郭达</Name> \n";
ssPkg += "                <CertType>10</CertType> \n";
ssPkg += "                <CertId>42218901</CertId> \n";
ssPkg += "                <Type>INVE</Type> \n";
ssPkg += "                <Sex>M</Sex> \n";
ssPkg += "                <Ntnl>CHN</Ntnl> \n";
ssPkg += "                <Addr>八卦四路22号</Addr>\n";
ssPkg += "                <PstCd>518029</PstCd> \n";
ssPkg += "                <Email></Email> \n";
ssPkg += "                <Fax></Fax> \n";
ssPkg += "                <Mobile>13843453</Mobile> \n";
ssPkg += "                <Tel></Tel> \n";
ssPkg += "              </Cust> \n";
ssPkg += "              <Agt> </Agt>\n";
ssPkg += "              < BkAcct> </ BkAcct>\n";
ssPkg += "              < ScAcct> </ ScAcct>\n";
ssPkg += "              < ScAcct> </ ScAcct>\n";
ssPkg += "          </Acmt.001.01>\n";
ssPkg += "     </MsgText>\n";
ssPkg += " </IFTS>\n";

STUCfgParam* pCfgParam = (STUCfgParam*)pvParam;

//调用Mr2Send函数发送数据给接收方，每隔1s发送。
while (1)
{
    STUMsgProperty3 oMsgPropery;
    memcpy(&oMsgPropery, &(pCfgParam->m_oSendMsgProperty), sizeof(STUMsgProperty3));
```

```
            int iSendRet = Mr3SendFunc(g_pHandle, ssPkg.c_str(), (int)ssPkg.length(), &oMsgPropery,
2000);
            if (iSendRet == MR3_ERRCODE_OK)
            {
                //返回值为MR3_ERRCODE_OK：发送成功
                printf("send ok: PkgID[%s], src[%s.%s], dest[%s.%s], pkgLen[%d], CorrPkgID[%s],
UserData1[%s], UserData2[%s] \n", oMsgPropery.m_szPkgID, oMsgPropery.m_szSourceUserID,
oMsgPropery.m_szSourceAppID, oMsgPropery.m_szDestUserID, oMsgPropery.m_szDestAppID,
(int)ssPkg.length(), oMsgPropery.m_szCorrPkgID, oMsgPropery.m_szUserData1,
oMsgPropery.m_szUserData2);
            }
            else
            {
                //返回值为其它值（参考mr3api.h错误码定义）。
                printf("send failed[%d]: PkgID[%s], src[%s.%s], dest[%s.%s], pkgLen[%d],
CorrPkgID[%s], UserData1[%s], UserData2[%s] \n", iSendRet, oMsgPropery.m_szPkgID,
oMsgPropery.m_szSourceUserID, oMsgPropery.m_szSourceAppID, oMsgPropery.m_szDestUserID,
oMsgPropery.m_szDestAppID, (int)ssPkg.length(), oMsgPropery.m_szCorrPkgID,
oMsgPropery.m_szUserData1, oMsgPropery.m_szUserData2);
            }

            MySleep(1000);
        }

    return 0;
}


int main(int argc, char* argv[])
{
    //加载mrapi.dll
    if (0 != loadmrapi(LIB_MRAPI_FILE))
    {
        printf("load %s error.\n", LIB_MRAPI_FILE);
        return -1;
    }

    //加载配置
    STUCfgParam oCfgParam;
    LoadCfg(&oCfgParam);

    const char* psUserData = "UserData";

    //调用Mr3Init函数进行初始化
```

```
	g_pHandle = Mr3InitFunc(oCfgParam.m_ssAppID.c_str(), oCfgParam.m_ssAppPasswd.c_str(),
NULL, oCfgParam.m_oIpAddress, 2, (void*)psUserData);
	if(g_pHandle==NULL)
	{
		//初始化失败。
		unloadmrapi();
		printf("ERROR: Mr3Init failed, exit.\n");
		return -1;
	}
	else
	{
		//初始化成功。
		printf("Mr3Init OK.\n");
	}

#if (G_PLATFORM_OS==G_PLATFORM_OS_WINDOWS)

	//启动接收线程
	if(_beginthreadex(NULL, 0,	RecvThrd, &oCfgParam, 0, NULL)==0)
	{
		unloadmrapi();
		printf("ERROR: create recv thread failed, exit.\n");
		return -1;
	}
	else
	{
		printf("create recv thread OK.\n");
	}

	//启动发送线程
	if(_beginthreadex(NULL, 0,	SendThrd, &oCfgParam, 0, NULL)==0)
	{
		unloadmrapi();
		printf("ERROR: create send thread failed, exit.\n");
		return -1;
	}
	else
	{
		printf("create send thread OK.\n");
	}

#else
	//启动接收线程
	pthread_t recvpid;
```

```
    if(pthread_create(&recvpid, NULL,    RecvThrd, &oCfgParam)!=0)
    {
        unloadmrapi();
        printf("ERROR: create recv thread failed\n");
        return -1;
    }
    else
    {
        printf("create recv thread OK.\n");
    }

    //启动发送线程
    pthread_t sendpid;
    if(pthread_create(&sendpid, NULL,    SendThrd, &oCfgParam)!=0)
    {
        unloadmrapi();
        printf("ERROR: create send thread failed\n");
        return -1;
    }
    else
    {
        printf("create send thread OK.\n");
    }
#endif

    while (1)
    {
        //间隔1s检查App与Mr的连接是否正常。
        if (Mr3IsLinkOKFunc(g_pHandle)==0)
        {
            printf("WARN: Link not ok.\n");
        }

