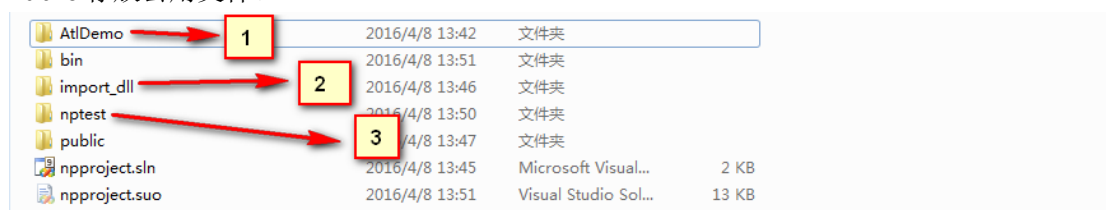


# Np 插件如何封装调用 ATL COM

## 1 demo 工程如下：

AtlDemo 是 ATL Com 控件工程，当然也可以是 ActiveX Com 控件，生成控件（dll 或 ocx）；  
Import\_dll 是简单的 MFC 工程，用来生成控件对应的 tlh/tli 两个文件；  
nptest 是 Win32 Dll 的 np 工程，用来封装控件；  
Bin 是成果物目录；  
Public 存放公用文件。



## 2 各工程简单说明：

### 1) AtlDemo 工程：ATL Project

A、有一个对外接口 **Test**

STDMETHOD(Test)(BSTR strXml, LONG nData, LONG\* lReturn)

Test 接口有两个入参：第一个参数是字符串，第二个是整型

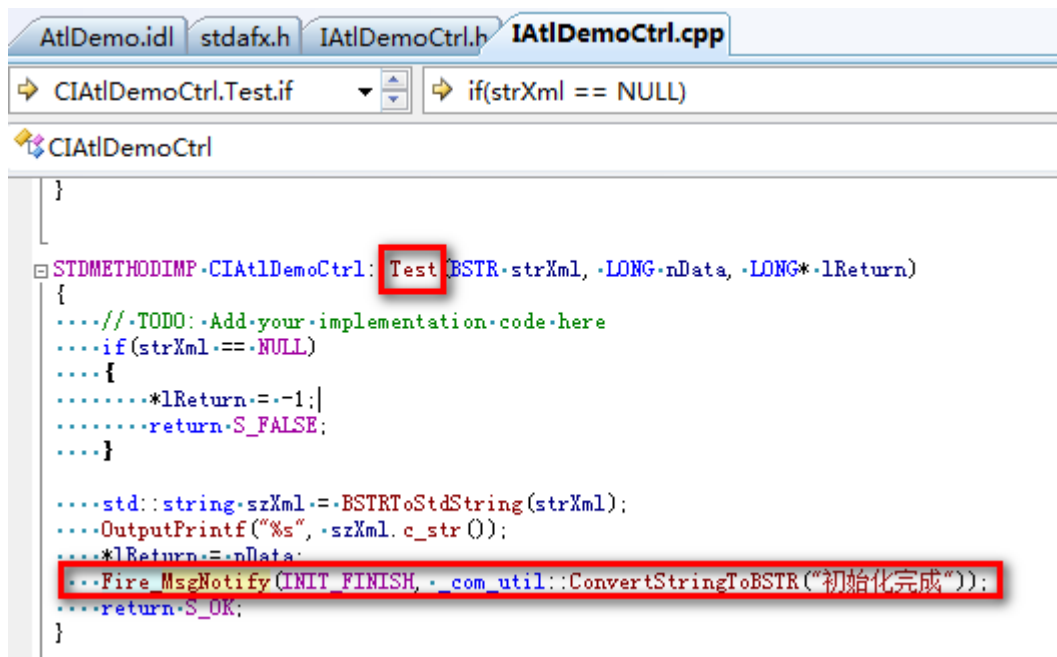
LONG\* lReturn 是返回值，类型为整型

B、有一个控件回调事件 **MsgNotify**

HRESULT MsgNotify([in] LONG lMsgId, [in] BSTR szDetail)

