PROP 数据交换系统
用户接口说明书
（V1.0）

中国证券登记结算有限责任公司上海分公司

2009 年 4 月

版本修订历史

日期	版本	说明	作者
2009/04/02	1.0	发布	李栋良

目　录

第一章 PROP 数据交换系统概述

一 前言

1. 文档所涉内容及适用对象

本文档是中国证券登记结算有限责任公司上海分公司（以下简称中国结算上海分公司）PROP 数据交换系统的用户接口说明书，包括系统概述、消息交换及文件交换的功能说明以及消息交换应用编程接口的详细说明。

本文档的适用对象为：通过 PROP 数据交换系统进行数据交换的系统设计人员、软件开发人员。

2. 术语释义

- PROP：参与人远程操作平台（Participant Remote Operating Platform），是中国结算上海分公司面向各类市场参与人推出的技术平台。
- PROP 用户：接入 PROP 系统的用户，一个市场参与人可能拥有一个或多个 PROP 用户，PROP 用户通过 PROP 用户名唯一标识。
- PROP 操作员：一个 PROP 用户可以包括多个 PROP 操作员，PROP 操作员通过 PROP 操作员号标识，用户号在该 PROP 用户下唯一。
- PROP 服务域、服务名及服务类型：PROP 服务通过服务域、服务名和服务类型的组合唯一标识。服务域（16 位）用以标识不同的服务提供者，如中国结算上海分公司的服务域为“SSCCRC”；服务名（16 位）用以标识用一服务域下的不同服务大类，如证券账户类服务的服务名为“ZHZLGLXT”；服务类型（2 位，其中‘FF’为系统保留值）则用以表示具体的服务类型，如证券账户开户的服务类型为“08”。

二 系统概述

1. PROP数据交换系统的概念

PROP 数据交换系统是中国结算上海分公司推出的面向 PROP 用户的数据交换服务，通过该系统可以在两个 PROP 用户之间建立安全、高效的数据交换通道，数据交换的模式包括实时消息的传递及文件的传输。

可以使用 PROP 数据交换系统的用户包括：各类 PROP 用户，包括券商、资金结算银行、

托管银行、基金管理公司、部分上市公司（使用网关模式的用户）、其他特殊用户等。

2. PROP数据交换系统的组成

PROP 数据交换系统由两个子系统组成，即处理实时消息的 PROP 消息交换子系统(PMX)以及负责文件传输的 PROP 文件交换子系统（PFX）。系统结构图如图 1.1 所示：

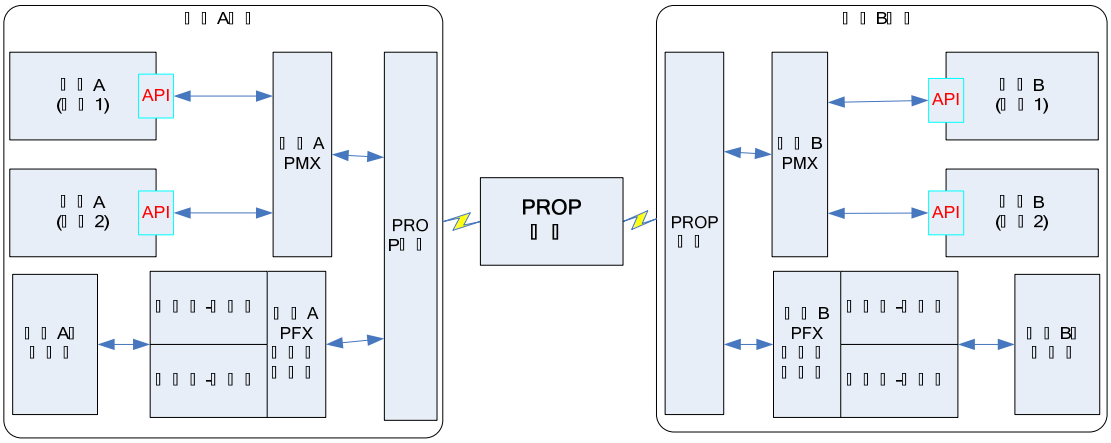


图 1.1 PROP 数据交换系统

PMX 是 PROP 消息交换子系统（PROP Message Exchange SubSystem）的缩写，是 PROP 系统客户端的一个应用模块。PROP 用户的应用系统可以通过 PMX 子系统提供的 API 在 PROP 用户之间完成各类实时消息的传递。消息的格式和内容由消息交换双方协商确定。

PFX 是 PROP 文件交换子系统（PROP File Exchange SubSystem）的缩写，也是 PROP 系统客户端的一个应用软件。PROP 用户可以通过 PFX 完成用户之间文件数据的交换。用户的文件应用和 PFX 之间为目录接口，即用户只需将文件存放在指定的目录即可完成向指定对象的文件发送；同样地，接收来自对方发送的文件存放在相应的目录下。文件交换双方协商确定文件的命名、格式、内容和传递约定等。

消息交换双方通过 PMX 进行消息交换时必须同时在线，PROP 系统并不暂存双方之间的实时消息。文件交换双方通过 PFX 进行文件交换时，不要求文件交换对方在线，PROP 系统暂存双方交换的数据文件。

PROP 数据交换系统具有丰富的监控功能，提供对 PFX 和 PMX 的运行状态、实时统计信息、交换用户的当前状态等的监控信息。

3. PROP数据交换系统的特点

PROP 数据交换系统是 PROP 系统的上层应用，具有如下特点和优势：

（1）快速建立用户间的数据交换通道。PROP 系统具有数量庞大的用户群，包括券商、资金结算行、托管银行（基金、QFII）、上市公司、基金管理公司、机构投资者以及其他各类特殊用户。PROP 用户之间如有数据交换的需要，只需通过本系统即可快速建立相应的数据交换通道，且无需申请新的通信链路。一个 PROP 用户可以与很多 PROP 用户建立数据交换通道，但通信链路只需要一条，即目前 PROP 系统使用的链路。

（2）安全、高效的数据交换能力。PROP 数据交换系统充分利用了 PROP 系统具有的高强度安全特性，保证了数据的安全和高效传输。

（3）扩展性强。PMX 具有灵活的横向扩展能力，用户可以同时部署多个 PMX 模块，以提高系统的整体处理能力。

第二章 PMX应用编程接口

一 PMX的应用模式

通过 PMX 可以实现两个 PROP 用户之间的消息交换，消息的格式和业务含义由消息交换双方共同确定。消息交换应用通过使用 PMX 提供的 API 实现与 PMX 系统的对接，并最终与对方用户建立消息交换通道，如下图所示。

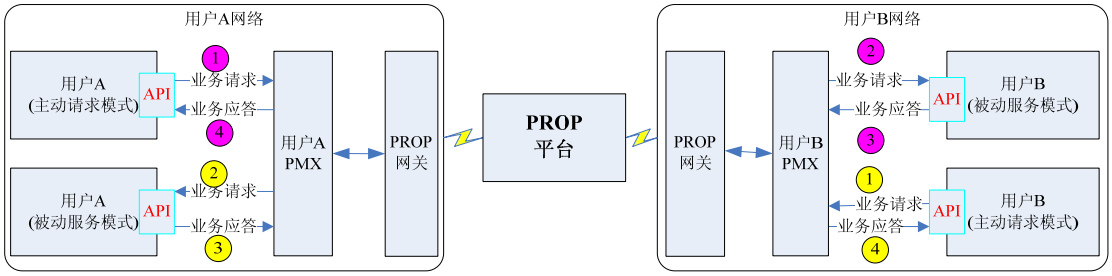


图 2.1 PMX 的应用模式

在消息传递应用的应用模式上，根据本方是否为请求的主动发起方可分为两类应用模式，即主动请求模式和被动服务模式，消息交换双方一般来说均需支持这两类模式。以第三方存管银证数据交换为例而言，券商通过主动请求模式向资金存管银行发起请求交易，银行采用被动服务模式接收来自券商发起的交易请求，并返回相应的应答。与此相对应的是，资金存管银行采用主动请求模式向券商发起交易请求，券商采用被动服务模式接收来自银行发起的交易请求并返回相应的应答。在图 2.1 中，当用户 A 为主动请求模式时首先向用户 B 发起业务请求并等待用户 B 返回应答，用户 B 则采用被动服务模式接收来自用户 A 的请求，并回应用户 A 的业务请求向用户 A 返回业务应答。

