2021/04/16_MFC原理_第5课_试图中进行序列化、消息处理

笔记本: MFC原理
创建时间: 2021/4/17 星期六 15:27
作者: ileemi

在试图中进行序列化

当序列化数据链表写在试图类中时，就需要在视图类中重写虚函数 "Serialize"，在文档类中的序列化函数中遍历试图，将文档类的序列换转到试图中的序列化进行操作。

MFC单文档程序可以在主框架的 "OnCreate" 函数中创建试图，多文档程序在主框架的 "CreateDockingWindows" 函数中创建试图（一个文档对象对应多个试图）。试图支持停靠功能需要继承 "CBasePane" 类。

代码示例:

```
void CTestDoc::Serialize(CArchive& ar)
{
  POSITION pos = GetFirstViewPosition();
  while(true)
  {
    CView* pView = GetNextView(pos);
    pView->Serialize();
  }
}
```

消息处理

在MFC中利用多态进行消息处理（在窗口类中，将所有的消息以虚函数的形式进行封装）。

> 主窗口以及子窗口，都需要 Hook 拦截消息，方便统一派发消息。拦截全部消息（在窗口创建时不拦截消息会导致 WM_NCCREATE、WM_CREATE、等消息错过。所以 Hook 过程函数的时机应该在创建主窗口的时候。
> CreateWindowEx 后取消 Hook。

WM_NCCREATE -- 非客户区域创建。调用 CreateWindowEx API的时候 WM_CREATE 消息才会来。

SetWindowsHookEx -- 拦截指定窗口的消息。WH_CBT 时机较早，可以拦截到 WM_CREATE、WM_NCCREATE、WM_DESTROY消息（HCBT_CREATEWND）。

代码示例：

```
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)
{
  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 = ::GetModuleHandle(NULL);
  cs.lpCreateParams = lpParam;

  // Hook 过程函数的时机应该在创建主窗口的时候

  // 保存窗口和对象的对应关系(此时会漏掉CreateWindowEx创建的消息)
  // 需要通过下钩子，截取需要的消息
  g_hHook = SetWindowsHookEx(WH_CBT, CBTProc,
    ::GetModuleHandle(NULL),
    ::GetCurrentThreadId());

  g_pInitWnd = this;

  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);

  // 取消钩子
  UnhookWindowsHookEx(g_hHook);

  if (hWnd == NULL) {
    return FALSE;
  }
```

```
  // 加表 -- 保存窗口和对象的对应关系(此时会漏掉CreateWindowEx创建的消息)
  //g_WndMap[hWnd] = this; // 所有创建窗口对应的CWnd指针以及窗口句柄

  // 查表
  //CWnd* pTmp = g_WndMsp[hWnd];
  return TRUE;
}
```

> Hook 函数在是 HCBT_CREATEWND 时执行，需要过滤掉输入法等与主窗口无关的窗口，其他窗口一律修改其窗口过程为主窗口过程函数 "MyWndProc"，在 MyWndProc 中进行查表以及消息的派发。