        MySleep(1000);
    }

    //程序退出，释放资源。
    Mr3DestroyFunc(g_pHandle);
    unloadmrapi();

    return 0;
}
```

6 Java 接口说明

6.1.1 方法清单

mrapi 类方法（本手册第 7 章有创建 mrapi 类的具体说明）共有 13 个，分别对应本文 4.3.1 提到的 18 个函数，具体如下：

对应序号	函数名称	函数功能
1	Mr3Init	初始化，获取相关资源，并尝试与接入客户端 FDAP 建立连接。
2	Mr3IsLinkOK	查看并判断当前与接入客户端 FDAP 的连接是否正常。
3	Mr3CreatePkgID	生成消息包标识。
4	Mr3Send	通过 FDAP 向消息中枢单发消息，请求转发。
5	Mr3Receive1	以方式 1 条件接收消息中枢转发来的消息。
6	Mr3Receive3	以方式 3 条件接收消息中枢转发来的消息。
7	Mr3Destroy	断开与 FDAP 的连接，释放相关资源。
8	Mr3GetVersion	取得该 API 的版本号。
9	Mr3RegRecvCondition	注册包下推条件，一次推送所有条件。
10	Mr3GetPeerUserStat	获取通信对端用户状态。
11	Mr3SendTopicMsg	通过 FDAP 向消息中枢单发主题消息，请求转发。
12	Mr3ReceiveTopicMsg	订阅主题后，接收中枢转发来的主题消息。
13	Mr3GetAppStatus	获取 app 实例的状态信息。

注意：MRAPI 动态链接库中也包含有 MR 和 MR2 开头的函数，是为了兼容旧版的接口而保留的；MR3 开头的函数是新接口，新旧接口函数不可混用。

6.1.2 Mr3Init

连接 FDAP 时的初始化方法。该方法对 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 在调用此初始化方法时，会尝试使用 TCP 方式连接接入客户端，一般情况下，在该方法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，调用此方法返回时，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：

```
public native static int Mr3Init(String sAppId, int iInstanceId, String sAppPwd, String
```

```
sIp, short uPort, String sIpbak, short uPortbak);
```

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。
sAppPwd[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。
sIp[in]	第一个接入客户端消息路由器的 IP 地址。
uPort[in]	第一个接入客户端消息路由器的连接端口。
sIpbak[in]	第二个接入客户端消息路由器的 IP 地址。
uPortbak[in]	第二个接入客户端消息路由器的连接端口。

注意：Mr3Init 首先尝试与指定的第一个消息路由器连接，如不成功再尝试连接第二个消息路由器。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的所有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有可能不是参数指定的这两个。

返回值说明：

返回值	说明
0	初始化失败。
非 0	初始化成功。（须结合 Mr3IsLinkOK 一起判断）

6.1.3 Mr3IsLinkOK

连接状态判断方法。该方法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。

注意：Mr3Init 获取返回值之后，并不代表网络已连接成功，有可能正在进行网络连接，所以使用 Mr3Init 方法后，应该等待数秒后再进行 Mr3IsLinkOK 判断。若在使用 Mr3Init 方法之后立即进行 Mr3IsLinkOK 判断，则有可能会认为没有正常连接。

函数原型：

```
public native static int Mr3IsLinkOK(String sAppId, int iInstanceId);
```

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。

返回值说明：

返回值	说明
0	连接不正常。
1	连接正常。

6.1.4 Mr3CreatePkgID

消息包标识生成方法。该方法生成一个在整个 FDEP 中全局唯一的 UUID，可用作消息包标识。

函数原型：

```
public native static byte[]    Mr3CreatePkgID(String sAppId, int iInstanceId);
```

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。

返回值说明：

返回值	说明
UUID 值	生成的消息包标识 ID。
带“NULL”的字符串	失败。

6.1.5 Mr3Send

单发消息包方法。该方法完成发送消息给单个目的用户的功能。

约定：

如果用户调用 Mr3Send 方法发送的是业务应答包， CorrPkgID 参数统一约定，填写为与对应请求包中的 PkgID 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，该字段为空。

函数原型：

```
public native static String Mr3Send( byte[] psPkg, String sSourceUserID,String sSourceAppID, int iSrcInstanceId, String sDestUserID, String sDestAppID,String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, String sMsgType, byte cFlag, int iBizType, byte cPriority, byte cSensitiveLevel, int iMillSecTimeo);
```

参数说明：

参数	说明
psPkg [in]	需要发送的数据。
sSourceUserID [in]	源用户标识。
sSourceAppID [in]	源应用标识。
iSrcInstanceId[in]	本应用的实例标识。
sDestUserID [in]	目的用户标识。
sDestAppID [in]	目的应用标识。
sPkgID [in]	消息包的包标识，（可由 Mr3CreatePkgID 方法生成）。该参数为唯一的消息包标识。可由用户调用 Mr3CreatePkgID 方法生成的唯一标识符，输出不变；也可以输入为空，那么系统会自动分配唯一的消息包标识。对每一个发送的包，该 PkgID 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 PkgID。虽然 Mr3CreatePkgID 方法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 PkgID 或 CorrPkgID 可能重复，因此，用户不能简单地把 PkgID 或 CorrPkgID 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。
sCorrPkgID [in]	相关包标识。
sUserData1 [in]	用户数据 1。
sUserData2 [in]	用户数据 2。
sMsgType [in]	消息类型标识
cFlag [in]	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；

	位 2 --为 1 表示应答包会自动推送到发送方。
iBizType [in]	业务类型标志：0～255。
cPriority [in]	优先级别标志：3～5，5 为最低，3 为最高。
cSensitiveLevel [in]	敏感级别标志：0～255，0 为最低，255 为最高。
iMillSecTimeo [in]	以毫秒为单位的发送最大超时时间，建议该值为 2000，即 2 秒。

返回值说明：

返回值	说明
PkgID 字符串	成功。
""	失败。

6.1.6 Mr3Receive1

消息包接收方法 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。该方法接收到的消息包占用的内存由 FDEAPI 内部分配。

函数原型：

