COM 中的可连接对象与连接点机制及其实现

Connectable Object in COM and Connection Points Mechanism and its Implemention

纪澍琴（长春工业大学现代教育技术学院 130012）
王树明 张 媛 寇 峰（长春 吉林大学计算机学院 130012）

摘要：论述可连接对象和连接点机制的原理，并用 ATL 编程实现可连接对象和内嵌于客户的事件接收器，实现组件服务器与客户间的通信。

关键词：COM ATL 连接点 事件接收器

1 引言

如果软件采用 COM(Component Object Model)组件对象模型，用户可以用组件搭建自己的应用系统，可实现与原有系统等的无缝融合。另外 Internet 可利用 COM 组件对象在中间层进行事务逻辑服务，处理各种复杂的商务逻辑计算和演算规则等，其中一系列普通的服务，包括 Web 服务、组件服务和信息服务都可通过 COM 以一种统一的方式展示出来，使诸多应用之间易于交互操作和共享组件。

为了在组件对象和客户之间提供更大的交互能力，组件对象也需要主动与客户进行通信，实现异步调用(Asynchronous calling)，COM 引入了连接点机制。本文利用 VC 平台下的 ATL 和 MFC 编程技术，演示实现了客户/服务器模式下的可连接组件服务对象与客户端的双向通信。

2 可连接对象和连接点机制的基本原理

组件对象通过出接口(Outgoing Interface)与客户进行通信[1]。如果一个组件对象定义了一个或者多个出接口则此组件对象叫做可连接点对象[2]。所谓出接口也是 COM 接口，每个出接口包含一组成员函数，每个成员函数代表了一个事件、一个通知或者一个请求。但是这些接口是在客户的事件接收器(sink)中实现的，所以叫出接口。事件接收器是被用于监听并处理组件对象的通知或请求的 COM 对象。

可连接对象必须实现一个连接点容器接口(IconnectionPointContainer)，它通过连接点枚举器管理所有的出接口即连接点(Connection Points)，并输出自己的出口信息。客户可调用 IConnectionPointContainer::EnumConnectionPoints(IEnumConnectionPoints * *)函数获取连接点枚举器接口指针，从而可遍历 COM 对象的所有连接点对象[3]。

每个出接口对应一个连接点对象，连接点对象实现了 IConnectionPoint 接口。客户正是通过该接口与可连接对象建立连接。通过连接点机制，客户程序把接收器的接口指针传给可连接对象，运用该指针可连接对象可以访问接收器的成员函数[3]。如图 1 所示：

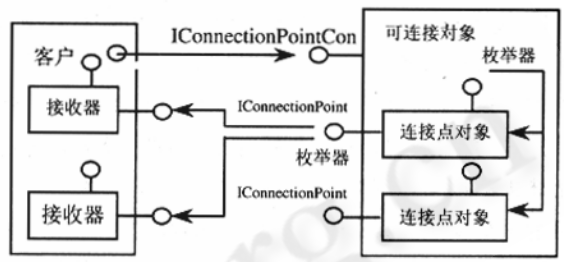

图 1 可连接机制基本结构

接口 IconnectionPoint 包含五个成员函数，用于连接点管理、返回源对象和出口 IID 信息及源对象与接收器之间的连接。客户可调用 EnumConnections 成员函数获取可连接对象实现的连接枚举器，连接枚举器内嵌一 CONNECTDATA 结构体来描述每一连接。CONNECTDATA 包含两个成员：IUnknown * pUnk 和 DWORD dwCookie，pUnk 对应于件接收器对象的接口指针，而 dwCookie 是由连接点对象生成的、用于唯一标识此连接的 32 位整数，Unadvise 函数将利用此标识找到相应的连接并取消连接点与接收器之间的连接。

综上所述，一个可连接对象必须实现 IConnectionPointContainer、IConnectionPoint、IEnumConnectionPoints、IenumConnections 四个接口。

下面是利用这些接口建立接收器到连接点之间连接的过程。客户在获取了可连接对象的 IUnknown 接口指针 pUnk 后：

(1) 调用 pUnk->QueryInterface (IID_IConnectionPointContainer,(void * *) & pConnPtCont)；如果调用成功，pConnPtCont 中将存放可连接对象的连接点容器

接口指针。如果调用不成功,则表明对象不是可连接对象。

(2) 调用 pConnPtCont -> FindConnectionPoint (IID_IsomeEventSet,&pConnPt)。如果调用成功,pConnPt 将存放对应于出接口 ISomeEventSet 的连接点对象所实现的连接点接口指针;如果调用不成功,说明可连接对象不支持出接口 ISomeEventSet。