二 PMX系统的特点

1、请求应答模式

PMX 子系统采用传统的 C/S 一应一答模式，客户端只能发起业务请求并且接收服务端返回的应答，服务端只能被动的接收客户端发起的业务请求，并返回应答到客户端，服务端不能主动向客户端发送请求。

2、应用与 PMX 之间的连接模式

PMX 系统支持与应用之间的长连接模式和短连接模式，PROP 用户可以根据需要自由选择两种模式，当采用长连接模式时可以连续在一个连接上发起一个或者多个请求但必须等到所有的请求均有应答时才能关闭连接，否则该应答将丢失；当采用短连接模式时一个连接上只能处理一个请求一个应答，处理完应答后必须关闭连接，当有一个新的请求时必须重新建立一个连接。

三 PMX-API的两种编程模式

1、主动请求模式

当需要主动向某指定用户发送交易请求时需要使用主动请求模式的编程方式。主动请求模式不能取代被动服务模式，也就是说主动请求模式接收不到来自其它用户发送的请求消息，但能收到其它用户返回的应答消息。

2、被动服务模式

跟主动请求模式相对应是被动服务模式，被动服务模式可以接收到来自其它用户发送的业务请求消息。在被动服务模式应用不能主动发起请求交易。

3、两种模式的关系

消息交换双方在消息交换过程中必须分别采用主动请求模式合被动服务模式，互相配合才能完成正常的实时消息交换。

四 消息交换程序处理流程

使用 PMX-API 编程的主动请求模式和被动服务模式采用两种不同的流程进行处理，主动请求模式的程序处理流程如下：

1）初始化环境；

2）连接上 PMX；

3）发送交易请求：发送交易请求到被动服务模式方；

4）等待交易应答： 等待被动服务模式方的应答；

5）如果为长连接模式，则重复步骤 3、4，否则关闭连接。

被动服务模式流程如下：

1）初始化环境；

2）等待并接受来自 PMX 的连接；

3）等待请求：等待来自主动请求方发来的交易请求；

4）发送应答：发送应答消息到主动请求方；

5）关闭连接。

上述流程如下图所示：

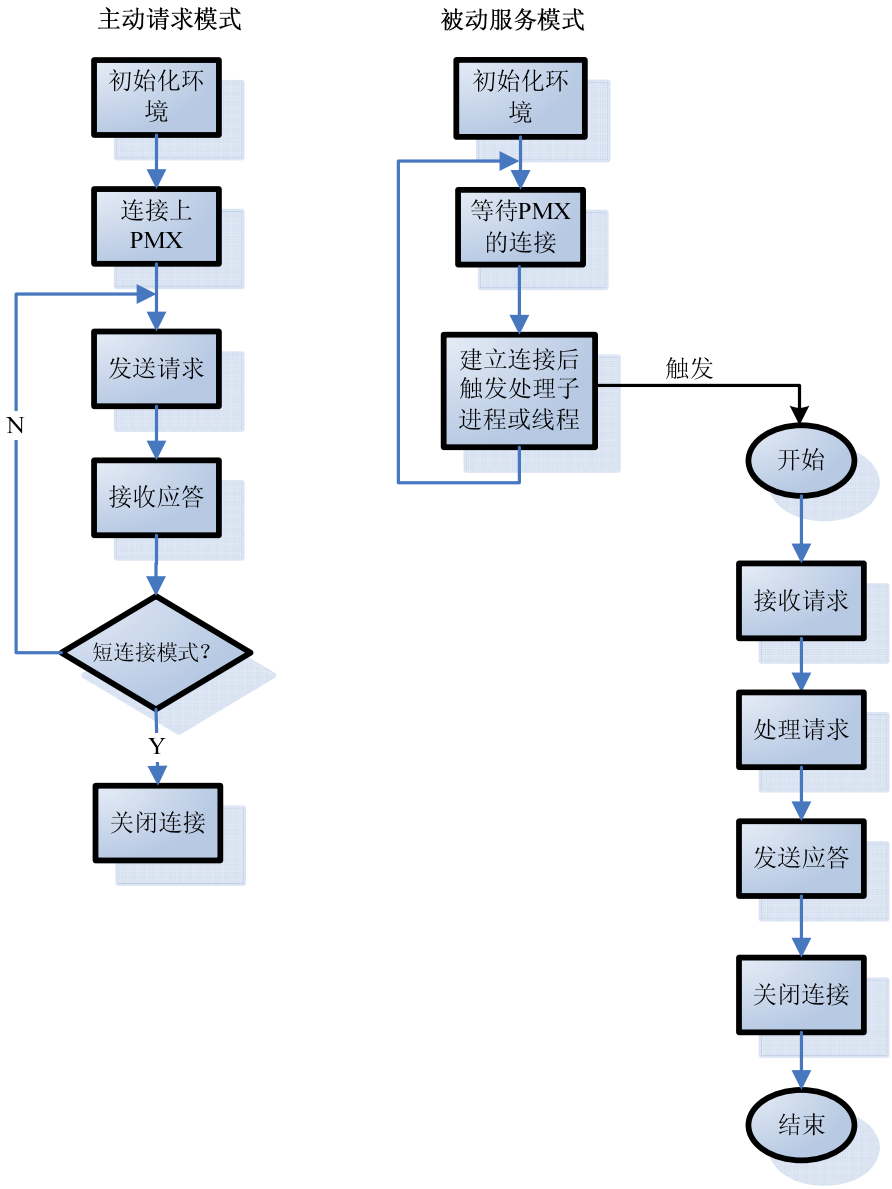


图 2.2 消息交换程序处理流程

五 消息寻址

消息发送者在每一次的消息交换中均要指定一个唯一的消息接收者以及相应的服务名称。消息用户名以 PROP 用户名标识，而服务名称则由中国结算上海分公司统一编制和管理，用来唯一标识一类业务。

PROP 用户名的长度为 8 位，大小写敏感。PROP 用户名在系统内唯一。部分 PROP 用户名在尾部包含了‘*’串，消息应用在指定这些用户时应去除尾部的‘*’串。如某 PROP 用户名为“Q12345**”，在指定该用户时应填写为“Q12345”。

第三章 PFX用户接口

一 PFX系统结构

PFX 是部署在 PROP 用户端的应用软件，它通过 PROP 网关与中国结算上海分公司相连。PFX 负责完成 PROP 用户之间的文件交换，用户和 PFX 之间采用目录接口的方式进行文件交换。PFX 将定期扫描指定的发件箱目录并把扫描到的文件发送到指定的接收用户。同样地，PFX 将接收到的文件存放到指定的目录中。PFX 的系统结构图如下所示。