```
public native static byte[] Mr3Receive1(String sAppId, int iInstanceId,String sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, int iMillSecTimeo);
```

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。
sSourceUserID [in]	源用户标识条件。
sSourceAppID [in]	源应用标识条件。
sDestUserID [in]	目的用户标识条件。
sDestAppID [in]	目的应用标识条件。
sPkgID [in]	消息包的包标识条件。
sCorrPkgID [in]	相关包标识条件。
sUserData1 [in]	用户数据 1 条件。
sUserData2 [in]	用户数据 2 条件。
iMillSecTimeo [in]	以毫秒为单位的接收最大超时时间。该时间是该方法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，而不是该方法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，该方法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。

注意：该方法参数中的 6 个字段作为接收的判断条件： SourceUserID、SourceAppID、PkgID、CorrPkgID、UserData1 和 UserData2。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。

对 CorrPkgID 的输入条件有特殊处理：当 CorrPkgID 字符串的值是空时，则可以匹配消息中 CorrPkgID 字段为任意值的消息；当 CorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 CorrPkgID 字段为空的消息；当 CorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 CorrPkgID 字段为非空的消息。

返回值说明：

返回值	说明
pkID(64byte)+CorrpkID(64byte)+sourceUserID(32byte) +sourceAppID(32byte)+sourceIntanceId(16byte)+MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte)+SensitiveLevel(8byte) +UserData1(256byte) +UserData2(256byte) +data，上述前 12 个字段与输入的对应参数相同，data 为真正的数据	接收成功。
"NULL"或"NULL,errmsg"	接收失败。

6.1.7 Mr3Receive3

消息包接收方法 3。该方法与 Mr3Receive1 方法功能类似，但此方法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，该方法返回值当中的 errcode 值表示错误的原因代码。该方法接收到的消息包占用的内存由 FDEAPI 内部分配。

函数原型：

```
public native static byte[] Mr3Receive3(String sAppId, int iInstanceId,String sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID, String
```

sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, int iMillSecTimeo);

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。
SourceUserID [in]	源用户标识条件。
SourceAppID [in]	源应用标识条件。
DestUserID [in]	目的用户标识条件。
DestAppID [in]	目的应用标识条件。
PkgID [in]	消息包的包标识条件。
CorrPkgID [in]	相关包标识条件。
UserData1 [in]	用户数据 1 条件。
UserData2 [in]	用户数据 2 条件。
iMillSecTimeo [in]	以毫秒为单位的接收最大超时时间。该时间是该方法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，而不是该方法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，该方法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。

注意：该方法参数中的 6 个字段作为接收的判断条件： SourceUserID、SourceAppID、PkgID、CorrPkgID、UserData1 和 UserData2。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。

对 CorrPkgID 的输入条件有特殊处理：当 CorrPkgID 字符串的值是空时，则可以匹配消息中 CorrPkgID 字段为任意值的消息；当 CorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 CorrPkgID 字段为空的消息；当 CorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 CorrPkgID 字段为非空的消息。

返回值说明：

返回值	说明
errcode(4byte) +pkID(64byte) +CorrpkID(64byte) +sourceUserID(32byte) +sourceAppID(32byte)	接收成功。其中 errcode= "0000" 为接收正常，如果 errcode 为非

+sourceIntanceId(16byte)+MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte) +SensitiveLevel(8byte)+UserData1(256byte) +UserData2(256byte) +data，其中 12 个字段与输入的对应参数相同，data 为真正的数据。其中 errcode= "0000"为接收正常，如果 errcode 为非"0000"字符串，则由系统返回的错误信息，详细错误在 data 字段中	"0000"字符串，则由系统返回的错误信息，详细错误在 data 字段中。
"NULL"或"NULL,errmsg"	接收失败。

6.1.8 Mr3Destroy

资源释放方法。在 FDEAPI 使用完毕后，释放所有资源。该方法通常是最后一个被调用的 FDEAPI 方法。

函数原型：

```
public native static void Mr3Destroy(String sAppId, int iInstanceId);
```

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。

返回值说明：

返回值	说明
无	

6.1.9 Mr3GetVersion

取得该 API 的版本号。该方法可以在装载该动态库后的任何时间调用。

函数原型：

```
public native static byte[]    Mr3GetVersion();
```

参数说明：

参数	说明
无	

返回值说明：

返回值	说明
版本号字符串	

6.1.10 Mr3RegRecvCondition

注册下推条件方法。

用户使用接收方法 Mr3Receive1 或 Mr3Receive3 时，每次提供多个接收条件，从而接收到满足条件的消息。如果使用了注册下推条件方法，则系统会主动将满足条件的包依次下推给用户，这样的会带来单链接传输速率的提高，代价是牺牲一定灵活性。此方法只在对单链接流量有较高要求时调用，此方法使用多次时，最后一次调用的下推条件才有效，之前的条件都被清除。

函数原型：

```
public native static int    Mr3RegRecvCondition(String sAppId, int iInstanceId,String sSrcUserId, String sSrcAppId, String sDestUserId, String sDestAppId, String sPkgId, String sCorrPkgId, String sUserData1,String sUserData2);
```

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。
szSrcUserId [in]	源用户标识条件。
szSrcAppId [in]	源应用标识条件。
szDestUserId [in]	目的用户标识条件。
szDestAppId [in]	目的应用标识条件。
szPkgId [in]	消息包的包标识条件。
szCorrPkgId [in]	相关包标识条件。
szUserData1 [in]	用户数据 1 条件。
szUserData2 [in]	用户数据 2 条件。

注意：该方法参数中的 6 个字段作为接收的判断条件： szSrcUserID、szSrcAppID、szPkgId、szCorrPkgID、szUserData1 和 szUserData2。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。

对 szCorrPkgID 的输入条件有特殊处理：当 szCorrPkgID 字符串的值是空时，则可以匹配消息中 szCorrPkgID 字段为任意值的消息；当 szCorrPkgID 字符串的值为"<EMPTY>"时，则只匹配消息中 szCorrPkgID 字段为空的消息；当 szCorrPkgID 字符串的值为"<NOEMPTY>"时，则只匹配消息中 szCorrPkgID 字段为非空的消息。

返回值说明：

返回值	说明
0	成功。
小于 0 的数值	失败。

6.1.11 Mr3GetPeerUserStat

获取通信对端用户状态。调用 Mr3Init 方法初始化成功后一段时间，使用该方法获取对端用户状态。

函数原型：

```
public native static int Mr3GetPeerUserStat(String sAppId, int iInstanceId,String sPeerUserID);
```

参数说明：

参数	说明
sAppId[in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。
szPeerUserID [in]	通信对端用户标识。

返回值说明：

返回值	说明
-1	方法调用失败。
0	方法调用成功，对端用户不在线。
1	方法调用成功，对端用户在线。

6.1.12 Mr3SendTopicMsg

该函数完成发送主题消息的功能。

约定：

如果用户调用 Mr3SendTopicMsg 方法发送的是主题消息包，不需要填写对端用户信息。

函数原型：

