MFC窗口的创建过程详细解析

作者：www.gomensoft.com

关于MFC的窗口创建过程一直感觉比较神秘，那么我们来看下MFC中的窗口创建到底是一个什么过程(窗口创建前的窗口类准备就直接忽略了，与标准Win32窗口类准备大同小异)。MFC中窗口创建主要涉及三个重要的函数，分别是CWnd::CreateEx（或者CWnd::Create）、AfxHookWindowCreate、AfxCbtFilterHook函数，首先是大概介绍下MFC的窗口创建过程，当CWnd::CreateEx被调用时，CWnd::CreateEx在调用API函数::CreateWindowEx创建窗口前会通过调用AfxHookWindowCreate安装一个名为\_AfxCbtFilterHook的线程钩子，并将需要创建的窗口的CWnd指针保存到线程状态结构中，在API函数::CreateWindowEx真正创建窗口前AfxCbtFilterHook会被调用，AfxCbtFilterHook会执行子类化操作，把要创建的窗口的窗口过程子类化为线程状态结构中的窗口过程(AfxGetModuleState()->m\_pfnAfxWndProc)，即MFC的标准窗口过程，这样MFC的窗口(CWnd及其派生类)都可以通过消息映射机制接收和响应包括从创建开始的各种各样的消息。(关于AfxCbtFilterHook可以参考MSDN中关于SetWindowsHookEx的解释。)

1. CWnd::CreateEx

BOOL CWnd::CreateEx(DWORD dwExStyle, LPCTSTR lpszClassName,

LPCTSTR lpszWindowName, DWORD dwStyle,

int x, int y, int nWidth, int nHeight,

HWND hWndParent, HMENU nIDorHMenu, LPVOID lpParam)

{

// allow modification of several common create parameters

准备一个结构给PreCreateWindow函数使用，这样就允许应用程序在创建窗口前修改窗口创建参数，比如给cs.lpszClass重新指定一个窗口类给CreateWindow函数。

CREATESTRUCT cs;

cs.dwExStyle = dwExStyle;

cs.lpszClass = lpszClassName;

cs.lpszName = lpszWindowName;

cs.style = dwStyle;

cs.x = x;

cs.y = y;

cs.cx = nWidth;

cs.cy = nHeight;

cs.hwndParent = hWndParent;

cs.hMenu = nIDorHMenu;

cs.hInstance = AfxGetInstanceHandle();

cs.lpCreateParams = lpParam;

//允许应用程序修改窗口创建的参数。

if (!PreCreateWindow(cs))

{

PostNcDestroy();

return FALSE;

}

Hook窗口的创建过程：主要是给当前现成安装一个名为\_AfxCbtFilterHook的线程钩子，并讲需要创建的窗口的CWnd指针保存到线程状态结构(\_AFX\_THREAD\_STATE)中。

AfxHookWindowCreate(this);

开始创建窗口，在该函数正真开始创建窗口之前，AfxCbtFilterHook会被调用，AfxCbtFilterHook会执行子类化操作，把要创建的窗口的窗口过程子类化为线程状态结构中的窗口过程(AfxGetModuleState()->m\_pfnAfxWndProc)，这样MFC的窗口(CWnd及其派生类)都可以接收和响应包括从创建开始的各种各样的消息

HWND hWnd = ::CreateWindowEx(cs.dwExStyle, cs.lpszClass,

cs.lpszName, cs.style, cs.x, cs.y, cs.cx, cs.cy,

cs.hwndParent, cs.hMenu, cs.hInstance, cs.lpCreateParams);

#ifdef \_DEBUG

if (hWnd == NULL)

{

TRACE1("Warning: Window creation failed: GetLastError returns 0x%8.8X/n",

GetLastError());

}

#endif

解除创建窗口的Hook

if (!AfxUnhookWindowCreate())

PostNcDestroy(); // cleanup if CreateWindowEx fails too soon

if (hWnd == NULL)

return FALSE;

ASSERT(hWnd == m\_hWnd); // should have been set in send msg hook

return TRUE;

}

2.AfxHookWindowCreate

AfxHookWindowCreate主要是Hook窗口的创建过程：主要是给当前现成安装一个名为\_AfxCbtFilterHook的线程钩子，并讲需要创建的窗口的CWnd指针保存到线程状态结构

void AFXAPI AfxHookWindowCreate(CWnd\* pWnd)

{

//获取线程状态

\_AFX\_THREAD\_STATE\* pThreadState = \_afxThreadState.GetData();

//如果线程状态里的m\_pWndInit成员与pWnd相等表明给窗口正在创建中，直接返回

if (pThreadState->m\_pWndInit == pWnd)

return;

if (pThreadState->m\_hHookOldCbtFilter == NULL)