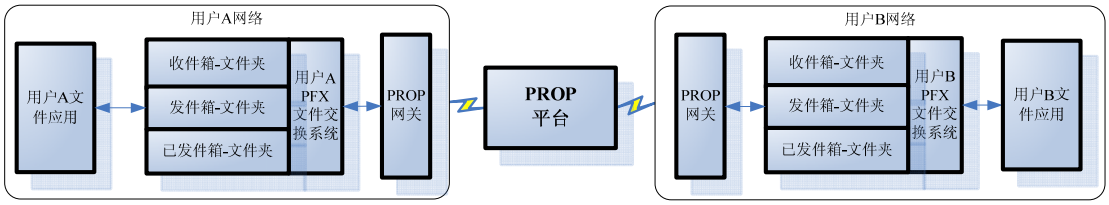


图 3.1 PFX 系统结构

二 PFX的功能

PFX 包括如下功能：

（1）文件的自动和手动发送/接收功能。

（2）可接收文件查询、取消已发送的文件等功能。

（3）系统配置功能，包括文件就绪规则配置、定时发送规则配置、文件收发目录配置以及文件交换用户配置等：

（4）文件状态的实时监控。

（5）日志查看及其他系统管理功能等。

三 文件目录接口

PFX 在文件交换目录（可配置）下根据不同的交换用户建立不同的用户目录，每个用户目录下又包括三个子目录（具体路径可配置），即收件箱、发件箱和已发件箱，如下图所示：

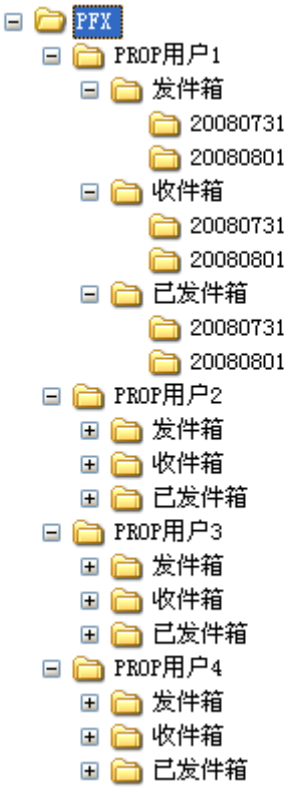


图 3.2 PFX 信箱目录结构

用户目录说明：

1．发件箱：本方发送至该用户的发送文件存放位置。以上图的 PROP 用户 1 目录为例，如本方需要向 PROP 用户 1 发送文件，则应在该目录的发件箱子目录中放置发送文件，PFX 将自动将该子目录下的文件发送到 PROP 用户 1。

2．收件箱：存放来自该 prop 用户发送的文件目录。以上图的 PROP 用户 1 目录为例，该目录下的收件箱子目录，存放了来自 PROP 用户 1 发送至本方文件。

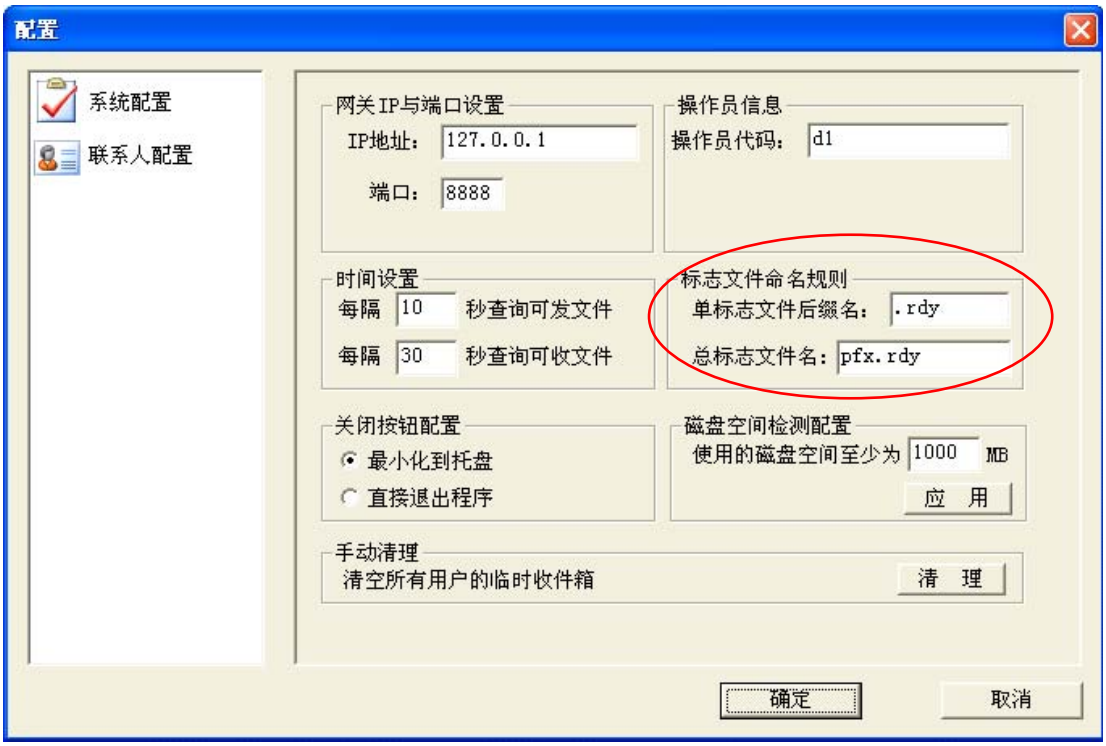
3．已发件箱：本方发送至该用户的文件发送完毕后，文件将从发件箱移动到已发件箱。因此本目录下存放的是已发送文件的备份。

此外，用户可以配置自动按照日期建立下级子目录，以方便用户的数据管理。

四 PFX文件就绪规则配置

所谓文件的就绪规则，是指 PFX 认定发件箱中的数据文件已处于就绪状态的判断规则。PFX 对处于就绪状态的文件进行自动发送。

用户可以选择两类文件就绪规则，即总标志文件规则和单个标志文件规则。如下图所示。



◆ 总标志文件就绪规则：总标志文件是一个发送清单文件，每行说明一个文件，每行的格式定义为：

文件名|文件大小

其中：

- ✧ 文件名：不定长，文件必须是存放在发件箱目录下的文件，相对路径；
- ✧ 文件大小：不定长，文件的字节数。

当 PFX 在发件箱目录下发现总标志文件存在时，根据清单文件的每一行的内容检查发件箱下的相应文件是否存在，文件大小是否一致。如通过检查即认为该文件就绪，该文件发送成功后即被移动到已发件箱下。当所有文件处理完毕后，发件箱目录下的总标志文件被删除，并在已发件箱目录下生成总标志文件的返回文件，其命名规则是总标志文件+“.ret”。返回文件描述了清单文件中每个文件的实际发送结果，每行一个文件，其格式定义为：

文件名|文件大小|结果代码|错误描述

其中：

- ✧ 文件名：不定长，同总标志文件；
- ✧ 文件大小：不定长，文件的字节数，同总标志文件；
- ✧ 结果代码：4 位定长，该值为‘0000’表示发送成功，其他表示发送失败；
- ✧ 结果说明：80 位定长，描述错误原因。

◆ 单个标志文件就绪规则：在发件箱中的任何一个数据文件如需 PFX 启动自动发送，则需要生成与其相应的标志文件，否则 PFX 不认定该文件处于就绪状态。标志文件的后缀可配置，标志文件名和数据文件名的关系是：标志文件名：=数据文件名+标志文件后缀。假设标志文件后缀配置为“.flag”，如数据文件名为 abc.txt，则其标志文件名为 abc.txt.flag。需要注意的是，必须避免标志文件名和数据文件名后缀相同的情况出现，如上例中，如有数据文名为“abc.flag”，则将引起混淆，PFX 将认定这是一个标志文件而非数据文件。在这种情况下，应该修改标志文件的配置，如修改为“*.flg”。