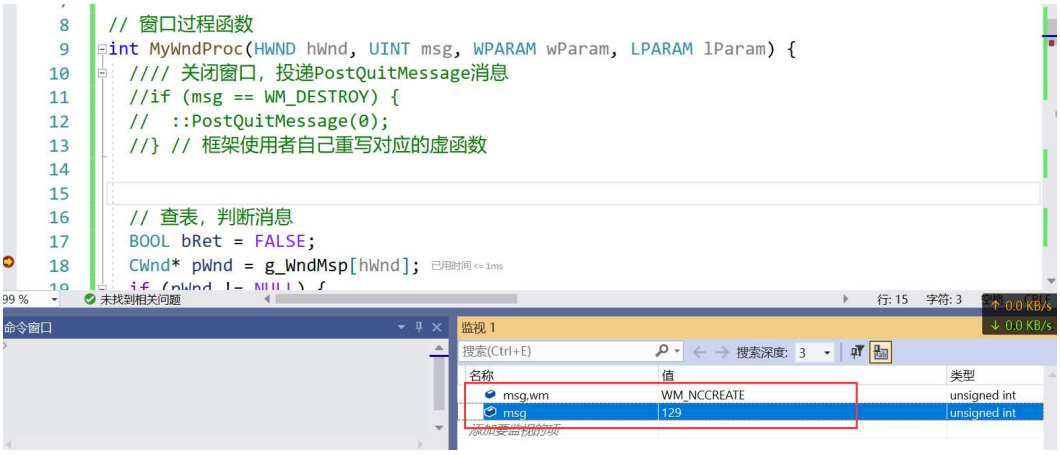
代码示例如下：

```
// 钩子
LRESULT CALLBACK CBTProc(int nCode, WPARAM wParam, LPARAM lParam) {
  if (nCode < 0) {
    return CallNextHookEx(g_hHook, nCode, wParam, lParam);
  }

  // 判断是否是输入法窗口,是:就不保存
  HWND hWnd = (HWND)wParam;
  char szClassName[MAXBYTE];
  GetClassName(hWnd, szClassName, sizeof(szClassName));
  // 判断窗口的类型并进行过滤 GCL_STYLE -- 输入法类型
  if (GetClassLong(hWnd, GCL_STYLE) & CS_IME) {
    return CallNextHookEx(g_hHook, nCode, wParam, lParam);
  }

  if (nCode == HCBT_CREATEWND) {
    if (g_pInitWnd != NULL) {

      // 判断窗口过程函数是否是主窗口的过程函数
      WNDPROC WndProc = (WNDPROC)GetWindowLong(hWnd, GWL_WNDPROC);
      if ((void*)WndProc != (void*)MyWndProc) {
        // 将控件的过程函数修改为自己的过程函数
        g_pInitWnd->m_pOldWndProc = (WNDPROC)SetWindowLong(hWnd,
GWL_WNDPROC, (LONG)&MyWndProc);
      }

      g_WndMap[hWnd] = g_pInitWnd;
      g_pInitWnd->m_hWnd = hWnd; // 保存主窗口句柄
      g_pInitWnd = NULL;
    }
  }
```

```
    // 调取下一个钩子
    return CallNextHookEx(g_hHook, nCode, wParam, lParam);
}

// 查表 -- 查询窗口句柄是否保存过
CWnd* CWnd::FromHandlePermanent(HWND hWnd)
{
    CWnd* pWnd = NULL;
    pWnd = g_WndMap[hWnd];
    return pWnd;
}

// 窗口过程函数 -- 利用多态进行消息处理
LRESULT MyWndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {
    //// 关闭窗口，投递PostQuitMessage消息
    //if (msg == WM_DESTROY) {
    //    ::PostQuitMessage(0);
    //} // 框架使用者自己重写对应的虚函数

    // 查表 -- 查询窗口句柄是否保存过
    BOOL bRet = FALSE;
    CWnd* pWnd = CWnd::FromHandlePermanent(hWnd);
    //CWnd* pWnd = g_WndMap[hWnd];
    if (pWnd != NULL) {
        // 判断消息
        switch (msg) {
        case WM_CREATE: {
            LPCREATESTRUCT lpCreate = (LPCREATESTRUCT)lParam;
            bRet = pWnd->OnCreate(lpCreate);
            break;
        }
        case WM_DESTROY: {
            bRet = pWnd->OnDestory();
            break;
        }
        case WM_COMMAND: {
            WORD wNotifyCode = HIWORD(wParam);
            WORD wID = LOWORD(wParam);
            HWND hwndCtl = (HWND)lParam;
            bRet = pWnd->OnCommand(wID, wNotifyCode, hwndCtl);
            break;
        }
        }
    }

    if (bRet) {
        return bRet;
```

