IP Messenger 是一款局域网内部聊天、文件传输工具，具有很多优点，如数据通讯不需要建立服务器、直接在两台电脑间通信和数据传输，支持文件及文件目录的传输，安全快捷以及小巧方便等优异特点，因此很多公司都采用它作为部门、公司内部的 IM 即时通信工具。

IP Messenger 在程序结构方面采用了 Windows SDK 处理结构，通信方面采用了 TCP/UDP 通信方式，在文件传输处理方面采用文件映射技术，等等。通过分析 IP Messenger 的运行、工作原理，可以提高并加深对 Windows 处理流程的理解，提高 SOCKET 编程技术等，因此特对其源码进行分析，以抛砖引玉，共同提高大家的编程技术。

1、IP Messenger 源代码的下载

在写这篇文章时，IP Messenger 的最新版本是 2.06，因此大家在下载时尽量选择最新版本下载。IP Messenger 源代码的下载地址是 http://www.ipmsg.org/，在网站的右上角，点击 English page，网站转换到英文界面，网站有英文版以及其它语言的版本，当然还有中文版的链接（http://www.azhi.net/IPMsg/），建议大家尽量下载原版的英文版源代码，以利于学习。

2、IP Messenger 源代码的目录及文件

IP Messenger 源代码的目录结构及文件详见列表 1.1 IP Messenger 源代码主要的目录及文件：

列表 1.1 IP Messenger 源代码主要的目录及文件

| 属性 | 名称 | 描述 |
|---|---|---|
| 目录 | SRC | 工程文件源代码和 INSTALL 目录 |
| 文件 | IPMSG.MAK | VC4 及以前版本使用的工程文件，用来指定如何建立一个工程，VC6 把 MAK 文件转换成 DSP 文件来处理 |
| | IPMSG.MDP | 后缀 MDP（Microsoft DevStudio Project 的简称）是老版本的项目文件，相当于 DSP 文件 |
| | PROT-ENG.TXT | 英文版的协议描述 |
| | PROTOCOL.TXT | 日文版的协议描述 |
| | README.TXT | 英文版的 README |
| | README-J.TXT | 日文版的 README |

IP Messenger 的 SRC 目录内容描述在列表 1.2，SRC 目录及主要文件。

| 属性 | 名称 | 描述 |
|---|---|---|
| 目录 | INSTALL | 安装程序源代码目录 |

| | | |
|---|---|---|
| 文件 | BLOWFISH.CPP | 加密算法源文件 |
| | BLOWFISH.H | 加密算法头文件 |
| | BLOWFISH.H2 | 加密算法随机数头文件 |
| | CFG.CPP | 系统配置源文件 |
| | IPMSG.CPP | IPMSG 的 APP 源文件 |
| | IPMSG.H | IPMSG 头文件 |
| | LOGDLG.CPP | 日志 DLG 源文件 |
| | LOGMNG.CPP | 日志处理源文件 |
| | MAINWIN.CPP | IP Messenger 主窗口源文件 |
| | MISCDLG.CPP | 消息 DLG，关于 DLG，以及控件子类化等源文件 |
| | MSGMNG.CPP | Socket 管理源文件 |
| | PLUGIN.CPP | Dll plugin 加载源文件 |
| | RECVDLG.CPP | 接收消息处理源文件 |
| | SENDDLG.CPP | 发送消息处理源文件 |
| | SETUPDLG.CPP | IP Messenger 属性配置源文件 |
| | SHARE.CPP | 文件传输接收管理源文件 |
| | TAPP.CPP | 应用程序类源文件 |
| | TDLG.CPP | DLG 类源文件 |
| | TLIST.CPP | 链表类源文件 |
| | TREGIST.CPP | 注册表操作类源文件 |
| | TWIN.CPP | 窗口类源文件 |
| | MSGSTR.H | 字符串常量头文件 |
| | TLIB.H | 类、结构等头文件 |

另外在 SRC 目录下，还有一个 INSTALL 目录，该目录中的文件是 IP Messenger 安装程序的源代码，其主要文件描述在列表 1.3 INSTALL 目录主要文件。

列表 1.3 INSTALL 目录主要文件

| 属性 | 名称 | 描述 |
|---|---|---|
| 文件 | INSTALL.CPP | 安装程序 |

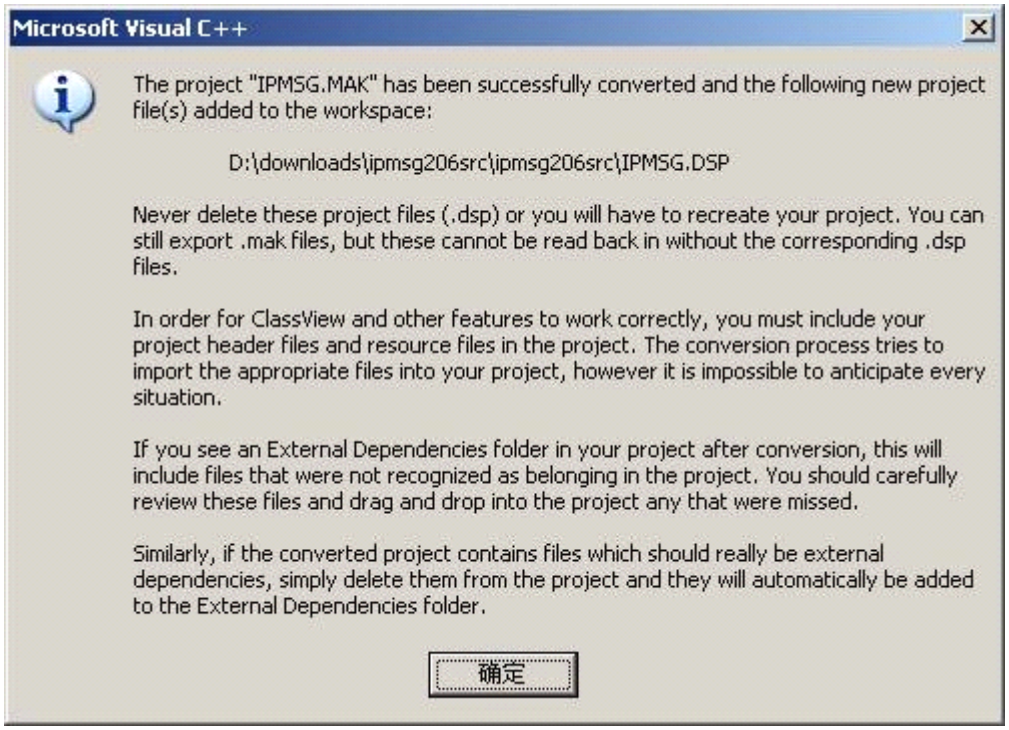
| | | |
|---|---|---|
| | | 源文件 |
| | INSTALL.H | 安装程序头文件 |

以上是 IP Messenger 的主要目录及其主要源文件的描述，通过列表我们可以清晰看出 IP Messenger 的功能分布。在以下的详细分析中，我们主要围绕着这些源文件而展开。

3、IP Messenger 源代码的工程转换

由于 IP Messenger 是使用以前版本的 VC 编写的，因此在打开工程文件时，需要转换该工程文件为 VC6 版本的工程文件。
双击打开 IPMSG.MDP 文件，系统提示转换 IPMSG.MAK 工程配置文件到 VC6 下的工程文件，如下图所示。


单击 “是”按钮，确定将工程转换成 VC6 的工程。系统提示将 MAK 工程文件转换成 DSP 工程文件，如下图所示。

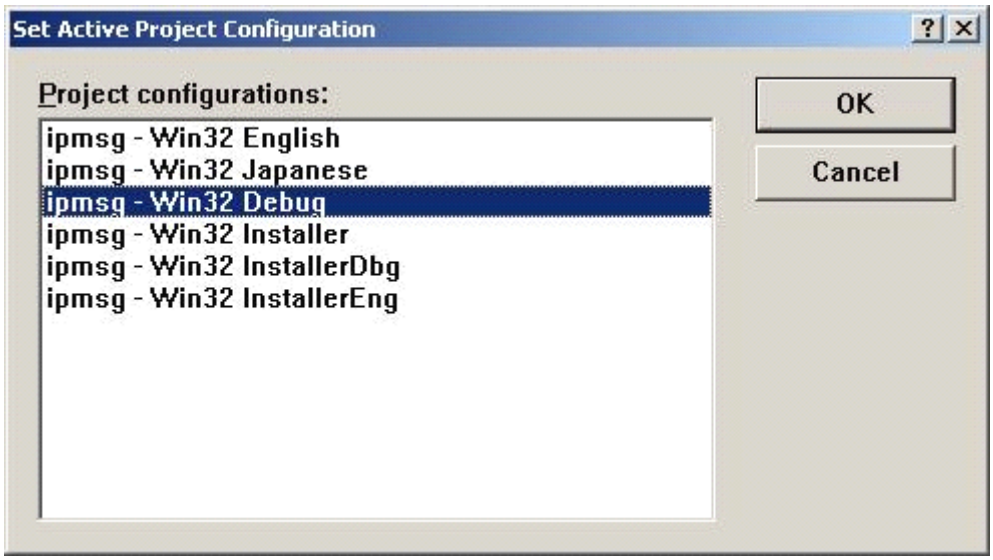



单击“确定”按钮，系统根据 MAK 文件生成 IPMSG.DSP 文件，以及 VC6 下的 IPMSG.DSW 文件，另外系统还生成了 IPMSG.NCB 和 IPMSG.OPT 两个文件。

4、IP Messenger 的工程配置

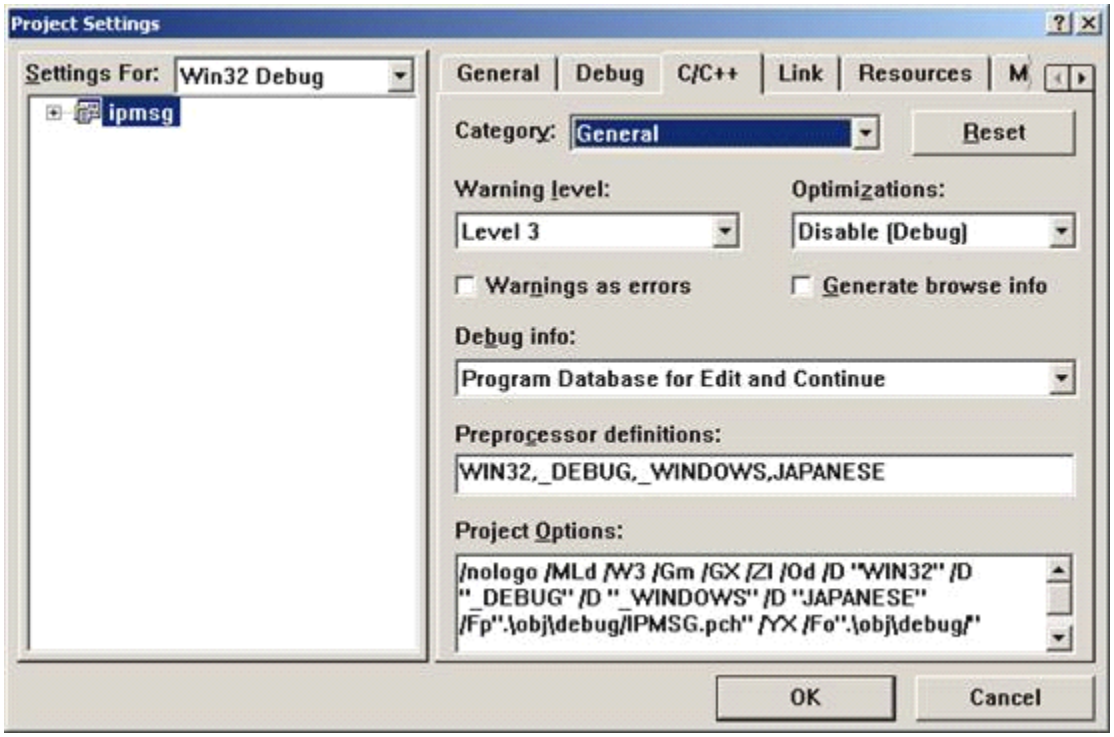
IP Messenger 的配置和目前 VC6 的工程配置有些不同，其安装工程文件和工程文件在混合在一起，不像 VC6 工程文件，不同的工程在不同的项目下，分别进行配置。

单击“Build”菜单，选择“Set Active Configuration…”，弹出“Set Active Project Configuration”对话框，如下图所示。



IP Messenger 的工程配置有 6 个，分别是 Release 的英文、日文以及 Debug 的 ipmsg 工程和安装工程。通过选择不同的配置，可以编译出不同的工程。根据需要，我们选择 ipmsg-Win32 Debug 配置作为当前活动工程，此编译配置可以编译出 Debug 版的 ipmsg。

单击“Project”菜单，选择“Settings…”，弹出“Project Settings”对话框，选择“C/C++”标签栏，如下图所示。




可以看到，在“Preprocessor definition：”中，工程使用了“JAPANESE”宏定义，也就是说，编译后的 Debug 版的 ipmsg 是日文版。当然，可以根据需要，修改此宏为“ENGLISH”，就可以编译出 Debug 版的英文 ipmsg。

5、系统概述完成上述操作之后，就可以使用 VC++ 6 来编译、调试 IP Messenger。

当然，在以后的分析中，我们将分析 ipmsg 的安装工程，那么就应当选择 ipmsg-Win32 InstallerDbg 配置作为当前活动工程，编译出 Debug 版的安装工程。

（转）**IPMSG 飞鸽传书——关于对话框**
2010-05-09 11:06


TWin 是所有窗口的父类，TDlg 是对话框子类，处理了 WM_INITDILOG 等消息，

创建过程：首先 aboutDlg = new TAboutDlg；然后 Create（），最后 Show（）；

Create（）是 TDlg 类的函数，首先把当前的 TWin 对象 aboutDlg 的指针加入到 wndArray 中，然后调用 SDK 函数

CreateDialog 来完成对话框的创建，然后进入 TApp 的 WinProc，调用 TDlg 的 WinProc 函数来处理消息，

发送 WM_INITDIALOG 消息，调用 TAboutDlg 类的 EvCreate 函数来处理，

接着是 Show()函数，发送 WM_NCPAINT 消息，接着发送 WM_PAINT 消息,后面是 CTLCOLOR 消息，调用

EventCtlColor（）函数来处理（5 次）。后面一直发送 WM_NCPAINT 消息；

接下来则是 TAboutDlg 来处理按钮消息了，按下按钮后会访问指定的网页。处理 EvCommand 消息，

访问 IPMSGURL_MSGSTR 指定的网页，

在 msgStr。h 文件中有定义：#define IPMSGURL_MSGSTR "http://www.ipmsg.org/index.html.en"

TAboutDlg 是关于对话框，这里仅分析 TDlg 类和 TAboutDlg 类的代码：

```
//对话框类，TWin 的子类

class TDlg : public TWin
{

protected:

LPSTR resName;//资源的名字

UINT resId;//资源的 id
```

```
BOOL modalFlg;//模式对话框还是非模式对话框

public:
TDlg(LPSTR _name, TWin *_parent = NULL);

TDlg(UINT resid, TWin *_parent = NULL);
virtual ~TDlg();

virtual BOOL Create(HINSTANCE hI = NULL);
virtual void Destroy(void);
virtual int    Exec(void);
virtual void EndDialog(int);

UINT     ResId(void)
{      return resId;
}

virtual BOOL EvCreate(LPARAM lParam);

virtual BOOL EvCommand(WORD wNotifyCode, WORD wID, LPARAM hwndCtl);

virtual BOOL EvSysCommand(WPARAM uCmdType, POINTS pos);

virtual BOOL EvQueryOpen(void);

virtual BOOL PreProcMsg(MSG *msg);

virtual LRESULT WinProc(UINT uMsg, WPARAM wParam, LPARAM lParam);
};
```

TDlg 的实现文件：

```
static char *tdlg_id =
"@(#)Copyright (C) H.Shirouzu 1996-2001    tdlg.cpp Ver0.95";
/*
========================================================================
Project Name     : Win32 Lightweight Class Library Test
Module Name      : Dialog Class
Create       : 1996-06-01(Sat)
Update       : 2001-12-06(Thu)
Copyright      : H.Shirouzu
Reference      :
========================================================================
*/
```

```
#include "tlib.h"

TDlg::TDlg(LPSTR _resName, TWin *_parent) : TWin(_parent)
{
// 将字符串_resName 复制到 resname 中

resName = strdup( _resName );

resId = 0;

modalFlg = FALSE;//默认的都是非模式对话框
}

TDlg::TDlg(UINT _resId, TWin *_parent) : TWin(_parent)
{
resName = NULL;

resId = _resId;

modalFlg = FALSE;//默认的都是非模式对话框
}

TDlg::~TDlg() // 调用 EndDialog 方法来关闭对话框
{
if ( hWnd )
{
   EndDialog(FALSE);
}

//释放 resName 资源

if ( resName )
{
   free( resName );
}

}

BOOL TDlg::Create(HINSTANCE hInstance)
{
//把 preWnd 指向当前窗口,这里并没有加入 wndarray，因为这时的 hwnd 还没有建立起来

TApp::AddWin( this );

/*
```

The CreateDialog macro creates a modeless
dialog box from a dialog box template resource.
The CreateDialog macro uses the CreateDialogParam function.

HWND CreateDialog(
HINSTANCE hInstance, // handle to module
LPCTSTR lpTemplate,  // dialog box template name
HWND hWndParent,  // handle to owner window
DLGPROC lpDialogFunc // dialog box procedure
);

Parameters
hInstance
[in] Handle to the module whose executable file contains
the dialog box template.
lpTemplate
[in] Specifies the dialog box template.
This parameter is either the pointer to a null-terminated
character string that specifies the name of the dialog box
template or an integer value that specifies the resource
identifier of the dialog box template. If the parameter
specifies a resource identifier, its high-order word must
be zero and its low-order word must contain the identifier.
You can use the MAKEINTRESOURCE macro to create this value.
hWndParent
[in] Handle to the window that owns the dialog box.
lpDialogFunc
[in] Pointer to the dialog box procedure.
Return Values
If the function succeeds, the return value is the handle

to the dialog box.

If the function fails, the return value is NULL.

注意这里的 addwin 只是简单的将 prewnd 指向 win，如果创建失败我们

在 DelWin 中只需要简单的将 prewnd 置空就行了，不用从链表中删除，

那样太慢

创建对话框的 hWnd 窗口!!!

*/

```
if ( ( hWnd = ::CreateDialog( hInstance ? hInstance : TApp::hI,
   resId ? (LPCSTR)resId : resName, parent ? parent->hWnd : NULL,
   (DLGPROC)TApp::WinProc) ) == NULL )
{
   return TApp::DelWin(this), FALSE;
}
else
{
   return TRUE;
}
}

int TDlg::Exec(void)
{
TApp::AddWin(this);

modalFlg = TRUE;//默认的是 modalFlg

/*
The DialogBox macro creates a modal dialog box
from a dialog box template resource. DialogBox does
not return control until the specified callback function
terminates the modal dialog box by calling the EndDialog
function. The DialogBox macro uses the DialogBoxParam function.

INT_PTR DialogBox(
HINSTANCE hInstance, // handle to module
LPCTSTR lpTemplate,    // dialog box template
HWND hWndParent,       // handle to owner window
DLGPROC lpDialogFunc // dialog box procedure
);

Parameters
hInstance
[in] Handle to the module whose executable file contains
the dialog box template.
lpTemplate
[in] Specifies the dialog box template. This parameter
is either the pointer to a null-terminated character
string that specifies the name of the dialog box
template or an integer value that specifies the resource
identifier of the dialog box template. If the parameter
specifies a resource identifier, its high-order word must be
zero and its low-order word must contain the identifier.
```

You can use the MAKEINTRESOURCE macro to create this value.
hWndParent
[in] Handle to the window that owns the dialog box.
lpDialogFunc
[in] Pointer to the dialog box procedure. For more information
about the dialog box procedure, see DialogProc.

Return Values
If the function succeeds, the return value is the nResult
parameter in the call to the EndDialog function used to terminate
the dialog box.

If the function fails because the hWndParent parameter is invalid,
the return value is zero. The function returns zero in this case
for compatibility with previous versions of Windows. If the
function fails for any other reason, the return value is –1.

创建一个模态对话框！！！这里把消息循环设置为 TApp::WinProc，以后的消息

都交给各个 dlg 的 EVCommand 函数来处理了，EndDialog 返回，参数指定了返回值

也就是这里的 result

*/

```
int result = ::DialogBox(TApp::hI, resId ? (LPCSTR)resId : resName,
    parent ? parent->hWnd : NULL, (DLGPROC)TApp::WinProc);

modalFlg = FALSE;

return result;
}

void TDlg::Destroy(void)
{
EndDialog(FALSE);
}

LRESULT TDlg::WinProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
LRESULT result = 0;

switch (uMsg)
{
```

```
case WM_INITDIALOG://create hwnd 完毕后发送 WM_INITDIALOG
    //完成其他的创建，这里 dlg 类处理了 initdialog 消息
    return EvCreate(lParam);
case WM_CLOSE:
    EvClose();
    return 0;
case WM_COMMAND: //  处理控件消息
    EvCommand(HIWORD(wParam), LOWORD(wParam), lParam);
    return 0;
case WM_SYSCOMMAND://处理系统菜单消息，比如右键菜单命令
    EvSysCommand(wParam, MAKEPOINTS(lParam));
    return 0;
case WM_TIMER://处理 timer 消息
    EvTimer(wParam, (TIMERPROC)lParam);
    return 0;
case WM_NCDESTROY://释放资源，在 wndarray 中删除自己对应的 TWin 对象
    GetWindowRect(&rect);
    EvNcDestroy();
    //在 app 的 winarray 中删除自己
    TApp::DelWin(this);
    hWnd = 0;//删除自己对应的窗口对象
    return 0;
case WM_QUERYENDSESSION: // Dialog
```

```
    return !EvQueryEndSession((BOOL)wParam, (BOOL)lParam);
case WM_ENDSESSION:
    EvEndSession((BOOL)wParam, (BOOL)lParam);
    return 0;
case WM_QUERYOPEN:
    return EvQueryOpen();
case WM_PAINT:
    EvPaint();
    return 0;
case WM_NCPAINT:
    EvNcPaint((HRGN)wParam);
    return 0;
case WM_SIZE: //  调整大小
    EvSize((UINT)wParam, LOWORD(lParam), HIWORD(lParam));
    return 0;
case WM_GETMINMAXINFO:
    EvGetMinMaxInfo((MINMAXINFO *)lParam);
    return 0;
case WM_SETCURSOR:
    return EvSetCursor((HWND)wParam, LOWORD(lParam), HIWORD(lParam));
case WM_MOUSEMOVE:
    return EvMouseMove((UINT)wParam, MAKEPOINTS(lParam));
case WM_NCHITTEST:
```

```
    return EvNcHitTest(MAKEPOINTS(lParam), &result), result;
case WM_MEASUREITEM:
    return EvMeasureItem((UINT)wParam, (LPMEASUREITEMSTRUCT)lParam);
case WM_DRAWITEM:
    return EvDrawItem((UINT)wParam, (LPDRAWITEMSTRUCT)lParam);
case WM_NOTIFY:
    return EvNotify((UINT)wParam, (LPNMHDR)lParam);
case WM_CONTEXTMENU://EvContextMenu 处理上下文菜单消息
    //弹出一个右键菜单
    return EvContextMenu((HWND)wParam, MAKEPOINTS(lParam));
case WM_HOTKEY:
    return EvHotKey((int)wParam);
case WM_LBUTTONUP:
case WM_RBUTTONUP:
case WM_NCLBUTTONUP:
case WM_NCRBUTTONUP:
case WM_LBUTTONDOWN:
case WM_RBUTTONDOWN:
case WM_NCLBUTTONDOWN:
case WM_NCRBUTTONDOWN:
case WM_LBUTTONDBLCLK:
case WM_RBUTTONDBLCLK:
case WM_NCLBUTTONDBLCLK:
case WM_NCRBUTTONDBLCLK:
    //EventButton 处理所有的鼠标消息
    EventButton(uMsg, wParam, MAKEPOINTS(lParam));
    return 0;
case WM_HSCROLL:
```

```
case WM_VSCROLL:
    EventScroll(uMsg, LOWORD(wParam), HIWORD(wParam), (HWND)lParam);
    return 0;
case WM_INITMENU:
case WM_INITMENUPOPUP: //  处理初始化 menu 消息
    EventInitMenu(uMsg, (HMENU)wParam, LOWORD(lParam), (BOOL)HIWORD(lParam));
    return 0;
case WM_MENUSELECT:
    EvMenuSelect(LOWORD(wParam), (BOOL)HIWORD(wParam), (HMENU)lParam);
    return 0;
case WM_DROPFILES:
    EvDropFiles((HDROP)wParam);
    return 0;
case WM_CTLCOLORBTN:
case WM_CTLCOLORDLG:
case WM_CTLCOLOREDIT:
case WM_CTLCOLORLISTBOX:
case WM_CTLCOLORMSGBOX:
case WM_CTLCOLORSCROLLBAR:
case WM_CTLCOLORSTATIC:
    return EventCtlColor(uMsg, (HDC)wParam, (HWND)lParam,
     (HBRUSH *)&result), result;
case WM_KILLFOCUS:
case WM_SETFOCUS:
    EventFocus(uMsg, (HWND)wParam);
    return 0;
default: //  默认的用户消息都使用 EventUser 来处理！比如 WM_DELAYSETTEXT 等消息
```

```
   if (uMsg >= WM_USER && uMsg <= 0x7FFF || uMsg >= 0xC000 && uMsg <= 0xFFFF)
   {
    return EventUser(uMsg, wParam, lParam);
   }
}

return FALSE;
}

BOOL TDlg::PreProcMsg(MSG *msg)
{
/*
The TranslateAccelerator function processes accelerator keys
for menu commands. The function translates a WM_KEYDOWN or
WM_SYSKEYDOWN message to a WM_COMMAND or WM_SYSCOMMAND message
(if there is an entry for the key in the specified accelerator table)
and then sends the WM_COMMAND or WM_SYSCOMMAND message directly to
the appropriate window procedure. TranslateAccelerator does not
return until the window procedure has processed the message.

Return Values
If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero.

*/
if ( hAccel && ::TranslateAccelerator(hWnd, hAccel, msg) )
{
   return TRUE;
}
    /*
The IsDialogMessage function determines whether a message
is intended for the specified dialog box and, if it is, processes the message.

   Return Values
If the message has been processed, the return value is nonzero.

If the message has not been processed, the return value is zero.


*/

if ( !modalFlg )
{
   return ::IsDialogMessage(hWnd, msg);
```

```
}

return FALSE;
}

BOOL TDlg::EvSysCommand(WPARAM uCmdType, POINTS pos)
{
return FALSE;
}

//处理对话框上的 ok  和 cancel 按钮

BOOL TDlg::EvCommand(WORD wNotifyCode, WORD wID, LPARAM hwndCtl)
{
/*

如果是 idok，idcancel，idyes，idno 则关闭对话框，返回真

否则返回假

*/

switch (wID)
{

case IDOK: case IDYES:

   EndDialog(TRUE);

   return TRUE;

case IDCANCEL: case IDNO:

   EndDialog(FALSE);

   return TRUE;
}

return FALSE;
}

BOOL TDlg::EvQueryOpen(void)
{
return FALSE;
}
```

```
BOOL TDlg::EvCreate(LPARAM lParam)
{
return TRUE;
}

void TDlg::EndDialog(int result)
{
/*
The EndDialog function destroys a modal dialog box,
causing the system to end any processing for the dialog box.

BOOL EndDialog(
HWND hDlg,         // handle to dialog box
INT_PTR nResult    // value to return
);

Parameters

hDlg
[in] Handle to the dialog box to be destroyed.
nResult
[in] Specifies the value to be returned to the application
from the function that created the dialog box.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

The IsWindow function determines whether the specified
window handle identifies an existing window.

EndDialog 干掉模式对话框, modalFlg 参数指定了是不是模式对话框

*/

if ( ::IsWindow(hWnd) )
{
   if ( modalFlg )
   {
    ::EndDialog(hWnd, result);
   }
   else //  非模式对话框，直接 destroyWindow
   {
```

```
    ::DestroyWindow(hWnd);
   }
}
}
```

