MFC

所有带`Afx`前缀的函数均是全局函数

兼容**Unicode**与**ASCII**用`_T`宏

DDX

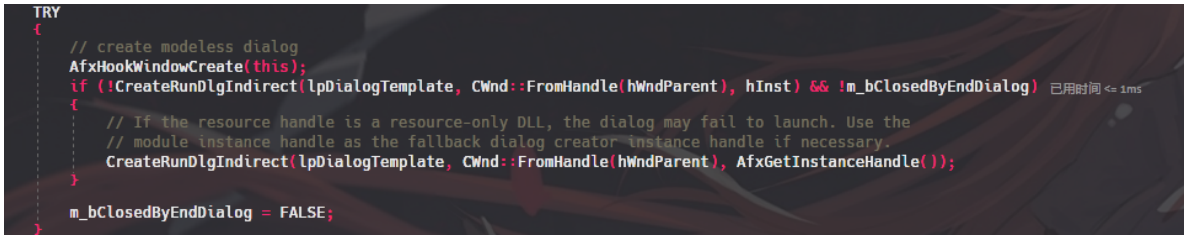
DDX是将一个控件和一个变量绑定在一起，对变量的操作就是对控件的操作

`UpdateData`

- 参数`TRUE`，数据从控件到变量
- 参数`FALSE`，数据从变量到控件

MFC的对话框分析

对话框创建



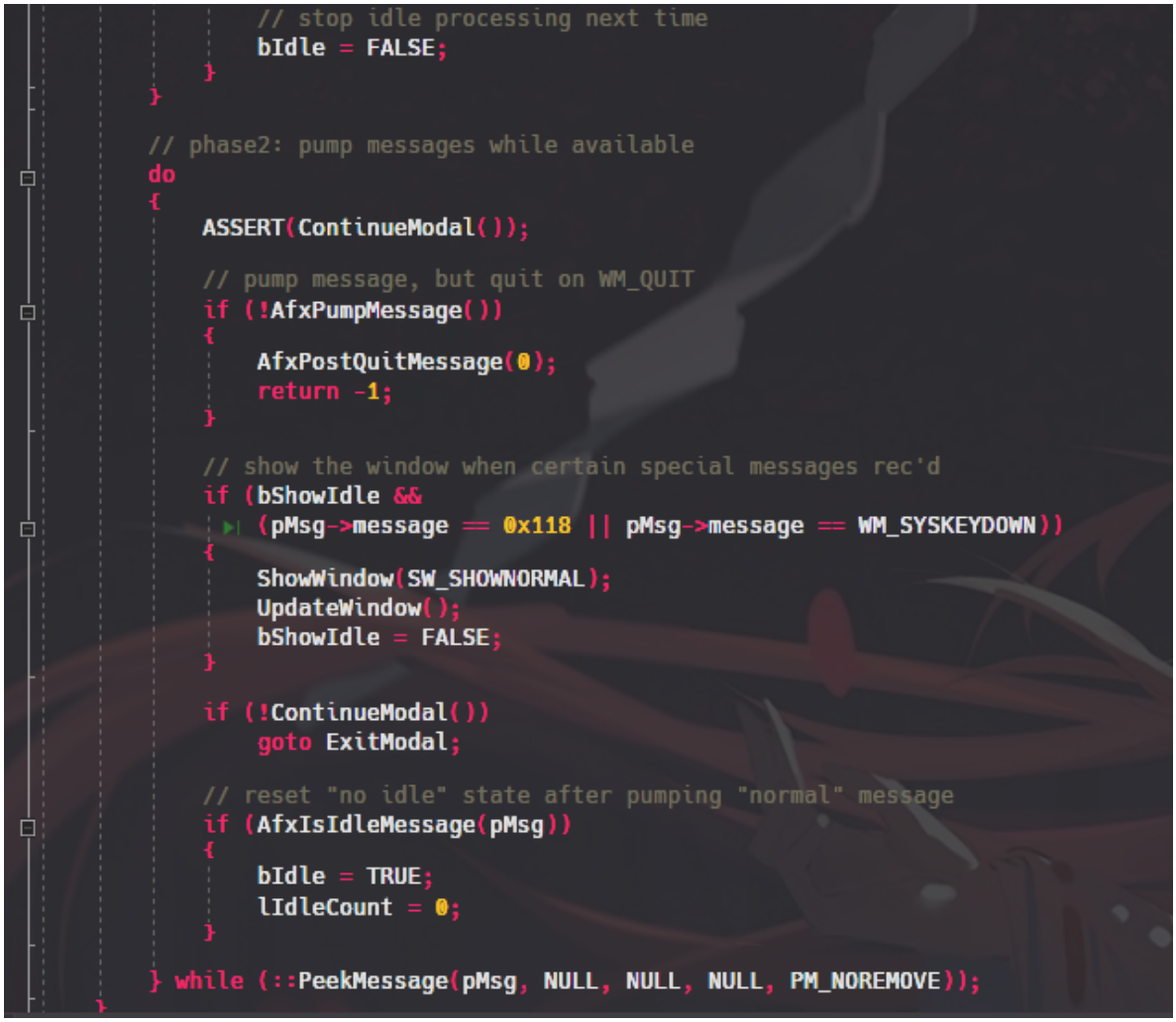
```
TRY
{
    // create modeless dialog
    AfxHookWindowCreate(this);
    if (!CreateRunDlgIndirect(lpDialogTemplate, CWnd::FromHandle(hWndParent), hInst) && !m_bClosedByEndDialog) 已用时间 <= 1ms
    {
        // If the resource handle is a resource-only DLL, the dialog may fail to launch. Use the
        // module instance handle as the fallback dialog creator instance handle if necessary.
        CreateRunDlgIndirect(lpDialogTemplate, CWnd::FromHandle(hWndParent), AfxGetInstanceHandle());
    }

    m_bClosedByEndDialog = FALSE;
}
```

```
BOOL CWnd::CreateRunDlgIndirect(LPCDLGTEMPLATE lpDialogTemplate, CWnd* pParentWnd, HINSTANCE hInst)
{ 已用时间 <= 1ms
    BOOL bRet = CreateDlgIndirect(lpDialogTemplate, pParentWnd, hInst);

    if (bRet)
    {
        if (m_nFlags & WF_CONTINUEMODAL)
        {
            // enter modal loop
            DWORD dwFlags = MLF_SHOWONIDLE;
            if (GetStyle() & DS_NOIDLEMSG)
                dwFlags |= MLF_NOIDLEMSG;
            VERIFY(RunModalLoop(dwFlags) == m_nModalResult);
        }

        // hide the window before enabling the parent, etc.
        if (m_hWnd != NULL)
            SetWindowPos(NULL, 0, 0, 0, 0, SWP_HIDEWINDOW|
            SWP_NOSIZE|SWP_NOMOVE|SWP_NOACTIVATE|SWP_NOZORDER);
    }

    return bRet;
}
```

```
        // create modeless dialog
        AfxHookWindowCreate(this);
        hWnd = ::CreateDialogIndirect(hInst, lpDialogTemplate,
            pParentWnd->GetSafeHwnd(), AfxDlgProc);
#ifdef _DEBUG
        dwError = ::GetLastError();
#endif
    }
    CATCH_ALL(e)
    {
    DELETE_EXCEPTION(e);
        m_nModalResult = -1;
    }
```