回调出去两个参数，第一个是事件 ID，整型；第二个详细内容，字符串

在需要通知回调时调用 Fire\_MsgNotify 函数（见 Test 接口的回调）



```
AtlDemo.idl  stdafx.h  IAtlDemoCtrl.h  IAtlDemoCtrl.cpp
CIAtlDemoCtrl.Test.if  if(strXml == NULL)

CIAtlDemoCtrl
{
}

STDMETHODIMP CIAtlDemoCtrl::Test(BSTR strXml, LONG nData, LONG* lReturn)
{
    ....//.TODO: Add your implementation code here
    ....if(strXml==NULL)
    ....{
    ....*lReturn=-1;|
    ....return S_FALSE;
    ....}

    ....std::string szXml=::BSTRToStdString(strXml);
    ....OutputPrintf("%s", szXml.c_str());
    ....*lReturn=nData;
    ....Fire_MsgNotify(INIT_FINISH, _com_util::ConvertStringToBSTR("初始化完成"));
    ....return S_OK;
}
```

## 2) Import\_dll 工程: MFC project

该工程只在 stdafx.h 添加了一行代码“#import “AtlDemo.dll””，然后编译，就会生成该 dll 对应的 th/thi 文件（import\_dll\t33\Release 目录下）

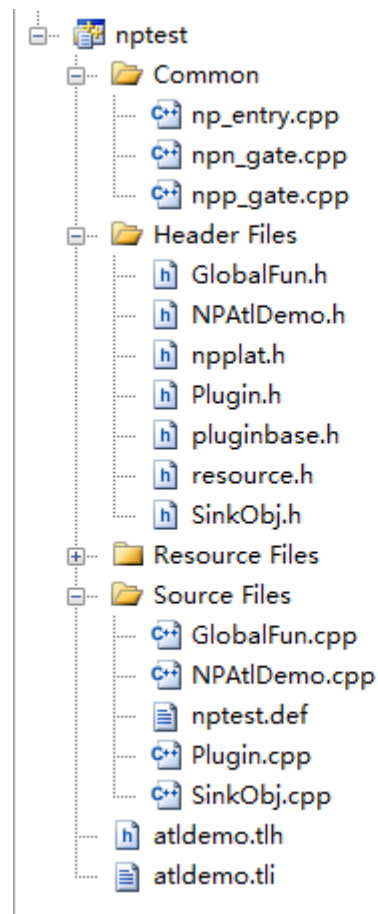


```
AtlDemo.idl  stdafx.h  IAtlDemoCtrl.h  IAtlDemoCtrl.cpp
stdafx.h  f:\源码\trunk\PreSearch\跨浏览器控件\

(Global Scope)
#include <afxcmn.h>.....//MFC support for Windows Common Cont
#include <afxcontrolbars.h>.....//MFC support for ribbons and control

#import "AtlDemo.dll"
```