TAboutDlg 类的代码很少：

```
//关于对话框,只需要重载 create 和响应 WM_COMMAND 消息的 EvCommand 函数

class TAboutDlg : public TDlg
{

public:

TAboutDlg(TWin *_parent = NULL);

virtual BOOL EvCommand(WORD wNotifyCode, WORD wID, LPARAM hwndCtl);

virtual BOOL EvCreate(LPARAM lParam);
};
```

TAboutDlg 类的实现代码：

```
/*
About Dialog 关于对话框,调用父类的构造函数，注意 ABOUT_DIALOG 是在 rc 文件中定义的

含有一个 icon，一个 button 和一些文字！！！这里的参数 ABOUT_DIALOG 是一个 UID，

在 resource.h 中有定义：

ABOUT_DIALOG DIALOGEX 0, 0, 126, 81
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About IP Messenger"
FONT 9, "Verdana"
BEGIN
    DEFPUSHBUTTON    "OK",IDOK,44,63,35,11
    ICON            IPMSG_ICON,IDC_STATIC,6,5,18,20,SS_SUNKEN
    CTEXT           "IP Messenger for Win\r\nversion 2.06",IDC_STATIC,29,4,
                    80,20,SS_SUNKEN
    CTEXT           "Copyright(c)1996-2004 H.Shirouzu",IDC_STATIC,2,50,118,9
    PUSHBUTTON      "http://www.ipmsg.org",IPMSGWEB_BUTTON,5,29,118,17,
                    BS_CENTER | BS_VCENTER,WS_EX_DLGMODALFRAME |
                    WS_EX_CLIENTEDGE | WS_EX_STATICEDGE
```

```
END

*/

TAboutDlg::TAboutDlg(TWin *_parent) : TDlg(ABOUT_DIALOG, _parent)
{
}

/*
Window CallBack
*/
BOOL TAboutDlg::EvCreate(LPARAM lParam)
{
//把窗口的图标设置为默认的主窗口的图标

SetDlgIcon(hWnd);

/*
nWidth:以设备单元指明窗口的宽度。对于层叠窗口，

nWidth 或是屏幕坐标的窗口宽度或是 CW_USEDEFAULT。若 nWidth 是 CW_USEDEFAULT,

    则系统为窗口选择一个缺省的高度和宽度：缺省宽度为从初始 X 坐标开始到屏幕的右边界,

缺省高度为从初始 X 坐标开始到目标区域的顶部。CW_USEDFEAULT 只参层叠窗口有效；

如果为弹出式窗口和子窗口设定 CW_USEDEFAULT 标志则 nWidth 和 nHeight 被设为零

   把窗口移动到 rect

*/

if ( rect.left == CW_USEDEFAULT )
{
  //获得 window 的 rect

  GetWindowRect(&rect);

  //xsize 和 ysize 是 window 的宽和高

  int xsize = rect.right - rect.left, ysize = rect.bottom - rect.top;
```

```
    //获得屏幕的高和宽

    int cx = ::GetSystemMetrics(SM_CXFULLSCREEN),
     cy = ::GetSystemMetrics(SM_CYFULLSCREEN);

    int x = ( cx - xsize ) / 2;

    int y = ( cy - ysize ) / 2;

    //调整窗口的位置

    MoveWindow( ( x < 0 ) ? 0 : x % ( cx - xsize ), ( y < 0 ) ? 0 : y % (cy - ysize),
     xsize, ysize, FALSE);
}
else
{
    MoveWindow(rect.left, rect.top, rect.right - rect.left,
     rect.bottom - rect.top, FALSE);
}

return TRUE;
}

BOOL TAboutDlg::EvCommand(WORD wNotifyCode, WORD wID, LPARAM hWndCtl)
{
switch (wID)
{

case IDOK:

case IDCANCEL: //  直接 EndDialog 干掉模态对话框,或者 destroywindow(非模态)

    EndDialog(TRUE);

    return TRUE;

case IPMSG_ICON:

case IPMSGWEB_BUTTON:

    //如果按下了 button IPMSGWEB_BUTTON  则  访问 IPMSGURL_MSGSTR 所指定的网站

    if ( wID == IPMSGWEB_BUTTON || wNotifyCode == 1 )
    {
     ::ShellExecute(NULL, NULL, IPMSGURL_MSGSTR, NULL, NULL, SW_SHOW);
```

```
    }

    return TRUE;
}

return FALSE;
}
```

（转）IPMSG 飞鸽传书——协议翻译
2010-05-09 11:04

最近看到一些朋友在编写网络程序是遇到一些问题，故把以前做 IPMSG 时翻译的文档贴过来，希望对网络编程新手有所帮助，在寻找编程项目的同学们也可参照此文档写出自己的 IPMSG。

本文只包含其中几个比较重要的命令以及运行机制的中文翻译，更详细的内容请参照文后的 IPMSG 协议英文文档

声明：下述协议内容略去了一些在编写程序过程中没有用到协议内容，最初的 Ipmsg 协议是用日文写的，下面协议内容由本人(cugb_cat)翻译自 Mr.Kanazawa 的英文文档。本翻译文档可任意传播和使用。

IP 信使传输协议(第 9 版草案)　　1996/02/21
2003/01/14 修订

H.Shirouzu
[email]shirouzu@h.email.ne.jp[/email]

关于 IP 信使：
IP 信使使用 TCP/UDP 协议提供收发消息及文件(目录)。
特性：
IP 信使能够安装在任何一个安装了 TCP/IP 协议栈的操作系统上，使用在线用户的动态识别机制，可以和在线所有用户进行信息交换。
运行机制介绍：
使用 TCP/UDP 端口(默认端口为 2425)，消息的收发使用 UDP 协议，文件(文件夹)的收发使用 TCP 协议。
1、 命令字：
1) 基本命令字(32 位命令字的低 8 位)
IPMSG_NOOPERATION　　不进行任何操作
IPMSG_BR_ENTRY　　用户上线
IPMSG_BR_EXIT　　用户退出
IPMSG_ANSENTRY　　通报在线
IPMSG_SENDMSG　　发送消息
IPMSG_RECVMSG　　通报收到消息

IPMSG_GETFILEDATA　　请求通过 TCP 传输文件
IPMSG_RELEASEFILES 停止接收文件
IPMSG_GETDIRFILES　　请求传输文件夹
2) 选项位(32 位命令字的高 24 位)
IPMSG_SENDCHECKOPT 传送检查(需要对方返回确认信息)
IPMSG_FILEATTACHOPT 传送文件选项
3) 附件类型命令(文件类型命令字的低 8 位)
IPMSG_FILE_REGULAR 普通文件
IPMSG_FILE_DIR　　目录文件
IPMSG_FILE_RETPARENT 返回上一级目录
2、 数据包格式(使用字符串)：
1) 数据包格式(版本 1 的格式)
版本号(1):包编号:发送者姓名:发送者主机名:命令字:附加信息
2) 举例如下
“1:100:shirouzu:Jupiter:32:Hello”
3、 数据包处理总述：
1) 用户识别
当 IPMSG 启动时，命令 IPMSG_BR_ENTRY 被广播到网络中，向所有在线的用户提示一个新用户的到达(即表示“我来了”)；所有在线用户将把该新上线用户添加到自己的用户列表中，并向该新上线用户发送 IPMSG_ANSENTRY 命令(即表示“我在线”)；该新上线用户接收到 IPMSG_ANSENTRY 命令后即将在线用户添加到自己的用户列表中。
2) 收发消息
使用 IPMSG_SENDMSG 命令发送消息，消息内容添加在附加信息中；在接收消息时，如果对方要求回信确认(IPMSG_SENDCHECKOPT 位打开)，则需发送 IPMSG_RECVMSG 命令并将对方发送的数据包的编号放在附加信息中一同发送至发送消息方
3) 附加文件的扩充(添加于第 9 版)
带有 IPMSG_FILEATTACHOPT 位的 IPMSG_SENDMSG 命令可用来传输文件，文件属性及内容添加在附加信息中，文件内容添加在消息内容后并以’\0’与之分隔开。传输文件时以下信息将被添加到消息内容之后(包括格式)：文件序号:文件名:大小(单位:字节):最后修改时间:文件属性[: 附加属性=val1[,val2…][:附加信息=…]]:\a:文件序号…
(文件大小、最后修改时间和文件属性为十六进制数，如果文件名中包含’:’则使用“::”代替)。
接收端开始接收文件时，请求传输文件命令 IPMSG_GETFILEDATA 将发送到发送端的 TCP 端口(和 UDP 的发送端口相同)，并将发送端发送的包编号:文件序号:偏移量(全为十六进制格式)写到附加信息区一同发送，文件发送端接收到该请求信息并进行校验正确后即开始发送文件(不使用任何格式，亦不进行加密)。
当接收端接收到目录文件时，将发送附加信息区为发送端发送的包编号:文件序号:偏移量(全为十六进制格式)的 IPMSG_GETDIRFILES 命令，以用来请求传输目录文件；发送端则将头信息长度:文件名:文件大小:文件属性:文件内容添加到附加信息区(除了文件名和文件内容外，其余皆为十六进制)，头信息长度是从头

信息长度开始到文件内容前的‘:’分割符为止的字符个数。
当文件属性为 IPMSG_FILE_DIR 时，IPMsg 能够自动识别其为目录，下一个文件的数据在该目录之后。
当文件属性为 IPMSG_FILE_RETPARENT 时，IPMsg 识别其动作为返回上一级目录，在这种情况下，文件名为‘.’其属性为当前目录的值。

（转）IPMSG 飞鸽传书——App 类
2010-05-09 11:03

经过几周的艰苦奋战，终于对飞鸽传书（以下简称 FG）的架构有了一个简单的了解，下面分类分析一下各个类的基本情况。

首先是 App 类。和 MFC 中其他程序一样，FG 中存在着一个 App 类，用来发送消息，创建和管理窗口等。

main 函数如下：

```
//这里是主函数！！！！！！！！！！！！！！！！调用 app 的 run 开始执行

//构造函数中有 initapp 等操作

int WINAPI WinMain(HINSTANCE hI, HINSTANCE, LPSTR cmdLine, int nCmdShow)
{
//建立一个 TMsgApp 类的对象

TMsgApp app( hI, cmdLine, nCmdShow );

return app.Run();//run 函数完成主要的工作，初始化窗口，循环获得和发送消息等
}
```

TMsgApp 是 TApp 类的子类，TApp 类的定义如下：

```
class TApp
{

protected:

//static 的一大堆变量

static TWin     **wndArray;

static int     wndCnt;

static TWin     *SearchWnd(HWND hW);
```

```
static BOOL     SearchWndIndex(TWin *win, int *index);

static void     AddWinSub(TWin *win);

HINSTANCE     hCtl3d;

LPSTR      cmdLine;

int       nCmdShow;

//有一个 TWin 的 mainwnd，指向 TMainWin 类创建的窗口

TWin      *mainWnd;

virtual BOOL    InitApp(void);//注意这里的 InitApp 是虚函数，子类可以重载它

public:

TApp(HINSTANCE _hI, LPSTR _cmdLine, int _nCmdShow);

virtual ~TApp();

virtual void    InitWindow() = 0;

virtual int     Run();

static BOOL    PreProcMsg(MSG *msg);

/*

我们需要在 TApp 的 WndProc 函数中把创建的 TWin 指针 win 的 hwnd 设置为 hwnd，然后加入

wndArray 中，但是我怎么使用参数已经固定的 WndProc 函数来传入一个 TWin 指针呢？

这个 TWin 指针在外面已经创建好了，而且我们需要把创建好的 TWin 指针传进来。

怎么做呢？ 我们首先把创建好的 TWin 指针使用 preWnd 保存起来，然后后面把 win 赋值为 preWnd

设置 hwnd 为 hwnd 就可以了，然后就可以安全的加入 wndarray 中了，因为 preWnd 已经创建好了

*/
```

```
static TWin     *preWnd;

static char     *defaultClass;

static HINSTANCE hI;

static void     AddWin(TWin *win);

static void     AddWinByWnd(TWin *win, HWND hWnd);

static void     DelWin(TWin *win);

static LRESULT CALLBACK WinProc(HWND hW, UINT uMsg, WPARAM wParam, LPARAM
lParam);
};
```

实现文件如下：

```
static char *tapp_id =
"@(#)Copyright (C) H.Shirouzu 1996-2001    tapp.cpp Ver0.95";
/*
========================================================================
Project Name     : Win32 Lightweight Class Library Test
Module Name      : Application Frame Class
Create       : 1996-06-01(Sat)
Update       : 2001-12-06(Thu)
Copyright      : H.Shirouzu
Reference      :
========================================================================
*/

#include "tlib.h"
/*

这是 TApp 类的一大拖静态变量

wndCnt 是 wndArray 中窗口的数目

居然保存了一大堆 TWin 的窗口！！

TApp 有两个子类，一个是 InstApp 一个是 MsgApp

*/

TWin**    TApp::wndArray    = NULL;
```

```
int     TApp::wndCnt    = 0;

TWin*    TApp::preWnd    = NULL;

HINSTANCE TApp::hI     = NULL;

char*    TApp::defaultClass = "tapp";

HINSTANCE LoadCtl3d(void);

/*
Initializes the COM library on the current thread and identifies
the concurrency model as single-thread apartment (STA).
Applications must initialize the COM library before they can
call COM library functions other than CoGetMalloc and memory allocation functions.
CoInitialize 初始化 com 库，这样就可以调用 com 库中的函数了
*/

TApp::TApp(HINSTANCE _hI, LPSTR _cmdLine, int _nCmdShow)
{
hI     = _hI;

cmdLine    = _cmdLine;

nCmdShow = _nCmdShow;

mainWnd    = NULL;

::CoInitialize(NULL);

hCtl3d = LoadCtl3d();

::InitCommonControls();
}

TApp::~TApp()
{
//删除 TWin 的主窗口指针，注意 new 是在 initApp 中 new 的！！！！！！！！！

delete mainWnd;

/*
The FreeLibrary function decrements the reference
count of the loaded dynamic-link library (DLL).
When the reference count reaches zero, the module
```

```
is unmapped from the address space of the calling
process and the handle is no longer valid.
注意 HINSTANCE hCtl3d；dll 文件用句柄来标志
*/

if ( hCtl3d )
{
   ::FreeLibrary(hCtl3d);
}

/*
Closes the COM library on the current thread,
unloads all DLLs loaded by the thread, frees any other resources
that the thread maintains, and forces all RPC connections on the thread to close.
在当前线程上关闭 com 库卸载掉所有当前线程加载的 dlls，释放所有线程维护的资源
关闭当前线程上的所有的 RPC 连接
*/

::CoUninitialize();
}

int TApp::Run(void)
{
MSG    msg;

InitApp();

InitWindow();//这里是纯虚拟构造函数！！！

/*
The GetMessage function retrieves a message
from the calling thread's message queue.
The function dispatches incoming sent messages
until a posted message is available for retrieval.

    if hWnd == NULL : GetMessage retrieves messages for any window that belongs

to the calling thread and thread messages posted to the calling

thread using the PostThreadMessage function.

If the function retrieves a message other than WM_QUIT, the return value is nonzero.

If the function retrieves the WM_QUIT message, the return value is zero.
```

If there is an error, the return value is -1. For example,

the function fails if hWnd is an invalid window handle or lpMsg is an invalid pointer.

The TranslateMessage function translates virtual-key messages into character messages.
The character messages are posted to the calling thread's message queue,
to be read the next time the thread calls the GetMessage or PeekMessage function.

BOOL TranslateMessage(
CONST MSG *lpMsg    // message information
);
Parameters
lpMsg
[in] Pointer to an MSG structure that contains message information
retrieved from the calling thread's message queue by using the GetMessage
or PeekMessage function.
Return Values
If the message is translated (that is, a character message is posted
to the thread's message queue), the return value is nonzero.

If the message is WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN, or WM_SYSKEYUP,
the return value is nonzero, regardless of the translation.

If the message is not translated (that is, a character message is
not posted to the thread's message queue), the return value is zero.

得到一个消息，首先看有没有预处理的窗口，然后发送消息

注意这里预处理的窗口可能是 msg 的目的窗口的父窗口

*/
```
while ( ::GetMessage(&msg, NULL, 0, 0) )
{
   if ( PreProcMsg(&msg) )
   {
    continue;
   }

   ::TranslateMessage(&msg);

   ::DispatchMessage(&msg);
}

return msg.wParam;
```

```
}

BOOL TApp::PreProcMsg(MSG *msg) // for TranslateAccel & IsDialogMessage
{
/*

这里将 msg 发送到 msg 中指定的那个窗口的父窗口链，如果在 wndarray 中找到了

窗口  hWnd,则调用 win 的 PreProcMsg 来处理消息


*/
for ( HWND hWnd = msg->hwnd; hWnd != NULL; hWnd = ::GetParent(hWnd) )
{
   //在 WndArray 中找到 hWnd 标志的那个 TWin*并将之返回

   TWin *win = SearchWnd(hWnd);

   if ( win != NULL )
   {
    return win->PreProcMsg(msg);
   }
}

return FALSE;
}

//WNDCLASS 窗口类的消息处理函数居然交给 TApp 的 WinProc 处理！！！

/*
所有发送给 TWin 的消息都要首先给 WinProc 处理！！！！
*/

LRESULT CALLBACK TApp::WinProc(HWND hWnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
//查找 hWnd 这个窗口,如果找到了,则调用 win 窗口的 WinProc 来处理

TWin *win = SearchWnd( hWnd );

if ( win ) //  找到了，不是第一次加入
{
   return win->WinProc(uMsg, wParam, lParam);
}
```

```
//是第一次加入，则加入 wndarray

//如果是第一次调用 TApp：：WinProc 则加入一个 TWin 到 TWinArray，

//把 PreWnd 设置为 NULL,然后创建一个 TWin,

//然后把窗口 hWnd 赋给 win 的 pWnd,然后调用 Win 的 winproc 处理

if ( ( win = preWnd ) != NULL )
{
   preWnd = NULL;//把 preWnd 设置为 NULL，以后要用到

   //注意 win 不是 NULL！win 保存了 prewnd 原来的值

   //这里的 hWnd 已经 create 完毕，可以加入到 hwndarray 中了

   AddWinByWnd(win, hWnd);//把 hWnd 加入 array 中

   //掉用 win 的 WinProc 来处理

   return win->WinProc(uMsg, wParam, lParam);
}

return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

void TApp::AddWin( TWin *win )
{
//  仅仅只是把 preWnd 指向了 win

preWnd = win;
}

//把 TWin 指针的 hWnd 设置为参数 hWnd,然后加入到 wndArray 中

void TApp::AddWinByWnd(TWin *win, HWND hWnd)
{
win->hWnd = hWnd;

AddWinSub(win);
}

//在 wndArray 中加入 TWin 指针  win

void TApp::AddWinSub(TWin *win)
```

```
{
/*

如果在 wndArray 中已经有那个 win 了

则什么也不做返回。如果 wndArray 数组的

长度达到 100 了，则再分配 100 个空间给 wndArray

然后把 wndCnt 这个位置放上 win

*/

if ( SearchWnd ( win->hWnd ) != NULL )
{
   return;
}

#define BIG_ALLOC 100

TWin **_wndArray = wndArray;

if ( ( wndCnt % BIG_ALLOC ) == 0 )
{
   _wndArray = (TWin **)realloc(wndArray, sizeof(TWin *)
    * (wndCnt + BIG_ALLOC));
}

if ( _wndArray == NULL ) //  重新分配空间失败,直接返回
{
   return;
}

(wndArray = _wndArray)[wndCnt++] = win;

return;
}

//在 wndArray 中删除 win 指定的 TWin 窗口

void TApp::DelWin(TWin *win)
{
/*

删除 win 指定的那个指针，如果 WinArray 中没有则直接返回；
```

如果在数组中没有 win 指向的那个指针，如果 prewnd 指向了 win，则把 preWnd 置位 NULL

如果 preWnd 没有指向 win，则直接返回；

为什么要这样考虑呢？因为有的时候我们仅仅是 AddWin 了，仅仅把 preWnd 指向了一个 TWin 的对象，但是并没有加入到 wndArray 中，这时我们 search 就返回 FALSE 了,但是我们还得删除 preWnd 指向的那个对象，所以这里仅仅是把 preWnd 设为了 NULL。

*/

```
int    index;

if ( SearchWndIndex(win, &index) == FALSE )
{
   if ( preWnd == win )
   {
    preWnd = NULL;
   }

   return;
}

//TWin 在 TWinArray 中存在

//  删除 wndArray＋index 指向的那个指针，后面的所有 TWin 指针向前移动一格

memmove(wndArray + index, wndArray + index + 1,
   sizeof(TWin *) * ( --wndCnt - index ) );

// wndArray = (TWin **)realloc(wndArray, sizeof(TWin *) * wndCnt);
}

BOOL TApp::InitApp(void) // reference kwc
{
/*
The WNDCLASS structure contains the window class
attributes that are registered by the RegisterClass function.
style:
CS_BYTEALIGNCLIENT    Aligns the window's client area on a byte boundary
                      (in the x direction). This style affects the width of
                 the window and its horizontal placement on the display.
    CS_BYTEALIGNWINDOW Aligns the window on a byte boundary (in the x direction).
                  This style affects the width of the window and
```

its horizontal placement on the display.
CS_DBLCLKS Sends a double-click message to the window procedure
when the user double-clicks the mouse while the cursor
is within a window belonging to the class.
*/

WNDCLASS wc;

memset( &wc, 0, sizeof(wc) );

wc.style = (CS_BYTEALIGNCLIENT | CS_BYTEALIGNWINDOW | CS_DBLCLKS);

wc.lpfnWndProc = WinProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = hI;

wc.hIcon = NULL;

wc.hCursor = LoadCursor(NULL, IDC_ARROW);

wc.hbrBackground = NULL;

wc.lpszMenuName = NULL;

wc.lpszClassName = defaultClass;// == "Tapp"

/*

The FindWindow function retrieves a handle

to the top-level window whose class name

and window name match the specified strings.