说明：数据文件名和标志文件名，其大小写不敏感。

五 文件交换对象

PROP 用户如需同别的 PROP 用户进行文件传输，需要在 PFX 中配置对方的 PROP 用户名，对方同样需要配置本方的 PROP 用户名。PFX 只发送已配置用户发件箱下的文件，也只接收来自已配置用户的文件。

PROP 用户名的长度为 8 位，大小写敏感。PROP 用户名在系统内唯一。部分 PROP 用户名在尾部包含了‘*’串，本方在配置这些用户时应去除尾部的‘*’串。如某 PROP 用户名为“Q12345**”，在配置该用户时应填写为“Q12345”。

六 PFX文件交换日志文件

PFX 在运行过程中生成文件交换的日志文件，便于用户的文件应用程序读取日志，以查看是否有文件到达或本方发送的文件是否发送完毕。

日志文件的存放路径根据用户的配置决定。发送日志文件和接收日志文件均每天生成，其命名规则为：

- 发送日志文件名：=pfxsndYYYYMMDD.log（YYYYMMDD 为 8 位日期）
- 接收日志文件名：=pfxrcvYYYYMMDD.log（YYYYMMDD 为 8 位日期）

日志文件的格式为文本文件，每行日志代表一个已发送完成或接收完成的文件，每行日志由几个数据域组成，数据域的长度不固定，各域之间由“|”作为分隔符。日志文件的格式如下：

序号	数据域名	说明
1	对方用户名	文件交换对方的 PROP 用户名，长度不定，但可能的最大长

		度为 8 位。
2	分隔符	“|”
3	文件序号	由中国结算上海分公司生成，系统内唯一，长度固定为 16 位。
4	分隔符	“|”
5	文件名	文件名，长度不定
6	分隔符	“|”
7	文件路径	文件存放路径，长度不定
8	分隔符	“|”
9	文件生成时间	文件生成时间，格式为“YYYYMMDDHHMMSS”。
10	分隔符	“|”
11	文件大小	文件字节数
12	换行符	回车换行符

第四章 PMX-API说明

一 头文件及链接库约定

（1）API 的头文件为 pmxapi.h，在该头文件中定义了公用的数据结构和常量定义。

（2）PMX 提供 WINDOWS 环境下和 AIX 环境下的链接库，链接库名为：

- WINDOWS 环境：动态链接库 pmxapi.dll；
- AIX 环境：动态链接库 libpmxapi.so。

二 API中使用的数据结构体说明

1、PROP_MessageInfo

PROP_MessageInfo 结构用来定义发送或接收消息的关键要素，每条消息均与一个 PROP_MessageInfo 结构体相关。

结构体定义：

```
typedef struct stru_PROP_MessageInfo
{
    char    strMsgType[2];
    char    strSender[8];
    char    strReceiver[8];
    char    strService[16];
    char    strServiceType[2];
    char    strMessageId[10];
    char    strTransId[16];
    char    strClientTransId[10];
    char    strDateTime[14];
    char    cSignFlag;
    char    strReserved[41];
} PROP_MessageInfo;
```

说明：

名称	说明
strMsgType	消息类型，00 表示请求，01 表示应答
strSender	消息的发送方
strReceiver	消息的接收方

strService	服务名
strServiceType	服务类型
strMessageId	请求消息发送方生成的消息序号
strTransId	PROP 通信中心生成的唯一交易流水号
strClientTransId	PMX 生成的唯一交易流水号
strDateTime	消息发送或接收的时间
cSignFlag	签名标志：‘1’-签名，‘0’-不签名，其他-不签名 本字段暂不启用。
strReserved	预留

2、PROP_ErrorInfo

结构体 PROP_ErrorInfo 包含了 PMX-API 函数调用失败后的错误描述信息。

结构体定义：

```
typedef struct stru_PROP_ErrorInfo
{
    char    iErrorCode[4];
    char    strErrorMsg[128];
} PROP_ErrorInfo;
```

说明：

名称	说明
iErrorCode	错误代码
strErrorMsg	出错信息

3、句柄类型说明

句柄类型	说明
PROP_ENV_HANDLE	环境句柄类型，PROP_InitEnv 函数的返回值类型及 PROP_FreeEnv 函数的输入参数类型。
PROP_CONNECT_HANDLE	连接句柄类型，PROP_Connect 及 PROP_Accept 函数的返回值类型。
PROP_LISTEN_HANDLE	侦听句柄类型，PROP_Listen 函数的返回值类型。
PROP_SOCKET_HANDLE	套接口句柄类型，PROP_Close 函数的输入参数类型。

备注：

连接句柄类型和侦听句柄类型实际上均为套接口句柄类型。

三 函数列表

以下所有函数均线程安全。

函数名称	说明
PROP_InitEnv	初始化 PROP 数据交换环境
PROP_Connect	建立和 PMX 系统的连接
PROP_Send	向指定用户发送消息
PROP_Recv	接收消息
PROP_Listen	开始服务侦听模式
PROP_Accept	接受一个连接请求
PROP_Close	关闭与 PMX 之间的连接
PROP_FreeEnv	释放 PROP 数据交换环境
PROP_Free	释放 PROP_Recv 分配的内存
PROP_Ping	检测到指定用户及指定服务是否可用
PROP_Select	检测给定的连接句柄是否可读或者可写
PROP_GetVersion	获取 PMX-API 的版本号

四　函数详细说明

1. 初始化环境函数PROP_InitEnv

功能描述：

初始化应用环境，应用程序向 PMX 进行签到操作。

函数原型：

```
PROP_ENV_HANDLE    PROP_InitEnv(const char *pstrPMXAddress,
                                unsigned short iPort,
                                const char *pstrUserName,
                                const char *pstrPassword,
                                PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
pstrPMXAddress [IN]	PMX 所在机器的 IP 地址。
iPort　[IN]	C-PMX 或 S-PMX 的端口号(需在 PMX 上进行配置)： 主动请求模式时，使用 C-PMX 端口号； 被动服务模式时，使用 S-PMX 端口号。
pstrUserName [IN]	PROP 系统操作员名。
pstrPassword [IN]	PROP 系统操作员密码。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结构空间。

返回值及出错处理：

成功返回 PROP 环境句柄，否则将返回 NULL，并在 pErrorInfo 结构中包含出错信息。常见错误代码及出错处理如下表所示：

错误代码	错误描述	错误原因	出错处理
L001	申请内存失败	系统内存资源不足	检查系统内存情况。
L002	连接 PMX 失败	（1）指定的 IP 地址或端口号不正确； （2）PMX 上的 C-PMX 或 S-PMX 服务未启动。	（1）使用正确的 IP 地址或端口号是否正确； （2）启动 C-PMX 或 S-PMX 服务。
L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。
C001	登录到 c-pmx 失败	操作员或密码错误	检查操作员或密码是否正确
S001	登录到 s-pmx 失败	操作员或密码错误	检查操作员或密码是否正确

使用约定：

应用程序在调用 PMX-API 的其它函数之前必须先调用本函数对环境进行初始化，并在程序结束之前调用 PROP_FreeEnv 释放相应的系统资源。同一进程可以调用该函数多次，每次返回一个新的 PROP 句柄（不会覆盖原有的 PROP 环境句柄）。

特别要注意的是，应用程序在调用本函数时应根据应用模式使用相应的 PMX 端口。当应用程序处于主动请求模式时，应连接 C-PMX 侦听端口；当应用程序处于被动服务模式时，应连接 S-PMX 侦听端口。

2. 连接PMX函数PROP_Connect

功能描述：

建立与 PMX 系统之间的会话。

函数原型：

```
PROP_CONNECT_HANDLE  PROP_Connect(PROP_ENV_HANDLE  hPropEnv,
                     PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
hPropEnv [IN]	PROP 环境句柄，填写调用 PROP_InitEnv 时返回的环境句柄。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结