### 3) nptest 工程：Win32 DLL 工程，封装了 np 框架代码



Common 下 3 个 cpp，npplat.h、pluginbase.h 是 np 框架的代码；

Atldemo.tlh/tli 就是 import\_dll 工程生成的文件，注意 tlh 中稍作修改（删除了命名空间），它们记录了控件的所有接口和事件；

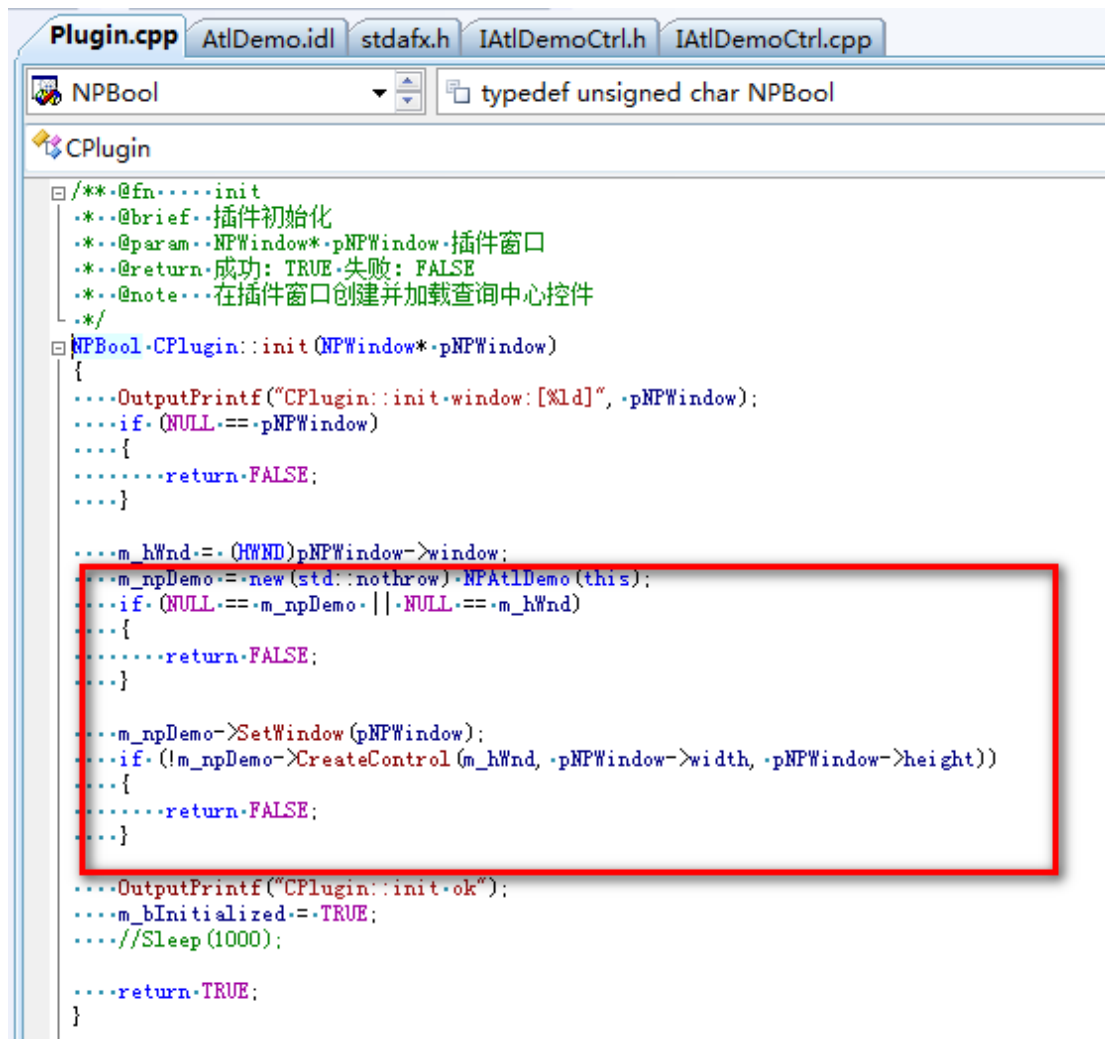
NPAtIDemo.h/NPAtIDemo.cpp 的 NPAtIDemo 类通过 ATL.dll 加载控件并封装了控件接口；

Plugin.h/Plugin.cpp 含有两个继承与 np 框架的两个类，通过调用 NPAtIDemo 类来创建 ATL 控件并调用 ATL 控件接口；

GlobalFun.h/GlobalFun.cpp 是公共文件，SinkObj.h/SinkObj.cpp 是接收控件事件的类，供 NPAtIDemo 类调用，如果 np 插件不需要接收控件事件，可以去掉 SinkObj 文件的引用。

重点在 NPAtIDemo、CPlugin、CPluginScriptObject 三个类，NPAtIDemo 封装了控件接口；CPlugin、CPluginScriptObject 是继承 np 框架的类，CPlugin 类的 init 函数创建 NPAtIDemo，进而调用到 ATL 控件；CPluginScriptObject 类的 hasMethod 函数判断页面调用的接口是否存在于控件里，如果存在，hasMethod 函数返回 true，之后就会进入 invoke 函数，去通过 CPlugin 类调用到 ATL 控件接口；如果不存在，hasMethod 返回 false，不会进入 invoke 函数。（控件属性对应函数为 hasProperty、setProperty）