This function does not search child windows.

This function does not perform a case-sensitive search.

FindWindow()不检查子窗口，搜索不是大小写敏感的

The RegisterClass function registers a window class

for subsequent use in calls to the CreateWindow or CreateWindowEx function.

ATOM RegisterClass(
CONST WNDCLASS *lpWndClass // class data
);

首先查找 title 是我们创立的这个 class 的，如果没有找到，则注册一下，找到了则什么也不做

*/

```
if ( ::FindWindow(defaultClass, NULL) == NULL )
{
   if ( ::RegisterClass(&wc) == 0 )
   {
    return FALSE;
   }
}

return TRUE;
}
```

TWin* TApp::SearchWnd(HWND hWnd) //在 wndArray 中查找 hwnd 是参数 hWnd 的 TWin 指针

```
{
/*
```

在 wndArray 中找到 hWnd 所标志的那个

窗口，将之返回

*/

```
for ( int cnt = 0; cnt < wndCnt; cnt++ )
{
   if ( wndArray[cnt]->hWnd == hWnd )
   {
    return wndArray[cnt];
   }
}

return NULL;
}
```

//找到 TWin win 指针的 index

```
BOOL TApp::SearchWndIndex(TWin *win, int *index)
{
/*

在 wndArray 中找到 win 标志的那个窗口，

将位置保存到 index 中。

*/

for ( *index = 0; *index < wndCnt; (*index) ++ )
{
   if ( wndArray[*index] == win )
   {
    return TRUE;
   }
}

return FALSE;
}

#if 0

IsXpManifest(void)
{
static BOOL done = FALSE;

static BOOL isXpManifest = FALSE;

//只查找一次，如果已经完成了查找，直接返回结果 isXpManifest

if ( done )
{
   return isXpManifest;
}

//是第一次判断

done = TRUE;

HMODULE    hComctl;

char    path[MAX_PATH];

if ( ( hComctl = ::GetModuleHandle("comctl32")) == NULL )
```

```
{
   return isXpManifest = FALSE;
}

if ( ::GetModuleFileName(hComctl, path, sizeof(path)) == 0 )
{
   return isXpManifest = FALSE;
}


GetFileVersionInfo();

VerQueryValue();

}
#endif

HINSTANCE LoadCtl3d(void)
{
if ( IsNewShell() )
{
   return NULL;
}

HINSTANCE hCtl3d;

BOOL (WINAPI *Ctl3dRegister)(HANDLE);

BOOL (WINAPI *Ctl3dAutoSubclass)(HANDLE);

hCtl3d = ::LoadLibrary("ctl3d32.dll");

/*

ctl3d32.dll 是 Windows 应用程序公用 GUI 图形用户界面 3D 维显示模块。

The GetProcAddress function retrieves the address of

   an exported function or variable from the specified dynamic-link library (DLL).

FARPROC GetProcAddress(
HMODULE hModule,     // handle to DLL module
LPCSTR lpProcName    // function name
);
```

调用 ctl3d32.dll 中的 2 个函数 Ctl3dRegister(TApp::hI); 和 Ctl3dAutoSubclass(TApp::hI);

*/

```
Ctl3dRegister = (BOOL (WINAPI *)(HANDLE))::GetProcAddress(hCtl3d, "Ctl3dRegister");

Ctl3dAutoSubclass = (BOOL (WINAPI *)(HANDLE))::GetProcAddress(hCtl3d,
"Ctl3dAutoSubclass");

if ( Ctl3dRegister && Ctl3dAutoSubclass )
{
   Ctl3dRegister(TApp::hI);

   Ctl3dAutoSubclass(TApp::hI);
}

return hCtl3d;
}
```

其中 TMsgApp 的代码如下：

```
class TMsgApp : public TApp
{

public:

TMsgApp(HINSTANCE _hI, LPSTR _cmdLine, int _nCmdShow);

virtual ~TMsgApp();


virtual void InitWindow(void);
};
```

实现文件如下：

```
static char *ipmsg_id =
"@(#)Copyright (C) H.Shirouzu 1996-2002    ipmsg.cpp Ver2.00";
/*
========================================================================
Project Name     : IP Messenger for Win32
Module Name      : IP Messenger Application Class
Create       : 1996-06-01(Sat)
Update       : 2002-11-03(Sun)
```

```
Copyright      : H.Shirouzu
Reference      :
================================================================================
*/

#include <time.h>
#include "tlib.h"
#include "resource.h"
#include "ipmsg.h"

#define IPMSG_CLASS "ipmsg_class"

TMsgApp::TMsgApp(HINSTANCE _hI, LPSTR _cmdLine, int _nCmdShow)
: TApp(_hI, _cmdLine, _nCmdShow)
{
//为后面的随机数做好准备

srand( (UINT)Time() );
}

TMsgApp::~TMsgApp()
{
}

void TMsgApp::InitWindow(void)
{
//注册一个窗口结构 wc

WNDCLASS wc;

HWND    hWnd;

char    class_name[MAX_PATH] = IPMSG_CLASS, *tok, *msg, *p;

ULONG    nicAddr = 0;

int     port_no = atoi(cmdLine);

if ( port_no == 0 )//命令行中没有设置端口号,则设置为默认的端口号
{
   //#define IPMSG_DEFAULT_PORT    0x0979

   port_no = IPMSG_DEFAULT_PORT;
}
```

```
//解析命令行,由于这里没有命令行，所以这个 if tok  里面的语句不会执行

if ( ( tok = strchr(cmdLine, '/')) && separate_token(tok, ' ', &p) )
{
   BOOL diag = TRUE;

   DWORD status = 0xffffffff;

   if ( stricmp(tok, "/NIC") == 0 ) // NIC
   {
    if ( tok = separate_token(NULL, ' ', &p ) )
    {
     nicAddr = ResolveAddr(tok);
    }
   }
   else if ( stricmp(tok, "/MSG") == 0 ) //
   {
    MsgMng msgMng(nicAddr, port_no);

    ULONG command = IPMSG_SENDMSG|IPMSG_NOADDLISTOPT|IPMSG_NOLOGOPT,
destAddr;

    while ( ( tok = separate_token(NULL, ' ', &p ) ) != NULL && *tok == '/' )
    {
     if ( stricmp(tok, "/LOG") == 0 )
     {
      command &= ~IPMSG_NOLOGOPT;
     }
     else if ( stricmp(tok, "/SEAL") == 0 )
     {
      command |= IPMSG_SECRETOPT;
     }
    }

    if ( ( msg = separate_token(NULL, 0, &p)) != NULL )
    {
     diag = FALSE;

     if ( ( destAddr = ResolveAddr(tok)) != NULL )
     {
      status = msgMng.Send(destAddr, htons(port_no), command, msg) ? 0 : -1;
     }
    }
   }
```

```
   if ( nicAddr == 0 )
   {
    if ( diag )
    {
     MessageBox(0, "ipmsg.exe [portno] [/MSG [/LOG] [/SEAL] \
      <hostname or IP addr> <message>]\r\nipmsg.exe [portno] \
      [/NIC nic_addr]", MSG_STR, MB_OK);
    }

    ::ExitProcess(status);

    return;
   }
}

//如果 port_no 不是默认的端口才这样做 class_name=“ipmsg_class_0X979”

if ( port_no != IPMSG_DEFAULT_PORT || nicAddr )//这里端口已经设置好了，所以也不会执行
{
   wsprintf(class_name, nicAddr ? "%s_%d_%s" : "%s_%d", IPMSG_CLASS,
    port_no, inet_ntoa(*(in_addr *)&nicAddr) );
}

//建立 WNDCLASS 结构

memset( &wc, 0, sizeof(wc) );

wc.style     = CS_DBLCLKS;

wc.lpfnWndProc    = TApp::WinProc;

wc.cbClsExtra      = 0;

wc.cbWndExtra    = 0;

wc.hInstance    = hI;

wc.hIcon     = ::LoadIcon(hI, (LPCSTR)IPMSG_ICON);

wc.hCursor     = ::LoadCursor(NULL, IDC_ARROW);

wc.hbrBackground = NULL;

wc.lpszMenuName    = NULL;
```

```
wc.lpszClassName = class_name;

/*

   使程序运行时只创建一个实例

HANDLE    hMutex=CreateMutex(
NULL, //    nosecurityattributes
FALSE, //    initiallynotowned
"yourApp");//    命名 Mutex 是全局对象

if ( hMutex == NULL || ERROR_ALREADY_EXISTS == ::GetLastError() )
{
     //程序第二次或以后运行时，会得到 Mutex 已经创建的错误

     MessageBox(NULL,"该程序已经运行","Info",MB_OK|MB_ICONINFORMATION);

     return    FALSE;
}

CreateMutex：

If the function succeeds, the return value is a handle to the mutex object.

If the named mutex object existed before the function call, the function returns

a handle to the existing object and GetLastError returns ERROR_ALREADY_EXISTS.

Otherwise, the caller created the mutex.

*/

HANDLE hMutex = ::CreateMutex( NULL, FALSE, class_name );

::WaitForSingleObject( hMutex, INFINITE );

//如果 class_name 名字的 window 已经存在了，则不创建新的程序，否则登记一下

//防止同时运行两个飞鸽！！

if ( ( hWnd = ::FindWindow( class_name, NULL ) ) != NULL
     || ::RegisterClass(&wc) == 0 )
{
   if ( hWnd != NULL )//找到了飞鸽，把飞鸽设置为前台程序
   {
    ::SetForegroundWindow(hWnd);
```

```
    }

    ::ExitProcess(0xffffffff);

    return;
}

//创建一个新的 TMainWin 窗口,这里的参数 nicAddr 为 0，port_no 为 0X979

//这里创建各个管理类对象和常用到的各个窗口，创建 udp socket  和 tcpsocket

mainWnd = new TMainWin( nicAddr, port_no );

//创建 TMainWin 的 CWnd 窗口，启动时最小化

mainWnd->Create( class_name, IP_MSG, WS_OVERLAPPEDWINDOW |
    ( IsNewShell() ? WS_MINIMIZE : 0 ) );

//释放全局的 hMutex 变量

::ReleaseMutex(hMutex);

::CloseHandle(hMutex);

//InitApp 完毕后进入 Run 函数里面的获得消息和派送消息了！
}
```

（转）IPMSG 飞鸽传书——日志文件管理类

2010-05-09 11:02

日志文件管理类主要的作用在于组装消息，然后写入 log 文件 ipmsg.txt.

```
//日志文件管理类

class LogMng
{

protected:

Cfg    *cfg;

BOOL Write(LPCSTR str);

public:

LogMng(Cfg *_cfg);
```

```
BOOL WriteSendStart(void);

BOOL WriteSendHead(LPCSTR head);

BOOL WriteSendMsg(LPCSTR msg, ULONG command, ShareInfo *shareInfo=NULL);

BOOL WriteRecvMsg(MsgBuf *msg, THosts *hosts, ShareInfo *shareInfo=NULL);

BOOL WriteStart(void);

BOOL WriteMsg(LPCSTR msg, ULONG command, ShareInfo *shareInfo=NULL);

    static void StrictLogFile(char *path);
};
```

实现类代码：

```
static char *logmng_id =
"@(#)Copyright (C) H.Shirouzu 1996-2002    logmng.cpp Ver2.00";
/*
========================================================================
Project Name     : IP Messenger for Win32
Module Name      : Log Manager
Create       : 1996-08-18(Sun)
Update       : 2002-11-03(Sun)
Copyright      : H.Shirouzu
Reference      :
========================================================================
*/

#include "tlib.h"
#include "resource.h"
#include "ipmsg.h"
#include "msgstr.h"

//管理 log 日志文件的类

LogMng::LogMng(Cfg *_cfg)
{
cfg = _cfg;
}

BOOL LogMng::WriteSendStart()
{
```

```
//写入"----------\r\n"

return WriteStart();
}

//写入  "TO : head 字符串\r\n"

BOOL LogMng::WriteSendHead(LPCSTR head)
{
char buf[MAX_LISTBUF];

wsprintf(buf, " To: %s\r\n", head);

return Write(buf);
}

BOOL LogMng::WriteSendMsg(LPCSTR msg, ULONG command, ShareInfo *shareInfo)
{
return WriteMsg(msg, command, shareInfo);
}

BOOL LogMng::WriteRecvMsg(MsgBuf *msg, THosts *hosts, ShareInfo *shareInfo)
{
//如果设置了不用写入接收到的文件，则直接返回

if ( msg->command & IPMSG_NOLOGOPT )
{
   return FALSE;
}

//写入表头

WriteStart();

char buf[MAX_PATH] = " From: ";

//根据 msg 创建一个 list 字符串，写入 buf

MakeListString(cfg, &msg->hostSub, hosts, buf + strlen(buf));

strcat(buf, "\r\n");

//将 buf 写入文件

Write(buf);
```

```
//msg 的 msgBuf 保存了要写入的内容

return WriteMsg(msg->msgBuf, msg->command, shareInfo);
}

BOOL LogMng::WriteStart(void)
{
return Write("==========================================\r\n");
}

BOOL LogMng::WriteMsg(LPCSTR msg, ULONG command, ShareInfo *shareInfo)
{
/*

首先写入时间：

at 时间 ， 如果是封装的则在加上（RSA）（封装）

*/
char buf[MAX_BUF * 2] = " at ";

strcat(buf, Ctime());

strcat(buf, " ");

//根据是多播、自动回复还是多播等拼接上不同的字符串

if ( command & IPMSG_BROADCASTOPT )//广播
{
   strcat(buf, BROADCASTLOG_MSGSTR);
}

if ( command & IPMSG_AUTORETOPT )//自动回复
{
   strcat(buf, AUTORETLOG_MSGSTR);
}

if ( command & IPMSG_MULTICASTOPT )//多播
{
   strcat(buf, MULTICASTLOG_MSGSTR);
}

if ( command & IPMSG_ENCRYPTOPT )//已加密
{
```

```
    //#define ENCRYPT_MSGSTR    "(RSA)"

    strcat(buf, ENCRYPT_MSGSTR);
}

if ( command & IPMSG_SECRETOPT )
{
    if ( command & IPMSG_PASSWORDOPT )//已经上锁了
    {
     //#define PASSWDLOG_MSGSTR "(locked)"

     strcat(buf, PASSWDLOG_MSGSTR);
    }
    else //  已经加入信封了
    {
     //#define SECRETLOG_MSGSTR "(sealed)"

     strcat(buf, SECRETLOG_MSGSTR);
    }
}

if ( shareInfo && (command & IPMSG_FILEATTACHOPT) ) //  用户发送的还有文件
{
    strcat(buf, "\r\n " FILEATTACH_MSGSTR " ");

    char fname[MAX_PATH], *ptr = buf + strlen(buf);

    for ( int cnt = 0; cnt < shareInfo->fileCnt
     && ptr-buf < sizeof(buf) - MAX_PATH; cnt++ )
    {
     //使得 fname 仅保存文件名部分

     ForcePathToFname( shareInfo->fileInfo[cnt]->Fname(), fname );

     //加入到 buf 中，加入文件分割符“，”或者什么也没有（到了最后了）

     ptr += wsprintf(ptr, "%s%s", fname, cnt + 1 == shareInfo->fileCnt ? "" : ", ");
    }
}

//写入结尾的结束符

strcat(buf, "\r\n-----------------------------------\r\n");
```

```
//首先写入 buf，然后写入 msg，最后写入换行

if ( Write(buf) && Write(msg) && Write("\r\n\r\n") )
{
   return TRUE;
}
else
{
   return FALSE;
}
}

inline int bit_cnt(unsigned char c)
{

//测试 char c 中有多少个 1，返回 bit 个 1

for ( int bit = 0; c; c >>= 1 )
{
   if ( c & 1 )
   {
    bit++;
   }
}

return bit;
}

// key  是字符串 password 中所有字符的 ascii 值之和

//返回 password 字符串中所有的字符的和，仅保留后 8 位

inline char make_key(char *password)
{
char key = 0;

while ( *password )
{
   key += *password++;
}

return key;
}
```

```
//将字符串 str 写入到  cfg 的 logFile 里面

BOOL LogMng::Write(LPCSTR str)
{
BOOL ret = FALSE;

int    len = strlen(str);//len 保存了字符串的长度

//如果文件名为空，则返回

if ( cfg->LogCheck == FALSE || *cfg->LogFile == 0 )
{
   return TRUE;
}

HANDLE    fh;

DWORD    size;

/*
The CreateFile function creates or opens the following objects and
returns a handle that can be used to access the object:
If the function fails, the return value is INVALID_HANDLE_VALUE.

打开要写入的文件

*/

if ((fh = ::CreateFile(cfg->LogFile,
   GENERIC_WRITE, FILE_SHARE_READ|FILE_SHARE_WRITE, NULL,
   OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0)) != INVALID_HANDLE_VALUE)
{
   //移动文件指针到文件的最后

   ::SetFilePointer(fh, 0, 0, FILE_END);

   //把 str 写入文件

   ret = ::WriteFile(fh, str, len, &size, NULL);

   //关闭文件句柄

   ::CloseHandle(fh);
}

return ret;
```

```
}

void LogMng::StrictLogFile(char *logFile)
{
/*
如果 logFile 路径名不完整则获得整个完整的路径名保存在
logFile 中。
如果\\ 没有在 logFile 中出现过，则 strstr 返回 NULL 0
The GetFullPathName function retrieves the full
path and file name of a specified file.

DWORD GetFullPathName(
LPCTSTR lpFileName, // file name
DWORD nBufferLength, // size of path buffer
LPTSTR lpBuffer,      // path buffer
LPTSTR *lpFilePart    // address of file name in path
);
Parameters
lpFileName
[in] Pointer to a null-terminated string that specifies
a valid file name. This string can use either short
(the 8.3 form) or long file names.
nBufferLength
[in] Specifies the size, in TCHARs, of the buffer for the
drive and path.
lpBuffer
[out] Pointer to a buffer that receives the null-terminated
string for the name of the drive and path.
lpFilePart
[out] Pointer to a buffer that receives the address
(in lpBuffer) of the final file name component in the path.

Return Values

If the function succeeds, the return value is the length,

in TCHARs, of the string copied to lpBuffer, not including

   the terminating null character.

If the lpBuffer buffer is too small to contain the path,
the return value is the size of the buffer, in TCHARs,
required to hold the path. Therefore, if the return value
is greater than nBufferLength, call the function again with
```

a buffer that is large enough to hold the path.

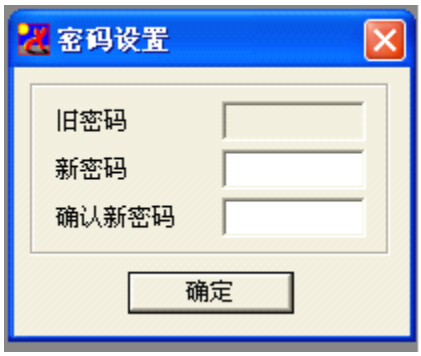
如果不是网络文件（路径中有、、），或者路径中没有：

则获得完整的路径名，保存到 logFile 中；

*/

```
if ( strstr(logFile, "\\\\") == 0 || strchr(logFile, ':') == 0 )
{
   char path[MAX_BUF], *tmp = NULL;

   if (::GetFullPathName(logFile, sizeof(path), path, &tmp) && tmp)
   {
    strncpyz(logFile, path, MAX_PATH);
   }
}
}
```

（转）IPMSG 飞鸽传书——密码设置与离开对话框
2010-05-09 11:01




密码设置对话框，用来设置察看消息时需要的密码，

资源 id 如下：

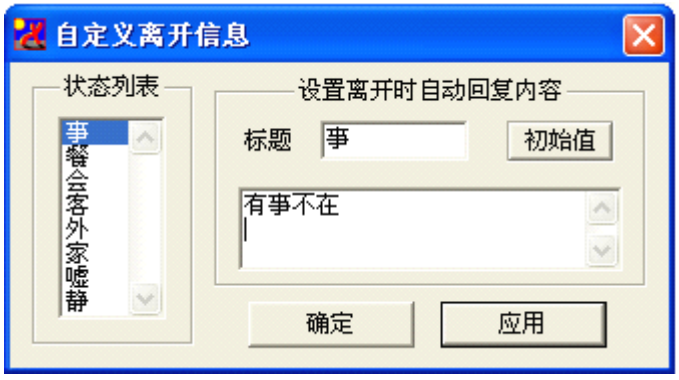
```
PASSWDCHANGE_DIALOG DIALOG DISCARDABLE 0, 0, 115, 87
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Password Settings"
FONT 8, "Verdana"
BEGIN
    EDITTEXT        OLDPASSWORD_EDIT,48,11,48,13,ES_LOWERCASE |
ES_PASSWORD |
                ES_AUTOHSCROLL
    EDITTEXT        NEWPASSWORD_EDIT,48,27,48,13,ES_LOWERCASE |
ES_PASSWORD |
```

```
                ES_AUTOHSCROLL
    EDITTEXT        NEWPASSWORD_EDIT2,48,43,48,13,ES_LOWERCASE |
ES_PASSWORD |
                ES_AUTOHSCROLL
    DEFPUSHBUTTON    "OK",IDOK,29,67,50,14
    LTEXT           "Old Password",IDC_STATIC,14,14,20,8
    LTEXT           "New Password",IDC_STATIC,14,29,21,8
    LTEXT           "Verify Password",IDC_STATIC,14,44,29,8
    GROUPBOX        "",IDC_STATIC,5,1,102,61
END

//修改密码对话框

class TPasswdChangeDlg : public TDlg
{

protected:

Cfg    *cfg;

public:

TPasswdChangeDlg(Cfg *_cfg, TWin *_parent = NULL);

virtual BOOL EvCommand(WORD wNotifyCode, WORD wID, LPARAM
hwndCtl);

//重载 create 和 command 两个函数

virtual BOOL EvCreate(LPARAM lParam);
};


TPasswdChangeDlg::TPasswdChangeDlg(Cfg *_cfg, TWin *_parent) :
TDlg(PASSWDCHANGE_DIALOG, _parent)
{
cfg = _cfg;
}

BOOL TPasswdChangeDlg::EvCommand(WORD wNotifyCode, WORD wID,
LPARAM hWndCtl)
{
switch ( wID )
{
```

```
case IDOK: //  修改完了密码,确定了
   char buf[MAX_NAMEBUF], buf2[MAX_NAMEBUF];
   //获得原来的老密码
   GetDlgItemText(OLDPASSWORD_EDIT, buf, sizeof(buf));
   //这里仅仅检查原来的密码字符的后 7 位,也就是说密码仅能使 ascii 字符
   if ( CheckPassword(cfg->PasswordStr, buf) ) //和原来的密码一致
   {
    //获得新密码
    GetDlgItemText(NEWPASSWORD_EDIT, buf, sizeof(buf));
    GetDlgItemText(NEWPASSWORD_EDIT2, buf2, sizeof(buf2));
    if ( strcmp(buf, buf2) == 0 ) //  两次输入的密码一致
    {
     //修改 cfg 的原来的密码
     MakePassword(buf, cfg->PasswordStr);
    }
    else //  弹出警告:两次输入的密码不一致
    {
     return MessageBox(NOTSAMEPASS_MSGSTR), TRUE;
    }
    //写入注册表
    cfg->WriteRegistry(CFG_GENERAL);
    EndDialog(TRUE);
   }
   else //  和原来的密码不一致，弹出设置失败对话框
   {
    SetDlgItemText(PASSWORD_EDIT, ""), MessageBox(CANTAUTH_MSGSTR);
   }
   return TRUE;
case IDCANCEL: //  取消  关闭对话框
   EndDialog(FALSE);
```

```
   return TRUE;
}
return FALSE;
}

BOOL TPasswdChangeDlg::EvCreate(LPARAM lParam)
{
GetWindowRect(&rect);

MoveWindow( rect.left + 50, rect.top + 100, rect.right - rect.left,
   rect.bottom - rect.top, FALSE );

//如果原来的密码是空，把原来的旧密码设置为不可用

if ( *cfg->PasswordStr == 0 )
{
   ::EnableWindow(GetDlgItem(OLDPASSWORD_EDIT), FALSE);
}

return TRUE;
}

//把 inputPasswd 简单加密后,判断两个密码是否相同,相同返回 true,不同返回 false

BOOL CheckPassword(const char *cfgPasswd, const char *inputPasswd)
{
char buf[MAX_NAMEBUF];

//首先把 inputpassword 转化为加密的 password

MakePassword( inputPasswd, buf );

return strcmp( buf, cfgPasswd ) == 0 ? TRUE : FALSE;
}

//把 inputpassword 的每个字符取反后取后 7 位进行简单的加密处理

void MakePassword(const char *inputPasswd, char *outputPasswd)
{
while ( *inputPasswd ) //  进行简单的加密处理
{
   *outputPasswd++ = ((~*inputPasswd++) & 0x7f); //
}

*outputPasswd = 0;
```