	构空间。

返回值及出错处理：

调用成功时返回一个连接句柄。失败则返回 NULL 值，并在 pErrorInfo 结构中存放出错信息。常见错误代码及出错处理如下表所示：

错误代码	错误描述	错误原因	出错处理
L001	申请内存失败	系统内存资源不足	检查系统内存情况。
L002	连接 PMX 失败	（1）不正确的环境句柄 （2）PMX 上的 C-PMX 服务未启动。	（1）修正传入的句柄； （2）启动 C-PMX 服务。
L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。
C009	客户连接数过多	作为主动请求模式的连接数已超过 PMX 上的配置值	调整 PMX 上的 C-PMX 连接数配置值，或控制应用的并发连接数。

使用约定：

函数调用成功时返回一个连接句柄，以供 PROP_Send, PROP_Recv, PROP_Close 调用时使用。每次调用本函数均会创建一个与 PMX 之间新的会话，该会话在调用 PROP_Close 函数时结束。调用 PROP_Close 将结束会话并回收内存，PROP_Connect 和 PROP_Close 必须一一配对，每调用一次 PROP_Connect 均需要调用一次 PROP_Close 以释放资源，否则将造成内存泄露和 socket 资源泄漏。

3. 发送函数PROP_ Send

描述：

向指定的用户发送实时消息。

函数原型：

```
int PROP_Send(PROP_CONNECT_HANDLE    hConnect,
                    PROP_MessageInfo *pMsgInfo,
                    void *pMsgBody,
                    unsigned int iMsgLen,
                    int iMillSecTimeOut,
                    PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
hConnect [IN]	PROP_Connect 调用成功时返回的连接句柄。
pMsgInfo [IN]	本次发送的消息信息。 **（1）当发送的是请求消息时必填内容如下：** strMsgType：指定本消息类型为 00（请求）； strReceiver：指定本条消息的接收者（PROP 用户名）； strService：指定本条消息相关的服务名（如 DSFCG）； strServiceType：指定本消息的服务类型（对第三方存管服务固定填写“00”）； strMessageId：发送方消息序号； **（2）当发送的是应答信息时必填内容如下：** strMsgType：指定本消息类型为 01（应答）； strReceiver：指定本条消息的接收者，必须是该请求的发起方； strService：原样返回； strServiceType：原样返回； strMessageId：原样返回； strClientTransId：原样返回； strTransId：原样返回；
pMsgBody [IN]	消息体内容。
iMsgLen [IN]	消息长度（最大 1M），以字节为单位。
iMillSecTimeOut [IN]	发送时的超时时间，单位毫秒，当该值小于 0 时函数将无限等待直至返回。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结构空间。

返回值：

如果本函数调用成功将返回 0，超时时返回-9，出错时返回-1，并在 pErrorInfo 返回出错信息。常见错误代码及出错处理如下表所示：

错误代码	错误描述	错误原因	出错处理
L001	申请内存失败	系统内存资源不足	检查系统内存情况。
L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。
L004	发送失败	与 C-PMX 或 S-PMX 的连接已断开	检查 C-PMX 或 S-PMX 的运行状态
L006	发送超时	C-PMX 或 S-PMX 存在消息拥堵；或传入的超时参数值过小	检查 C-PMX 或 S-PMX 是否存在消息拥堵

使用约定：

请求消息发送方可以在 strMessageId 字段中填入本条消息的本地编号，该编号由用户自行编码、自行使用。应答消息发送方必须原样返回“strMessageId”、“strClientTransId”、

“strTransId”三个消息序号字段的内容，否则请求方将无法收到应答消息。

4. 接收函数PROP_Recv

描述：

接收发送给本方的实时消息。

函数原型：

```
int PROP_Recv(PROP_CONNECT_HANDLE    hConnect,
              PROP_MessageInfo *pMsgInfo,
              void **ppMsgBody,
              unsigned int *piMsgLen,
              int iMillSecTimeOut,
              PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
hConnect [IN]	PROP_Connect 调用成功时返回的连接句柄。
pMsgInfo [OUT]	本次接收到的消息信息。 **（1）当接收到的是请求消息时各成员描述如下：** strMsgType：本消息类型为 00,表示请求消息； strSender：本条消息的发送方； strReceiver：本条消息的接收方； strService：本条消息相关的服务名； strServiceType：本条消息相关的服务类型； strMessageId：本条消息的发送方消息序号； strTransId：本条消息的 PROP 交易流水号； strClientTransId：本条消息发送方的 PMX 流水号； strDateTime：本条消息的发送方发送时间； **（2）当接收到的是应答信息时各成员描述如下：** strMsgType：本消息类型为 01，表示应答消息； strSender：本条消息的发送方； strReceiver：本条消息的接收方； strService：本条消息的服务名； strServiceType：本条消息的服务类型； strMessageId：与本条应答消息对应的请求消息的发送方消息序号； strTransId：与本条应答消息对应的请求消息的 PROP 交易流水号； strClientTransId：与本条应答消息对应的请求消息的 PMX 流水号； strDateTime：本条消息的发送方发送时间；

ppMsgBody [OUT]	双指针，用来存放接收到的消息。
piMsgLen [OUT]	本次收到消息的长度。
iMillSecTimeOut [IN]	接收超时时间，单位毫秒，当该值小于 0 时表示函数将无限等待直至返回。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结构空间。

返回值：

如果本函数调用成功将返回 0，接收超时返回-9，PMX 主动关闭连接时返回-2，其他情况则返回-1。当返回值小于 0 时，在 pErrorInfo 返回出错信息。常见错误代码及出错处理如下表所示：

错误代码	错误描述	错误原因	出错处理
L001	申请内存失败	系统内存资源不足	检查系统内存情况。
L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。
L005	接收失败	与 C-PMX 或 S-PMX 的连接已断开	检查 C-PMX 或 S-PMX 的运行状态
L007	接收超时	C-PMX 或 S-PMX 存在消息拥堵；或传入的超时参数值过小	检查 C-PMX 或 S-PMX 是否存在消息拥堵
L00F	对方主动关闭连接	C-PMX 或 S-PMX 可能停止了服务	检查 C-PMX 或 S-PMX 是否已经停止了服务

使用约定：

调用者从 ppMsgBody 读取消息内容后，必须调用 PROP_Free 以释放 API 为接收本条消息而申请的内存资源。

特别说明：

在被动服务模式下，S-PMX 在接收完服务方发送的应答消息后将主动关闭连接，此时服务方如调用本函数再次接收请求，函数返回-2，错误代码为“L00F”，这是正常情况，服务方不应认为这是错误。

5. 侦听函数PROP_ Listen

描述：

在指定的地址和端口上侦听指定的服务。

函数原型：

```
PROP_LISTEN_HANDLE    PROP_Listen(PROP_ENV_HANDLE    hPropEnv,
                                  const char* pstrIpAddress,
                                  unsigned short iPort,
                                  const char* pstrServiceName,
                                  const char* pstrAcceptUser;
                                  PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
hPropEnv [IN]	调用 PROP_InitEnv 时返回的 PROP 环境句柄。
pstrIpAddress [IN]	侦听的 IP 地址，当该参数为 NULL 时，则侦听本机的所有地址。
iPort [IN]	本机侦听的端口号。
pstrServiceName [IN]	指定侦听的服务名。
pstrAcceptUser [IN]	指定可接受的请求用户（可使用本服务的 PROP 用户名），如该参数为 NULL，表示可接受所有的请求者。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结构空间。

返回值：

调用成功将返回侦听句柄，否则将返回 NULL，并在 pErrorInfo 结构中包含出错信息。常见错误代码及出错处理如下表所示：

错误代码	错误描述	错误原因	出错处理
L001	申请内存失败	系统内存资源不足	检查系统内存情况。
L002	连接 PMX 失败	PMX 上的 S-PMX 服务未启动。	启动 S-PMX 服务。
L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。
L004	发送失败	与 S-PMX 的连接已断开	检查 S-PMX 的运行状态。
L005	接收失败	与 S-PMX 的连接已断开	检查 S-PMX 的运行状态。
L00A	监听套接字出错	创建或监听套接字发生系统错误	分析系统日志，检查异常原因。
L00F	对方主动关闭 SOCKET 连接	使用了错误的环境句柄	检查环境句柄是否正确，作为被动服务模式工作时，初始化环境必须连接 S-PMX 的端口。

使用约定：

在程序退出时需要调用 PROP_Close 以关闭侦听句柄。

6. 接受连接函数PROP_ Accept

描述：

接受来自 PMX 的连接请求。

函数原型：