(3) 调用 pConnPt->Advise(pSomeEventSet,&m_dwCookie) 以建立事件接收器与连接点的连接。其中 pSomeEventSet 是客户事件接收器 IUnknown 接口的指针,此指针通过此函数传递给了可连接对象以便可连接对象发起对客户的通信;m_dwCookie 是连接标识,此值由可连接对象设置由客户保存,客户还将使用此值以断开连接。可连接对象可以通过连接点调用客户事件接收器中的方法。在客户与连接点成功建立连接后,连接点中已经保存了客户事件接收器接口的指针并可以调用 pConnPt->GetConnections()来获取。

(4) 当客户要取消连接时,调用 pConnPt->Unadvise(m_dwCookie)来,同时调用 pConnPt->Release() 释放连接点对象。

3 组件服务器可连接对象编程实现

在 VC++6.0 下利用 ATL COM AppWizard 建立一个命名为 HelloServ 的远程服务器。在字符表中添加一 ID 为 ID_DISPLAY_NAME 的入口,并在 CServiceModule::Init(…)中对其进行加载。然后在 .cpp 文件中定义一全局函数 InitApplication()初始化组件对象及其安全属性。

```
extern "C" BOOL WINAPI InitApplication()
{ HRESULT hResult = CoInitialize(NULL);
    if (FAILED(hResult))
      return FALSE; //初始化 COM 失败
  //初始化安全属性并降低存取门限
  CoInitializeSecurity(NULL, -1, NULL, NULL,
RPC_C_AUTHN_LEVEL_NONE,
    RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL);
  //初始化成功
  return TRUE; }
```

将 CServiceModule::Run()中有关初始化组件对象和其安全性的代码删除代之以:

```
if (! InitApplication()) return;
```

应用 ATL Object Wizard 添加一名为 HelloWorld 的简单对象并确保选定连接点支持选项。从 ClassView 和 IDL 中可以发现由于选择了连接点支持选项,比通常多了一个 dispinterface 类型的事件接口_IHelloWorldEvents。将来需用此接口实现连接点。

给接口 IHelloWorld 添加一返回类型为 HRESULT 的成员函数 SayHello(),其实现代码为:

```
STDMETHODIMP CHelloWorld::SayHello()
{ //获取网络服务器机器名称
  TCHAR szComputerName[MAX_COMPUTERNAME_LENGTH + 1];
  DWORD dwSize = MAX_COMPUTERNAME_LENGTH + 1;
  if (! GetComputerName ( szComputerName, &dwSize))
    return E_FAIL; //获取操作失败
  return S_OK; }
```

同样给接口_IHelloWorldEvents 添加一方法即事件 OnSayHello(),输入参数为[in] BSTR bstrHost。

图 2　实现连接点后的类视结构

要实现连接点首先编译 IDL 文件,其次右击 CHelloWorld,点击实现连接点菜单项,实现连接点对话框出现,选中_IHelloWorldEvents 并确认 Ok,如图 2 所示,类视中多了一个继承自 IconnectionPointImpl 的模板类 CProxy_IHelloWorldEvents。它实现了 COM 中的代理/存根机制和 Fire_OnSayHello(…)成员函数,双击该模板类可发现 Fire_OnSayHello(…)函数实现主体为一遍历各连接并向对应客户接收器发送通知事件的 FOR 循环体。其中变量 m_vec 是 CDV 类型的向量,它记录客户事件接收器与服务器可连接对象的连接。然而这所有的一切都被 ATL 封装实现,几乎不需要我们去编程。

最后,只需对前面我们实现的 SayHello()稍示修改,在返回前对 Fire_OnSayHello()进行调用即可。其中调用语句为:Fire_OnSayHello(T2OLE(szComputerName));

4 客户方编程实现

利用向导创建一基于对话框的标准 MFC 应用程序 HelloCli。在文件 STDAFX.H 中添加 #define _WIN32_WINNT 0x0400 语句以使分布式组件对象模型(DCOM)正常运行。

可有两种方法告诉客户端服务器端的接口及其实现[5]。

(1) 通过 #import 方法引入类型库(.tlb);

(2) 包含头文件。

第二种方法非常好,可得到各接口的 C 或 C++ 声明,但需对所有接口编码定义全局唯一表识符(GUID)。而用第一种方法,程序自动实现这些 GUIDs,并且还提供一智能指针。我们采用 #import 方法,在文件 STDAFX.H 中加入如下一行语句:

#import "HelloServ.tlb" no_namespace named_guids raw_interfaces_only

为了实现出接口,我们通过 ClassWizard 添加一基类为 CcmdTarget 并选中 Automation 选项的类 CHelloWorldEvents,并删除 HelloWorldEvents.cpp 文件中有关消息映射宏和分发映射宏及对 IID_IhelloWorldEvents 接口 IID 的定义。