}




自定义离开信息对话框，可以设置自己的离开状态，以及自动回复的文字。

资源 id：

```
ABSENCE_DIALOG DIALOGEX 0, 0, 219, 105
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Absence Setting"
FONT 8, "Verdana", 0, 0, 0x1
BEGIN
    EDITTEXT        ABSENCE_EDIT,74,35,133,36,ES_MULTILINE |
ES_AUTOVSCROLL |
                ES_NOHIDESEL | ES_WANTRETURN | WS_VSCROLL
    DEFPUSHBUTTON    "Set && Absence",IDOK,71,83,70,15
    PUSHBUTTON       "Set",SET_BUTTON,177,84,35,14
    LISTBOX         ABSENCE_LIST,12,17,43,67,LBS_SORT |
LBS_NOINTEGRALHEIGHT |
                LBS_DISABLENOSCROLL | WS_VSCROLL | WS_TABSTOP
    EDITTEXT        ABSENCEHEAD_EDIT,103,17,47,13,ES_AUTOHSCROLL
    LTEXT           "Title",IDC_STATIC,81,19,18,10
    GROUPBOX        "Absence List",IDC_STATIC,7,4,54,87,BS_CENTER
    GROUPBOX        "Setting contents",IDC_STATIC,68,4,144,73,BS_CENTER
    PUSHBUTTON      "Initialize",ABSENCEINIT_BUTTON,168,16,38,13,NOT
                WS_TABSTOP,WS_EX_STATICEDGE
END

//自定义的离开对话框

class TAbsenceDlg : public TDlg
{

protected:
```

```
Cfg    *cfg;

int    currentChoice;

char (*tmpAbsenceStr)[MAX_PATH];

char (*tmpAbsenceHead)[MAX_NAMEBUF];

void SetData(void);

void GetData(void);

public:

TAbsenceDlg(Cfg *_cfg, TWin *_parent = NULL);

virtual ~TAbsenceDlg();

virtual BOOL EvCreate(LPARAM lParam);

virtual BOOL EvNcDestroy(void);

virtual BOOL EvCommand(WORD wNotifyCode, WORD wID, LPARAM
hwndCtl);

virtual BOOL EventUser(UINT uMsg, WPARAM wParam, LPARAM lParam);
};

/*
自定义的离开对话框
*/

extern char *DefaultAbsence[], *DefaultAbsenceHead[];

TAbsenceDlg::TAbsenceDlg(Cfg *_cfg, TWin *_parent) :
TDlg(ABSENCE_DIALOG, _parent)
{
cfg = _cfg;

//hAccel 保存了加速键

hAccel = ::LoadAccelerators(TApp::hI, (LPCSTR)IPMSG_ACCEL);
}

TAbsenceDlg::~TAbsenceDlg()
{
```

```
}

BOOL TAbsenceDlg::EvCreate(LPARAM lParam)
{
SetDlgIcon( hWnd );

if ( rect.left == CW_USEDEFAULT )
{
   DWORD val = GetMessagePos();

   POINTS pos = MAKEPOINTS(val);

   GetWindowRect(&rect);

   int cx = ::GetSystemMetrics(SM_CXFULLSCREEN),
    cy = ::GetSystemMetrics(SM_CYFULLSCREEN);

   int xsize = rect.right - rect.left,
    ysize = rect.bottom - rect.top;

   int x = pos.x - xsize / 2,
    y = pos.y - ysize;

   if ( x + xsize > cx )
   {
    x = cx - xsize;
   }

   if ( y + ysize > cy )
   {
    y = cy - ysize;
   }

   MoveWindow(x > 0 ? x : 0, y > 0 ? y : 0, xsize, ysize, FALSE);
}
else
{
   MoveWindow(rect.left, rect.top, rect.right - rect.left,
    rect.bottom - rect.top, FALSE);
}

typedef char MaxBuf[MAX_PATH];

typedef char MaxHead[MAX_NAMEBUF];
```

```
tmpAbsenceStr = new MaxBuf[cfg->AbsenceMax];

tmpAbsenceHead = new MaxHead[cfg->AbsenceMax];

//根据 cfg 来设置各个变量的值

SetData();

return TRUE;
}

//释放 tmp 数组占用的空间

BOOL TAbsenceDlg::EvNcDestroy(void)
{
delete [] tmpAbsenceHead;

delete [] tmpAbsenceStr;

return TRUE;
}

BOOL TAbsenceDlg::EvCommand(WORD wNotifyCode, WORD wID, LPARAM
hWndCtl)
{
switch ( wID )
{

case IDOK:

   //根据用户的设置修改相应的变量

   GetData();

   //把 absence 状态写入注册表

   cfg->WriteRegistry(CFG_ABSENCE);

   //把 AbsenceCheck 设置为 FALSE，也就是状态设置为离开状态了！！

   cfg->AbsenceCheck = FALSE;

   //向父窗口发送 MENU_ABSENCE 消息！修改图标等

   ::PostMessage(GetMainWnd(), WM_COMMAND,
```

```
    MENU_ABSENCE, 0);
    EndDialog(TRUE);
    return TRUE;
case SET_BUTTON: //  应用按钮
    //获得用户的设置
    GetData();
    //写入注册表
    cfg->WriteRegistry(CFG_ABSENCE);
    if ( cfg->AbsenceCheck )
    {
     //设置为离开状态
     cfg->AbsenceCheck = FALSE;
     //修改为离开的图标
     ::PostMessage(GetMainWnd(), WM_COMMAND, MENU_ABSENCE, 0);
    }
    EndDialog(FALSE);
    return TRUE;
case IDCANCEL: //  取消了
    EndDialog(FALSE);
    return TRUE;
case ABSENCEINIT_BUTTON: //  设置为初始的默认值
    //DefaultAbsenceHead 和 DefaultAbsence 保存了默认的标题和内容
    SetDlgItemText(ABSENCEHEAD_EDIT, DefaultAbsenceHead[currentChoice]);
    SetDlgItemText(ABSENCE_EDIT, DefaultAbsence[currentChoice]);
```

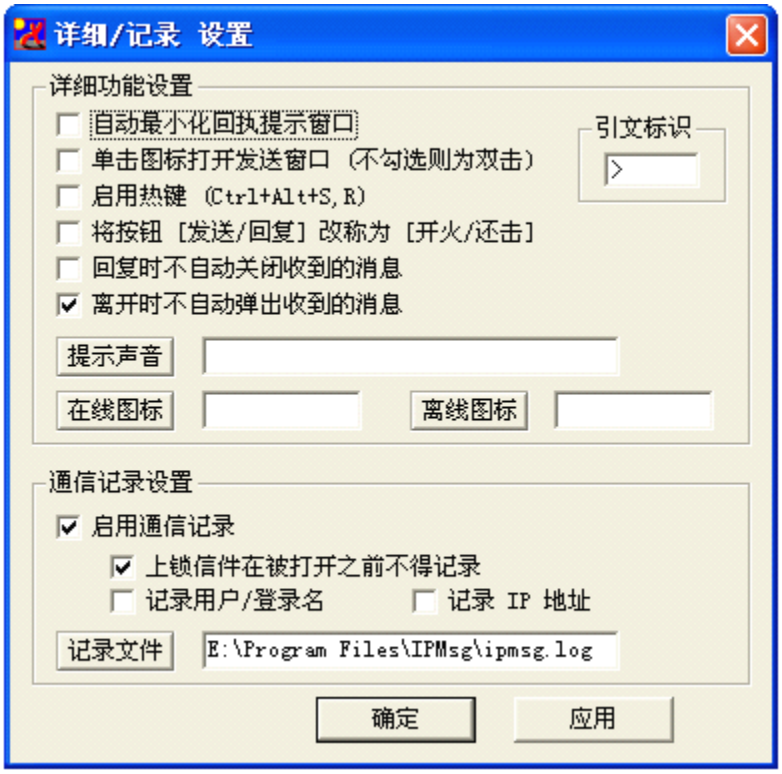
```
    return TRUE;

case HIDE_ACCEL: // 隐藏加速健，隐藏/显示所有的子窗口 这里就是 ctr + d ，
哈哈

    ::PostMessage(GetMainWnd(), WM_HIDE_CHILDWIN, 0, 0);

    return TRUE;

case ABSENCE_LIST: // 用户在离开列表中的单击事件

    if (wNotifyCode == LBN_SELCHANGE) // 选择的 list 项变化了
    {
     int    index;

     if ( ( index = (int)SendDlgItemMessage(ABSENCE_LIST, LB_GETCURSEL, 0,
0))
            != LB_ERR ) // 获得用户选择的那一项，保存到 index 中
     {
      //首先如果用户修改了 currentChoice 项的内容，则保存之

      char oldAbsenceHead[MAX_NAMEBUF];

      strcpy(oldAbsenceHead, tmpAbsenceHead[currentChoice]);

      GetDlgItemText(ABSENCEHEAD_EDIT, tmpAbsenceHead[currentChoice],
       MAX_NAMEBUF);

      GetDlgItemText(ABSENCE_EDIT, tmpAbsenceStr[currentChoice],
       MAX_PATH);

      //新的标题和原来的标题不同

      if ( strcmp(oldAbsenceHead, tmpAbsenceHead[currentChoice]) )
      {
       //删除 ABSENCE_LIST 中原来的标题，加入新的标题

       SendDlgItemMessage(ABSENCE_LIST, LB_DELETESTRING,
        currentChoice, 0);

       SendDlgItemMessage(ABSENCE_LIST, LB_INSERTSTRING,
        currentChoice, (LPARAM)tmpAbsenceHead[currentChoice]);

       if ( currentChoice == index )
       {
```

```
        //设置为选中当前的 currentChoice 项

        SendDlgItemMessage(ABSENCE_LIST, LB_SETCURSEL,
         currentChoice, 0);

        return TRUE;
       }
      }

      //把 currentChoice 设置为当前的 index 项

      currentChoice = index;

      //选中 currentChoice 项，并且根据 tmp 数组设置标题和内容

      PostMessage(WM_DELAYSETTEXT, 0, 0);
     }
    }
    else if ( wNotifyCode == LBN_DBLCLK )
    {
     //双击 list 中的某一项相当于 ok！！！

     PostMessage(WM_COMMAND, IDOK, 0);
    }

    return TRUE;
}

return FALSE;
}

/*

default 的用户事件处理函数 EventUser(UINT uMsg, WPARAM wParam,
LPARAM lParam)

User  事件的回调函数  Event CallBack

如果是 WM_DELAYSETTEXT 消息，则把编辑框的内容设置为
tmpAbsenceStr[currentChoice]

的内容，而且选中 currentChoice 这一项；

*/

BOOL TAbsenceDlg::EventUser(UINT uMsg, WPARAM wParam, LPARAM
```

```
lParam)
{
switch ( uMsg )
{

case WM_DELAYSETTEXT:
   {
    int    len = strlen(tmpAbsenceStr[currentChoice]);

    SetDlgItemText(ABSENCEHEAD_EDIT, tmpAbsenceHead[currentChoice]);

    SetDlgItemText(ABSENCE_EDIT, tmpAbsenceStr[currentChoice]);

    SendDlgItemMessage(ABSENCE_EDIT, EM_SETSEL, (WPARAM)len,
(LPARAM)len);
   }

   return TRUE;

}

return FALSE;
}

//根据 cfg 的内容来设置添加各种离开状态，拷贝到 tmp 数组中

void TAbsenceDlg::SetData(void)
{
for ( int cnt = 0; cnt < cfg->AbsenceMax; cnt++ )
{
   strcpy(tmpAbsenceHead[cnt], cfg->AbsenceHead[cnt]);

   strcpy(tmpAbsenceStr[cnt], cfg->AbsenceStr[cnt]);

   SendDlgItemMessage(ABSENCE_LIST, LB_INSERTSTRING, (WPARAM)-1,
    (LPARAM)cfg->AbsenceHead[cnt]);
}

//当前的选择设置为 cfg->AbsenceChoice

currentChoice = cfg->AbsenceChoice;

//发送一个自定义的 WM_DELAYSETTEXT 消息？？不知道为什么？？

PostMessage(WM_DELAYSETTEXT, 0, 0);
```

```
//选中第 currentChoice 个离开状态

SendDlgItemMessage(ABSENCE_LIST, LB_SETCURSEL, currentChoice, 0);
}

void TAbsenceDlg::GetData(void)
{
//获得离开时的标题

GetDlgItemText(ABSENCEHEAD_EDIT, tmpAbsenceHead[currentChoice],
MAX_NAMEBUF);

//离开时的自动回复

GetDlgItemText(ABSENCE_EDIT, tmpAbsenceStr[currentChoice], MAX_PATH);

//把 tmpAbsenceHead 和 tmpAbsenceStr 的各项内容拷贝到 cfg 的对应数组中

for ( int cnt = 0; cnt < cfg->AbsenceMax; cnt++ )
{
   strcpy(cfg->AbsenceHead[cnt], tmpAbsenceHead[cnt]);

   strcpy(cfg->AbsenceStr[cnt], tmpAbsenceStr[cnt]);
}

//设置当前选择的 absence 状态

cfg->AbsenceChoice = currentChoice;
}
```

（转）IPMSG 飞鸽传书——详细\记录  设置对话框
2010-05-09 11:00




详细功能设置对话框，保存了诸如最小化回执窗口和声音图标等选项。

资源 id 定义如下：

```
LOG_DIALOG DIALOG DISCARDABLE 0, 0, 194, 226
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Details & Log Settings"
FONT 8, "Verdana"
BEGIN
    CONTROL         "OpenCheckDialog is icon",MSGMINIMIZE_CHECK,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,15,16,102,10
    CONTROL         "Don't close ReceiveDialog, when reply",NOERASE_CHECK,
                    "Button",BS_AUTOCHECKBOX | WS_TABSTOP,15,64,150,10
    CONTROL         "Use Hotkey (Alt+Ctl+S,R)",HOTKEY_CHECK,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,15,40,109,10
    CONTROL         "Change Send/Reply -> Fire/Intercept",ABNORMALBTN_CHECK,
                    "Button",BS_AUTOCHECKBOX | WS_TABSTOP,15,52,152,10
    CONTROL         "One click open SendDialog",ONECLICK_CHECK,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,15,28,106,10
    CONTROL         "Don't popup ReceiveDialog, if absence mode",
                    ABSENCENONPOPUP_CHECK,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,15,76,166,10
    EDITTEXT        QUOTE_EDIT,136,24,39,12,ES_AUTOHSCROLL
    PUSHBUTTON      "Sound",SOUNDFILE_BUTTON,13,91,30,14
```

```
    EDITTEXT        SOUND_EDIT,48,91,133,13,ES_AUTOHSCROLL
    PUSHBUTTON      "Icon",MAINICON_BUTTON,13,109,29,14
    EDITTEXT        MAINICON_EDIT,46,110,50,13,ES_AUTOHSCROLL
    PUSHBUTTON      "RevIcon",REVICON_BUTTON,100,109,29,14
    EDITTEXT        REVICON_EDIT,132,110,49,13,ES_AUTOHSCROLL
    CONTROL         "Logging available",LOG_CHECK,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,15,147,110,10
    CONTROL         "Locked seal message is logged after opened",
                    PASSWDLOG_CHECK,"Button",BS_AUTOCHECKBOX | WS_TABSTOP,21,
                    160,159,10
    PUSHBUTTON      "LogFile",LOGFILE_BUTTON,13,185,47,14
    EDITTEXT        LOG_EDIT,64,185,118,13,ES_AUTOHSCROLL
    DEFPUSHBUTTON   "OK",IDOK,70,207,54,15
    GROUPBOX        "Details",IDC_STATIC,7,4,180,124
    GROUPBOX        "Log settings",LOG_GROUP,7,133,180,71
    GROUPBOX        "QuoteStr",IDC_STATIC,131,12,50,29
    PUSHBUTTON      "apply",SET_BUTTON,158,206,24,13
    CONTROL         "LogonName",NICKNAME_CHECK,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,21,172,63,9
    CONTROL         "IP address",IPADDR_CHECK,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,95,172,68,9
END

//详细\纪录设置对话框

class TLogDlg : public TDlg
{

protected:

Cfg    *cfg;

void SetData(void);

void GetData(void);

public:

TLogDlg(Cfg *_cfg, TWin *_parent = NULL);

BOOL OpenSoundFileDlg(void);

virtual BOOL EvCommand(WORD wNotifyCode, WORD wID, LPARAM hwndCtl);

virtual BOOL EvCreate(LPARAM lParam);
```

};

实现文件：

```
static char *logdlg_id =
"@(#)Copyright (C) H.Shirouzu 1996-2002    logdlg.cpp Ver2.00";
/*
========================================================================
Project Name     : IP Messenger for Win32
Module Name      : Log Dialog
Create       : 1996-08-12(Mon)
Update       : 2002-11-03(Sun)
Copyright      : H.Shirouzu
Reference      :
========================================================================
*/

#include "tlib.h"
#include "resource.h"
#include "ipmsg.h"
#include "msgstr.h"

//详细/记录设置对话框

TLogDlg::TLogDlg(Cfg *_cfg, TWin *_parent) : TDlg(LOG_DIALOG, _parent)
{
cfg = _cfg;
}

BOOL TLogDlg::EvCommand(WORD wNotifyCode, WORD wID, LPARAM hWndCtl)
{
switch (wID)
{

case IDOK:

case SET_BUTTON:

   //获得用户的设置

   GetData();

   //如果是网络驱动器而且严格检查 log 而且 logFile 中没有\则弹出警告
```

```
    //"log 文件可能在远端驱动器上!!!"

    if ( GetDriveType(NULL) == DRIVE_REMOTE && cfg->LogCheck
     && strchr(cfg->LogFile, '\\') == NULL )
    {
     MessageBox(LOGALERT_MSGSTR);
    }

    //如果需要保存完整的 log 文件路径名，则把 log 文件的完整路径设置到 cfg->LogFile 中

    if ( cfg->LogCheck )
    {
     LogMng::StrictLogFile(cfg->LogFile);
    }

    //把 cfg 的内容写入注册表

    cfg->WriteRegistry(CFG_GENERAL);

    //重新设置主窗口的 icon

    ::SendMessage(GetMainWnd(), WM_IPMSG_INITICON, 0, 0);

    //设置 hWnd 的 icon,也就是当前窗口的 icon

    SetDlgIcon(hWnd);

    //设置 cfg 的 hotkey

    SetHotKey(cfg);

    if ( wID == IDOK )
    {
     EndDialog(TRUE);
    }

    return TRUE;

  case IDCANCEL:

    EndDialog(FALSE);

    return TRUE;

  case MAINICON_BUTTON:
```

```
case REVICON_BUTTON: // 选择 icon 对话框

   //注意这里设置了离线图标和在线图标两个图标！！！！

   OpenFileDlg(this).Exec(wID == MAINICON_BUTTON ? MAINICON_EDIT
    : REVICON_EDIT, OPENFILEICON_MSGSTR );

   return TRUE;

case SOUNDFILE_BUTTON: // 打开声音文件，windows 目录下的 media 文件夹的内容

   OpenSoundFileDlg();

   return TRUE;

case LOGFILE_BUTTON: // 设置 log 文件，注意这里并没有保存到 log 文件中！！

   OpenFileDlg(this, OpenFileDlg::SAVE).Exec(LOG_EDIT, OPENFILELOG_MSGSTR);

   return TRUE;
}

return FALSE;
}

BOOL TLogDlg::EvCreate(LPARAM lParam)
{
GetWindowRect(&rect);

int cx = ::GetSystemMetrics(SM_CXFULLSCREEN),
   cy = ::GetSystemMetrics(SM_CYFULLSCREEN);

int x = rect.left + 50, y = rect.top + 50;

int xsize = rect.right - rect.left,
   ysize = rect.bottom - rect.top;

if (x + xsize > cx)
{
   x = cx - xsize;
}
if (y + ysize > cy)
{
   y = cy - ysize;
}
```

```
MoveWindow(x, y, xsize, ysize, FALSE);

//把 cfg 的 LogFile 设置为完整路径

if ( cfg->LogCheck )
{
   LogMng::StrictLogFile(cfg->LogFile);
}

//根据 cfg 来设置各项的内容

SetData();

return TRUE;
}

//根据 cfg 来设置各个项的内容

void TLogDlg::SetData(void)
{
SendDlgItemMessage(CONTROLIME_CHECK, BM_SETCHECK, cfg->ControlIME, 0);

SendDlgItemMessage(MSGMINIMIZE_CHECK, BM_SETCHECK, cfg->MsgMinimize, 0);

SendDlgItemMessage(HOTKEY_CHECK, BM_SETCHECK, cfg->HotKeyCheck, 0);

SendDlgItemMessage(NOERASE_CHECK, BM_SETCHECK, cfg->NoErase, 0);

SendDlgItemMessage(ABNORMALBTN_CHECK, BM_SETCHECK, cfg->AbnormalButton, 0);

SendDlgItemMessage(ONECLICK_CHECK, BM_SETCHECK, cfg->OneClickPopup, 0);

SendDlgItemMessage(ABSENCENONPOPUP_CHECK, BM_SETCHECK,
cfg->AbsenceNonPopup, 0);

SendDlgItemMessage(LOG_CHECK, BM_SETCHECK, cfg->LogCheck, 0);

SendDlgItemMessage(PASSWDLOG_CHECK, BM_SETCHECK, cfg->PasswdLogCheck, 0);

SetDlgItemText(QUOTE_EDIT, cfg->QuoteStr);

SetDlgItemText(SOUND_EDIT, cfg->SoundFile);

SetDlgItemText(MAINICON_EDIT, cfg->IconFile);
```

```
SetDlgItemText(REVICON_EDIT, cfg->RevIconFile);

SetDlgItemText(LOG_EDIT, cfg->LogFile);

SendDlgItemMessage(NICKNAME_CHECK, BM_SETCHECK, !cfg->NickNameCheck, 0);

SendDlgItemMessage(IPADDR_CHECK, BM_SETCHECK, cfg->IPAddrCheck, 0);
}

//获得用户设置的各项的内容

void TLogDlg::GetData(void)
{
cfg->ControlIME = (int)SendDlgItemMessage(CONTROLIME_CHECK, BM_GETCHECK, 0, 0);

cfg->MsgMinimize = (int)SendDlgItemMessage(MSGMINIMIZE_CHECK, BM_GETCHECK, 0,
0);

cfg->HotKeyCheck = (int)SendDlgItemMessage(HOTKEY_CHECK, BM_GETCHECK, 0, 0);

cfg->NoErase = (int)SendDlgItemMessage(NOERASE_CHECK, BM_GETCHECK, 0, 0);

cfg->AbnormalButton = (int)SendDlgItemMessage(ABNORMALBTN_CHECK,
BM_GETCHECK, 0, 0);

cfg->OneClickPopup = (int)SendDlgItemMessage(ONECLICK_CHECK, BM_GETCHECK, 0, 0);

cfg->AbsenceNonPopup = (int)SendDlgItemMessage(ABSENCENONPOPUP_CHECK,
BM_GETCHECK, 0, 0);

cfg->LogCheck = (int)SendDlgItemMessage(LOG_CHECK, BM_GETCHECK, 0, 0);

cfg->PasswdLogCheck = (int)SendDlgItemMessage(PASSWDLOG_CHECK, BM_GETCHECK, 0,
0);

GetDlgItemText(QUOTE_EDIT, cfg->QuoteStr, sizeof(cfg->QuoteStr));

GetDlgItemText(MAINICON_EDIT, cfg->IconFile, sizeof(cfg->IconFile));

GetDlgItemText(SOUND_EDIT, cfg->SoundFile, sizeof(cfg->SoundFile));

GetDlgItemText(REVICON_EDIT, cfg->RevIconFile, sizeof(cfg->RevIconFile));
```

```
GetDlgItemText(LOG_EDIT, cfg->LogFile, sizeof(cfg->LogFile));

cfg->NickNameCheck = !(int)SendDlgItemMessage(NICKNAME_CHECK, BM_GETCHECK, 0,
0);

cfg->IPAddrCheck = (int)SendDlgItemMessage(IPADDR_CHECK, BM_GETCHECK, 0, 0);
}

BOOL TLogDlg::OpenSoundFileDlg(void)
{
char szDirName[MAX_PATH] = "", dirNameBak[MAX_PATH];

//把当前路径备份起来

::GetCurrentDirectory(sizeof(dirNameBak), dirNameBak);

//获得 windows 的路径

::GetWindowsDirectory(szDirName, sizeof(szDirName));

//szDirName 现在指向了 windows 目录下的 media 文件夹

if ( IsNewShell() )
{
   strcat(szDirName, "\\Media");
}

//弹出一个打开文件对话框

BOOL ret = OpenFileDlg(this).Exec(SOUND_EDIT, OPENFILESND_MSGSTR, szDirName);

//恢复当前目录

::SetCurrentDirectory(dirNameBak);

return ret;
}
```