```
public native static String Mr3SendTopicMsg(byte[] psPkg, String sSourceUserID,String sSourceAppID, int iSrcInstanceId, String sTopicName,String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, byte cFlag, byte cPriority, byte cSensitiveLevel);
```

参数说明：

参数	说明
psPkg [in]	需要发送的数据。
sSourceUserID [in]	源用户标识。
sSourceAppID [in]	源应用标识。
iSrcInstanceId[in]	本应用的实例标识。
sTopicName[in]	主题名。
sPkgID [in]	消息包的包标识，（可由 Mr3CreatePkgID 方法生成）。 该参数为唯一的消息包标识。可由用户调用 Mr3CreatePkgID 方法生成的唯一标识符，输出不变；也可以输入为空，那么系统会自动分配唯一的消息包标识。 对每一个发送的包，该 PkgID 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 PkgID。虽然 Mr3CreatePkgID 方法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 PkgID 或 CorrPkgID 可能重复，因此，用户不能简单地把 PkgID 或 CorrPkgID 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。
sCorrPkgID [in]	相关包标识。
sUserData1 [in]	用户数据 1。
sUserData2 [in]	用户数据 2。
cFlag [in]	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； 位 2 --为 1 表示应答包会自动推送到发送方。
cBizType [in]	业务类型标志：0～255。
cPriority [in]	优先级别标志：3～5，5 为最低，3 为最高。
cSensitiveLevel [in]	敏感级别标志：0～255，0 为最低，255 为最高。

返回值说明：

返回值	说明
PkgID 字符串	成功。
""	失败。

6.1.13 Mr3ReceiveTopicMsg

主题消息包接收函数。用户需要输入主题消息包长度，最大值为 5M。

主题消息采用下推模式，由中枢分配，无须提交注册条件。

函数原型：

```
public native static byte[] Mr3ReceiveTopicMsg(String sAppId, int iInstanceId, String sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,String sTopicName, String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2);
```

参数说明：

参数	说明
sAppId [in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。
sSourceUserID[in]	源用户标识条件。
sSourceAppID[in]	源应用标识条件。
sDestUserID [in]	目的用户标识条件。
sDestAppID [in]	目的应用标识条件。
sTopicName[in]	主题名。
sPkgID [in]	消息包的包标识条件。
sCorrPkgID [in]	相关包标识条件。
sUserData1 [in]	用户数据 1 条件。
sUserData2 [in]	用户数据 2 条件。

注意：该方法参数中的 7 个字段作为接收的判断条件： SourceUserID、

SourceAppID、TopicName、PkgID、CorrPkgID、UserData1 和 UserData2。如果这 7 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 7 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 7 个字段全为空，则收取队列中第一个消息包。

返回值说明：

返回值	说明
pkID(64byte)+CorrpkID(64byte)+topicName(17byte)+sourceUserID(32byte)+sourceAppID(32byte)+sourceInstanceID(16byte) +flag(8byte)+Priority(8byte)+SensitiveLevel(8byte)+ UserData1(256byte) + UserData2(256byte) +data 其中 11 个字段同 C 接口中 pMsgPropery 中各个字段 ，data 为真正的数据；	接收成功。
“NULL”	接收失败。

6.1.14 Mr3GetAppStatus

函数原型：

```
public native static byte[] Mr3GetAppStatus(String sAppId, int iInstanceId, String sQueryAppId);
```

参数说明：

参数	说明
sAppId [in]	本应用的应用标识。
iInstanceId[in]	本应用的实例标识。
sQueryAppId[in]	需要查询的 appid。

返回值说明：

返回值	说明
AppID(32byte) +LinkID(32byte) +InstanceID(16byte) +IP(16byte) +Port(8byte) +MrName(32byte) +LoginTime(256byte) +QueueLength(64byte) +TopicQueueLength(64byte)+FileQueueLength(64byte) +SendPkgCountTotal(64byte) +RecvPkgCountTotal(64byte)	成功。
NULL	失败。

注：

1、查询单个 app 且只有一个实例，成功返回的状态信息如表格所示；

2、如果查询的是单个 app 且只有一个实例，则无竖线"|"分隔；如果查询全部 app，则用竖线分隔每个 app 每个实例的状态信息；

6.1.15 调用顺序

详见 4.3.21。

7 Java 编程示例

7.1 示例说明

Java 调用 mrapi.dll 动态链接库有多种方式，本手册提供一种基于 JNI 的方法，仅供参考，用户可自行选择 JNA 等其他方式。

7.2 具体步骤

7.2.1 建立工程

建立工程目录，将从 mrapi_05.01_win32_java_bin_20190812.rar 包中取出的 mrapi.dll, ini/mr.ini, mrapi.h, mr2api.h, mr3api.h 这五个文件拷贝到工程目录下，并在该目录下依次建立 com/sscc/fdep 目录结构。在 fdep 目录下创建 mrapi.java 文件，mrapi.java 文件代码如下（该 rar 包中已提供 mrapi.java 文件）：

```
package    com.sscc.fdep;


public class mrapi
{
static
{
System.loadLibrary("mrapi");
}

public static void main(String[] args)
```

```
{

}

/********************************** FDEP V3
**********************************/

/* 初始化，返回为整形
 * return: 0-failed; 非0-OK
   App和AppPwd对应c接口中的AppId和AppPwd， 后四个参数对应C接口中的结构体
oConnInfo中各个字段
 */
public native static int MrInit(String App,String AppPwd, String Ip,short Port,String
Ipbak, short Portbak);



/* 发送消息 ，参数中SourceAppID填初始化的时候用的App
 * return:     ""-发送失败    pkgId-发送成功
   psPkg中是要发送的数据，其余字段同C接口中pMsgPropery中各个字段
 */

public native static String MrSend( byte[] psPkg, String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, String
ExpiredAbsTime, byte flag, byte Protocltype,int iMillSecTimeo);




//return "NULL"-接收失败      否则返回pkID(64byte) + CorrpkID(64byte) +
sourceUserID(64byte) + sourceAppID(64byte) +destUserID(64byte) +
UserData1(256byte) + UserData2(256byte) + data       上述前7个字段同C接口中
pMsgPropery中各个字段 ，data为真正的数据

public native static byte[] MrReceive1(String sAppID,String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, String
ExpiredAbsTime, byte flag, byte Protocltype,int iMillSecTimeo);