```
PROP_CONNECT_HANDLE   PROP_Accept(PROP_LISTEN_HANDLE   hListen,
                                  int iMillSecTimeOut,
                                  PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
hListen [IN]	调用 PROP_Listen 时返回的句柄。
iMillSecTimeOut [IN]	Accept 超时时间，当该值小于 0 时表示无限等待。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结构空间。

返回值：

成功将返回连接句柄，否则将返回 NULL，并在 pErrorInfo 结构中包含出错信息。常见错误代码及出错处理如下表所示：

错误代码	错误描述	错误原因	出错处理
L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。
L008	Accept 超时	N/A	N/A

使用约定：

调用该函数等待并接受来自 PMX 系统的连接请求，调用本函数之前必须先调用 PROP_Listen 函数以生成 hListen 句柄。不再使用连接句柄时必须调用 PROP_Close 以关闭句柄。

7. 关闭连接函数PROP_ Close

描述：

关闭连接。

函数原型：

```
void PROP_Close(PROP_SOCKET_HANDLE   hConnect);
```

参数说明：

名称	说明
hConnect [IN]	调用 PROP_Connect、PROP_Accept 或 PROP_Listen 时返回的句柄。

返回值：

无。

使用约定：

调用 PROP_Connect、PROP_Accept 和 PROP_Listen 后生成的句柄不再使用时必须调用本函数以释放套接口资源。

8. 释放环境函数PROP_ FreeEnv

描述：

释放由 PROP_InitEnv 申请的环境句柄。

函数原型：

```
void PROP_FreeEnv(PROP_ENV_HANDLE hPropEnv);
```

参数说明：

名称	说明
hPorpEnv [IN]	PROP_InitEnv 返回的句柄。

返回值：

无。

使用约定：

调用 PROP_InitEnv 生成的环境句柄不再使用时必须调用本函数以释放系统资源。

9. 释放函数PROP_ Free

描述：

释放指定的接收缓冲区。

函数原型：

```
void PROP_Free(void *pMsgBody);
```

参数说明：

名称	说明
pMsgBody [IN]	调用 PROP_Recv 后返回的消息体指针。

返回值：

无。

使用约定：

调用 PROP_Recv 后必须调用 PROP_Free 以释放指定的缓冲区资源。

10.检测函数PROP_ Ping

描述：

检测指定用户提供的指定服务当前是否可用。

函数原型：

```
int PROP_Ping(PROP_CONNECT_HANDLE    hConnect,
              const char *pstrReceiver,
              const char *pstrService,
              int iMillSecTimeOut,
              PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
hConnect [IN]	调用 PROP_Connect 后返回的句柄。
pstrReceiver [IN]	对方用户名称。
pstrService [IN]	对方用户提供的服务名称。
iMillSecTimeOut [IN]	超时时间单位：毫秒；如果小于 0 表示无限等待。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结构空间。

返回值：

成功返回 0，超时时返回-9，失败时将返回错误代码。常见错误代码及出错处理如下表所示：

错误代码	错误描述	错误原因	出错处理

L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。
L00D	PING 超时	超时值设置太小	调整超时值的大小
S006	服务未提供	（1）输入的对方用户名称或服务名称有误； （2）对方服务不可用	（1）检查输入参数 （2）联系对方，确认对方服务是否可用

使用约定：

本函数只能由服务请求方使用，服务提供者不能使用本函数。

11.连接句柄I/O状态检测函数PROP_ Select

描述：

检测指定的连接句柄组当前的 I/O 状态。

函数原型：

```
int PROP_Select(PROP_SOCKET_HANDLE*    pHandles,
                int    nHandleNum,
                int    nOperType,
                int    iMillSecTimeOut,
                PROP_ErrorInfo *pErrorInfo);
```

参数说明：

名称	说明
pHandles [IN]	调用 PROP_Connect 或 PORP_Accept 后返回的连接句柄组成的数组指针，至少有一个。
nHandleNum [IN]	参数 pHandles 数组中句柄的数量，必须大于等于 1。
nOperType [IN]	操作类型：0-检测是否可读。1-检测是否可写。2-检测是否有错误
iMillSecTimeOut [IN]	超时时间单位：毫秒；如果小于 0 表示无限等待。
pErrorInfo [OUT]	调用失败时返回的出错信息，由调用者负责申请该结构空间。

返回值：

成功返回 0～nHandleNum-1，表示的是某个符合检测条件的句柄在 pHandles 中的偏移量（从 0 开始），超时时返回-9，失败时返回-1，并在 pErrorInfo 结构中包含出错信息。常见错误代码及出错处理如下表所示：

错误代码	**错误描述**	**错误原因**	**出错处理**
L003	参数错误	检查传入的参数是否为空指针	修正传入的函数参数。

L009	Select 失败	句柄 socket 资源可能出错	检查与 C-PMX 或者 S-PMX 之间的连接是否断开
L010	Select 超时	超时值设置太小	调整超时值的大小

使用约定：

使用该函数之前需调用 PROP_Connect 或者 PROP_Accept 函数生成检测的信号句柄。

12.获取版本号函数PROP_ GetVersion

描述：

获取 PMX-API 的当前版本号。

函数原型：

```
void    PROP_GetVersion(char *pstrVersion);
```

参数说明：

名称	说明
pstrVersion [OUT]	API 的版本号，格式为： XX.XX.XX_YYYYMMDD；由调用者负责分配空间（不少于 18 字节）。

返回值：

无。

使用约定：

无。

附录 1：PMX应用示例程序

1. 客户端示例程序

```
/*
        PMX 系统客户端调用示例程序
*/
#include "pmxapi.h"
#include <stdio.h>
#include <string.h>
#ifndef   NULL
#define   NULL   0
#endif    //NULL

int main()
{
      PROP_ENV_HANDLE hProp;
      PROP_CONNECT_HANDLE hConnect;
      int iMillSecTimeOut;
      char strPMXAddress[20];
      unsigned short iPort;
      char strUserName[9];
      char strPassword[9];
      int ret;
      PROP_ErrorInfo ErrorInfo;
      PROP_MessageInfo MessageInfo;
      char MsgBuffer[1024];
      char *pRecvBuffer;
      unsigned int iMsgBodyLen;

      char szErrCode[5]={0};
      char szErrMsg[128]={0};
      char strSender[8+1]={0};     //消息的发送方
      char strReceiver[8+1]={0};   //消息的接收方
      char strService[16+1]={0};   //服务名
      char strServiceType[2+1]={0};     //服务类型
      char strMessageId[10+1]={0};      //请求消息发送方生成的消息序号
      char strPMX_MsgId[10+1]={0};  //PMX 交易序号
      char strTransId[16+1]={0};   //PROP 通迅中心生成的唯一交易流水号
      char strDateTime[14+1]={0};       //消息发送或接收的时间
      char* pRecvBufferx = 0;
```