{

给本线程安装一个名为AfxCbtFilterHook的钩子，AfxCbtFilterHook会在::CreateWindowEx真正开始创建窗口前被调用，更多信息请参考MSDN关于SetWindowsHookEx的解释。

pThreadState->m\_hHookOldCbtFilter = ::SetWindowsHookEx(WH\_CBT,

\_AfxCbtFilterHook, NULL, ::GetCurrentThreadId());

if (pThreadState->m\_hHookOldCbtFilter == NULL)

AfxThrowMemoryException();

}

ASSERT(pThreadState->m\_hHookOldCbtFilter != NULL);

ASSERT(pWnd != NULL);

ASSERT(pWnd->m\_hWnd == NULL); // only do once

ASSERT(pThreadState->m\_pWndInit == NULL); // hook not already in progress

把需要Hook创建的窗口指针保存到线程状态中，AfxCbtFilterHook被调用时会用到。

pThreadState->m\_pWndInit = pWnd;

}

3. AfxCbtFilterHook

AfxCbtFilterHook会执行子类化操作，把要创建的窗口的窗口过程子类化为线程状态结构中的窗口过程(AfxGetModuleState()->m\_pfnAfxWndProc)，即MFC的标准窗口过程，这样MFC的窗口(CWnd及其派生类)都可以接收和响应包括从创建开始的各种各样的消息

LRESULT CALLBACK

\_AfxCbtFilterHook(int code, WPARAM wParam, LPARAM lParam)

{

\_AFX\_THREAD\_STATE\* pThreadState = \_afxThreadState.GetData();

if (code != HCBT\_CREATEWND)

{

// wait for HCBT\_CREATEWND just pass others on...

return CallNextHookEx(pThreadState->m\_hHookOldCbtFilter, code, wParam, lParam);

}

ASSERT(lParam != NULL);

LPCREATESTRUCT lpcs = ((LPCBT\_CREATEWND)lParam)->lpcs;

ASSERT(lpcs != NULL);

//获取要创建的窗口

CWnd\* pWndInit = pThreadState->m\_pWndInit;

BOOL bContextIsDLL = afxContextIsDLL;

if (pWndInit != NULL || (!(lpcs->style & WS\_CHILD) && !bContextIsDLL))

{

// Note: special check to avoid subclassing the IME window

if (\_afxDBCS)

{

// check for cheap CS\_IME style first...

if (GetClassLong((HWND)wParam, GCL\_STYLE) & CS\_IME)

goto lCallNextHook;

//获取窗口类

// get class name of the window that is being created

LPCTSTR pszClassName;

TCHAR szClassName[\_countof("ime")+1];

if (HIWORD(lpcs->lpszClass))

{

pszClassName = lpcs->lpszClass;

}

else

{

szClassName[0] = '/0';

GlobalGetAtomName((ATOM)lpcs->lpszClass, szClassName, \_countof(szClassName));

pszClassName = szClassName;

}

// a little more expensive to test this way, but necessary...

if (lstrcmpi(pszClassName, \_T("ime")) == 0)

goto lCallNextHook;

}

ASSERT(wParam != NULL); // should be non-NULL HWND

//获取即将创建的窗口句柄

HWND hWnd = (HWND)wParam;

WNDPROC oldWndProc;

if (pWndInit != NULL)

{

#ifdef \_AFXDLL

AFX\_MANAGE\_STATE(pWndInit->m\_pModuleState);

#endif

//确保窗口句柄未与窗口(CWnd)关联

// the window should not be in the permanent map at this time

ASSERT(CWnd::FromHandlePermanent(hWnd) == NULL);

//使窗口句柄与窗口(CWnd)关联，

// connect the HWND to pWndInit...

pWndInit->Attach(hWnd);

//CWnd子类可以重载这个函数以在子类化进行前做一些处理

// allow other subclassing to occur first

pWndInit->PreSubclassWindow();

//获取窗口的原始窗口过程，该窗口过程保存在CWnd:: m\_pfnSuper里，用CWnd::GetSuperWndProcAddr()函数可以获取，该函数为虚函数，可以重载把原来的窗口过程替换。

WNDPROC \*pOldWndProc = pWndInit->GetSuperWndProcAddr();

ASSERT(pOldWndProc != NULL);

#ifndef \_AFX\_NO\_CTL3D\_SUPPORT

\_AFX\_CTL3D\_STATE\* pCtl3dState;

DWORD dwFlags;

if (!afxData.bWin4 && !bContextIsDLL &&

(pCtl3dState = \_afxCtl3dState.GetDataNA()) != NULL &&

pCtl3dState->m\_pfnSubclassDlgEx != NULL &&

(dwFlags = AfxCallWndProc(pWndInit, hWnd, WM\_QUERY3DCONTROLS)) != 0)

{

// was the class registered with AfxWndProc?