/* 判断与交换中枢的连接是否正常
 * return: 0-link_not_ok; 1-link_ok
```

```
    参数为mrinit中设置的app
 */
public native static int MrIsLinkOK(String sAppID);

/*释放资源
  参数为mrinit中设置的app*/
public native static void MrDestroy(String sAppID);


/********************************** FDEP V4
**********************************/

/* 初始化，返回为整形
 * return: 0-failed; 非0-OK
    App和AppPwd对应c接口中的AppId和AppPwd， 后四个参数对应C接口中的结构体
oConnInfo中各个字段
 */
public native static int Mr2Init(String App,String AppPwd, String Ip,short Port,String
Ipbak, short Portbak);


/* 发送消息 ，参数中SourceAppID填初始化的时候用的App
 * return:     ""-发送失败    pkgId-发送成功
    psPkg中是要发送的数据，其余字段同C接口中pMsgPropery中各个字段
 */
public native static String Mr2Send( byte[] psPkg, String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, String MsgType,
byte flag, byte BizType, byte Priority, byte SensitiveLevel, int iMillSecTimeo);


//return "NULL"-接收失败     否则返回pkID(64byte) + CorrpkID(64byte) +
sourceUserID(32byte) + sourceAppID(32byte) +destUserID(32byte) +
UserData1(256byte) + UserData2(256byte) + data       上述前7个字段同C接口中
pMsgPropery中各个字段 ，data为真正的数据
public native static byte[] Mr2Receive1(String sAppID,String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, int iMillSecTimeo);



/* 判断与交换中枢的连接是否正常
 * return: 0-link_not_ok; 1-link_ok
    参数为mrinit中设置的app
```

```
 */
public native static int Mr2IsLinkOK(String sAppID);

/*释放资源
  参数为mrinit中设置的app*/
public native static void Mr2Destroy(String sAppID);


/* 获取版本信息
 * return 版本号字符串
 */
public native static byte[]  Mr2GetVersion();


/* 创建包ID
 * return PkgID
 */
public native static byte[]  Mr2CreatePkgID(String szHandleAppID);

/* 获取对端用户是否在线状态
/ return: -1:函数调用失败  0:不在线     1:在线
*/
public native static int  Mr2GetPeerUserStat(String szHandleAppID,String
szPeerUserID);


/* 调用注册条件函数
 * return: 0：注册成功,<0其它失败
 */
public native static int  Mr2RegRecvCondition(String szHandleAppId,String
szSrcUserId, String szSrcAppId, String szDestUserId, String szDestAppId,
String szPkgId, String szCorrPkgId, String szUserData1,String szUserData2) ;


///  "NULL，errmsg"-接收失败    否则返回errcode(4byte)+pkID(64byte) +
CorrpkID(64byte) + sourceUserID(32byte) + sourceAppID(32byte)
+destUserID(32byte) + destAppID(32byte)+UserData1(256byte) +
UserData2(256byte) + data      上述前7个字段同C接口中pMsgPropery中各个字段 ，data
为真正的数据
///   其中errcode= "0000"为接收正常，如果errcode为非"0000"字符串，则由系统返回的错误
信息，详细错误在data字段中；
public native static byte[] Mr2Receive3(String sAppID,String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, int iMillSecTimeo);
```

```
/********************************** FDEP V5
**********************************/
/* 初始化，返回为整形
 * return: 0-failed; 1-OK
   App,usInstanceId和AppPwd对应c接口中的AppId, InstanceId和AppPwd， 后四个参数对应C接口中的结构体oConnInfo中各个字段
 */
public native static int Mr3Init(String sApp, short uInstanceId, String sAppPwd, String sIp, short uPort, String sIpbak, short uPortbak);


/* 发送消息 ，参数中SourceAppID填初始化的时候用的App
 * return:    ""-发送失败    pkgId-发送成功
   psPkg中是要发送的数据，其余字段同C接口中pMsgPropery中各个字段
 */
public native static String Mr3Send( byte[] psPkg, String sSourceUserID,String sSourceAppID, short uSrcInstanceId, String sDestUserID, String sDestAppID,
                        String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, String sMsgType, byte cFlag, byte cBizType, byte cPriority, byte cSensitiveLevel, int iMillSecTimeo);


/* 接收消息
 * return: 接收成功：pkID(64byte) +CorrpkID(64byte) +sourceUserID(32byte) +sourceAppID(32byte) +sourceIntanceId(16byte)
            +MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte) +SensitiveLevel(8byte)
            +UserData1(256byte) +UserData2(256byte) +data，上述前12个字段与输入的对应参数相同，data为真正的数据
            接收失败："NULL"或"NULL,errmsg"；
 */
public native static byte[] Mr3Receive1(String sAppID, short uHandleInstanceId,String sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,
                                        String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, int iMillSecTimeo);


/* 判断与交换中枢的连接是否正常
 * return: 0-link_not_ok; 1-link_ok
 参数为mrinit中设置的app和InstanceId
*/
```

```
public native static int Mr3IsLinkOK(String sAppID, short uHandleInstanceId);


/*释放资源
  参数为mrinit中设置的app和InstanceId
*/
public native static void Mr3Destroy(String sAppID, short uHandleInstanceId);


/* 获取版本信息
 * return 版本号字符串
*/
public native static byte[]  Mr3GetVersion();


/* 创建包ID
 * return PkgID
*/
public native static byte[]  Mr3CreatePkgID(String sHandleAppID, short
uHandleInstanceId);


/* 获取对端用户是否在线状态
    return: -1:函数调用失败  0:不在线     1:在线
*/
public native static int  Mr3GetPeerUserStat(String sHandleAppID, short
uHandleInstanceId,String sPeerUserID);


/* 调用注册条件函数
 * return: 0：注册成功,<0其它失败
*/
public native static int  Mr3RegRecvCondition(String sHandleAppId, short
uHandleInstanceId,String sSrcUserId, String sSrcAppId, String sDestUserId, String
sDestAppId,
                                                String sPkgId, String sCorrPkgId,
String sUserData1,String sUserData2);