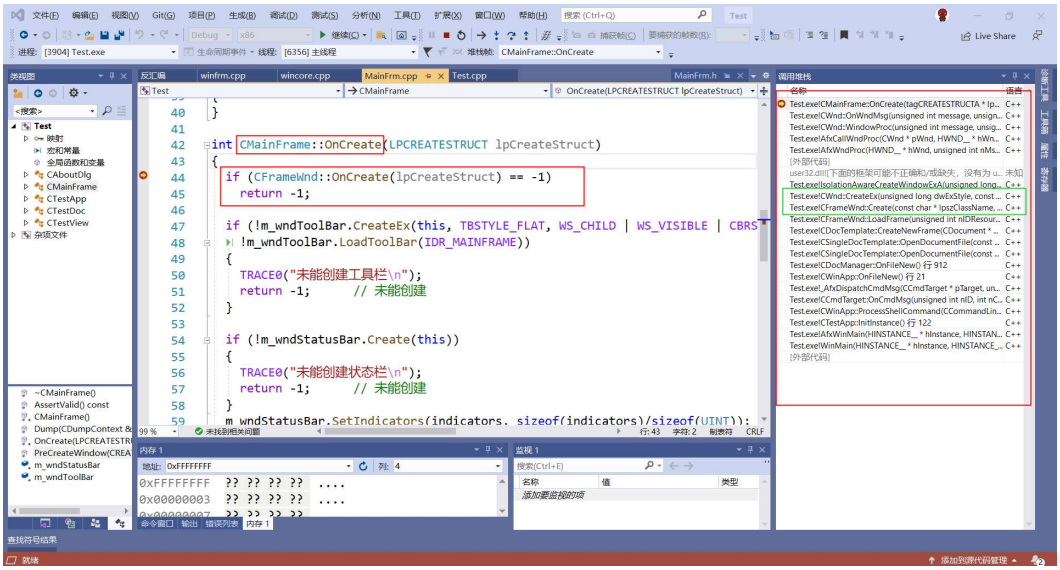
```
    }

    // 旧的过程函数不为空，就调用
    if (pWnd->m_pOldWndProc != NULL) {
      return pWnd->m_pOldWndProc(hWnd, msg, wParam, lParam);
    }
    else {
      // 调用默认的过程函数
      return ::DefWindowProc(hWnd, msg, wParam, lParam);
    }
}
```