URL 链接设置对话框，设置了 http 协议的打开程序等。

资源 id 定义如下：

```
URL_DIALOG DIALOG DISCARDABLE 0, 0, 215, 166
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Clickable URL Settings"
FONT 8, "Verdana"
```

```
BEGIN
    CONTROL         "Useing default browser, if not specified",
                    DEFAULTURL_CHECK,"Button",BS_AUTOCHECKBOX | WS_TABSTOP,
                    15,21,179,10
    LISTBOX         URL_LIST,14,57,49,35,LBS_SORT | LBS_NOINTEGRALHEIGHT |
                    LBS_DISABLENOSCROLL | WS_VSCROLL | WS_TABSTOP
    PUSHBUTTON      "File",ADD_BUTTON,173,57,26,13,BS_CENTER
    EDITTEXT        URL_EDIT,67,75,134,14,ES_AUTOHSCROLL
    CONTROL         "Double click execute(like double click in explore)",
                    SHELLEXEC_CHECK,"Button",BS_AUTOCHECKBOX | WS_TABSTOP,16,
                    121,184,10
    DEFPUSHBUTTON   "OK",IDOK,86,145,50,14
    GROUPBOX        "Double click behavior except URL",IDC_STATIC,7,107,201,
                    32
    GROUPBOX        "Clickable URL -- Default Settings",IDC_STATIC,7,7,201,
                    31
    GROUPBOX        "Clickable URL -- Protocol Settings",IDC_STATIC,7,44,201,
                    55
    LTEXT           "Execute program",IDC_STATIC,71,60,36,10
    PUSHBUTTON      "apply",SET_BUTTON,182,146,24,13
END

//url 设置对话框，根据用户的设定，双击给定协议的信息时会打开对应的程序

class TUrlDlg : public TDlg
{

protected:

Cfg    *cfg;

TList tmpUrlList;

char currentProtocol[MAX_NAMEBUF];

void SetData(void);

void GetData(void);

void SetCurrentData(void);

public:

TUrlDlg(Cfg *_cfg, TWin *_parent = NULL);
```

```
virtual ~TUrlDlg();

virtual BOOL EvCreate(LPARAM lParam);

virtual BOOL EvNcDestroy(void);

virtual BOOL EvCommand(WORD wNotifyCode, WORD wID, LPARAM hwndCtl);
};
```

实现文件：

```
/*
TURL 对话框，这里设定了 currentProtocol 为 0
*/
TUrlDlg::TUrlDlg(Cfg *_cfg, TWin *_parent) : TDlg(URL_DIALOG, _parent)
{
cfg = _cfg;

*currentProtocol = 0;
}

TUrlDlg::~TUrlDlg()
{
}

BOOL TUrlDlg::EvCreate(LPARAM lParam)
{
GetWindowRect(&rect);

int cx = ::GetSystemMetrics(SM_CXFULLSCREEN),
   cy = ::GetSystemMetrics(SM_CYFULLSCREEN);

int x = rect.left + 10, y = rect.top + 50;

int xsize = rect.right - rect.left,
   ysize = rect.bottom - rect.top;

if (x + xsize > cx)
{
   x = cx - xsize;
}

if (y + ysize > cy)
{
```

```
    y = cy - ysize;
}

//移动窗口到指定位置

MoveWindow(x, y, xsize, ysize, FALSE);

//根据 cfg 类设置各项的内容

SetData();

//选中 urllist 中的第 0 项

SendDlgItemMessage(URL_LIST, LB_SETCURSEL, 0, 0);

//发送一个 URL_LIST 消息显示第 0 个 protocol 对应的 program

EvCommand(0, URL_LIST, 0);

return TRUE;
}

//删除 tmpUrlList 中的所有项，释放资源

BOOL TUrlDlg::EvNcDestroy(void)
{
UrlObj *urlObj;

while ((urlObj = (UrlObj *)tmpUrlList.TopObj()) != NULL)
{
    tmpUrlList.DelObj(urlObj);

    delete urlObj;
}

return TRUE;
}

BOOL TUrlDlg::EvCommand(WORD wNotifyCode, WORD wID, LPARAM hWndCtl)
{
switch (wID)
{

case IDOK: case SET_BUTTON:
```

```
    //获得用户的输入

    GetData();

    //写入注册表

    cfg->WriteRegistry(CFG_CLICKURL);

    if ( wID == IDOK )
    {
     EndDialog(TRUE);
    }

    return TRUE;

  case IDCANCEL: //  取消，直接返回

    EndDialog(FALSE);

    return TRUE;

  case ADD_BUTTON: //  浏览按钮
    {
     char protocol[MAX_LISTBUF], buf[MAX_LISTBUF];

     int    index;

     UrlObj *obj;

     //获得当前选中的 protocol 的 index，保存到 index 中

     //获得当前选中的 protocol 的内容不是空，而且这个新协议在原来中有

     if ( (index = (int)SendDlgItemMessage(URL_LIST, LB_GETCURSEL, 0, 0))
       != LB_ERR && SendDlgItemMessage(URL_LIST, LB_GETTEXT,
          (WPARAM)index, (LPARAM)protocol) != LB_ERR
       && (obj = SearchUrlObj(&tmpUrlList, protocol)) != NULL )
     {
      //#define EXECPROGRAM_MSGSTR "%s Execute Program"

      wsprintf(buf, EXECPROGRAM_MSGSTR, protocol);

      //这里的 buf 是标题比如：“http  打开程序”

      //#define OPENFILEPRG_MSGSTR "Program(*.exe)\0*.exe\0All Files(*.*)\0*.*\0\0"
```

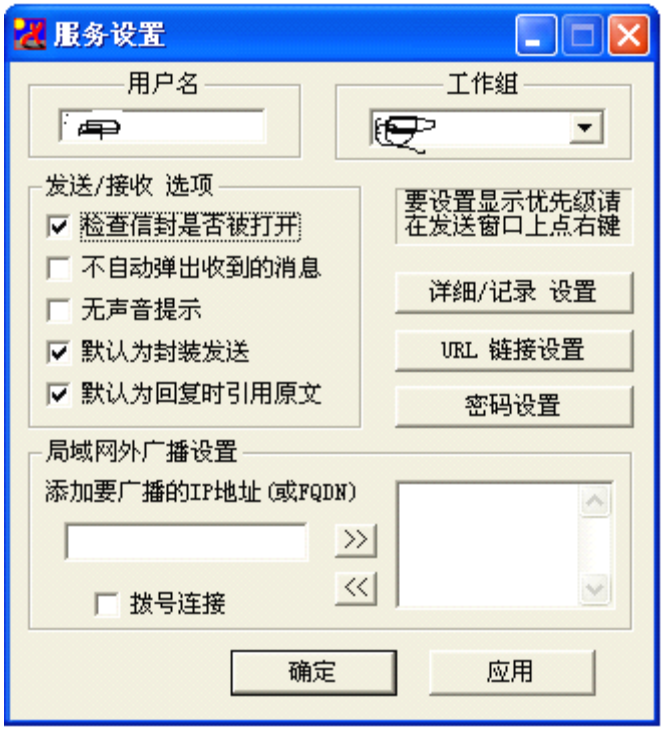
```
    //调用 OpenFileDlg 打开对话框，最后路径保存在 edit 控件中

    //也就是说双击指定的网址打开对应的程序！！！！和网址！！！

    OpenFileDlg(this).Exec(URL_EDIT, buf, OPENFILEPRG_MSGSTR);
   }
  }

  return TRUE;

case URL_LIST: // 根据用户选中的 url 来显示对应的 program，当用户选择了某一个 urllist 项时触发
  {
   char protocol[MAX_LISTBUF];

   int   index;

   UrlObj *obj;

   if ( ( index = (int)SendDlgItemMessage(URL_LIST, LB_GETCURSEL, 0, 0)) != LB_ERR
    && SendDlgItemMessage(URL_LIST, LB_GETTEXT, (WPARAM)index,
    (LPARAM)protocol) != LB_ERR
    && (obj = SearchUrlObj(&tmpUrlList, protocol)) != NULL )
   {
    //在 urllist 中获得 program 的文字，显示到 url_edit 中

    SetCurrentData();

    SetDlgItemText(URL_EDIT, obj->program);

    //把当前的协议拷贝到 currentProtocol 中

    strncpyz(currentProtocol, protocol, sizeof(currentProtocol));
   }
  }

  return TRUE;
}

return FALSE;
}

//如果当前协议在 tmpUrlList 中可以找到，则设置对应项的程序的文字

void TUrlDlg::SetCurrentData(void)
```

```
{
UrlObj *obj;

if ( ( obj = SearchUrlObj(&tmpUrlList, currentProtocol)) != NULL )
{
   GetDlgItemText(URL_EDIT, obj->program, sizeof(obj->program));
}
}

//把 cfg->urlList 中所有的项的协议加入 URL_LIST 中

void TUrlDlg::SetData(void)
{
for ( UrlObj *obj = (UrlObj *)cfg->urlList.TopObj(); obj != NULL;
                    obj = (UrlObj *)cfg->urlList.NextObj(obj) )
{
   UrlObj *tmp_obj = new UrlObj;

   strcpy(tmp_obj->protocol, obj->protocol);

   strcpy(tmp_obj->program, obj->program);

   tmpUrlList.AddObj(tmp_obj);

   SendDlgItemMessage(URL_LIST, LB_INSERTSTRING,
    (WPARAM)-1, (LPARAM)obj->protocol);
}

//根据 cfg 来设置 checkbox 的状态

SendDlgItemMessage(DEFAULTURL_CHECK, BM_SETCHECK, cfg->DefaultUrl, 0);

SendDlgItemMessage(SHELLEXEC_CHECK, BM_SETCHECK, cfg->ShellExec, 0);
}

//插入 cfg 的 urllist 中新加入的项

void TUrlDlg::GetData(void)
{
//首先根据用户选择的程序来设置 tmpUrlList 中对应项的 program 的文字

SetCurrentData();

for ( UrlObj *tmp_obj = (UrlObj *)tmpUrlList.TopObj(); tmp_obj != NULL;
                    tmp_obj = (UrlObj *)tmpUrlList.NextObj(tmp_obj) )
```

```
{
   //在 cfg 中找 tmp 的 protocol，如果没有找到，则在 cfg 的 urlist 中插入新项

   UrlObj *obj = SearchUrlObj(&cfg->urlList, tmp_obj->protocol);

   if ( obj == NULL )
   {
    obj = new UrlObj;

    cfg->urlList.AddObj(obj);

    strcpy(obj->protocol, tmp_obj->protocol);
   }

   strcpy(obj->program, tmp_obj->program);
}

//根据 checkbox 来设置对应的 cfg 中的内容

cfg->DefaultUrl = (int)SendDlgItemMessage(DEFAULTURL_CHECK, BM_GETCHECK, 0, 0);

cfg->ShellExec = (int)SendDlgItemMessage(SHELLEXEC_CHECK, BM_GETCHECK, 0, 0);
}
```

（转）IPMSG 飞鸽传书——设置对话框

2010-05-09 10:58



设置对话框保存了用户的各种设置，如 nickname 组名 检查是否信封被打开等各种选项。

创建过程类似于前面的关于对话框。

设置对话框的资源定义如下：

```
SETUP_DIALOG DIALOGEX 0, 0, 186, 210
STYLE DS_MODALFRAME | DS_3DLOOK | WS_MINIMIZEBOX | WS_POPUP |
WS_CAPTION |
    WS_SYSMENU
CAPTION "IP Messenger Settings"
FONT 8, "Verdana", 0, 0, 0x1
BEGIN
    CONTROL         "Message open check",OPEN_CHECK,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,12,50,79,10
    CONTROL         "None Popup Check",NOPOPUP_CHECK,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,12,64,79,10
    CONTROL         "No receive sound",NOBEEP_CHECK,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,12,78,79,10
    CONTROL         "Default sealed check",SECRET_CHECK,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,12,92,79,10
    CONTROL         "Default quoted check",QUOTE_CHECK,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,12,106,79,10
    EDITTEXT        NICKNAME_EDIT,11,15,68,12,ES_AUTOHSCROLL
    COMBOBOX        GROUP_COMBO,97,14,77,66,CBS_DROPDOWN |
CBS_AUTOHSCROLL |
                    CBS_SORT | CBS_DISABLENOSCROLL | WS_VSCROLL | WS_TABSTOP
    PUSHBUTTON      "Details/Log Settings",LOG_BUTTON,100,71,79,14,BS_CENTER |
                    BS_VCENTER
    PUSHBUTTON      "Clickable URL",URL_BUTTON,100,88,79,14
    PUSHBUTTON      "Password Settings",PASSWORD_BUTTON,100,105,79,14
    EDITTEXT        BROADCAST_EDIT,11,154,66,13,ES_AUTOHSCROLL
    PUSHBUTTON      ">>",ADD_BUTTON,82,147,14,11
    PUSHBUTTON      "<<",DEL_BUTTON,82,161,14,11
    LISTBOX         BROADCAST_LIST,102,139,70,42,LBS_SORT |
                    LBS_NOINTEGRALHEIGHT | LBS_EXTENDEDSEL |
                    LBS_DISABLENOSCROLL | WS_VSCROLL | WS_TABSTOP
    CONTROL         "Dial-up check",DIALUP_CHECK,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,13,170,59,10
    DEFPUSHBUTTON   "OK",IDOK,62,191,52,14
    GROUPBOX        "User name",IDC_STATIC,6,3,80,29,BS_CENTER
    GROUPBOX        "Group name",IDC_STATIC,90,3,89,29,BS_CENTER
    GROUPBOX        "Send/Receive Option",IDC_STATIC,6,37,89,85
    GROUPBOX        "Broadcast Setup for Wide Area Network",IDC_STATIC,6,126,
```

```
                173,60
    LTEXT           "Ex) 192.168.10.255\r\n       (or FQDN)",IDC_STATIC,9,
                135,71,16
    PUSHBUTTON      "apply",SET_BUTTON,148,189,24,13
    LTEXT           "Sort/Filter settings are available by Right click on Send/RecvDlg",
                IDC_STATIC,99,39,80,28,0,WS_EX_STATICEDGE
END

//设置对话框

class TSetupDlg : public TDlg
{

protected:

Cfg    *cfg;

THosts *hosts;

void SetData(void);

void GetData(void);

public:

TSetupDlg(Cfg *cfg, THosts *hosts, TWin *_parent = NULL);

virtual ~TSetupDlg();

virtual BOOL EvCommand(WORD wNotifyCode, WORD wID, LPARAM hwndCtl);

virtual BOOL EvCreate(LPARAM lParam);
};
```

实现文件：

```
static char *setupdlg_id =
"@(#)Copyright (C) H.Shirouzu 1996-2002    setupdlg.cpp Ver2.00";
/*
========================================================================
Project Name     : IP Messenger for Win32
Module Name      : Setup Dialog
Create       : 1996-06-01(Sat)
Update       : 2002-11-03(Sun)
Copyright      : H.Shirouzu
```

```
Reference     :
================================================================================
*/

#include <stdio.h>
#include "tlib.h"
#include "resource.h"
#include "ipmsg.h"
#include "msgstr.h"

TSetupDlg::TSetupDlg(Cfg *_cfg, THosts *_hosts, TWin *_parent) : TDlg(SETUP_DIALOG,
_parent)
{
cfg    = _cfg;

hosts = _hosts;
}

TSetupDlg::~TSetupDlg()
{
}

//完成创建窗口的过程

BOOL TSetupDlg::EvCreate(LPARAM lParam)
{
//设置 icon 为默认的 icon

SetDlgIcon(hWnd);

//根据 cfg 来设置显示各个变量

SetData();

if ( rect.left == CW_USEDEFAULT )
{
   GetWindowRect(&rect);

   int cx = ::GetSystemMetrics(SM_CXFULLSCREEN),
    cy = ::GetSystemMetrics(SM_CYFULLSCREEN);

   int xsize = rect.right - rect.left,
    ysize = rect.bottom - rect.top;

   int x = ( cx - xsize ) / 2,
```

```
    y = ( cy - ysize ) / 2;

   if ( x + xsize > cx )
   {
    x = cx - xsize;
   }
   if ( y + ysize > cy )
   {
    y = cy - ysize;
   }

   MoveWindow ( x < 0 ? 0 : x, y < 0 ? 0 : y, xsize, ysize, FALSE );
}
else
{
   MoveWindow( rect.left, rect.top, rect.right - rect.left,
    rect.bottom - rect.top, FALSE);
}

//设置为前台窗口

SetForegroundWindow();

return TRUE;
}

BOOL TSetupDlg::EvCommand(WORD wNotifyCode, WORD wID, LPARAM hWndCtl)
{
switch ( wID )
{
case IDOK: case SET_BUTTON: // ok 按钮和应用按钮

   GetData(); //  根据用户的设置更新 cfg

   //如果是 idok 则把 broadcast 列表写入注册表,如果是应用按钮,则把全部写入注册表

   cfg->WriteRegistry(wID == IDOK ? (CFG_GENERAL|CFG_BROADCAST) : CFG_ALL);

   if ( wID == IDOK ) //  如果是 ok,直接关闭对话框了
   {
    EndDialog(TRUE);
   }

   return TRUE;
```

```
case ADD_BUTTON: //  加入新的广播地址
    {
     char buf[MAX_PATH], buf2[MAX_PATH];

     //获得用户添加的广播地址,保存到 buf 中

     if ( GetDlgItemText(BROADCAST_EDIT, buf, sizeof(buf)) <= 0 )
     {
      return TRUE;
     }

     //如果地址无效,弹出对话框警告之

     if ( ResolveAddr(buf) == 0 )
     {
      return MessageBox(CANTRESOLVE_MSGSTR), TRUE;
     }

     //如果 buf 在 BROADCAST_LIST 链表中已经存在,直接返回

     for ( int cnt = 0; SendDlgItemMessage(BROADCAST_LIST, LB_GETTEXT,
             cnt, (LPARAM)buf2) != LB_ERR; cnt++ )
     {
      if ( stricmp(buf, buf2) == 0 )
      {
       return TRUE;
      }
     }

     //buf 是新加入的地址,加入之

     SendDlgItemMessage(BROADCAST_LIST, LB_ADDSTRING, 0, (LPARAM)buf);

     SetDlgItemText(BROADCAST_EDIT, "");
    }

    return TRUE;

case DEL_BUTTON: //  删除已经加入的广播地址
    {
     char buf[MAX_PATH];

     int    index;
```

```
    //这里用户可能选中多项,所以使用了一个 while 循环

    while ( (int)SendDlgItemMessage(BROADCAST_LIST,
            LB_GETSELCOUNT, 0, 0) > 0 )
    {
     //如果没有选中的项了,直接返回

     if ( SendDlgItemMessage(BROADCAST_LIST,
      LB_GETSELITEMS, 1, (LPARAM)&index) != 1 )
     {
      break;
     }

     //获得选中的 index 项的文字到 buf 中

     SendDlgItemMessage(BROADCAST_LIST, LB_GETTEXT,
      (WPARAM)index, (LPARAM)buf);

     //把要加入的 ip 广播地址编辑框设置为删除的内容

     SetDlgItemText(BROADCAST_EDIT, buf);

     //在 broadcast 列表中删除指定的项

     if ( SendDlgItemMessage(BROADCAST_LIST, LB_DELETESTRING,
              (WPARAM)index, (LPARAM)buf) == LB_ERR )
     {
      break;
     }
    } //  循环删除直到没有选中项了为止
   }

   return TRUE;

case LOG_BUTTON: //  打开详细\设置对话框

   TLogDlg(cfg, this).Exec();

   return TRUE;

case PASSWORD_BUTTON: //  修改密码对话框

   TPasswdChangeDlg(cfg, this).Exec();

   return TRUE;
```

```
case URL_BUTTON://弹出 url 设置对话框

   TUrlDlg(cfg, this).Exec();

   return TRUE;

case IDCANCEL: case IDNO: //  取消对话框

   EndDialog(FALSE);

   return TRUE;
}

return FALSE;
}

//根据 cfg 中的各项内容来设置 item 的文字，GROUP_COMBO 是 ID

void TSetupDlg::SetData(void)
{
//设置 combo 框中当前用户所属的组名

SetDlgItemText(GROUP_COMBO, cfg->GroupNameStr);

//设置编辑框的用户名

SetDlgItemText(NICKNAME_EDIT, cfg->NickNameStr);

//根据 cfg 的对应项的内容来设置几个 checkbox 的选中状态

SendDlgItemMessage(OPEN_CHECK, BM_SETCHECK, cfg->OpenCheck, 0);

SendDlgItemMessage(NOPOPUP_CHECK, BM_SETCHECK, cfg->NoPopupCheck, 0);

SendDlgItemMessage(NOBEEP_CHECK, BM_SETCHECK, cfg->NoBeep, 0);

SendDlgItemMessage(QUOTE_CHECK, BM_SETCHECK, cfg->QuoteCheck, 0);

SendDlgItemMessage(SECRET_CHECK, BM_SETCHECK, cfg->SecretCheck, 0);

//把 cfg 中要广播的 ip 地址添加到 BROADCAST_LIST 中

for ( TBroadcastObj *obj = cfg->broadcastList.Top(); obj;
                        obj = cfg->broadcastList.Next(obj) )
{
   SendDlgItemMessage(BROADCAST_LIST, LB_ADDSTRING, 0, (LPARAM)obj->Host());
```

```
}

//对 hosts 中所有的工作组，如果 host->groupName 在 GROUP_COMBO 控件工作组中没有，则加入之

for ( int cnt = 0; cnt < hosts->HostCnt(); cnt++ )
{
   Host *host = hosts->GetHost(cnt);

   if ( *host->groupName && SendDlgItemMessage(GROUP_COMBO, CB_FINDSTRING,
           (WPARAM)-1, (LPARAM)host->groupName) == CB_ERR )
   {
    SendDlgItemMessage(GROUP_COMBO, CB_ADDSTRING, (WPARAM)0,
     (LPARAM)host->groupName);
   }
}

SendDlgItemMessage(DIALUP_CHECK, BM_SETCHECK, cfg->DialUpCheck, 0);
}

void TSetupDlg::GetData(void)
{
char buf[MAX_PATH];

//获得用户设定的用户名

GetDlgItemText(NICKNAME_EDIT, cfg->NickNameStr, sizeof(cfg->NickNameStr));

//获得用户设定的组名

GetDlgItemText(GROUP_COMBO, cfg->GroupNameStr, sizeof(cfg->GroupNameStr));

//根据 checkbox 的选中状态来设置 cfg 中对应项的内容

cfg->OpenCheck = (int)SendDlgItemMessage(OPEN_CHECK, BM_GETCHECK, 0, 0);

cfg->NoPopupCheck = (int)SendDlgItemMessage(NOPOPUP_CHECK, BM_GETCHECK, 0, 0);

cfg->NoBeep = (int)SendDlgItemMessage(NOBEEP_CHECK, BM_GETCHECK, 0, 0);

cfg->QuoteCheck = (int)SendDlgItemMessage(QUOTE_CHECK, BM_GETCHECK, 0, 0);

cfg->SecretCheck = (int)SendDlgItemMessage(SECRET_CHECK, BM_GETCHECK, 0, 0);

//删除 broadcastList 中的所有内容
```

```
cfg->broadcastList.Reset();

//把用户设定的要广播地址加入到 broadcastlist 中

for ( int cnt = 0; SendDlgItemMessage(BROADCAST_LIST,
        LB_GETTEXT, cnt, (LPARAM)buf) != LB_ERR; cnt++ )
{
   cfg->broadcastList.SetHostRaw(buf, ResolveAddr(buf));
}

//获得拨号连接的选中状态

cfg->DialUpCheck = (int)SendDlgItemMessage(DIALUP_CHECK, BM_GETCHECK, 0, 0);
}
```