/* 接收消息 含错误码
 * return:  接收成功：errcode(4byte) +pkID(64byte) +CorrpkID(64byte)
+sourceUserID(32byte) +sourceAppID(32byte) +sourceIntanceId(16byte)
                        +MsgType(8byte) +Flag(8byte) +BizType(8byte)
+Priority(8byte) +SensitiveLevel(8byte)
                        +UserData1(256byte) +UserData2(256byte) +data，其中12个
```

```
字段与输入的对应参数相同，data为真正的数据
                        其中errcode= "0000"为接收正常，如果errcode为非"0000"字符串，
则由系统返回的错误信息，详细错误在data字段中；
            接收失败:"NULL"或"NULL，errmsg";；
 */
public native static byte[] Mr3Receive3(String sAppID, short uHandleInstanceId,String
sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,
                                                String sPkgID, String sCorrPkgID, String
sUserData1, String sUserData2, int iMillSecTimeo);

/* 发送主题消息函数
 * return: PkgId, 空:发送失败；非空:发送成功
*/
public native static String Mr3SendTopicMsg( byte[] psPkg, String sSourceUserID,String
sSourceAppID, short uSrcInstanceId, String sTopicName,
                                        String sPkgID, String sCorrPkgID, String
sUserData1, String sUserData2, byte cFlag, byte cBizType, byte cPriority, byte
cSensitiveLevel);

/* 接收主题消息函数
            接收成功:  pkID(64byte) +CorrpkID(64byte) + topicName(17byte)
+sourceUserID(32byte) +sourceAppID(32byte) +sourceInstanceID(16byte)
                        +flag(8byte) +Priority(8byte) +SensitiveLevel(8byte)+
UserData1(256byte) + UserData2(256byte) +data      其中11个字段同C接口中
pMsgPropery中各个字段 ，data为真正的数据
            接收失败: "NULL"或"NULL，errmsg";
*/
public native static byte[] Mr3ReceiveTopicMsg(String sAppId, int iInstanceId, String
sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,String
sTopicName, String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2);

/*
 * 函数说明：获取App状态信息函数
 * return 查询单个app且只有一个实例，成功返回的状态信息：AppID(32byte)
+LinkID(32byte) +InstanceID(16byte) +IP(16byte)
 *          +Port(8byte) +MrName(32byte) +LoginTime(256byte)
+QueueLength(64byte) +TopicQueueLength(64byte)
 *          +FileQueueLength(64byte) +SendPkgCountTotal(64byte)
+RecvPkgCountTotal(64byte)
 *          失败："NULL"或"NULL,errmsg"
 * note：1、如果查询的是单个app且只有一个实例，则无竖线"|"分隔；如果查询全部app，则用
竖线分隔每个app每个实例的状态信息；
 *          2、sQueryAppId：查询的AppId。若填写AppId，则查询的是单个app的状态信息；若
不填写，查询所有全部app；
```

```
 *        3、参数sAppId和iInstanceId为Mr3Init中设置的参数，为必填项；
*/
public native static   byte[] Mr3GetAppStatus(String sAppId, int iInstanceId, String
sQueryAppId);
```

编写 mrapi.java 文件之后，使用 javac 命令编译该文件，在 fdep 目录下生成 mrapi.class 文件。

7.2.2 示例代码

生成 mrapi.class 文件之后，可通过 mrapi 类方法来使用本手册 4.3 章节介绍的各种函数。示例代码如下（rar 包中的 Demo.java 文件内已提供了该示例代码）：

```
/**
 * Created by sscc on 2020/1/13.
 */


import com.sscc.fdep.*;

import java.io.UnsupportedEncodingException;

public class MsgDemo extends Thread
{
    //主线程:初始化
    public static void main( String agrs[])
    {
        CfgParam oCfgParam = new CfgParam();
        run(oCfgParam);
        mrapi.Mr3Destroy(oCfgParam.m_ssAppID,
oCfgParam.m_usSourceInstanceID);
    };

    public static void run(CfgParam oCfgParam)
    {
         //调用Mr3Init初始化
        int iErr = mrapi.Mr3Init(oCfgParam.m_ssAppID,
oCfgParam.m_usSourceInstanceID, oCfgParam.m_ssAppPasswd,
oCfgParam.m_ssMrIP1, oCfgParam.m_usMrPort1, oCfgParam.m_ssMrIP2,
oCfgParam.m_usMrPort2);
        if (iErr == 0)
        {
            System.out.println("ERROR: Mr3Init failed, exit.");
            return;
```

```
        }
        else
        {
            System.out.println("Mr3Init OK.");
        }

        byte[] Version = mrapi.Mr3GetVersion();
        String sVersion = null;
        try {
            sVersion = new String(Version, "GBK");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        System.out.println("Mr3GetVersion[" + sVersion + "]");
        //启动发送线程
        Runnable RunSendThrd = new SendThrd(oCfgParam);
        Thread SndTd = new Thread( RunSendThrd );
        SndTd.start();
        //启动接收线程
        Runnable RunRecvThrd = new RecvThrd(oCfgParam);
        Thread RcvTd = new Thread( RunRecvThrd );
        RcvTd.start();

        while(true)
        {
            //每隔1s调用Mr3IsLinkOK判断AppID与mr连接是否正常：若不正常则报警，待网络
连接正常后，mrapi会自动重连上mr
            if (mrapi.Mr3IsLinkOK(oCfgParam.m_ssAppID,
oCfgParam.m_usSourceInstanceID) == 0)
            {
                System.out.println("WARN: Link not ok.");
            }