调试MFC程序时，在监视窗口输入 "msg, wm" 可以查看当前过程函数参数 "msg" 的值(消息)，如下图所示：

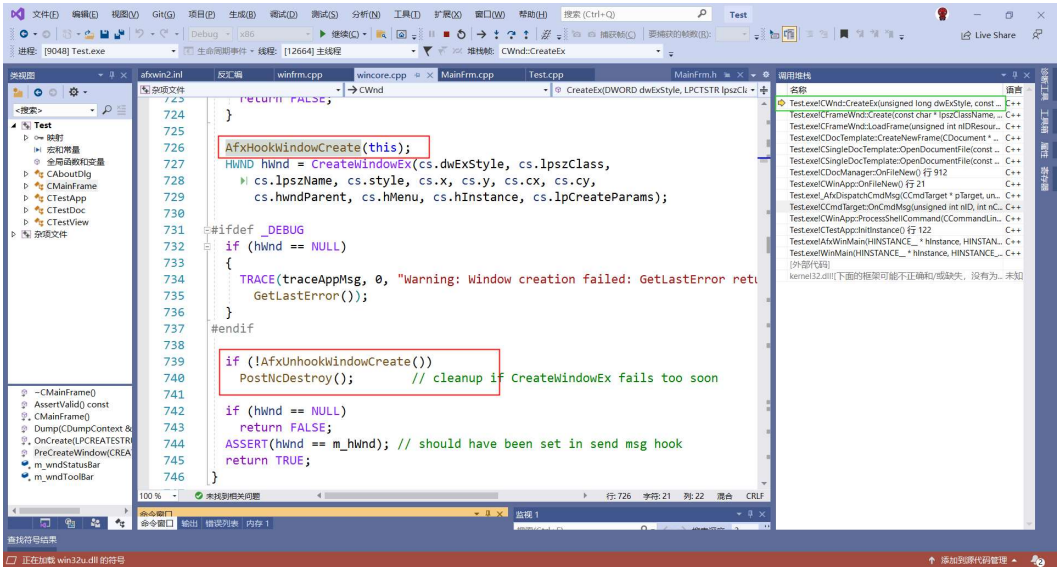


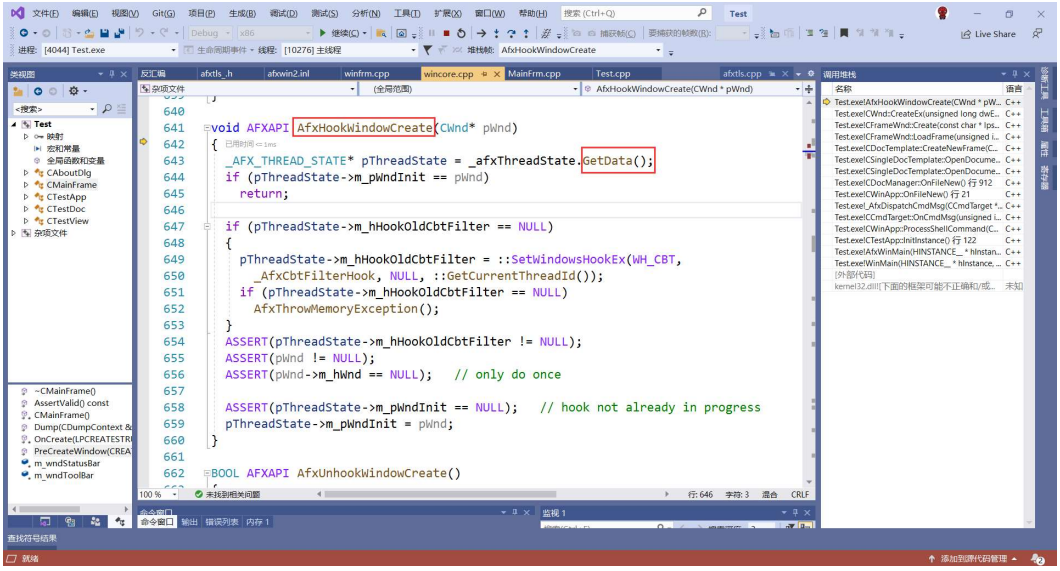
窗口过程函数的参数 "lParam" 支持自定义消息。所有的窗口都必须是框架进行注册的。

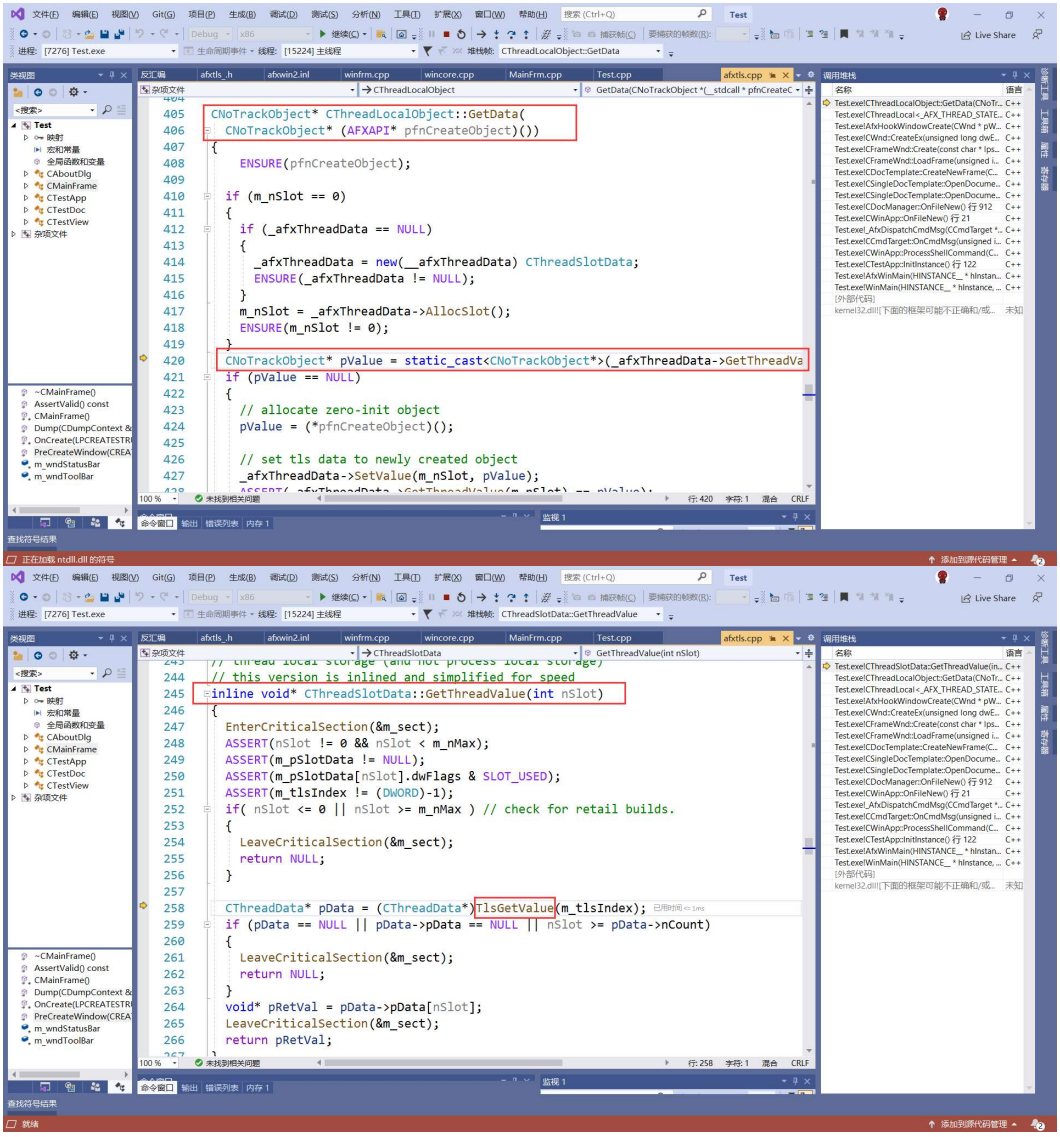
> 在MFC单文档程序中可通过在主框架 "CMainFrame" 类的 "OnCreate" 函数中下断，之后通过栈回溯跟踪主窗口的创建，如下图所示：