（转）IPMSG 飞鸽传书——发送对话框

2010-05-09 10:56


飞鸽传书最主要的界面：发送消息对话框，选择一个用户，然后发送，经由 UDP

socket 发送到指定的用户了。

```
static char *senddlg_id =
"@(#)Copyright (C) H.Shirouzu 1996-2004    senddlg.cpp Ver2.05";
/*
========================================================================
Project Name     : IP Messenger for Win32
Module Name      : Send Dialog
Create       : 1996-06-01(Sat)
Update       : 2004-01-01(Thu)
Copyright      : H.Shirouzu
Reference      :
========================================================================
*/
#include "tlib.h"
#include "resource.h"
```

```
#include "ipmsg.h"
#include "msgstr.h"
#include "blowfish.h"

/*
SendDialog
*/
TSendDlg::TSendDlg(MsgMng *_msgmng, ShareMng *_shareMng, THosts *_hosts,
      Cfg *_cfg, LogMng *_logmng, HWND _hRecvWnd, MsgBuf *_msg)
: TListDlg(SEND_DIALOG), editSub(_cfg, this), separateSub(this),
hostListView(this)
{
//发送对话框对应的 recv 对话框的窗口

hRecvWnd   = _hRecvWnd;

msgMng    = _msgmng;//udp 消息管理器

shareMng   = _shareMng;//shareinfo 管理器

shareInfo   = NULL;//当前的 shareinfo，发送的有文件时才不是 NULL

shareStr   = NULL;//保存要发送文件的各种信息的字符串

hosts    = _hosts;//所有的主机列表

hostArray   = NULL;//显示的主机的数组

cfg     = _cfg;//设置类

logmng    = _logmng;//日志文件管理类

memberCnt   = 0;//所有在线的用户数

sendEntry   = NULL;//发送的 entry

sendEntryNum = 0;//sendentry 的总数

//调用 msgMng 的 MakePacketNo 返回一个编号

//一个 packetno 对应一个 TSendDlg！！！，这里创建好 packetno 后在 senddlg 类中

//就没有改变过

packetNo   = msgMng->MakePacketNo();
```

```
retryCnt    = 0;//已经尝试发送了多少次了

timerID     = 0;//为了关闭 senddlg 而设置的 timer

hEditFont    = NULL;//编辑框的字体

hListFont    = NULL;//列表框的字体

captureMode    = FALSE;//用户在拖动分割条时是否捕获了鼠标

listOperateCnt = 0;//互斥变量

hiddenDisp    = FALSE;//是否隐藏优先级比较小的用户

*selectGroup = 0;//用户选中的组名对话框

currentMidYdiff = cfg->SendMidYdiff;//y 的坐标偏移量

memset( &orgFont, 0, sizeof(orgFont) );//默认字体

maxItems = 0;//

lvStateEnable = FALSE;

sortItem = -1;//排序的根据，比如用户名，主机名，优先级等

sortRev = FALSE;//是从小到大还是从大到小

findDlg = NULL;//查找对话框

msg.Init(_msg);//根据参数 msg 构建 msg 对象

hAccel = ::LoadAccelerators(TApp::hI, (LPCSTR)IPMSG_ACCEL);
}

/*
释放 SendDialog  所占用的资源
*/
TSendDlg::~TSendDlg()
{
if ( findDlg )//如果有删除对话框，删除之
{
   delete findDlg;
```

```
}

//删除 list 中的所有对象

SendDlgItemMessage(HOST_LIST, LVM_DELETEALLITEMS, 0, 0);

//删除 list 和 edit 的字体

if ( hListFont )
{
   ::DeleteObject(hListFont);
}

if ( hEditFont )
{
   ::DeleteObject(hEditFont);
}

//删除 sendEntry 数组

delete [] sendEntry;

delete [] shareStr;

if ( hostArray )//删除 hostArray
{
   free(hostArray);
}
}

/*
SendDialog Create 函数
*/
BOOL TSendDlg::EvCreate(LPARAM lParam)
{
SetDlgIcon( hWnd );//设置 icon

if ( msg.hostSub.addr )
{
   Host *host =
    cfg->priorityHosts.GetHostByName(&msg.hostSub );

   //如果 host 有效并且 host 的优先级小于 0，设置 hiddenDisp 为真
```

```
    //如果一个用户的优先级小于等于 0，则隐藏该用户

    if ( host && host->priority <= 0 )
    {
     hiddenDisp = TRUE;
    }
}

//如果用户设置把发送设置成了“开火”，就把发送按钮的文字设置为开火

if ( cfg->AbnormalButton )
{
    SetDlgItemText(IDOK, FIRE_MSGSTR);
}

if ( cfg->SecretCheck )//如果 cfg 中设置了封装，则设置之
{
    SendDlgItemMessage( SECRET_CHECK, BM_SETCHECK,
     cfg->SecretCheck, 0 );
}
else //  如果不封装，则上锁按钮无效
{
    ::EnableWindow(GetDlgItem(PASSWORD_CHECK), FALSE);
}

//根据 cfg 来设置字体

SetFont();

//设置调整大小

SetSize();

//显示在线用户数目

DisplayMemberCnt();

if ( IsNewShell() != TRUE )
{
    ULONG style;

    style = GetWindowLong(GWL_STYLE);

    style &= 0xffffff0f;
```

```
    style |= 0x00000080;
    SetWindowLong(GWL_STYLE, style);
    style = ::GetWindowLong(GetDlgItem(SEPARATE_STATIC), GWL_STYLE);
    style &= 0xffffff00;
    style |= 0x00000007;
    ::SetWindowLong(GetDlgItem(SEPARATE_STATIC), GWL_STYLE, style);
}
//设置为前台窗口
SetForegroundWindow();
//根据 msg 的 msgBuf 设置 text，如果需要加上引用则加上“〉〉”
//这里是发送对话框，必须要加上引用字符串！！
PostMessage( WM_DELAYSETTEXT, 0, 0 );
//创建各个控件对应的 win 窗口
editSub.CreateByWnd( GetDlgItem(SEND_EDIT) );
separateSub.CreateByWnd( GetDlgItem(SEPARATE_STATIC) );
hostListView.CreateByWnd( GetDlgItem(HOST_LIST) );
#if 0
SendDlgItemMessage(HOST_LIST, LVM_SETTEXTBKCOLOR, 0, 0x222222);
SendDlgItemMessage(HOST_LIST, LVM_SETTEXTCOLOR, 0, 0xeeeeee);
SendDlgItemMessage(HOST_LIST, LVM_SETBKCOLOR, 0, 0x222222);
#endif
//设置标头
InitializeHeader();
```

```
//有序的加入 list，加载进所有的用户到 list 和 hostArray 中

for ( int cnt = 0; cnt < hosts->HostCnt(); cnt++ )
{
   AddHost( hosts->GetHost(cnt) );
}

//list 的窗口获得了焦点

::SetFocus( hostListView.hWnd );

//如果 msg.hostSub.addr 不是空，则选中这个 host 用户，回复消息时

//选中要回复的用户

if ( msg.hostSub.addr )
{
   SelectHost( &msg.hostSub );
}

return TRUE;
}

/*
Construct/Rebuild Column Header

建立各行的标题，这里使用 cfg->ColumnItems 被设置的那些位来显示对应的列，

比如第 2，5，7 位被设置了，我们首先把这个顺序的对应关系使用 items 数组保存起来，

比如  items[0] = 2，  items[1] = 5，items[2] = 7；这里有一个问题，由于用户可能调整各列

的顺序，比如用户把最后一列搬到了最前面了，调整显示的各列为 2，0，1 了，我们要记录这个

对应关系，怎么办？记录 0，1，2 是不行的，我们得把顺序 7，2，5 使用 FullOrder 数组保存起来，

create 时首先顺序的设置 items 数组，然后根据 FullOrder 的内容来调整显示的顺序（Order 数组

保存）。从 FullOrder 找到 items 中数组的下标，这里就得用到了 revitems 数组的内容，使得可以根据

items 中项的内容找到项的坐标，revitems 就是把 items 中项的内容映射到了 items 中项的下标
```

的！！！

然后把 Order 数组中对应项的内容设置为找到的这个下标。显示时根据 order 数组的下标显示第 2，0，1 列

*/

```
void TSendDlg::InitializeHeader(void)
{
int    order[MAX_SENDWIDTH];

int    revItems[MAX_SENDWIDTH];

//  首先删除 HOST_LIST 中所有的列 ，maxItems 默认是 0

while ( maxItems > 0 )
{
   SendDlgItemMessage(HOST_LIST, LVM_DELETECOLUMN,
    --maxItems, 0 );
}

//不考虑 absence 那一列

ColumnItems = cfg->ColumnItems & ~(1 << SW_ABSENCE);

//把 cfg 的 sendorder 数组拷贝到 FullOrder 数组中

memcpy( FullOrder, cfg->SendOrder, sizeof(FullOrder) );

//判断 ColumnItems 中从右边数第 cnt 位是否被设置

//比如 ColumnItem 的 0，2，4，5，6 位可能被设置了，则 items 数组中的前 5 个元素

//的内容就是[0] = 0；[1] = 2；[2] = 4；[3] = 6；[4] = 7；

//revItems 数组的内容是[0] = 0；[2] = 1；[4] = 2；[6] = 3；[7] = 4；

//如果说 items 定义了一种从左边到右边的关系，

//revItems 则定义了一种从右边到左边的映射关系！！

/*

整体的第 2，5，7 列被显示了，则 items[0] = 2，  items[1] = 5，items[2] = 7；
```

items 数组中下标是显示的 list 的各列的编号，比如 items[1] = 5 表示列表中第 1 列

对应于整体的第 5 列；revitems 定义的是一个 items 数组中的逆反关系，比如 revitems[2] = 0；

revitems[5] = 1；revitems[7] = 2; 整体的第 7 列对应于显示的列表中的第 2 列。

由于用户可以更改 list 中显示的各列的顺序，所以原来是 0，1，2 列现在由于用户拖动修改了顺序，可能

变为 2，0，1，对应于整体的列中的 7，2，5，我们保存的 SendOrder 就是列表中显示的列对应于整体的列，

比如我们要保存的就是 SendOrder，这里是 FullOrder[0] = 7，FullOrder[1] = 2，FullOrder[2] = 5；

在 create 函数中，我们要根据 SendOrder 数组，也就是 FullOrder 数组来设置显示的各列的顺序。

order 数组保存了最后我们要显示的各列的下标；

我们首先按照整体的默认顺序来设置 items 和 revItems 数组的内容，比如 ColumnItems 的第 2, 5，7 位被设置了

我们顺序建立 items 数组：items[0] = 2， items[1] = 5，items[2] = 7；然后建立 revItems 数组：

revitems[2] = 0；revitems[5] = 1；revitems[7] = 2;然后我们根据 SendOrder（FullOrder）数组的内容

来设置要显示的顺序，例如 FullOrder 数组的内容为 FullOrder[0] = 7，FullOrder[1] = 2，FullOrder[2] = 5；

表示要显示的第 0 列对应于整体的第 7 列，要显示的第 1 列对应于整体的第 2 列等等。

我们的目的，就是要显示的顺序为 2，0，1，也就是需要把刚才建立的 items 数组中的 0，1，2 顺序调整一下，

变成 2，0，1， 即：Order[0] = 2，Order[1] = 0，Order[2] = 1，

怎么才能达到我们的目的呢？首先从 FullOrder[i]表示了整体的列，我们根据整体的列找到对应于 items

中的那一列，然后依次设置到 order 中的各列中。比如 FullOrder[0] = 7，那我们根据 FullOrder[0], 也就是

7，在 revitems 中找到 7 对应于 items 数组中的哪一列我们上面可以看到 items[2] = 7；所以我

```
们把 order

的第 0 项设置为 items 中 7 对应的下标（items 中的哪一列），

*/

for ( int cnt = 0; cnt < MAX_SENDWIDTH; cnt++ )
{
   if ( GetItem( ColumnItems, cnt ) ) //  第 cnt 位被设置了
   {
    items[maxItems] = cnt;

    revItems[cnt] = maxItems++;
   }
}

int    orderCnt = 0;

for ( cnt = 0; cnt < MAX_SENDWIDTH; cnt++ )
{
   if ( GetItem(ColumnItems, FullOrder[cnt]) )
   {
    order[orderCnt++] = revItems[FullOrder[cnt]];
   }
}

//  设置 list 的格式

DWORD dw = ::GetWindowLong(GetDlgItem(HOST_LIST), GWL_STYLE)
   | LVS_SHOWSELALWAYS;

::SetWindowLong( GetDlgItem(HOST_LIST), GWL_STYLE, dw );

//  这里设置为 LVS_EX_HEADERDRAGDROP，使得可以随意拖动各列的位置。厉害！！！

DWORD style = SendDlgItemMessage( HOST_LIST,
   LVM_GETEXTENDEDLISTVIEWSTYLE, 0, 0) | LVS_EX_FULLROWSELECT
   | LVS_EX_HEADERDRAGDROP;

if ( cfg->GlidLineCheck )
{
   style |= LVS_EX_GRIDLINES;
}
else
{
```

```
    style &= ~LVS_EX_GRIDLINES;
}

//设定 list 的 style

SendDlgItemMessage( HOST_LIST,
    LVM_SETEXTENDEDLISTVIEWSTYLE, 0, style );

//设置各列的字符串

static char *headerStr[MAX_SENDWIDTH];

if ( headerStr[SW_NICKNAME] == NULL )
{
    headerStr[SW_NICKNAME] = USER_MSGSTR;

    headerStr[SW_PRIORITY] = PRIORITY_MSGSTR;

    headerStr[SW_ABSENCE] = ABSENCE_MSGSTR;

    headerStr[SW_GROUP]    = GROUP_MSGSTR;

    headerStr[SW_HOST]    = HOST_MSGSTR;

    headerStr[SW_USER]    = LOGON_MSGSTR;

    headerStr[SW_IPADDR] = IPADDR_MSGSTR;
}

LV_COLUMN lvC;

memset(&lvC, 0, sizeof(lvC));

lvC.fmt = LVCFMT_LEFT;

lvC.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM;

//根据 items 数组[cnt]依次插入各列

//例如 items 数组的内容依次为 0，2，4，6，7

//则把文字设置为 headerStr 中第 0，2，4，6，7 项的文字

for ( cnt = 0; cnt < maxItems; cnt++ )
{
```

```
    lvC.pszText = headerStr[items[cnt]];

    lvC.cx = cfg->SendWidth[items[cnt]];

    SendDlgItemMessage(HOST_LIST, LVM_INSERTCOLUMN,
     lvC.iSubItem = cnt, (LPARAM)&lvC);

}

//把 Order 设置为 list 的 array ，order 定义了各列的显示顺序

//例如 order 数组的内容为 2，0，1，则先显示第 2 列，然后显示第 0 列，最后显示第 1 列

SendDlgItemMessage( HOST_LIST, LVM_SETCOLUMNORDERARRAY,
    maxItems, (LPARAM)order );
}
```

（转）IPMSG 飞鸽传书——开发指南
2010-05-09 10:55
官方站点: http://www.ipmsg.org/

编译指南：http://www.vckbase.com/document/viewdoc/?id=1748

1. 软件简介

飞鸽传书 (IP Messenger V2.06)是一款基于 TCP/IP(UDP)的局域网内即时通信软件，2.00 以上版本局域网内发信息、传送文件、文件夹、多文件(或文件夹)，通讯数据采用 RSA/Blofish 加密 (2.00上)，速度非常快，采用 BSD License 开源许可协议发布。。最新版（2.50 alpha6）的 Log 文件支持 U编码格式，增加了气球提示。

2. 系统要求

IP Messenger V2.06 for Windows 支持 Microsoft Windows 2000/XP/2003 操作系统平台，要求公共comctl32.dll(Windows 应用程序公用 GUI 图形用户界面模块) 的版本为 5.x 或更高。(如果你安装是 5.x 或更高版本, 那么就不必担心这个问题了)

==========

3. 使用说明

< 安装/卸载 >

执行 setup.exe 你可以将 IPMsg 安装到指定目录, 或者重新注册到启动组.

如果需要删除 IPMsg, 请先从控制面板中卸载, 再手动删除 IPMsg 目录.

< 系统托盘区图标 >

左键双击图标, 即可打开发送消息窗口.

右键点击图标, 可进入 [服务设置], [离开] 等.

< 发送消息窗口 >

- 发送消息时, 若勾选了[封装],则接收者要先打开信封才能看到消息; 如果你还勾选了[上锁], 则接
在打开信封时还需要输入密码。(密码是由接收者自己在 [服务设置] 中设置的)

- 若要发送文件/文件夹, 可直接将文件/文件夹拖入发送消息窗口，或在发送消息窗口上点击右键,
择发送文件或发送文件夹

- 传送文件/文件夹时, 当接收者还没有保存(下载)文件/文件夹时,若发送者关闭或重启了 IPMsg,
带的文件信息将被清除, 接收者将不能继续接收(下载)到此文件。

- 用户列表前缀符号说明

":" 表示用户处于离开模式;

"|" 表示用户使用的不是 2.0 以上版本的 IPMsg，将不支持文件/文件夹的传送, 并且不支持通信
加密;

"|"(短线) 表示只支持文件/文件夹的传送;

- 拖动列表表头标题项可改变其顺序, 并可点击右键选择"保存列表顺序"

- 在消息输入窗口中, 可使用 Ctrl+Tab 输入制表符 Tab.

- 在发送消息窗口上点击右键, 可以进入设置显示优先级, 选择工作组，搜索用户(Ctrl+F), 传送文
传送文件夹, 保存列表顺序, 字体设置，窗口大小设置, 固定窗口位置, 列表显示设置。

- 在用户名上点击右键, 可设置其显示优先级，优先级按由小到大的顺序排序

- 通过设置显示优先级, 可将经常联系的用户至于列表顶端, 或者隐藏不需要联系的用户

< 接收消息窗口 >

- 在接收消息窗口上点击右键, 可以进入字体设置，窗口大小设置, 固定窗口位置.

- 接收消息窗口标题栏中的 "+" 或 "-" 表示通信数据使用了加密算法

"+" 表示 RSA/1024 位, blowfish/128 位加密

"-" 表示 RSA/512 位, RC2/40 位加密

- 如果你收到的消息附带了文件, 将会显示出附件按钮，点击按钮即可保存文件。

< 其他 >

- 如果需要(通过路由器)连接到广域网, 则需要设置广播地址，详见广域网设置(广播设置)。

4. 相关信息

- 所有的设置信息都保存在注册表的以下位置:

\\HKEY_CURRENT_USER\Software\HSTools\

用户密码以不可逆加密方式存储.

**********

* 如果你忘记了密码, 可以在注册表中将此键值删除 *

* \\HKEY_CURRENT_USER\Software\HSTools\IPMsgEng\PasswordStr *

**********

- 本软件使用的默认端口为 2425。若仅使用 UDP 协议的端口, 将不能传送文件/文件夹(如果安装
火墙, 则必须打开相应的 TCP 和 UDP 端口)

- 本软件为自由软件, 你可以随意传播, 但源码使用请参见许可协议.

- 只有在启动或退出程序, 使用离开模式, 刷新在线用户时 IPMsg 才会进行消息广播.

---

5. 广域网设置(广播设置)

- 主机号全部为 1 的 IP 地址, 即广播地址。例如, 连接到一个 C 类子网(即 24 位网络号, 8 位
号), IP 地址为 aaa.bbb.ccc.ddd, 其广播地址即为: aaa.bbb.ccc.255；若对方处于另一个私有子网中,
可能无效。

- 若两台主机的连接经过了多个路由器, 请直接指定对方 IP 地址

- 拨号上网用户请勾选 [拨号连接]，当刷新在线用户列表时, 列表不会被清空

---

6. 补充

- 启动飞鸽传书前, 你可以指定其运行时使用的端口,

  且可以使用不同的端口打开多个窗口. 用法如下：

  ipmsg.exe 2426 (你可在快捷方式上设置)

  但是你只能与同时也使用该端口的用户通信.

- 所以你尽可选用你喜欢的端口运行本软件.

  介于 10000 至 60000 可能更安全些.

  你也可以咨询你的网络管理员.

- 如果有多个网卡(IP), 你可以将飞鸽传书与指定的网卡(IP)进行绑定.

  命令格式如下(你可以在快捷方式上设置):

  ipmsg.exe [端口] /NIC IP 地址

  例如: C:\>ipmsg.exe /NIC 192.168.10.100

- 支持命令方式发送消息，命令格式如下：

  ipmsg.exe [端口] /MSG [/LOG][/SEAL] <主机名或 IP 地址> <消息>

  例如：C:\>ipmsg.exe /MSG /SEAL localhost Hello.

- 操作技巧

  1. 隐藏/显示窗口 Ctrl + D

  2. 按住 Ctrl 键再点 [刷新] 可保持现有用户, 搜索新上线的用户

  3. 打开发送/接收消息窗口 Ctrl + Alt + S/R (需要进行详细设置)

  4. 打开搜索窗口 Ctrl + F

  5. 接收到多个文件, 保存时可勾选 [全部]

---

7. 支持

- IPMsg 的技术讨论区是开放的.

  如果你想订阅相关邮件, 请联系 ipmsg-subscribe@ring.gr.jp

- 欢迎报告 bug, 以及提出建议

- 如果你有任何疑问, 请 E-mail 联系.

  shirouzu@h.email.ne.jp

  中文版可联系阿志: http://www.azhi.net/IPMsg/

- 发送错误报告, 请务必记录以下信息:

  软件版本, 操作系统, 故障描述, 以及故障重现方法等.

---

8. 更新历史

ver 1.00 ... 日文版 (1996/08/19)

ver 1.31 ... 英文版/日文版 (1997/09/01)

ver 2.00 ... 英文版/日文版 (2002/11/19)

支持文件/文件夹传送

支持通信数据加密

ver 2.03 ... Bug 修正 (文件传送引起缓冲溢出)

广播设置支持主机地址(FQDN)解析

ver 2.04 ... 增加绑定网卡(IP)功能

ver 2.05 ... Bug 修正 (2.04 版当激活发送/接收消息窗口时,无法注销/关闭系统)

ver 2.06 ... 很小的调整

ver 2.5alpha6 ...Log 文件支持 UTF-8 编码格式，增加了气球提示。

---

9. 感谢

- IPMsg 技术讨论区的所有成员

- Mr.Kanazawa (英文信息修正)

- 所有报告软件 bug 以及提出建议的朋友.

---

官方站点: http://www.ipmsg.org/

- 中文版站点: http://www.azhi.net/IPMsg/

- 中文版由 阿志 制作 2004-11-15 欢迎批评指正

- QQ: 5602433

- 中文版二次开发 QQ：327524866

（转）IPMSG 飞鸽传书——UDP 通讯类 2

2010-05-09 10:52

```
//首先关闭原来的 socket，然后重新初始化创建各个 socket

BOOL MsgMng::WSockReset(void)
{
WSockTerm();

return WSockInit(TRUE);
}

//向 hostsub 发送 command 和附加信息 val

BOOL MsgMng::Send( HostSub *hostSub, ULONG command, int val )
{
char buf[MAX_NAMEBUF];

wsprintf(buf, "%d", val);

return Send( hostSub->addr, hostSub->portNo, command, buf );
}

//注意这里的 hostSub 是目的地的 ip 地址

BOOL MsgMng::Send(HostSub *hostSub, ULONG command,
      const char *message, const char *exMsg )
```

```
{
return Send( hostSub->addr, hostSub->portNo, command, message, exMsg ) ;
}

BOOL MsgMng::Send(ULONG host, int port_no, ULONG command,
      const char *message, const char *exMsg)
{
char buf[MAX_UDPBUF];

int    trans_len;

MakeMsg(buf, command, message, exMsg, &trans_len);

return UdpSend(host, port_no, buf, trans_len);
}

//把 hWnd 和 tcp socket 和 udp socket 关联起来，注册 hwnd 需要处理的 tcp 和 udp 消息

BOOL MsgMng::AsyncSelectRegist(HWND hWnd)
{
if ( hAsyncWnd == 0 ) //  设置 hAsyncWnd 窗口为 hWnd
{
   hAsyncWnd = hWnd;
}

/*

The WSAAsyncSelect function requests Windows message-based
notification of network events for a socket.

int WSAAsyncSelect(
SOCKET s,
HWND hWnd,
unsigned int wMsg,
long lEvent
);

Parameters
s
[in] Descriptor identifying the socket for which event notification is required.
hWnd
[in] Handle identifying the window that will receive a message when a network event
occurs.
wMsg
[in] Message to be received when a network event occurs.
```

lEvent
[in] Bitmask that specifies a combination of network events in which the application is interested.