为添加当服务器激发 OnSayHello 事件时客户端相应的处理函数,在类向导中选定 Automation 标签,并确认在类名称一栏中类 CHelloWorldEvents 被选定。单击添加方法按扭,添加方法对话框打开,为方法添加一外部名,并必须与服务器端事件处理方法名称相一致,这里为 OnSayHello,内部名可采用默认名。返回类型必须为 void,再添加一类型为 LPCTSTR 的参数 lpszHost。下面是函数的实现,并将 HelloCliDlg.h 文件包含进来,同时将 CHelloCliDlg 类设为友类:

```
void CHelloWorldEvents:: OnSayHello ( LPCTSTR lpszHost)
{ CHelloCliDlg* pDlg = (CHelloCliDlg*)AfxGetMainWnd();
    if (pDlg ! = NULL)
    { pDlg->m_strStatus += "The OnSayHello() connection point method has been called\r\n";
     pDlg->UpdateData(FALSE); }
    // Show a message box saying 'Hello, world, from host' + lpszHost;
    CString strMessage = "Hello, world, from ";
    strMessage += lpszHost;
    AfxMessageBox(strMessage, MB_ICONINFORMATION);
}
```

在 CHelloCliDlg 类中声明如下几个保护型数据成员:

```
HICON m_hIcon;
DWORD m_dwCookie; // Cookie to keep track of connection point
BOOL m_bSinkAdvised; // Were we able to advise the server?
IHelloWorldPtr* m_pHelloWorld; // Pointer to the IHelloWorld interface pointer
IUnknown* m_pHelloWorldEventsUnk; // Pointer to the IUnknown of the event "sink"
CHelloWorldEvents m_events;
```

并在对话框构造函数中对它们进行适当初始化。

其他,方法 CHelloCliDlg::AdviseEventSink()通过调用 AfxConnectionAdvise()建立连接,编码实现如下:

```
BOOL CHelloCliDlg::AdviseEventSink()
{ if (m_bSinkAdvised)
    return TRUE;
  IUnknown* pUnk = NULL;
  CComPtr< IUnknown > spUnk = (* m_pHelloWorld);
  pUnk = spUnk.p;
  // Advise the connection point
  BOOL bResult = AfxConnectionAdvise(pUnk,
  IID_IHelloWorldEvents,
      m_pHelloWorldEventsUnk, TRUE, &m_dwCookie);
  return bResult;
}
```

对应的断开连接调用 AfxConnectionUnadvise()。

通过函数 OnStartServer()可以打开连接服务器并获取远端 IhelloWorld 接口指针,其主要实现代码有:

```
void CHelloCliDlg::OnStartServer()
{
    COSERVERINFO serverInfo;
    ZeroMemory(&serverInfo, sizeof(COSERVERINFO));
    ...
    serverInfo.pwszName = L"\\Viz-06";
```

```
...
MULTI_QI qi = {&IID_IHelloWorld, NULL, S_OK};
...
try
{
m_pHelloWorld = new IHelloWorldPtr;
}
catch(...)
{
  AfxMessageBox(AFX_IDP_FAILED_MEMORY_
ALLOC, MB_ICONSTOP);
  ...
  return;
}
HRESULT hResult = CoCreateInstanceEx(CLSID
_HelloWorld, NULL,
CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_
SERVER, &serverInfo, 1, &qi);
if (FAILED(hResult))
{...
  return;
}
m_pHelloWorld -> Attach((IHelloWorld * ) qi.
pItf);
  // Now we have a live pointer to the IHelloWorld
interface on the remote host
  ...
  return;
}
```

5 结论

本文根据可连接对象和连接点机制的原理，采用 ATL 编程实现可连接对象和内嵌于客户的事件接收器，演示组件服务器与客户间的通信。该技术应用到系统开发中，可增强系统扩展性，使系统集成升级更加方便，使用户完全可以按照公司的规模和硬件网络设施情况，对现有功能随意拼和、集成和拆分，将系统的性能扩充到最佳状态。

参考文献

1 Jesse Liberty 等著，COM 技术内幕，清华大学出版社，1996。

2 潘爱民 著，COM 原理与应用，清华大学出版社，2001。

3 Ashish Dhar. Connection Points And Asynchronous calls. http://www.codeproject.com/。

4 David J. Krulinski 等著，Visual C++ 6.0 技术内幕（第五版），北京希望电子出版社，1999。

5 刘浩兵、金国栋，使用 COM 技术在 Visual C++ MFC 编程，