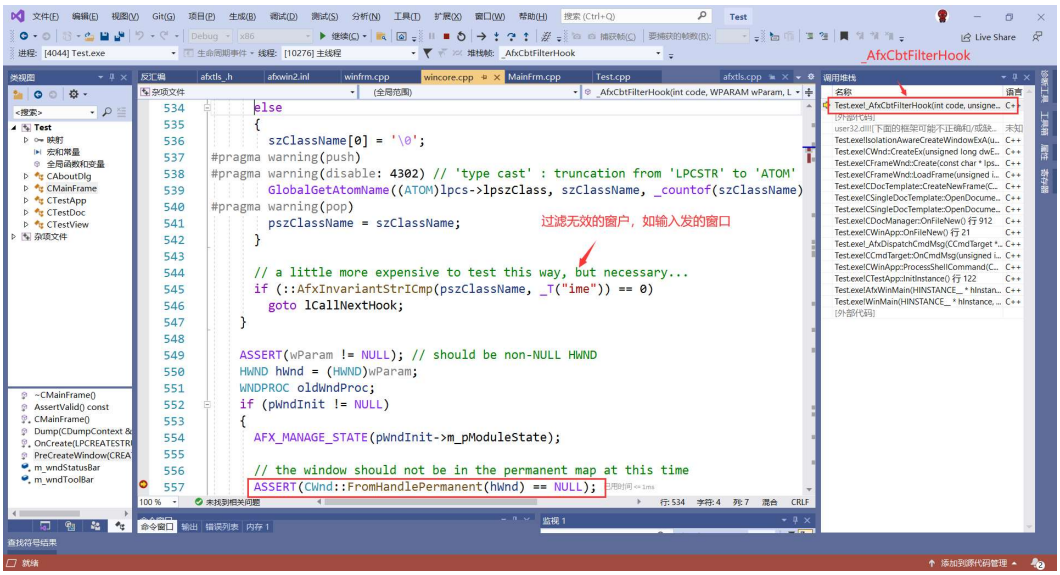
> MFC中防止全局变量出现同步问题，就会将所有的全局变量写入到TLS中，每个线程都会有一个对应的全局变量来解决全局变量的同步问题。会使用 TLS 的 "TlsGetValue" 函数，如下图所示：