指定 udp_sd 这个 socket 要处理 FD_READ 消息

注册 hWnd 处理自定义的 WM_UDPEVENT 和 WM_TCPEVENT 消息！！！

*/

```
if ( ::WSAAsyncSelect(udp_sd, hWnd, WM_UDPEVENT,
                FD_READ ) == SOCKET_ERROR )
{
   return FALSE;
}
```

//指定 tcp 端口要处理 accept 和 close 消息 tcp 用来建立连接等，udp 用来读取数据！！！

```
if ( ::WSAAsyncSelect( tcp_sd, hWnd, WM_TCPEVENT,
      FD_ACCEPT | FD_CLOSE ) == SOCKET_ERROR )
{
   return FALSE;

}

return TRUE;
}
```

//尝试在 udpsocket 上接受数据保存到 recvbuf 里面，然后解析消息

```
BOOL MsgMng::Recv(MsgBuf *msg)
{
RecvBuf    buf;

if ( UdpRecv(&buf) != TRUE || buf.size == 0 )
{
   return FALSE;
}

return ResolveMsg(&buf, msg);
}
```

//根据参数创建一个消息包，buf 保存了最后组装好的缓冲区，返回_packetNo

```
//这里的 exmsg 指的是诸如所有的文件名拼装而成的一个长字符串

ULONG MsgMng::MakeMsg( char *buf, int _packetNo, ULONG command,
       const char *msg, const char *exMsg, int *packet_len )
{
int    len, ex_len = exMsg ? strlen(exMsg) + 1 : 0, max_len = MAX_UDPBUF;

if ( packet_len == NULL )
{
   packet_len = &len;
}

//buf 的格式为：ip 版本：包的序号：本地的用户名：本地的主机名：命令：exmsg

*packet_len = wsprintf(buf, "%d:%ld:%s:%s:%ld:", IPMSG_VERSION,
   _packetNo, local.userName, local.hostName, command);

//如果 packet 的长度 +   ex_len 超出了 udp 包的最大长度，则把 ex_len 设置为 0

if ( ex_len + *packet_len + 1 >= MAX_UDPBUF )
{
   ex_len = 0;
}

max_len -= ex_len;// max_len 保存了剩余部分的最大长度

if ( msg != NULL ) // sprintf msg 不是空，需要把 msg 加入到 buf 中，去掉 msg 中的\r
{
   *packet_len += LocalNewLineToUnix(msg, buf + *packet_len,
    max_len - *packet_len);
}

//为什么要++，因为最后多了一个\0

(*packet_len)++;

if ( ex_len ) //  还有附加长度消息，把 ex_msg 的内容拷贝到 buf 中
{
   memcpy(buf + *packet_len, exMsg, ex_len);

   *packet_len += ex_len;
}

return _packetNo;
```

```
}

//去掉 src 中所有的\r 符号，把剩余的部分保存到缓冲区 dest 中，返回转化的长度

int MsgMng::LocalNewLineToUnix(const char *src, char *dest, int maxlen)
{
int    len = 0;

maxlen--; //  去掉末尾的\0

//如果 src 还没有到最后而且长度合法

while ( *src != '\0' && len < maxlen )
{
   if ( ( dest[len] = *src++ ) != '\r' )
   {
    len++;
   }
}

//最后加上结束符\0

dest[len] = 0;

return len;
}

//把 unix 中的\n 转化为本地 windows 系统中的\r\n

int MsgMng::UnixNewLineToLocal(const char *src, char *dest, int maxlen)
{
int    len = 0;

char *tmpbuf = NULL;

//如果起始方和目的方的地址相同，则首先把 src 拷贝到一个临时缓冲区 tmpbuf 中

if ( src == dest )
{
   tmpbuf = strdup(src), src = tmpbuf;
}


maxlen--; //去掉  \0 所占用的位置
```

```
while ( *src != '\0' && len < maxlen )
{
   if ( ( dest[len] = *src++ ) == '\n' )
   {
    dest[len++] = '\r';

    if ( len < maxlen )
    {
     dest[len] = '\n';
    }
   }

   len++;
}

//加上结束符\0

dest[len] = 0;

//如果用到了 tmpbuf，则释放缓冲区 tmpbuf

if ( tmpbuf )
{
   free (tmpbuf);
}

return len;
}

//解析 buf，结果保存到 msg 中

BOOL MsgMng::ResolveMsg(RecvBuf *buf, MsgBuf *msg)
{
char *exStr = NULL, *tok, *p;

int    len;

//msgBuf 保存了消息的长度，exStr 指向消息后面的附加信息的开始！！

// buf->size  比 strlen(buf->msgBuf)) + 1 大说明有附加消息！！

if ( buf->size > (len = strlen(buf->msgBuf)) + 1 )
{
   exStr = buf->msgBuf + len + 1;
```

```
}

//msg->hostSub.addr 保存了发送消息的主机的地址

msg->hostSub.addr = buf->addr.sin_addr.s_addr;

msg->hostSub.portNo = buf->addr.sin_port;

//把第一个：之前的数据作为 ip 的 version，保存到 tok 中

if ( ( tok = separate_token( buf->msgBuf, ':', &p ) ) == NULL )
{
   return FALSE;
}

//如果 version 不是 IPMSG_VERSION，直接返回，如果是，保存到 msg 的 version 中

if ( ( msg->version = atoi(tok)) != IPMSG_VERSION )
{
   return FALSE;
}

//下一个：前面是 packetno

if ( ( tok = separate_token(NULL, ':', &p) ) == NULL )
{
   return FALSE;
}

msg->packetNo = atol(tok);

//下一个：前面是用户名

if ( ( tok = separate_token(NULL, ':', &p)) == NULL )
{
   return FALSE;
}

strncpyz( msg->hostSub.userName, tok, sizeof(msg->hostSub.userName) );

//下一个：前面是主机名

if ( ( tok = separate_token(NULL, ':', &p)) == NULL )
{
   return FALSE;
```

```
}

strncpyz(msg->hostSub.hostName, tok, sizeof(msg->hostSub.hostName));

//下一个：前面是命令

if ( ( tok = separate_token(NULL, ':', &p)) == NULL )
{
   return FALSE;
}

msg->command = atol(tok);

int    cnt = 0, ex_len;

*msg->msgBuf = 0;

if ( ( tok = separate_token(NULL, 0, &p)) != NULL ) //  这个\0 前面是消息的内容
{
   //把所有的\n 转化为\r\n,保存到 msg 的 msgBuf 里面

   while (*tok != '\0' && cnt < MAX_UDPBUF -1)
   {
    if ( ( msg->msgBuf[cnt++] = *tok++) == '\n' )
    {
     msg->msgBuf[cnt-1] = '\r';

     if ( cnt < MAX_UDPBUF -1 )
     {
      msg->msgBuf[cnt++] = '\n';
     }
    }
   }

   msg->msgBuf[cnt] = '\0';
}

//exOffset 保存了附加消息的开始位置！！

msg->exOffset = cnt;

if ( exStr && (ex_len = strlen(exStr) + 1) < MAX_UDPBUF - 1 )
{
   //如果有附加消息，超出了最大值，则设置 exOffset 为 MAX_UDPBUF - ex_len
```

```
    if ( ++msg->exOffset + ex_len >= MAX_UDPBUF )
    {
     msg->msgBuf[ (msg->exOffset = MAX_UDPBUF - ex_len) -1 ] = '\0'; // exStr
    }

    //在 exOffset 位置吧 exStr 拷贝到 msg 的 msgBuf 中

    memcpy( msg->msgBuf + msg->exOffset, exStr, ex_len );
}

return TRUE;
}


BOOL MsgMng::UdpSend(ULONG host_addr, int port_no, const char *buf)
{
return UdpSend(host_addr, port_no, buf, strlen(buf) +1);
}

//host_addr 是目的地的 addr，port_no 是目的地的端口，buf 是要发送的缓冲区，len 是
要发送的长度

BOOL MsgMng::UdpSend(ULONG host_addr, int port_no, const char *buf, int len)
{
struct sockaddr_in addr;

memset( &addr, 0, sizeof(addr) );

addr.sin_family     = AF_INET;

addr.sin_port     = port_no;

addr.sin_addr.s_addr = host_addr;

//通过本地的 udp socket 向目的地发送消息

if ( ::sendto(udp_sd, buf, len, 0, (LPSOCKADDR)&addr, sizeof(addr)) ==
SOCKET_ERROR )
{
    switch ( WSAGetLastError() )
    {
     //WSAENETDOWN ：  The network subsystem has failed.

    case WSAENETDOWN:
```

```
     break;

     //WSAEHOSTUNREACH ： The remote host cannot be reached from this host at this time.

    case WSAEHOSTUNREACH:

     static BOOL done;

     if ( done == FALSE )
     {
      done = TRUE;

//      MessageBox(0, HOSTUNREACH_MSGSTR, inet_ntoa(*(LPIN_ADDR)&host_addr), MB_OK);
     }

     return FALSE;

    default:

     return FALSE;
    }

    //发送失败了，重新尝试发送！！

    if ( WSockReset() != TRUE )
    {
     return FALSE;
    }

    //注册监听事件

    if ( hAsyncWnd && AsyncSelectRegist(hAsyncWnd) != TRUE )
    {
     return FALSE;
    }

    //再次尝试使用 udp 发送，如果失败了，则返回 false！！

    if ( ::sendto(udp_sd, buf, len, 0, (LPSOCKADDR)&addr, sizeof(addr)) == SOCKET_ERROR )
    {
     return FALSE;
    }
```

```
}

return TRUE;
}

//保存数据到参数 buf 的 msgBuf 变量中，buf 变量的 addr 指明了目的地的 addr

BOOL MsgMng::UdpRecv(RecvBuf *buf)
{
buf->addrSize = sizeof(buf->addr);

if ( ( buf->size = ::recvfrom(udp_sd, buf->msgBuf,
   sizeof(buf->msgBuf) -1, 0,
   (LPSOCKADDR)&buf->addr, &buf->addrSize)) == SOCKET_ERROR )
{
   return FALSE;
}

//在最后的数据后面加上结束符\0

buf->msgBuf[buf->size] = 0;

return TRUE;
}

BOOL MsgMng::Accept(HWND hWnd, ConnectInfo *info)
{
struct sockaddr_in addr;

int    size = sizeof(addr), flg = TRUE;

//在本地的 tcp socket 上接受连接

if ( ( info->sd = ::accept(tcp_sd, (LPSOCKADDR)&addr, &size)) ==
INVALID_SOCKET )
{
   return FALSE;
}

/*

TCP_NODELAY

The TCP_NODELAY option is specific to TCP/IP service providers.
```

The Nagle algorithm is disabled if the TCP_NODELAY option is enabled (and vice versa). The process involves buffering send data when there is unacknowledged data already "in flight" or buffering send data until a full-size packet can be sent. It is highly recommended that TCP/IP service providers enable the Nagle Algorithm by default, and for the vast majority of application protocols the Nagle Algorithm can deliver significant performance enhancements. However, for some applications this algorithm can impede performance, and TCP_NODELAY can be used to turn it off. These are applications where many small messages are sent, and the time delays between the messages are maintained. Application writers should not set TCP_NODELAY unless the impact of doing so is well-understood and desired because setting TCP_NODELAY can have a significant negative impact on network and application performance.

禁止接收方把数据进行合并！！

下面假设你的操作都是对 sock 这个有效的 socket 作的

1.

Q:我利用 recv/recvfrom 函数来接收数据，但是我不想等待数据来了才返回，希望有个超时值，怎么办?

A:调用 setsockopt 函数:

```
int timeout = YOUR_TIMEOUT_VALUE_IN_MILLISECONDS;
setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout,
                sizeof(timeout));
```

另外你也可以利用 select 函数解决，但是比较复杂，请参看 MSDN

2.

Q:我希望在调用 connect 的时候能够设置超时值，怎么办？

A:利用 ioctlsocket 和 select 函数:

首先利用 ioctlsocket 函数吧 socket 设置为非堵塞的，然后利用 select 函数设置超时值，如下(假定你要联接的远程地址的 sockaddr_in 是 sa)：

```
fd_set readfd;
FD_ZERO(&readfd);
u_long iomode = 1;
struct timeval timeout;
timeout.tv_sec = TIMEOUT_IN_SECONDS;
timeout.tv_usec = TIMEOUT_IN_MILLISECONDS;
ioctlsocket(sock, FIONBIO, &iomode);
FD_SET(sock, &readfd);
connect(sock, (struct sockaddr*)&sa, sizeof(sa));
select(0, &readfd, NULL, NULL, &tv_sec);
```

3.
Q:我使用 TCP 通信，发现发送端发送的数据报可能在接收端合并，我不想让他合并，怎
么办？
A:在发送端使用 setsockopt 函数：

```
BOOL sendnodelay = TRUE;
setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (char *)&sendnodelay,
                    sizeof(sendnodelay));

*/

::setsockopt(info->sd, SOL_SOCKET, TCP_NODELAY,
   (char *)&flg, sizeof(flg));

//info 保存了本地的对应于客户端通信的 socket 的端口和地址

info->addr = addr.sin_addr.s_addr;

info->port = addr.sin_port;

//由于是别人连到本地，所以 info 结构的 server 属性设置为 true

info->server = info->complete = TRUE;

//尝试设置 tcp socket 的发送缓冲区，如果不成功则把 size/2 之后再次尝试，直到设置
成功为止

for ( int buf_size = cfg->TcpbufMax; buf_size > 0; buf_size /= 2 )
{
   if ( ::setsockopt(info->sd, SOL_SOCKET, SO_SNDBUF,
    (char *)&buf_size, sizeof(buf_size)) == 0 )
   {
    break;
   }
```

```
}

//把窗口 hWnd 和 info 关联起来

if ( AsyncSelectConnect(hWnd, info) )
{
   info->startTick = info->lastTick = ::GetTickCount();

   return TRUE;
}

//关闭连接所用的 socket！！！ 居然关闭了！！！！

::closesocket( info->sd );

return FALSE;
}

BOOL MsgMng::Connect( HWND hWnd, ConnectInfo *info )
{
//本地连接到外部的 socket，server 属性设置为 false

info->server = FALSE;

//创建一个本地 socket tcp

if ( ( info->sd = ::socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET )
{
   return FALSE;
}

BOOL flg = TRUE; // Non Block

//设置本地刚创建的这个 socket 为非阻塞的

if ( ::ioctlsocket( info->sd, FIONBIO, (unsigned long *)&flg ) != 0 )
{
   return FALSE;
}

//设置本地的这个 socket 不允许目的方把数据包进行合并

::setsockopt( info->sd, SOL_SOCKET, TCP_NODELAY,
   (char *)&flg, sizeof(flg) );
```

```
//设置 socket 的发送缓冲区大小

for ( int buf_size = cfg->TcpbufMax;
      buf_size > 0; buf_size /= 2 )
{
   if ( ::setsockopt(info->sd, SOL_SOCKET, SO_RCVBUF,
    (char *)&buf_size, sizeof(buf_size) ) == 0 )
   {
    break;
   }
}

//把窗口 hWnd 和 info 结构关联起来

if ( AsyncSelectConnect( hWnd, info ) )
{
   struct sockaddr_in addr;

   memset( &addr, 0, sizeof(addr) );

   addr.sin_family     = AF_INET;

   addr.sin_port     = info->port;//  目的方  server 的端口

   addr.sin_addr.s_addr = info->addr;//目的方 server 的地址

   //尝试连接到 server 端口

   if ( ( info->complete = (::connect(info->sd,
    (LPSOCKADDR)&addr, sizeof(addr)) == 0 ) )
    || WSAGetLastError() == WSAEWOULDBLOCK )
   {
    //发送成功了，记录下当前的时间

    info->startTick = info->lastTick = ::GetTickCount();

    return TRUE;
   }
}

//关闭连接所用的 socket

::closesocket(info->sd);

return FALSE;
```

```
}

//关联 hWnd 和 info 结构，如果 info 是 server，则监听 read 事件，如果 info 是客户端，
则监听 connect 事件

//两个都监听 close 事件

BOOL MsgMng::AsyncSelectConnect(HWND hWnd, ConnectInfo *info)
{
if ( ::WSAAsyncSelect( info->sd, hWnd, WM_TCPEVENT,
     (info->server ? FD_READ : FD_CONNECT )| FD_CLOSE ) == SOCKET_ERROR )
{
   return FALSE;
}

return TRUE;
}

/*

连接完成了，把 info 的 socket 和 hWnd 关联起来，设置 info 的 sd 设置为非阻塞的

在 mainwin 和 recvdlg 中调用

*/

BOOL MsgMng::ConnectDone(HWND hWnd, ConnectInfo *info)
{
::WSAAsyncSelect( info->sd, hWnd, 0, 0 ); //

BOOL flg = FALSE;

//把 info 的 sd 结构设置为非阻塞的！！！

::ioctlsocket(info->sd, FIONBIO, (unsigned long *)&flg);

return TRUE;
}
```

（转）IPMSG 飞鸽传书——UDP 通讯类 1

2010-05-09 10:48

FG 中使用了创建了两个 socket，一个是 UDP socket，一个是 TCP socket，

UDP socket 用来发送聊天时的文本信息等，tcp socket 用来传送文件和文件相关的命令。

MsgMng 类是用来管理端口的，并封装了 udp 包的组装和解码等函数，类代码如下：

```
/*
管理底层通讯的类，这个类主要管理的是 tcp 和 udp socket 的创建，
和窗口的登记关联，UDP 类的包的组装、封装，发送等
*/
class MsgMng
{
protected:
//本地的 udpsocket
SOCKET    udp_sd;
//处理连接和关闭消息的 tcp socket
SOCKET    tcp_sd;
BOOL    status;//状态
ULONG    packetNo;//唯一的标志着一个 UDP 包
Cfg    *cfg;//全局的设置类对象
HWND    hAsyncWnd;
UINT    uAsyncMsg;
UINT    lAsyncMode;
HostSub    local;//本地的地址
//初始化各个 socket
BOOL    WSockInit(BOOL recv_flg);
//关闭 socket
void    WSockTerm(void);
```

```
//关闭 socket，然后重新创建各个 socket

BOOL    WSockReset(void);

public:

//构造函数

MsgMng(ULONG nicAddr, int portNo, Cfg *cfg = NULL);

~MsgMng();//析构函数

BOOL GetStatus(void)
{
   return status;
}

HostSub *GetLocalHost(void)
{
   return &local;
}

void CloseSocket(void);

BOOL IsAvailableTCP() //  判断本地的 tcpsocket 是否有效
{
   return tcp_sd != INVALID_SOCKET ? TRUE : FALSE;
}

BOOL Send(HostSub *hostSub, ULONG command, int val);

BOOL Send(HostSub *hostSub, ULONG command,
   const char *message = NULL, const char *exMsg = NULL);

BOOL Send(ULONG host_addr, int port_no, ULONG command,
   const char *message = NULL, const char *exMsg = NULL);

BOOL AsyncSelectRegist(HWND hWnd);

BOOL Recv(MsgBuf *msg);

BOOL ResolveMsg(RecvBuf *buf, MsgBuf *msg);

ULONG MakePacketNo(void)
{
```

```
    return packetNo++;
}

ULONG MakeMsg(char *udp_msg, int packetNo, ULONG command,
    const char *msg, const char *exMsg = NULL, int *packet_len = NULL);

ULONG MakeMsg(char *udp_msg, ULONG command,
    const char *msg, const char *exMsg = NULL, int *packet_len = NULL)
{
    return MakeMsg(udp_msg, MakePacketNo(), command, msg, exMsg, packet_len);
}

BOOL UdpSend(ULONG host_addr, int port_no, const char *udp_msg);

BOOL UdpSend(ULONG host_addr, int port_no,
    const char *udp_msg, int len);

BOOL UdpRecv(RecvBuf *buf);

BOOL Accept(HWND hWnd, ConnectInfo *info);

BOOL Connect(HWND hWnd, ConnectInfo *info);

BOOL AsyncSelectConnect(HWND hWnd, ConnectInfo *info);

BOOL ConnectDone(HWND hWnd, ConnectInfo *info);

static int LocalNewLineToUnix(const char *src, char *dest, int maxlen);

static int UnixNewLineToLocal(const char *src, char *dest, int maxlen);
};
```

实现代码如下：

```
static char *msgmng_id =
"@(#)Copyright (C) H.Shirouzu 1996-2002    msgmng.cpp Ver2.00";
/*
========================================================================
Project Name     : IP Messenger for Win32
Module Name      : Message Manager
Create       : 1996-06-01(Sat)
Update       : 2002-11-03(Sun)
Copyright      : H.Shirouzu
Reference      :
========================================================================
```

```
*/

#include "tlib.h"
#include <stdio.h>
#include "resource.h"
#include "ipmsg.h"
#include "msgstr.h"

//这个类是用来创建 tcp 和 udp 端口，注册窗口，封装发送和接收 UDP 消息的

//管理底层连接的通讯函数

MsgMng::MsgMng(ULONG nicAddr, int portNo, Cfg *_cfg)
{
status = FALSE;

//默认的 packetno 为（当前时间 - unixbase）/ 10000000

packetNo = Time();

//默认的 socket 都是 invalid 的

udp_sd = tcp_sd = INVALID_SOCKET;

hAsyncWnd = 0;

local.addr = nicAddr;//设置为本地的地址

local.portNo = htons(portNo);//设置为本地的端口号

cfg = _cfg;

//创建 socket， tcp 和 udp 的 socket

if ( WSockInit( cfg ? TRUE : FALSE ) == FALSE )
{
   return;
}
/*
GetComputerName

The GetComputerName function retrieves the NetBIOS name of the local computer.
This name is established at system startup, when the system reads it from the registry.
```

If the local computer is a node in a cluster, GetComputerName returns the name of the node.

Windows 2000/XP: GetComputerName retrieves only the NetBIOS name of the local computer. To retrieve the DNS host name, DNS domain name, or the fully qualified DNS name, call the GetComputerNameEx function.

Return Values
If the function succeeds, the return value is a nonzero value.

If the function fails, the return value is zero.

*/
DWORD size = sizeof( local.hostName );

//获得计算机的名称

if ( ::GetComputerName(local.hostName, &size) == FALSE )
{
  GetLastErrorMsg("GetComputerName()");

  return;
}

//获得主机的 ip 地址

if ( nicAddr == INADDR_ANY )
{
  char host[MAX_BUF];
  /*
  The gethostname function returns the name of the local host
  into the buffer specified by the name parameter. The host name is
  returned as a null-terminated string. The form of the host name is
  dependent on the Windows Sockets provider — it can be a simple host name,
  or it can be a fully qualified domain name. However, it is guaranteed that
  the name returned will be successfully parsed by gethostbyname and
WSAAsyncGetHostByName.

Note If no local host name has been configured, gethostname must succeed and
return a token host name that gethostbyname or WSAAsyncGetHostByName can resolve.

获得主机名称

  */

  if ( ::gethostname(host, sizeof(host)) == -1 )

```
    {
     strcpy(host, local.hostName);
    }

    hostent *ent = ::gethostbyname(host);

    if ( ent )
    {
     local.addr = *(ULONG *)ent->h_addr_list[0];
    }
}

/*
The GetUserName function retrieves the user name of the current thread.
This is the name of the user currently logged onto the system.
Return Values
If the function succeeds, the return value is a nonzero value,
and the variable pointed to by nSize contains the number of TCHARs
copied to the buffer specified by lpBuffer, including the terminating null character.

If the function fails, the return value is zero.
失败返回 0

获得当前线程的 user name

*/

size = sizeof( local.userName );

if ( ::GetUserName( local.userName, &size ) != TRUE )
{
   strncpyz( local.userName, NO_NAME, sizeof( local.userName ) );
}

status = TRUE;
}

MsgMng::~MsgMng()
{
WSockTerm();
}

BOOL MsgMng::WSockInit(BOOL recv_flg)
{
```

```
WSADATA    wsaData;

/*
WSAStartup
The WSAStartup function initiates use of Ws2_32.dll by a process.
The WSAStartup function returns zero if successful. Otherwise, it
returns one of the error codes listed in the following.

Parameters
wVersionRequested
[in] Highest version of Windows Sockets support that the caller
can use. The high-order byte specifies the minor version (revision) number;
the low-order byte specifies the major version number.
lpWSAData
[out] Pointer to the WSADATA data structure that is to receive details
of the Windows Sockets implementation.

An application cannot call WSAGetLastError to determine the error code
as is normally done in Windows Sockets if WSAStartup fails.
The Ws2_32.dll will not have been loaded in the case of a failure so
the client data area where the last error information is stored could not be established.

建立 2 个 socket udpsd  和  tcpsd，

分别进行绑定

*/
if ( ::WSAStartup(0x0101, &wsaData) != 0 )
{
   return GetSockErrorMsg("WSAStart()"), FALSE;
}

//创建一个 UDP socket

if ( (udp_sd = ::socket(AF_INET, SOCK_DGRAM, 0))
         == INVALID_SOCKET )
{
   return GetSockErrorMsg("Please setup \
    TCP/IP(controlpanel->network)\r\n"), FALSE;
}

if ( recv_flg != TRUE )
{
   return TRUE;
```

```
}

//创建一个 TCP socket

if ( (tcp_sd = ::socket(AF_INET, SOCK_STREAM, 0))
        == INVALID_SOCKET )
{
   return GetSockErrorMsg("Please setup2 TCP/IP\
    (controlpanel->network)\r\n"), FALSE;
}

struct sockaddr_in addr;

memset( &addr, 0, sizeof(addr) );

addr.sin_family     = AF_INET;

addr.sin_addr.s_addr = local.addr;

addr.sin_port     = local.portNo;

//把地址绑定了两次！！！

//把 addr 绑定到本地 UDP socket 端口 portNo 上

//UDP 的 socket 发送和接收都必须 bind

if ( ::bind( udp_sd, (LPSOCKADDR)&addr,
             sizeof(addr) ) != 0 )
{
   return GetSockErrorMsg("bind()"), FALSE;
}

//把 addr 绑定到本地端口 tcp_sd 上！！！

if ( ::bind( tcp_sd, (LPSOCKADDR)&addr, sizeof(addr) ) != 0 )
{
   //如果 tcp 绑定失败则需要关闭 socket，udp 不需要！！！！！！

   ::closesocket(tcp_sd);

   tcp_sd = INVALID_SOCKET;

   GetSockErrorMsg("bind(tcp) error. Can't support file attach");
```

```
}

BOOL flg = TRUE; // Non Block

/*
ioctlsocket
The ioctlsocket function controls the I/O mode of a socket.

int ioctlsocket(
SOCKET s,
long cmd,
u_long FAR *argp
);

Parameters
s
[in] Descriptor identifying a socket.
cmd
[in] Command to perform on the socket s.
argp
[in, out] Pointer to a parameter for cmd.

Return Values
Upon successful completion, the ioctlsocket returns zero.
Otherwise, a value of SOCKET_ERROR is returned,

Remarks
The ioctlsocket function can be used on any socket in any state.
It is used to set or retrieve operating parameters associated with
the socket, independent of the protocol and communications subsystem.
Here are the supported commands to use in the cmd parameter and their semantics:

FIONBIO
Use with a nonzero argp parameter to enable the nonblocking mode
of socket s. The argp parameter is zero if nonblocking is to be disabled.
The argp parameter points to an unsigned long value. When a socket is created,
it operates in blocking mode by default (nonblocking mode is disabled).
This is consistent with BSD sockets.
The WSAAsyncSelect and WSAEventSelect functions automatically set a socket
to nonblocking mode. If WSAAsyncSelect or WSAEventSelect has been issued on
a socket, then any attempt to use ioctlsocket to set the socket back to
blocking mode will fail with WSAEINVAL.

To set the socket back to blocking mode, an application must first disable
```

WSAAsyncSelect by calling WSAAsyncSelect with the lEvent parameter equal to zero, or disable WSAEventSelect by calling WSAEventSelect with the lNetworkEvents parameter equal to zero.