```
    sprintf(strPMXAddress, "201.4.1.144"); //PMX 地址
    iPort = 6666;//PMX 端口号

    memcpy(strUserName, "99999999",8);// 网关操作员 Id
    memcpy(strPassword, " Pass#001",8); //网关操作员密码

    memset(&ErrorInfo,0,sizeof(PROP_ErrorInfo));
    hProp=PROP_InitEnv(strPMXAddress, iPort, strUserName, strPassword, &ErrorInfo);

    if ( NULL== hProp )
    {
        printf("初始化 PROP 环境失败!错误代码:%s,错误说明:%s\r\n",ErrorInfo.iErrorCode,
ErrorInfo.strErrorMsg);
        return -1;
    }

    memset(&ErrorInfo,0,sizeof(PROP_ErrorInfo));
    hConnect = PROP_Connect(hProp,&ErrorInfo);

    if (NULL == hConnect)
    {
        printf("连接 PMX 失败!错误代码: %s,错误说明:%s\r\n",ErrorInfo.iErrorCode,
ErrorInfo.strErrorMsg );
        PROP_FreeEnv(hProp);
        return -1;
    }
    memset(&MessageInfo,0,sizeof(PROP_MessageInfo));
    sprintf(MessageInfo.strMsgType, "%-2s","00");//请求交易
    sprintf(MessageInfo.strReceiver, "%-8s","Q54900  ");//交易的接收方
    sprintf(MessageInfo.strService, "%-16s","DSFCG");//三方存管业务
    sprintf (MessageInfo.strServiceType, "%-2s","00");//三方存管数据交换业务
    sprintf (MessageInfo.strMessageId, "%-10s","0000000001");//消息 ID 号必须唯一

    sprintf (MsgBuffer,"<xml><body>保证金存管业务</body></xml>");
    iMillSecTimeOut = 0;
    ret=PROP_Send(hConnect,&MessageInfo,&MsgBuffer,
    strlen(MsgBuffer),iMillSecTimeOut,&ErrorInfo);

    if(ret < 0)
    {
        printf("发送数据失败，错误代码：%-.4s,错误说明:%.128s\r\n",ErrorInfo.iErrorCode,
ErrorInfo.strErrorMsg);
        PROP_Close(hConnect);
        PROP_FreeEnv(hProp);
```

```
        return -1;
    }

    iMillSecTimeOut = -1;
    memset(&MessageInfo, 0, sizeof(PROP_MessageInfo));
   ret=PROP_Recv(hConnect, &MessageInfo,&pRecvBuffer,&iMsgBodyLen,iMillSecTimeOut,
&ErrorInfo);
    if(ret < 0)
    {
        memcpy(szErrCode, ErrorInfo.iErrorCode, 4);
        memcpy(szErrMsg, ErrorInfo.strErrorMsg, 128);
        printf("接收数据失败，错误代码：%s,错误说明:%s\r\n",szErrCode, szErrMsg);
        PROP_Close(hConnect);
        PROP_FreeEnv(hProp);
        return -1;
    }

    printf("接收数据成功！\r\n");
    memcpy(strSender, MessageInfo.strSender, 8);
    printf("收到数据，发送方：%s\r\n",strSender);
    memcpy(strReceiver, MessageInfo.strReceiver, 8);
    printf("          接收方：%s\r\n",strReceiver);
    memcpy(strService, MessageInfo.strService, 16);
    printf("          服务名称：%s\r\n",strService);
    memcpy(strServiceType, MessageInfo.strServiceType, 2);
    printf("          服务子类：%s\r\n",strServiceType);
    memcpy(strMessageId, MessageInfo.strMessageId, 10);
    printf("          消息序号：%s\r\n",strMessageId);
    memcpy(strPMX_MsgId, MessageInfo.strClientTransId, 10);
    printf("          PMX 消息号：%s\r\n",strPMX_MsgId);
    memcpy(strTransId, MessageInfo.strTransId, 16);
    printf("          PROP 消息号：%s\r\n",strTransId);
    memcpy(strDateTime, MessageInfo.strDateTime, 14);
    printf("          发送时间：%s\r\n",strDateTime);

    pRecvBufferx = malloc(iMsgBodyLen+1);
    memset(pRecvBufferx, 0, iMsgBodyLen+1);
    memcpy(pRecvBufferx, pRecvBuffer, iMsgBodyLen);
    printf("消息内容：\r\n%s\r\n 消息长度:%d\r\n",pRecvBufferx,iMsgBodyLen);
    free(pRecvBufferx);
    PROP_Free (pRecvBuffer);
    PROP_Close(hConnect);
    PROP_FreeEnv(hProp);
```

```
    getchar();
    return 0;
}
```

2. 服务端示例程序

```
/*
        PMX 系统服务端调用示例程序
*/

#include <stdio.h>
#include <process.h>
void       ChildProc(void *dummy);
#include "pmxapi.h"
#ifndef    NULL
#define    NULL    0
#endif     //NULL

int main()
{
    PROP_ENV_HANDLE    hProp;
    PROP_CONNECT_HANDLE hConnect;
    PROP_LISTEN_HANDLE hListen;
    int iMillSecTimeOut= 10;
    char strPMXAddress[20];
    unsigned short iPort;
    char strUserName[9];
    char strPassword[9];
    unsigned short iListenPort;

    strcpy (strPMXAddress, "201.4.1.144"); //PMX 地址
    iPort = 6668;              //PMX 端口号
    iListenPort = 5000;
    memcpy(strUserName, "99999999",8);//  网关操作员 Id
    memcpy(strPassword, " Pass#001",8); //网关操作员密码
    memset(&ErrorInfo,0,sizeof(PROP_ErrorInfo));
    hProp = PROP_InitEnv(strPMXAddress,iPort,strUserName, strPassword, &ErrorInfo);

    if(NULL == hProp)
    {
        printf("初始化 PROP 环境失败!错误代码:%s,错误说明:%s\r\n",ErrorInfo.iErrorCode,
ErrorInfo.strErrorMsg);
        return -1;
```

```
    }

    memset(&ErrorInfo,0,sizeof(PROP_ErrorInfo));
    hListen = PROP_Listen(hProp,(char*)NULL,iListenPort, "DSFCG",(char
*)NULL,&ErrorInfo);

    if(NULL == hListen)
    {
        printf(" 侦 听 失 败 ! 错 误 代 码 ： %s, 错 误 说 明 :%s\r\n",ErrorInfo.iErrorCode,
ErrorInfo.strErrorMsg );
        PROP_FreeEnv(hProp);
        return -1;
    }

    while(1)
    {
        memset(&ErrorInfo,0,sizeof(PROP_ErrorInfo));
        hConnect = PROP_Accept(hListen, iMillSecTimeOut, &ErrorInfo);
        if(NULL ==  hConnect)
        {
            //接收连接超时，表示没有连接
            if(memcmp(ErrorInfo.iErrorCode,PMXEACCEPTTIMEOUT,4)==0)
            {
                continue;
            }

            printf(" 接 受 PMX 连 接 失 败 ! 错 误 代 码 ： %s, 错 误 说
明:%s\r\n",ErrorInfo.iErrorCode, ErrorInfo.strErrorMsg );
            continue;
        }
        _beginthread(ChildProc,0, hConnect);
    }