Init 方法：



```
Plugin.cpp  AtlDemo.idl  stdafx.h  IAtlDemoCtrl.h  IAtlDemoCtrl.cpp
NPBool
typedef unsigned char NPBool
CPlugin
/**.@fn.....init
.*.@brief..插件初始化
.*.@param..NPWindow*.pNPWindow..插件窗口
.*.@return..成功: TRUE..失败: FALSE
.*.@note...在插件窗口创建并加载查询中心控件
.*/
NPBool.CPlugin::init(NPWindow*.pNPWindow)
{
....OutputPrintf("CPlugin::init.window:[%ld]", .pNPWindow);
....if.(NULL==.pNPWindow)
....{
.....return.FALSE;
....}

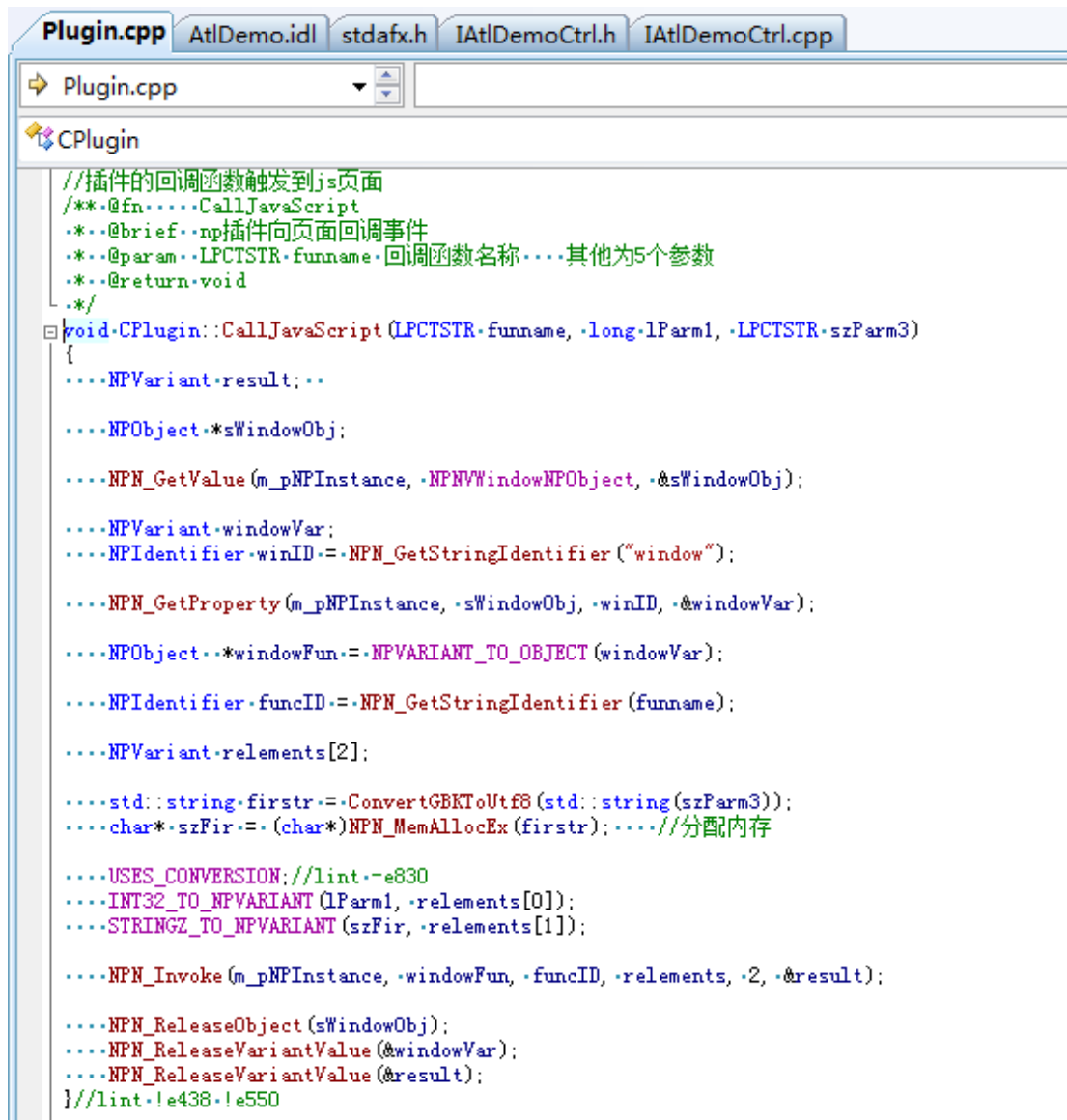
....m_hWnd=.(HWND)pNPWindow->window;
....m_npDemo=.(new(std::nothrow).NPAtlDemo(this);
....if.(NULL==.m_npDemo.||.NULL==.m_hWnd)
....{
.....return.FALSE;
....}

....m_npDemo->SetWindow(pNPWindow);
....if.(!m_npDemo->CreateControl(m_hWnd, .pNPWindow->width, .pNPWindow->height))
....{
.....return.FALSE;
....}

....OutputPrintf("CPlugin::init.ok");
....m_bInitialized=TRUE;
....//Sleep(1000);

....return.TRUE;
}
```