把 UDP socket 和 TCP socket 都设置为非阻塞的

```
*/
if ( ::ioctlsocket(udp_sd, FIONBIO, (unsigned long *)&flg) != 0 )
{
   return GetSockErrorMsg("ioctlsocket(nonblock)"), FALSE;
}

if ( IsAvailableTCP() && ::ioctlsocket(tcp_sd, FIONBIO,
        (unsigned long *)&flg) != 0 )
{
   return GetSockErrorMsg("ioctlsocket tcp(nonblock)"), FALSE;
}

flg = TRUE;     // allow broadcast

/*
setsockopt
The setsockopt function sets a socket option.

int setsockopt(
SOCKET s,
int level,
int optname,
const char FAR *optval,
int optlen
);
Parameters
s
[in] Descriptor identifying a socket.
level
[in] Level at which the option is defined; the supported
levels include SOL_SOCKET and IPPROTO_TCP. See Windows Sockets 2 Protocol-Specific Annex for more information on protocol-specific levels.
optname
[in] Socket option for which the value is to be set.
optval
[in] Pointer to the buffer in which the value for the requested option is supplied.
optlen
[in] Size of the optval buffer.
```

Return Values
If no error occurs, setsockopt returns zero. Otherwise, a value of SOCKET_ERROR is returned,

SO_BROADCAST ： Allows transmission of broadcast messages on the socket.

允许 udp_sd UDP socket 发送广播消息

*/

```
if (::setsockopt(udp_sd, SOL_SOCKET, SO_BROADCAST,
        (char *)&flg, sizeof(flg)) != 0 )
{
  return GetSockErrorMsg("setsockopt(broadcast)"), FALSE;
}

int buf_size = MAX_SOCKBUF, buf_minsize = MAX_SOCKBUF / 2;    // UDP

/*
```

SO_SNDBUF ： Specifies the total per-socket buffer space reserved for sends.

This is unrelated to SO_MAX_MSG_SIZE or the size of a TCP window.

设置 UDP socket 的发送的最大数据 buffer 的值

*/

```
if ( ::setsockopt(udp_sd, SOL_SOCKET, SO_SNDBUF,
        (char *)&buf_size, sizeof(int)) != 0
     && ::setsockopt(udp_sd, SOL_SOCKET, SO_SNDBUF,
        (char *)&buf_minsize, sizeof(int)) != 0 )
{
  GetSockErrorMsg("setsockopt(sendbuf)");
}

buf_size = MAX_SOCKBUF, buf_minsize = MAX_SOCKBUF / 2;

/*
```

SO_RCVBUF ： Specifies the total per-socket buffer space reserved for receives.

This is unrelated to SO_MAX_MSG_SIZE or the size of a TCP window.

设置 udp socket 发送的最大数据的值

```
*/

if (::setsockopt(udp_sd, SOL_SOCKET, SO_RCVBUF,
        (char *)&buf_size, sizeof(int)) != 0
    && ::setsockopt(udp_sd, SOL_SOCKET, SO_RCVBUF,
        (char *)&buf_minsize, sizeof(int)) != 0 )
{
   GetSockErrorMsg("setsockopt(recvbuf)");
}


flg = TRUE; // REUSE ADDR

/*

   SO_REUSEADDR：Allows the socket to be bound to an address that is already in use. (See bind.)


   Not applicable on ATM sockets.

设置 tcp socket 的地址是可以重用的

*/

if ( IsAvailableTCP() && ::setsockopt(tcp_sd, SOL_SOCKET,
        SO_REUSEADDR,(char *)&flg, sizeof(flg)) != 0 )
{
   GetSockErrorMsg("setsockopt tcp(reuseaddr)");
}

//在 tcp 上监听 5 个连接

if ( IsAvailableTCP() && ::listen(tcp_sd, 5) != 0 )
{
   return FALSE;
}

return TRUE;
}

//关闭各个 socket，清理一下

void MsgMng::WSockTerm(void)
{
```

```
CloseSocket();

if ( IsNewShell() ) //NT3.51
{
   WSACleanup();
}
}

//关闭 udp  端口和 tcp 端口

void MsgMng::CloseSocket(void)
{
if ( udp_sd != INVALID_SOCKET )
{
   ::closesocket(udp_sd);

   udp_sd = INVALID_SOCKET;
}

if ( tcp_sd != INVALID_SOCKET )
{
   ::closesocket(tcp_sd);

   tcp_sd = INVALID_SOCKET;
}
}
```

（转）IPMSG 飞鸽传书--文件传输解析
2010-05-09 10:45

这里继续讨论 IPMSG 飞鸽传书，前几天，详细的了解了 IPMSG 飞鸽传书的网络协议，详细分析并且实现了 IPMSG 飞鸽传书的消息传递过程，这里就 IPMSG 飞鸽传书中的文件和文件夹的传输做详细的说明。

在 IPMSG 飞鸽传书中，如果要进行文件或者文件夹的传输，首先需要做的就是消息的应答，通过 UDP 发送文件传输报文，另外的 IPMSG 飞鸽传书客户端收到报文后，使用 TCP 协议发送应答报文，这样就开始进行文件的传输了。

这里开启两个线程，线程 1 负责文件的传送，线程 2 负责文件的接受，下边做一一说明。

首先，我们知道，可以发送多个文件，这些文件需要建立链表保存信息。

```
typedef struct file_info
```

```
{
    struct file_info *next; //双向链表
    struct file_info *forward;
    char *file; //文件名
    int type; //文件类型：文件或者文件夹
    int size; //文件大小
    struct sockaddr_in addr; //目标网络信息
} FILE_INFO;
```

文件传送链表由专门的线程维护，这样，线程 1 就可以专心进行处理文件传输队列就可以了，线程 1 始终查询这个链表，如果表头为 NULL，就说明没有要传输的文件，表头非空，就开始发送报文，获得正确的应答后，就可以开始文件的传输了，文件传输结束，将相应的链表节点删除。这里借鉴了很多程序中使用非常广泛的“**命名池**”相关的概念，只不过这里我们使用的是**“文件池”**，可以这么认为，如果有文件要传输，可以不用考虑是否有文件正在传输，只要把要传输的文件放入 “**文件池”**就可以了 ， 同时，不用考虑“**文件池”的大小，线程 1** 是文件池的服务线程，它检测文件池的大小，如果非空，就会逐次传输文件。

这里需要特别注意的就是关于文件夹的传输，这个 IPMSG 飞鸽传书一个难点，应为对与文件夹的内容是没有显示的，需要我们逐次的判断，在飞鸽传书中是这么处理的，如果是一个文件夹，就发送文件属性为 IPMSG_FILE_DIR 的信息包，IPMSG 飞鸽传书客户端收到这个信息包后，就创建这个文件夹，然后发起发送的 IPMSG 飞鸽传书客户端，进入文件夹，传送文件夹内的文件，如果该文件夹下，还有文件夹，使用相同的方法，在文件夹内的文件传送结束后，就发送 IPMSG_FILE_RETPARENT 信息包，接受的 IPMSG_FILE_RETP 报文的 IPMSG 飞鸽传书客户端，执行返回上一级目录，IPMSG 飞鸽传书发送端，就需发送目录下的文件。这样循环操作，最终完成文件的传输，这个过程比较难以理解。

有了上边的知识，文件的接受也可以类推了，同样开启一个线程维护接受文件链表，逐次接受身下的文件，链表为空时，等待。

**感悟：**

有很多的程序都是“池”这个概念，其实就是队列，而之所以称之为“池”，是因为这个队列有一个服务函数，可以这么认为：池＝队列＋服务函数，在以后的编程中，如果遇到多任务做逐次的处理，可以使用这个方法，开启一个线程用于维护这个队列，开启另外的线程用于处理这个队列，这样，就不用在加入新的任务的时候考虑是否旧任务已经完成，这样的话，程序的框架思路就非常的清晰，而且会非常的高效，如果在一些

特殊情况下，任务的执行需要耗费很长的时间，而又对任务的快速执行要比较严格的要求，可以开启多个线程来服务这个队列。

（转）IPMSG 飞鸽传书——网络协议解析手记 2

2010-05-09 10:44

每次 IPMSG 在收到上线通告报文后，都要查找相同 ip 的节点是否已经存在，只要和结构体成员 h
这样整个用户列表当中的成员是不会重复的。报文的发送主要依靠下边的函数实现，这里推荐下
是对与命令比较多的情况下，使用下边的好处就在与结构非常的清晰。

```
mode：  命令  msg：  附加信息  struct sockaddr *p：网络信息  fd：网络套接字描述符

int msg_send(const int mode,const char *msg,const struct sockaddr *p,int fd)
{
   int udp_fd=fd;
   int broadcast_en=1;
   char msg_buf[SND_BUF_LEN];
   char *use="test",*group="sunplusapp";
   socklen_t broadcast_len=sizeof(broadcast_en);
   long int msg_id=time((time_t *)NULL);
   struct sockaddr_in udp_addr;
   struct sockaddr client;
    bzero(msg_buf,SND_BUF_LEN);
    bzero(&udp_addr,sizeof(struct sockaddr_in));
    udp_addr.sin_family=AF_INET;
    udp_addr.sin_port=htons(IPMSG_UDP_PORT);
    inet_pton(AF_INET,BR_IP,&udp_addr.sin_addr.s_addr);
//下边的 if  与 else if  ：对于上线通告  下线等使用广播地址，其他的则否
   if( (p==NULL)&&(mode!=IPMSG_NOOPERATION)&&(mode!=IPMSG_BR_ENTRY)&&(mode!
   {
```

```
        printf("p is NULL,only mode = IPMSG_NOOPERATICNA IPMSG_BR_ENTRY IPMSG_EXIT i
\n");
        return 1;
    }
    else
if( (p!=NULL)&&(mode!=IPMSG_NOOPERATION)&&(mode!=IPMSG_BR_ENTRY)&&(mode!=IP
          client=*p;
//打开广播
    if( setsockopt(udp_fd,SOL_SOCKET,SO_BROADCAST,&broadcast_en,broadcast_len)<0 )
    {
        perror("setsockopt error");
        exit(1);
    }
    switch (mode)
    {
        case IPMSG_NOOPERATION:
        sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,NULL);
        sendto(udp_fd,msg_buf,strlen(msg_buf),0,(struct sockaddr *)&udp_addr,sizeof(struct sockaddr));
        break;
        case IPMSG_BR_ENTRY:
        sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
        sendto(udp_fd,msg_buf,strlen(msg_buf),0,(struct sockaddr *)&udp_addr,sizeof(struct sockaddr));
        break;
        case IPMSG_BR_EXIT:
        sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
        sendto(udp_fd,msg_buf,strlen(msg_buf),0,(struct sockaddr *)&udp_addr,sizeof(struct sockaddr));
        break;
        case IPMSG_ANSENTRY:
        sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
```

```
sendto(udp_fd,msg_buf,strlen(msg_buf),0,&client,sizeof(struct sockaddr));
break;
case IPMSG_SENDMSG:
sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
sendto(udp_fd,msg_buf,strlen(msg_buf),0,&client,sizeof(struct sockaddr));
break;
case IPMSG_SENDMSG_OPT:
sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
sendto(udp_fd,msg_buf,strlen(msg_buf),0,&client,sizeof(struct sockaddr));
break;
case IPMSG_RECVMSG:
sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
sendto(udp_fd,msg_buf,strlen(msg_buf),0,&client,sizeof(struct sockaddr));
break;
case IPMSG_GETFILEDATA:
sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
sendto(udp_fd,msg_buf,strlen(msg_buf),0,&client,sizeof(struct sockaddr));
break;
case IPMSG_RELEASEFILES:
sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
sendto(udp_fd,msg_buf,strlen(msg_buf),0,&client,sizeof(struct sockaddr));
break;
case IPMSG_GETDIRFILES:
sprintf(msg_buf,"1:%d:%s:%s:%d:%s",msg_id,use,group,mode,msg);
sendto(udp_fd,msg_buf,strlen(msg_buf),0,&client,sizeof(struct sockaddr));
break;
default:
printf("no match mode !\n");
break;
```

```
    }
     broadcast_en=0;
// 关掉广播
    if( setsockopt(udp_fd,SOL_SOCKET,SO_BROADCAST,&broadcast_en,broadcast_len)<0 )
    {
        perror("setsockopt error");
        exit(1);
    }
    printf("msg send ok ! \n");
    return 0;
}
```

通过上边的报文就可以实现消息的传递，可以发起文件、文件夹的传输，传输文件时，首先需要
在 UDP 报文联络好之后，随即发起 TCP 文件传输，文件传输是不带格式的。IPMSG 的一个难点
今天就写这里，而却也做到这里。

（转）IPMSG 飞鸽传书——网络协议解析手记 1

2010-05-09 10:43

相信很多人都使用过飞鸽传书，这个小工具在局域网传输数据高效而便捷，自己在大二的时候就想看看飞鸽传书的源码，但那时候自己的水平有限，这几天有机会重写飞鸽传书，也对 IPMSG 的网络协议做了深入的研究，这里也要感谢 IPMSG 的作者公开源代码。

首先需要明确 IPMSG 的主要功能，IPMSG 可以局域网通信、传输文件、传输文件夹，可以通过添加局域网外 IP 来实现网外的聊天与文件传输功能。我们先分析下 IPMSG 的聊天功能，IPMSG 通过 UDP 协议实现聊天，当一个 IPMSG 的客户端运行开始，首先它向整个局域网广播上线报文，局域网内的其他 IPMSG 客户端收到上线报文后，回复该报文，回复报文中包含了该客户端的 IP PORT 用户名 机器名。这样在上线客户端通过广播发送上线报文后，局域网内的其他所有 IPMSG 客户端都发送一个回复报文，这样，所有 IPMSG 的客户端都更新自己的在线用户列表。这样 IPMSG 的上线就算结束了，接下来，如果有客户端发送消息，而消息是通过 UDP 来完成的，客户端通过查询自己用户链表获取其他用户的网络地址信息，发送消息给其他用户。总结一下：

ipmsg 可以用于收发消息和文件（夹）

使用 UDP 协议收发消息使用 TCP 协议收发文件（夹）

默认使用 2425 端口做数据传输(TCP/UDP)

包含以下功能

用户上下线识别

消息收发

文件传输文件夹传输

IPMSG 的报文格式：版本号:包编号:发送者姓名:发送者主机名:命令字:附加信息

整个报文通过字符串的形式发送，IPMSG 的版本号为 1，而包编号必须是不重复的数字，这里可以是用比较简洁的方式，就是通过 linux 的库函数 timer 来完成，time 函数返回从 1970 年 1 月 1 日 0 点以来的秒数.所以每个运行 timer()的结果都是不一样的，可以放心使用。报文中的命令字是指明这个报文是消息、上线通告、传输文件、传输文件夹还是其他的东西，附加信息在不同的命令字下是不一样的，如果命令字是消息，那么附加信息就是消息内容，如果命令字是传输文件，那么附加信息就是文件的信息了，我们来看一下命令字，这是 IPMSG 最为重要的内容。

```
/*	@(#)Copyright (C) H.Shirouzu 1996-1998 ipmsg.h	Ver1.34 */

#ifndef IPMSG_H
#define IPMSG_H

/* IP Messenger Communication Protocol version 1.0 define */
/* macro */
#define GET_MODE(command)	(command & 0x000000ffUL)
#define GET_OPT(command)	(command & 0xffffff00UL)

/* header */
#define IPMSG_VERSION		0x0001
#define IPMSG_DEFAULT_PORT	0x0979
```

```
/* command */
#define IPMSG_NOOPERATION       0x00000000UL

#define IPMSG_BR_ENTRY          0x00000001UL
#define IPMSG_BR_EXIT           0x00000002UL
#define IPMSG_ANSENTRY          0x00000003UL
#define IPMSG_BR_ABSENCE        0x00000004UL

#define IPMSG_BR_ISGETLIST      0x00000010UL
#define IPMSG_OKGETLIST         0x00000011UL
#define IPMSG_GETLIST           0x00000012UL
#define IPMSG_ANSLIST           0x00000013UL
#define IPMSG_FILE_MTIME        0x00000014UL
#define IPMSG_FILE_CREATETIME       0x00000016UL
#define IPMSG_BR_ISGETLIST2     0x00000018UL

#define IPMSG_SENDMSG           0x00000020UL
#define IPMSG_RECVMSG           0x00000021UL
#define IPMSG_READMSG           0x00000030UL
#define IPMSG_DELMSG            0x00000031UL

/* option for all command */
#define IPMSG_ABSENCEOPT        0x00000100UL
#define IPMSG_SERVEROPT         0x00000200UL
#define IPMSG_DIALUPOPT         0x00010000UL
#define IPMSG_FILEATTACHOPT     0x00200000UL

/* file types for fileattach command */
```

```
#define IPMSG_FILE_REGULAR      0x00000001UL
#define IPMSG_FILE_DIR          0x00000002UL
#define IPMSG_LISTGET_TIMER         0x0104
#define IPMSG_LISTGETRETRY_TIMER    0x0105


#define HS_TOOLS      "HSTools"
#define IP_MSG        "IPMsg"
#define NO_NAME         "no_name"
#define URL_STR         "://"
#define MAILTO_STR      "mailto:"
#endif      /* IPMSG_H */
```

报文中的命令字是一个 32 位无符号整数，包含命令（最低字节）和选项（高三字节）两部分
常用基本命令（带有 BR 标识的为广播命令），下边是一些重要的命令字。
IPMSG_NOOPERATION 不进行任何操作
IPMSG_BR_ENTRY 用户上线
IPMSG_BR_EXIT 用户退出
IPMSG_ANSENTRY 通报在线
IPMSG_SENDMSG 发送消息
IPMSG_RECVMSG 通报收到消息
IPMSG_GETFILEDATA 请求通过 TCP 传输文件
IPMSG_RELEASEFILES 停止接收文件
IPMSG_GETDIRFILES 请求传输文件夹
在 IPMSG 上线时，首先发送的是 IPMSG_NOOPERATION，默认是不做任何处理，然后上线通告报文 IPMSG_BR_ENTRY 。
用户列表通过链表来实现，看看结构体：

```
typedef struct use_date
{
 char use_name[USE_NAME_LEN];   //用户名
 char host_name[HOST_NAME_LEN]; //机器名
 int id;              //节点 ID。
 long int host_ip;    //存储 IP 信息，避免重复添加
```

```
 struct sockaddr_in inet; //存储网络信息
 struct use_data *next;
}IPMSG_USE;
```

转）IPMSG 飞鸽传书——文件发送源码分析
2010-05-09 10:40

```
1.  DWORD WINAPI TMainWin::SendFileThread(void *_sendFileObj)
2.  {
3.      SendFileObj *obj = (SendFileObj *)_sendFileObj;
4.      fd_set      fds;
5.      fd_set      *rfds = NULL, *wfds = &fds;
6.      timeval     tv;
7.    int         sock_ret;
8.    BOOL        ret = FALSE, completeWait = FALSE;
9.    // 这里 SendFileFunc 根据 command 类型自动选择两种函数 : send file
    or send directory
10.   BOOL        (TMainWin::*SendFileFunc)(SendFileObj *obj) =
11.             obj->command == IPMSG_GETDIRFILES ?
    TMainWin::SendDirFile : TMainWin::SendFile;
12.
13.    FD_ZERO(&fds);
14.    FD_SET(obj->conInfo->sd, &fds);
15.   // 这里 for 条件引入了一个简单的超时机制
16.   // 正常情况下，只要文件未传送完，循环不会退出
17.   for (int waitCnt=0; waitCnt < 180 && obj->hThread != NULL;
    waitCnt++)
18.    {
19.      tv.tv_sec = 1, tv.tv_usec = 0;
20.     // 这里 select 有什么用途呢？对于 select 功能我还不是完全明白
21.     // 根据我的分析，这里主要是利用了 select 函数的等待功能
22.     // 如果 sd 描述符没有就绪，则在 select 中最久等待 1 秒
23.     // 如此反复等待最多 180 次，也就是 3 分钟，超过三分钟后，for 循
    环结束
24.     if ((sock_ret = ::select(obj->conInfo->sd + 1, rfds, wfds, NULL, &tv)) >
    0)
25.      {
26.       // 套接字可用，清除等待
27.        waitCnt = 0;
28.
```

```
29.         //下面的代码是一个有限状态机
30.         /*
31.          * 控制变量  completeWait, obj->status
32.          * 状态机迁移过程(一般状态下):
33.          * (1)    if ((mainWin->*SendFileFunc)(obj) != TRUE)
34.          * (2)    if (obj->status == FS_COMPLETE)
35.          * (3)    if (completeWait)
36.          *
37.          * 1 首先被反复执行，直到文件发送完毕。
38.          * 在没有错误的情况下，1 总是等价于 if(false)
39.          * 于是其后的 2 每次都会被执行，判断是否完成
40.          * 一旦完成，completeWait 被设置为 True，
41.          * 在下一次循环里，进入 3
42.          * 在 3 的语句体内执行 recv 函数，等待对方应答
43.          *
44.          */
45.         if (completeWait)
46.           {
47.            // 本分支在文件发送完后执行
48.            if (::recv(obj->conInfo->sd, (char *)&ret, sizeof(ret), 0) >= 0)
49.                ret = TRUE;
50.            break;
51.           }
52.         else if ((mainWin->*SendFileFunc)(obj) != TRUE)
53.           {
54.            //本分支仅在发送出错时进行
55.            break;
56.           }
57.         else if (obj->status == FS_COMPLETE)
58.           {
59.            // 本分支在发送完成后执行
60.              completeWait = TRUE, rfds = &fds, wfds = NULL;
61.            if (obj->fileSize == 0) { ret = TRUE; break; }
62.           }
63.       }
64.       else if (sock_ret == 0) {
65.         // select 超时，重置 fds
66.          FD_ZERO(&fds);
67.          FD_SET(obj->conInfo->sd, &fds);
68.       }
69.       else if (sock_ret == SOCKET_ERROR) {
70.         // select 错误，算了，离去吧~
71.         break;
```

```
72.         }
73.     }
74.
75.   // 如果发送的是文件夹，还需要擦一下屁股
76.   if (obj->isDir)
77.     {
78.         mainWin->CloseSendFile(obj);
79.       while (--obj->dirCnt >= 0)
80.             ::FindClose(obj->hDir[obj->dirCnt]);
81.     }
82.
83.   // ret 是对方发回的返回值，告知发送方是否完成接收
84.     obj->status = ret ? FS_COMPLETE : FS_ERROR;
85.   // 发送 TCPEVENT 消息，关闭句柄
86.   // 消息处理流程: EventUser->TcpEvent->EndSendFile
87.     mainWin->PostMessage(WM_TCPEVENT, obj->conInfo->sd,
    FD_CLOSE);
88.   // 退出发送线程
89.     ::ExitThread(0);
90.   return    0;
91. }
92.
```

上面传送数据最重要的一句是：

**else if** ((mainWin->*SendFileFunc)(obj) != TRUE)

SendFileFunc 的实际内容是什么呢？由函数开始赋值的指针知道：

```
1.
2.  BOOL TMainWin::SendFile(SendFileObj *obj)
3.  {
4.     if (obj == NULL || obj->hFile == INVALID_HANDLE_VALUE)  //判断
    文件句柄是否合法
5.         return    FALSE;
6.
7.     int       size = 0;
8.       _int64    remain = obj->fileSize - obj->offset;   //取得还需要传递的总字
    节数
9.
10.   //传数据
11.   if (remain > 0 && (size = ::send(obj->conInfo->sd, obj->mapAddr +
    (obj->offset % cfg->ViewMax), remain > cfg->TransMax ? cfg->TransMax :
```

```
(int)remain, 0)) < 0)
12.         return    FALSE;
13.
14.    // 根据本次成功发送的数据量，调整 offset
15.      obj->offset += size;
16.
17.    // 如果 offset 等于文件大小了，那么设置 obj 状态为完成
18.    // 由于存在传文件夹模式和传文件模式，所以状态分情况设置
19.    if (obj->offset == obj->fileSize)
20.          obj->status = obj->command == IPMSG_GETDIRFILES ?
    FS_ENDFILE : FS_COMPLETE;
21.    else if ((obj->offset % cfg->ViewMax) == 0)//没有完成，但是已经传送完
    成了本部分数据映射，需要调整映射窗口
22.      {
23.          ::UnmapViewOfFile(obj->mapAddr);    // 删除旧映射
24.          remain = obj->fileSize - obj->offset;   // 计算新的剩余量
25.        // 映射下一块，一次 8M ，如果只剩下最后一点了，则少于 8M
    (remain)
26.          obj->mapAddr = (char *)::MapViewOfFile(obj->hMap,
    FILE_MAP_READ, (int)(obj->offset >> 32), (int)obj->offset, (int)(remain >
    cfg->ViewMax ? cfg->ViewMax : remain));
27.      }
28.    // 更新总消耗时间
29.      obj->conInfo->lastTick = ::GetTickCount();
30.
31.    return    TRUE;
32. }
```