            try
            {
                sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

```
//配置参数
class CfgParam
{
    public String m_ssAppID;
    public String m_ssAppPasswd;
    public String m_ssMrIP1;
    public short   m_usMrPort1;
    public String m_ssMrIP2;
    public short   m_usMrPort2;
    public String m_ssSourceUserID;
    public short   m_usSourceInstanceID;
    public String m_ssDestUserID;
    public String m_ssDestAppID;
    public short   m_usDestInstanceID;
    public String m_ssTopicName;
    public String m_ssUserData1;
    public String m_ssUserData2;
    public byte m_ucFlag;
    public int    m_iBizType;

    public CfgParam()
    {
        m_ssAppID = "app1";
        m_ssAppPasswd = "1";
        m_ssMrIP1 = "10.10.218.64";
        m_usMrPort1 = 5852;
        m_ssMrIP2 = "10.10.218.64";
        m_usMrPort2 = 5852;
        m_ssSourceUserID = "FTCSTEST1021";
        m_usSourceInstanceID = 100;
        m_ssDestUserID = "FTCSTEST1020";
        m_ssDestAppID = "app1";
        m_usDestInstanceID = 0;
        m_ssTopicName = "1021sscc";
        m_ssUserData1 = "UserData1";
        m_ssUserData2 = "UserData2";
        m_ucFlag = 0;
        m_iBizType = 0;
    }
}

//发送线程
class SendThrd extends Thread
```

```
{
    private CfgParam m_oCfgParam;
    public static int m_iCount = 0;

    public static String bytesToString(byte[] b)
    {
        StringBuffer result = new StringBuffer("");
        int length = b.length;
        for (int i = 0; i < length; i++)
        {
            result.append((char)(b[i] & 0xff));
        }
        return result.toString();
    }

    public SendThrd(CfgParam oCfgParam)
    {
        this.m_oCfgParam = oCfgParam;
    }

    public void run()
    {
        while(true)
        {
            int iPeerStat = mrapi.Mr3GetPeerUserStat(m_oCfgParam.m_ssAppID,
m_oCfgParam.m_usSourceInstanceID, m_oCfgParam.m_ssDestUserID);
            System.out.println("PeerUser[" +m_oCfgParam.m_ssDestUserID +"]
Stat["+ iPeerStat + "]");

            byte[] szCreatePkgId =
mrapi.Mr3CreatePkgID(m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstance
ID);
            String ssCreatePkgId = bytesToString(szCreatePkgId);
            System.out.println("PkgId ["+ssCreatePkgId + "]");
            //发送的数据
            byte[] senddata = new byte[] { ('H'), ('E'), ('L'), ('L'), ('O') };
           //调用Mr3Send发送给接收方，返回值为PkgID
            String ssPkgID = mrapi.Mr3Send(senddata,
m_oCfgParam.m_ssSourceUserID,
m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstanceID,
m_oCfgParam.m_ssDestUserID, m_oCfgParam.m_ssDestAppID,
                    "", "", m_oCfgParam.m_ssUserData1,
m_oCfgParam.m_ssUserData2, "", m_oCfgParam.m_ucFlag,
m_oCfgParam.m_ucBizType, (byte)0,(byte)0,2000);
```

```
            System.out.println(ssPkgID);
            if (ssPkgID==null||ssPkgID.length()<=0)
            {
                System.out.println("ERROR: send failed: PkgID[],
src["+m_oCfgParam.m_ssSourceUserID+"."+m_oCfgParam.m_ssAppID+"],
dest["+m_oCfgParam.m_ssDestUserID+"."+m_oCfgParam.m_ssDestAppID+"]");
            }
            else
            {
                System.out.println("send ok: PkgID["+ssPkgID+"],
src["+m_oCfgParam.m_ssSourceUserID+"."+m_oCfgParam.m_ssAppID+"],
dest["+m_oCfgParam.m_ssDestUserID+"."+m_oCfgParam.m_ssDestAppID+"]");
            }
            //每隔1s发送一次
            try
            {
                sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}

//接收线程
class RecvThrd extends Thread
{
    public static String bytesToString(byte[] b)
    {
        StringBuffer result = new StringBuffer("");
        int length = b.length;
        for (int i = 0; i < length; i++)
        {
            result.append((char)(b[i] & 0xff));
        }
        return result.toString();
    }

    private CfgParam m_oCfgParam;

    public RecvThrd(CfgParam oCfgParam)
    {
```

```
        this.m_oCfgParam = oCfgParam;
    }

    public void run()
    {
        //int iRegRet =
mrapi.Mr3RegRecvCondition(m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceIns
tanceID, "", "", "", "", "", "", "", "") ;
        //System.out.println("Mr3RegRecvCondition=" + iRegRet);

        while(true)
        {
           //调用Mr3Receive1接收对方的发送过来的消息包
            byte[] recvdata =
mrapi.Mr3Receive1(m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstanceID,
"", "","", "", "", "", "","", 2000);
            String result = bytesToString(recvdata);
            if(!result.equals("NULL\0") && !result.contains("NULL"))
            {
                //接收到数据包
                //截取所需字段数据：recvdata = pkID(64byte) +CorrpkID(64byte)
+sourceUserID(32byte) +sourceAppID(32byte) +sourceIntanceId(16byte)
                //+MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte)
+SensitiveLevel(8byte)
                //+UserData1(256byte) +UserData2(256byte) +data
                int iStartIdx = 0;
                int iEndIdx = iStartIdx+64;
                String ssPkgID = result.substring(iStartIdx,iEndIdx).trim(); //pkID

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+64;
                String ssCorrPkgID = result.substring(iStartIdx,iEndIdx).trim();
//CorrpkID

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+32;
                String ssSourceUserID = result.substring(iStartIdx,iEndIdx).trim();
//sourceUserID

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+32;
                String ssSourceAppID = result.substring(iStartIdx,iEndIdx).trim();
//sourceAppID
```

```
                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+16;
                String ssIntanceId = result.substring(iStartIdx,iEndIdx).trim();
//sourceIntanceId

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+8;
                String ssMsgType = result.substring(iStartIdx,iEndIdx).trim();
//MsgType

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+8;
                String ssFlag = result.substring(iStartIdx,iEndIdx).trim(); //Flag

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+8;
                String ssBizType = result.substring(iStartIdx,iEndIdx).trim();
//BizType

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+8;
                String ssPriority = result.substring(iStartIdx,iEndIdx).trim(); //Priority

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+8;
                String ssSensitiveLevel = result.substring(iStartIdx,iEndIdx).trim();
//SensitiveLevel

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+256;
                String ssUserData1 = result.substring(iStartIdx,iEndIdx).trim();
//UserData1

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+256;
                String ssUserData2 = result.substring(iStartIdx,iEndIdx).trim();
//UserData2
                //接收的数据
                String ssData = result.substring(iEndIdx);

                System.out.println("recv ok: PkgContent[" + ssData + "]");
                System.out.println("pkgID[" + ssPkgID + "], src[" + ssSourceUserID +
"." + ssSourceAppID + "."+ssIntanceId+ "],  pkgLen[" + ssData.length() + "],
CorrPkgID[" + ssCorrPkgID + "], UserData1[" + ssUserData1 + "], UserData2[" +
```

```
ssUserData2 + "]");
                System.out.println("MsgType[" + ssMsgType + "], Flag[" + ssFlag + "],
BizType[" + ssBizType + "], Priority[" + ssPriority + "], SensitiveLevel[" +
ssSensitiveLevel + "]");
              //判断是请求还是应答
                if (ssCorrPkgID==null||ssCorrPkgID.length()<=0)
                {
                    //一般情况：若CorrPkgID为空，则说明该消息包是请求包；若不为空，则说
明该消息包是应答包。
                      //处理请求包:建议以下处理将消息包抛到业务处理线程进行处理。
                      //...


                      //构造应答包数据，并调用Mr3Send函数发送应答包。

                    byte[] PesponsePkg = new byte[] { ('R'), ('e'), ('s'), ('p'), ('o'), ('n'),
('s'), ('e'), (' '), ('p'), ('k'), ('g') };
                   //返回应答包：将应答包属性CorrPkgID的值赋值为对应请求包属性PkgID的
值。
                    mrapi.Mr3Send(PesponsePkg, m_oCfgParam.m_ssSourceUserID,
m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstanceID, ssSourceUserID,
ssSourceAppID,
                                  "", ssPkgID, ssUserData1, ssUserData2, "",
m_oCfgParam.m_ucFlag, m_oCfgParam.m_ucBizType, (byte)0,(byte)0,2000);
                }
                else
                {
//一般情况：若m_szCorrPkgID为空，则说明该消息包是请求包；若不为空，则说明该消息包是应
答包。
                      //处理应答包:建议以下处理将消息包抛到业务处理线程进行处理。
                      //...
                }
//继续接收下一个消息包
                continue;
            }

//没有数据包，则sleep一会
            try
            {
                sleep(20);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
```

```
            }
        }
    }
}
```