WNDPROC afxWndProc = AfxGetAfxWndProc();

BOOL bAfxWndProc = ((WNDPROC)GetWindowLong(hWnd, GWL\_WNDPROC) == afxWndProc);

pCtl3dState->m\_pfnSubclassDlgEx(hWnd, dwFlags);

// subclass the window if not already wired to AfxWndProc

if (!bAfxWndProc)

{

// subclass the window with standard AfxWndProc

oldWndProc = (WNDPROC)SetWindowLong(hWnd, GWL\_WNDPROC,

(DWORD)afxWndProc);

ASSERT(oldWndProc != NULL);

\*pOldWndProc = oldWndProc;

}

}

else

#endif

{

// 通过AfxGetAfxWndProc()获取MFC的标准窗口过程，该窗口过程保存在线程状态的m\_pfnAfxWndProc成员中

// subclass the window with standard AfxWndProc

WNDPROC afxWndProc = AfxGetAfxWndProc();

//子类化窗口，即把要创建的窗口的窗口过程设为标准的MFC窗口过程，这样标准MFC窗口过程就会通过MFC的消息映射给窗口(CWnd)发送各种消息了。

oldWndProc = (WNDPROC)SetWindowLong(hWnd, GWL\_WNDPROC,

(DWORD)afxWndProc);

ASSERT(oldWndProc != NULL);

//保存子类化前的窗口过程，保存在CWnd:: m\_pfnSuper成员中

if (oldWndProc != afxWndProc)

\*pOldWndProc = oldWndProc;

}

//结束窗口创建的Hook

pThreadState->m\_pWndInit = NULL;

}

else

{

ASSERT(!bContextIsDLL); // should never get here

// subclass the window with the proc which does gray backgrounds

oldWndProc = (WNDPROC)GetWindowLong(hWnd, GWL\_WNDPROC);

if (oldWndProc != NULL && GetProp(hWnd, \_afxOldWndProc) == NULL)

{

SetProp(hWnd, \_afxOldWndProc, oldWndProc);

if ((WNDPROC)GetProp(hWnd, \_afxOldWndProc) == oldWndProc)

{

GlobalAddAtom(\_afxOldWndProc);

SetWindowLong(hWnd, GWL\_WNDPROC,

(DWORD)(pThreadState->m\_bDlgCreate ?

\_AfxGrayBackgroundWndProc :\_AfxActivationWndProc));

ASSERT(oldWndProc != NULL);

}

}

}

}

lCallNextHook:

LRESULT lResult = CallNextHookEx(pThreadState->m\_hHookOldCbtFilter, code, wParam, lParam);

#ifndef \_AFXDLL

if (bContextIsDLL)

{

::UnhookWindowsHookEx(pThreadState->m\_hHookOldCbtFilter);

pThreadState->m\_hHookOldCbtFilter = NULL;

}

#endif

return lResult;

}

5.结语

从上述解析可以看出，MFC窗口创建过程并不复杂，主要就是：

1.在窗口创建前把窗口句柄和窗口(CWnd)关联

2.通过钩子技术在窗口创建前把窗口过程替换为MFC的标准窗口过程以利用消息映射机制。

最后，我来介绍下利用MFC的这种窗口创建机制的一个应用：

我们知道，在一个普通的MFC的MDI窗口应用程序中，一般都包括CMDIFrameWnd(MDI框架窗口)及CMDIChildWnd(MDI子窗口)，其实，MDI框架窗口还有一个MDI客户窗口，该窗口是MDI子窗口的直接父窗口，这个MDI客户窗口并没有相应的窗口类(CWnd)来表现，这个MDI客户窗口的句柄保存在CMDIFrameWnd::m\_hWndMDIClient成员里，在CMDIFrameWnd被创建时，CMDIFrameWnd::CreateClient会被调用以创建这个MDI客户窗口，CMDIFrameWnd::CreateClient创建这个MDI客户窗口时是直接使用API函数创建的，其窗口句柄就保存在m\_hWndMDIClient中，这样就给我们操作这个MDI客户窗口带来了不便。下面我们来看下如何通过CWnd来操作这个MDI客户窗口。

(1).从CWnd派生一个类，如 CMDIClientWnd : public CWnd

(2).在CMDIFrameWnd的派生类(如MFC自动生成的CMainFrame)中重载CMDIFrameWnd::CreateClient函数，如在CMainFrame重载如下：

BOOL CMainFrame::CreateClient(LPCREATESTRUCT lpCreateStruct, CMenu\* pWindowMenu)

{

AfxHookWindowCreate(&m\_wndMDIClientWnd);

BOOL bSucess = CMDIFrameWnd::CreateClient(lpCreateStruct,pWindowMenu);

if( !AfxUnhookWindowCreate() )

{

PostNcDestroy();

}

return bSucess;

}

(3)在CMDIClientWnd中添加各种消息映射函数就可以方便的操作MDI客户区窗口了。