    PROP_Close(hListen);
    PROP_FreeEnv(hProp);

    return 0
}

void ChildProc(void *dummy)
{
    PROP_CONNECT_HANDLE hConnect;
    int ret;
```

```
    PROP_ErrorInfo ErrorInfo;
    PROP_MessageInfo MessageInfo, SendMsgInfo;

    char MsgBuffer[1024];
    char *pRecvBuffer;
    unsigned int iMsgBodyLen;
    int iMillSecTimeOut =-1;
    char szErrCode[5]={0};
    char szErrMsg[128]={0};
    char strSender[8+1]={0};     //消息的发送方
    char strReceiver[8+1]={0};   //消息的接收方
    char strService[16+1]={0};   //服务名
    char strServiceType[2+1]={0};     //服务类型
    char strMessageId[10+1]={0};      //请求消息发送方生成的消息序号
    char strTransId[16+1]={0};   //PROP 通迅中心生成的唯一交易流水号
    char strDateTime[14+1]={0};       //消息发送或接收的时间
    char* pRecvBufferx = 0;

    hConnect = (PROP_CONNECT_HANDLE)dummy;
    memset(&ErrorInfo,0,sizeof(PROP_ErrorInfo));
    memset(&MessageInfo, 0, sizeof(PROP_MessageInfo));

   ret=PROP_Recv(hConnect, &MessageInfo,&pRecvBuffer,&iMsgBodyLen,iMillSecTimeOut,
&ErrorInfo);
    if(ret < 0)
    {
        printf("接收数据失败，错误代码：%s,错误说明:%s\r\n",ErrorInfo.iErrorCode,
ErrorInfo.strErrorMsg);
        PROP_Close(hConnect);
        return ;
    }
    printf("接收数据成功！\r\n");
    memcpy(strSender, MessageInfo.strSender, 8);
    printf("收到数据，发送方：%s\r\n",strSender);
    memcpy(strReceiver, MessageInfo.strReceiver, 8);
    printf("          接收方：%s\r\n",strReceiver);
    memcpy(strService, MessageInfo.strService, 16);
    printf("          服务名称：%s\r\n",strService);
    memcpy(strServiceType, MessageInfo.strServiceType, 2);
    printf("          服务子类：%s\r\n",strServiceType);
    memcpy(strMessageId, MessageInfo.strMessageId, 10);
    printf("          消息序号：%s\r\n",strMessageId);
    memcpy(strTransId, MessageInfo.strTransId, 16);
    printf("          发送时间：%s\r\n",strDateTime);
```

```
    pRecvBufferx = malloc(iMsgBodyLen+1);
    memset(pRecvBufferx, 0, iMsgBodyLen+1);
    memcpy(pRecvBufferx, pRecvBuffer, iMsgBodyLen);
    printf("消息内容：\r\n%s\r\n 消息长度:%d\r\n",pRecvBufferx,iMsgBodyLen);
    free(pRecvBufferx);

    PROP_Free (pRecvBuffer);

    memset(&SendMsgInfo,0,sizeof(PROP_MessageInfo));
    sprintf(SendMsgInfo.strMsgType,"01");//应答
    memcpy(SendMsgInfo.strReceiver,MessageInfo.strSender,sizeof(SendMsgInfo.strReceiver));
    //原样返回
    memcpy (SendMsgInfo.strService,MessageInfo.strService, sizeof(SendMsgInfo.strService));
    //三方存管业务
    memcpy(SendMsgInfo.strServiceType,MessageInfo.strServiceType,sizeof(SendMsgInfo.strS
erviceType));//三方存管数据交换业务
    memcpy(SendMsgInfo.strMessageId,MessageInfo.strMessageId,sizeof(MessageInfo.strMess
ageId));//客户端发送来的消息号
    memcpy(SendMsgInfo.strTransId,  MessageInfo.strTransId,sizeof(MessageInfo.strTransId));
//PROP 发送来的交易序号，必须原样返回
    memcpy(SendMsgInfo.strClientTransId,MessageInfo.strClientTransId,
sizeof(MessageInfo.strClientTransId)); //PMX 生成的交易序号，必须原样返回
    sprintf(MsgBuffer,"[xml][body]保证金存管业务应答[/body][/xml]");

    iMillSecTimeOut = 0;
    memset(&ErrorInfo,0,sizeof(PROP_ErrorInfo));

    ret = PROP_Send(hConnect, &SendMsgInfo, &MsgBuffer, strlen(MsgBuffer),
iMillSecTimeOut,&ErrorInfo);
    if(ret < 0)
    {
        printf("发送数据失败，错误代码：%s,错误说明:%s\r\n",ErrorInfo.iErrorCode,
ErrorInfo.strErrorMsg);

        PROP_Close(hConnect);
        return ;
    }
    PROP_Close(hConnect);
    return;
}
```

附录 2：PMX-API错误代码表

错误代码	错误描述
0010	现在不提供此类服务
0011	服务器忙
0012	服务器已关闭
0014	交易序列号非法
0015	无此权限
0016	签名错误
0017	非法操作员
0019	其他错误
0090	数据包格式错误
L001	申请内存失败
L002	连接 PMX 失败
L003	无效参数
L004	发送失败
L005	接收失败
L006	发送超时
L007	接收超时
L008	Accept 超时
L009	Select 出错
L00A	侦听套接字出错
L00B	接收套接字出错
L00C	本地令牌校验失败
L00D	PING 超时
L00E	setsockopt 出错
L00F	对方主动关闭 socket 连接
L010	select 超时
L011	线程互斥锁操作失败
P001	网络环境未初始化
P002	网络操作错误
P003	运行日志文件操作错误
P004	业务日志文件操作错误
P005	运行日志线程启动失败
P006	业务日志线程启动失败
C001	C_PMX:登录失败，用户名密码错误
C002	C_PMX:登录协议不合法
C003	C_PMX:服务请求方令牌校验失败
C004	C_PMX:系统资源不足

C005	C_PMX:网络环境未初始化
C006	C_PMX:网络操作错误
C007	C_PMX:PMX 交易序号池满
C008	C_PMX:信号量操作失败
C009	客户连接数过多
C010	C_PMX:不支持的交易子类型，目前只支持 00，01
C011	C_PMX:交易临时文件操作错误
C012	C_PMX:队列操作错误，如：队列满等
C013	C_PMX:PMX 交易序号错误，如：找不到等
C014	C_PMX:客户端管理 session 相关错误
C015	C_PMX:与网关之间的连接断开
C016	C_PMX:加密交易数据错误
C017	C_PMX:解密交易数据错误
C018	C_PMX:引擎初始化失败
C019	C_PMX:客户端管理启动失败
C020	C_PMX:网关端管理启动失败
C021	C_PMX:客户端交易请求报文大小超过最大报文(1M)
S001	S_PMX:登录失败，用户名密码错误
S002	S_PMX:登录协议不合法
S003	S_PMX:服务提供方令牌校验失败
S004	S_PMX:系统资源不足，如内存、信号量等
S005	S_PMX:连接失败指定的服务器失败
S006	S_PMX:服务未提供
S007	S_PMX:网络环境未初始化
S008	S_PMX:网络操作错误
S009	S_PMX:信号量操作失败
S010	S_PMX:不支持的交易子类型，目前只支持 00，01
S011	S_PMX:交易临时文件操作错误
S012	S_PMX:队列操作错误，如队列满等
S013	S_PMX:与网关之间的连接断开
S014	S_PMX:加密交易数据错误
S015	S_PMX:解密交易数据错误
S016	S_PMX:引擎初始化失败
S017	S_PMX:服务管理启动失败
S018	S_PMX:网关管理启动失败
S019	S_PMX:环境管理启动失败
S020	S_PMX:服务申明失败
S021	S_PMX:令牌服务申明文件操作失败(打开、读写等)
S022	S_PMX:环境管理 session 错误
S023	S_PMX:服务管理 session 错误
S024	S_PMX:网关管理 session 错误
S025	S_PMX:服务端交易应答报文大小超过最大报文(1M)