消息循环

```
    for (;;)
    {
        ASSERT(ContinueModal());

        // phase1: check to see if we can do idle work
        while (bIdle &&
            !::PeekMessage(pMsg, NULL, NULL, NULL, PM_NOREMOVE))
        {
            ASSERT(ContinueModal());

            // show the dialog when the message queue goes idle
            if (bShowIdle)
            {
                ShowWindow(SW_SHOWNORMAL);
                UpdateWindow();
                bShowIdle = FALSE;
            }

            // call OnIdle while in bIdle state
            if (!(dwFlags & MLF_NOIDLEMSG) && hWndParent != NULL && lIdleCount == 0)
            {
                // send WM_ENTERIDLE to the parent
                ::SendMessage(hWndParent, WM_ENTERIDLE, MSGF_DIALOGBOX, (LPARAM)m_hWnd);
            }
            if ((dwFlags & MLF_NOKICKIDLE) ||
                !SendMessage(WM_KICKIDLE, MSGF_DIALOGBOX, lIdleCount++))
            {
                // stop idle processing next time
                bIdle = FALSE;
            }
        }

        // phase2: pump messages while available
        do
        {
            ASSERT(ContinueModal());

            // pump message, but quit on WM_QUIT
```



```
            // stop idle processing next time
            bIdle = FALSE;
        }
    }

    // phase2: pump messages while available
    do
    {
        ASSERT(ContinueModal());

        // pump message, but quit on WM_QUIT
        if (!AfxPumpMessage())
        {
            AfxPostQuitMessage(0);
            return -1;
        }

        // show the window when certain special messages rec'd
        if (bShowIdle &&
            (pMsg->message == 0x118 || pMsg->message == WM_SYSKEYDOWN))
        {
            ShowWindow(SW_SHOWNORMAL);
            UpdateWindow();
            bShowIdle = FALSE;
        }

        if (!ContinueModal())
            goto ExitModal;

        // reset "no idle" state after pumping "normal" message
        if (AfxIsIdleMessage(pMsg))
        {
            bIdle = TRUE;
            lIdleCount = 0;
        }

    } while (::PeekMessage(pMsg, NULL, NULL, NULL, PM_NOREMOVE));
}
```

QT

发布程序

使用命令`windeployqt xxx.exe`

信号槽

信号槽用来解决对象与对象之间的交互问题

- 要继承自`QObject`
- 添加宏`Q_OBJECT`
- 信号`signals:`，信号只有声明没有实现
- 槽`public slots:`，槽函数可以当正常函数使用，也可以用来接收信号
- 发送信号`emit`
- `Object::connect(发送者, SIGNAL(func1(参数列表)), 接收者, SLOT(func2(参数列表)))`连接信号与槽

一个信号可以有多个槽来接收，多个信号也可以只被一个槽来接收