回调函数的处理在 CPlugin::CallJavaScript 函数，回调通知到页面：



```
Plugin.cpp AtlDemo.idl stdafx.h IAtlDemoCtrl.h IAtlDemoCtrl.cpp
Plugin.cpp
CPlugin
//插件的回调函数触发到js页面
/**@fn....CallJavaScript
...@brief...np插件向页面回调事件
...@param...LPCTSTR funname:回调函数名称....其他为5个参数
...@return void
*/
void CPlugin::CallJavaScript(LPCTSTR funname, long lParam1, LPCTSTR szParm3)
{
    ....NPVariant result;..

    ....NPObj* sWindowObj;

    ....NPN_GetValue(m_pNPInstance, NPNVWindowNPObj, &sWindowObj);

    ....NPVariant windowVar;
    ....NPIdentifier winID NPN_GetStringIdentifier("window");

    ....NPN_GetProperty(m_pNPInstance, sWindowObj, winID, &windowVar);

    ....NPObj* windowFun NPVARIANT_TO_OBJECT(windowVar);

    ....NPIdentifier funcID NPN_GetStringIdentifier(funname);

    ....NPVariant relements[2];

    ....std::string firststr ConvertGBKToUtf8(std::string(szParm3));
    ....char* szFir (char*)NPN_MemAllocEx(firststr);....//分配内存

    ....USES_CONVERSION;//lint -e830
    ....INT32_TO_NPVARIANT(lParam1, relements[0]);
    ....STRINGZ_TO_NPVARIANT(szFir, relements[1]);

    ....NPN_Invoke(m_pNPInstance, windowFun, funcID, relements, 2, &result);

    ....NPN_ReleaseObject(sWindowObj);
    ....NPN_ReleaseVariantValue(&windowVar);
    ....NPN_ReleaseVariantValue(&result);
} //lint !e438 !e550
```

hasMethod 方法:

Plugin.cpp
AtlDemo.idl
stdafx.h
IAtlDemoCtrl.h
IAtlDemoCtrl.cpp

CPlugin.CallJavaScript

CPluginScriptObject

```

/**.@fn.....hasMethod
*..@brief..判断是否有该方法
*..@param..NPIdentifier.methodName-函数名称
*..@return..存在返回true，不存在返回false
*..@note...与method相关的函数为hasMethod和Invoke，如果要为插件创建
*/
bool CPluginScriptObject::hasMethod(NPIdentifier.methodName)
{
    ....char.*pFunc.=.NPN_UTF8FromIdentifier(methodName);
    ....OutputPrintf("hasMethod:%s", .pFunc);
    ....if.(strcmp(pFunc, "Test").==.0)
    ....{
    ....return.true;
    ....}
    ....//else-if.(strcmp(pFunc, "ClearResult").==.0)
    ....//{
    ....//....return.true;
    ....//}
    ....return.false;
}