对 MsgDemo .java 文件编译并执行，可查看程序执行效果。

8 备份解决方案

深证通提供的金融数据交换平台的 API，称为 FDEAPI。FDEAPI 提供了一主一备两个 IP 地址和端口，当 API 连接其中一个 bsmr 软件不通时，会自动连接另外一个 bsmr 软件。通过这种方式，当 bsmr 软件发生故障时，用户的应用系统无需任何更改，就能连接到 bsmr 进行正常工作。

此外，金融数据交换平台消息传输系统的 API 主要功能就是发送和接收消息。对于发送消息，API 可以从任意一台机器发送出去；对于接收消息，金融数据交换平台 API 提供的是主动按照条件到 bsmr 去取的方式，可以只取自己需要的消息包。因此，金融数据交换平台的 API 是可以支持应用的负载均衡和多机并行处理的。但是，用户的应用系统是否确定支持负载均衡和多机并行处理，与软件开发商提供的程序有关，用户应当咨询调用 FDEAPI 的软件开发商了解相关技术细节。

9 注意事项

1、在 Windows 环境下 ini 目录需要和 mrapi.dll 放置在同一目录下；在非 Windows 平台环境下按照解包后的层次结构放置，若配置文件 mrapi.ini 不在 mrapi 动态链接库的同一级目录下，则需要设置环境变量“MRAPI_LOGCONF_PATH”为配置文件 mrapi.ini 的存放路径。如果配置文件不存在，则启动失败。

2、开发过程中，只能使用 MrXXXX 或 Mr2XXXX 或 Mr3XXXX 一类方法，不能将三者混着使用开发过程中，只能使用 MrXXXX 或 Mr2XXXX 或 Mr3XXXX 一类方法。

3、在非 Windows 平台环境下，配置环境变量，确保将项目工程目录加入 LD_LIBRARY_PATH 中。

10 常见问题

10.1 使用长连接

用户在使用 FDEAPI 的过程中，正确的用法是应当使用长连接。即一般只在程序的初始化部分调用 MrInit 函数(建立连接)，然后进行正常的收发包的操作，在程序退出时调用 MrDestroy。而不是每次来一个请求都调用 MrInit 连接一次，处理完成后释放。

10.2 关于 CorrPkgID 的使用约定

如果用户调用 MrSend 函数发送的是业务应答包，对于 STUMsgProperty 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')。

接收时，如果 STUMsgProperty 中的 m_szCorrPkgID 字段的值为空，则表示是一个请求包；否则表示为一个应答包。

10.3 如果是同步处理，则当发出请求后，需要循环等待接收

如果用户对于发往 FDEP 消息传输系统平台的包使用的是同步接收方式，即通过 MrSend 发送到 FDEP 消息传输系统平台后，然后等着接收应答的方式。对于这种处理方式，在开发时需要注意，发送完成后，如果立即接收，基本上可以肯定第一次是接收不到应答包的，因为应答包不可能象在本机运行一样这么快的速度就可以回来，必须等待一定时间才可以收到。其伪代码如下：

```
MrSend(); //发送

time_t    ltBeginSecond = time(NULL);    //取得当前时间秒数

while(1)

{

        int iRet = MrReceive();

        if(iRet==0) break;    //收到，OK
```

```
        //每次循环时，则计算当前时间减去开始接收时间，如果大于 20 秒，则表示超时
        if(time(NULL)- ltBeginSecond>20) break;
}
```

10.4 app1 和 app2 的约定

一般来说，用户接收请求的 appID 约定为 app1。自己使用任意一个 appID 发送请求包到对方的 app1，对方的 app1 收到请求后，进行处理，处理完成后，根据请求方的源 UserID 和 appID，将应答发回给请求方。

10.5 API 是线程安全的，但不是进程安全的

API 是线程安全的，但不是进程安全的。通过调用 MrInit 一旦创建了句柄，该句柄在调用 MrDestroy 之前不会再改变。同一个句柄可以在同一个进程内的多个不同线程间使用，但不能在不同进程间使用，例如在 Unix 下 fork 之后，在主进程中创建的句柄不能再使用。

深圳证券通信有限公司

2020 年 4 月