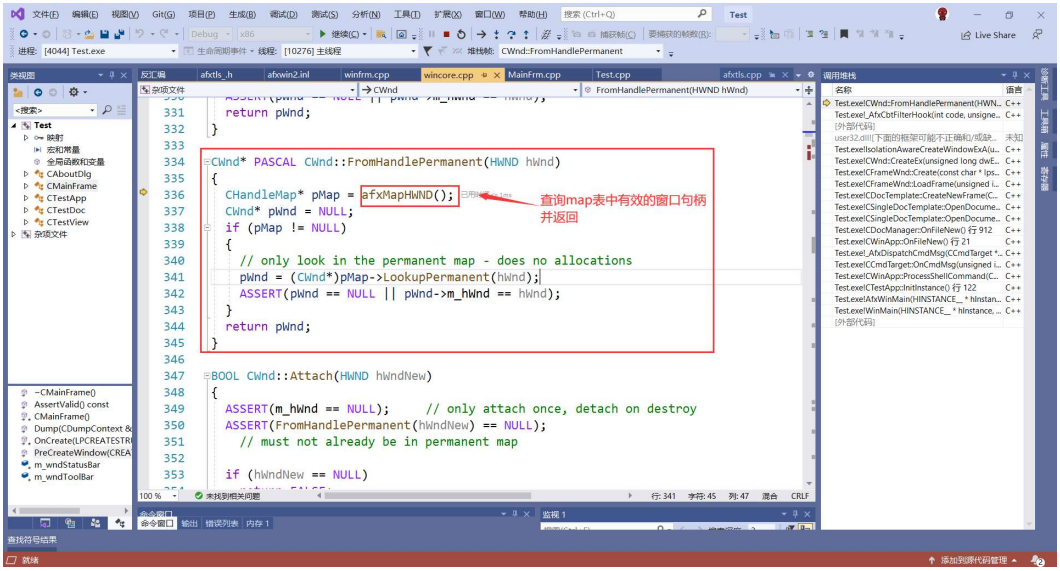

AfxHookWindowCreate(this);
HWND hWnd = CreateWindowEx(cs.dwExStyle, cs.lpszClass,
cs.lpszName, cs.style, cs.x, cs.y, cs.cx, cs.cy,
cs.hwndParent, cs.hMenu, cs.hInstance, cs.lpCreateParams);
#ifdef _DEBUG
if (hWnd == NULL)
{
TRACE(traceAppMsg, 0, "Warning: Window creation failed: GetLastError retu
GetLastError());
}
#endif
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;
}

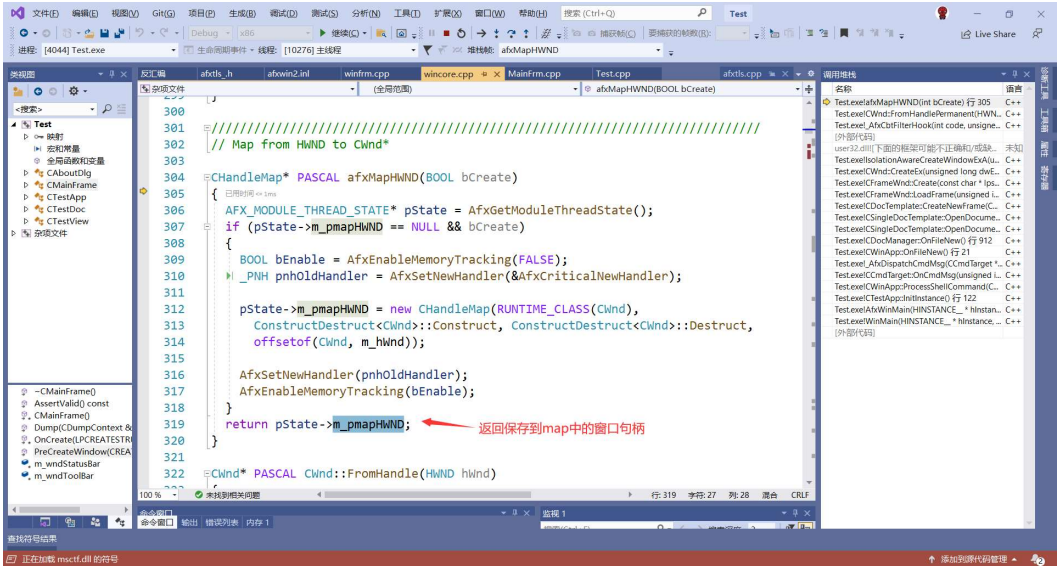

void AFXAPI AfxHookWindowCreate(CWnd* pWnd)
{
_AFX_THREAD_STATE* pThreadState = _afxThreadState.GetData();
if (pThreadState->m_pWndInit == pWnd)
return;
if (pThreadState->m_hHookOldCbtFilter == NULL)
{
pThreadState->m_hHookOldCbtFilter = ::SetWindowsHookEx(WH_CBT,
_AfxCbtFilterHook, NULL, ::GetCurrentThreadId());
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
pThreadState->m_pWndInit = pWnd;
}
BOOL AFXAPI AfxUnhookWindowCreate()



通过保存窗口和对象对应关系的哈希表获取有效的窗口句柄，如下图所示







MFC 中出现 "pThreadStart" 的就表示在操作TLS。WM_COMMAND 消息由子窗口发送给父窗口的，在试图中不能响应，需要在父窗口进行响应。