```

Invoke 方法:

Plugin.cpp
AtlDemo.idl
stdafx.h
IAtlDemoCtrl.h
IAtlDemoCtrl.cpp

CPluginScriptObject.invol
if(args != NULL && argCount >= 2)

CPluginScriptObject
invoke(NPId

```

*..@Function:.....CPluginScriptObject::invoke
*..@Description:.....NPAPI接口调用
*..@param-input:.....methodName:.....方法名
*.....args:.....接口参数列表
*.....argCount:.....接口参数个数
*..@param-output:.....result:.....接口返回值
*..@return-value:.....bool:.....true-成功, false-失败
*..@note:.....<member.Func>
**/
bool CPluginScriptObject::invoke(NPIdentifier.methodName, .const.NPVariant.*args, .uint32_t.argCount, .NPVariant.*result)
{
    ....//OutputPrintf("invokeinvokeinvoke");
    ....if.(NULL==.npp)
    ....{
    ....return.false;
    ....}
    ....CPlugin*.pPlugin.=.(CPlugin*).npp->pdata;
    ....if.(NULL==.pPlugin)
    ....{
    ....//.....OutputPrintf("pPlugin-is-NULL!!!.[test]");
    ....return.false;
    ....}

    ....char.*pFunc.=.NPN_UTF8FromIdentifier(methodName);
    ....OutputPrintf("invoke:%s", .pFunc);
    ....long.nRet;

    ....if.(strcmp(pFunc, "Test").==.0)
    ....{
    ....if.(args!=.NULL.&&.argCount.>=.2)
    ....{
    ....std::string.firststr.=.NPVARIANT_TO_GBK(args[0]);|
    ...._bstr_t.szXml.=._com_util::ConvertStringToBSTR(firststr.c_str());
    ....long.nData.=.NPVARIANT_TO_LONG(args[1]);

    ....nRet.=.pPlugin->Test(.szXml, .nData);
    ....INT32_TO_NPVARIANT(nRet, .*result);
    ....return.true;
    ....}
    ....OutputPrintf("%s.function.argument.error", .pFunc);
    ....return.false;
    ....}
}

```

### 3 控件页面调用

所有工程设置成果物生成到 bin 目录下，iereg.bat 是注册 ATL 控件的；npreg.bat 注册 np 插件；ie-test.htm 是 ie 示例页面，调用 ATL 控件（前提是 atl 控件已注册）；np-test.htm 是 chrome 示例页面，调用 np 插件（前提是 atl 控件/np 插件均已注册）

ie 示例和 np 示例页面的差别：

ie 示例通过 object 加载控件：

```
<div id="ocxContainer" style="width:100%;height:70%">
    <object classid="clsid:8F55A11E-35B9-4B5F-98BD-53502C1590DD" id="ocx" width="100%"
height="100%" name="ocx" >
    </object>
</div>
```

回调函数在单独的 script 中接收 id 为 ocx 的控件的事件：

```
<script language="javascript" for="ocx" event="MsgNotify(iMsg,szDetail)">
    szMsg = "msg:" + iMsg + ",detail:" + szDetail;
    document.frmApp.Result.value = szMsg;
</script>
```

Np 示例通过 embed 加载 np 插件：

```
<div id="ocxContainer" style="width:100%;height:70%">
    <embed type="application/hik-demo-plugin" id="ocx" width="800" height="600"
name="ocx"></embed>
</div>
```

回调函数跟普通 js 函数一样：

```
function MsgNotify(iMsg,szDetail)
{
    szMsg = "msg:" + iMsg + ",detail:" + szDetail;
    document.frmApp.Result.value = szMsg;
}
```

还有个注意事项：

Chrome 浏览器，页面传给控件的字符串必须有终止符 '\0'，不然控件收到的字符串后面会是乱码