一种异构语言系统间数据通信的实现①

Implementation on Communication of Data Between Applications Developed by Different Language

张　金　（武汉华中科技大学机械科学与工程学院 430074）
（武汉开目信息技术有限责任公司 430223）
游承启　（武汉华中科技大学机械科学与工程学院 430074）

摘要：制造业企业内部各个部门使用着不同开发语言所开发的应用系统，要实现这些异构语言系统的良好集成，解决异构语言系统的进程间数据通信就是一个关键的问题。本文提出了通信底层、通信代理控件和上层应用的异构语言系统间通信三层模型的设计思想，并以 VC 与 PB 程序为例详细介绍了一种能实现异构语言系统本地和远程的进程间通信的解决方案。

关键词：异构系统　命名管道　企业应用集成　代理

1　引言

随着计算机技术的不断普及，制造业企业信息化程度日益提高，许多由不同开发语言开发而成的异构应用系统都已经应用在诸如产品设计、生产管理、财务统计等各个部门，发挥着很大的作用[1]。由于企业信息化程度越来越高，越来越需要企业中各个不同应用系统间能够实现无缝集成，从而实现不同 IT 系统间协同工作和信息共享，进而提高企业工作效率。然而这些应用系统所使用的开发语言和开发工具不尽相同，语言开放性各有差异，因而如何使异种语言间的应用方便地实现数据通信是解决异构系统集成的关键问题。本文以 VC 与 PB 程序为例，探讨在底层采用命名管道、在中间层采用通信代理组件 OCX 控件、在上层是应用程序的异构语言系统间数据通信三层模型，并加以实现。

2　异构语言系统三层通信模型的实现思想

在实现异构语言应用系统集成中，按通信、管理和应用的功能把异构语言应用系统集成划分为通信底层、通信代理和上层应用三个层次，使得数据传送、管理与上层应用分离开来，这样使得应用系统只需解决自身需要解决的业务逻辑，不需要关心数据的通信和其他管理活动。

通信底层只需负责基本的数据传送和接收功能，将数据从一个进程传送到另一个进程，如果这两个进程分别运行在不同的计算机上，就通过网络，传送到另一个进程中。通信底层提供基本的创建服务、连接、数据发送和接收接口。通信底层可以有多种实现方法，其中命名管道是一种不错的实现方式。

通信代理层位于通信底层与上层应用之间，负责代理上层应用，调用通信底层提供的接口，进行数据的收发；另外数据的缓存、转发以及其他管理功能也在此层上实现，从而简化通信底层的设计和减轻底层负担。

通信代理采用 OCX 控件形式，有以下优点：

（1）屏蔽上层应用程序对通信底层的差异，开发应用程序的语言有多种，其中对命名管道的支持各不相同。VC 对命名管道的支持得较好，而 PB 则不支持命名管道。为了便于各种语言开发的应用都能通过管道地层通信，采用 OCX 控件是一种较好的选择；

（2）便于上层应用使用，OCX 控件注册后，在各种开发环境中都可以方便的使用；

（3）使得进程间通信的功能有着良好的可扩展性，只要保持接口不变，上层应用程序无需做任何改动，就可以实现通信功能的升级[4]；

上层应用程序使用通信代理提供的对外接口进行数据的接收和发送；通信代理通过调用上层应用设置的回调函数或者向上层应用发送消息来通知上层应用程序有数据接收。

三层结构层次清晰、结构明确，各层次各负其职，从而简化了异种语言进程间数据交互的实现。

通信原理如图 1 所示。

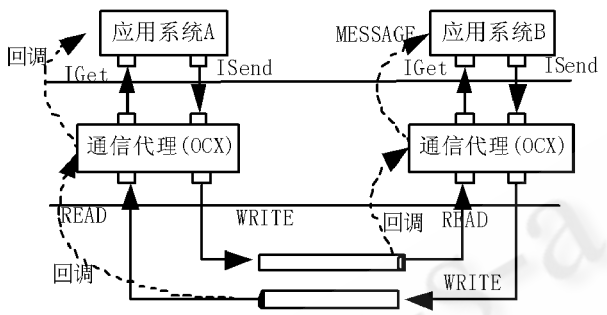
3　三层模型结构的实现

3.1　通信底层的数据传递[3]

通信底层的数据传送可以由 SOCK、命名管道等方式来

① 基金项目：本课题得到国家高技术研究发展计划 863 计划（2003AA411011、2003AA411042）资助

实现,这里采用命名管道来实现底层数据传送。管道是一块特殊的内存区域,由两个或更多进程共享,并使这些进程能够互相通信。它分为匿名管道和命名管道。匿名管道用来在具有亲缘关系的两个进程之间,如父子进程之间、兄弟进程之间数据传输。而命名管道则可以实现在同一台计算机不同进程之间,或不同计算机的不同进程之间,支持可靠的、单向或双向的数据通信。用命名管道实现通信底层十分方便,设计者不需要深入了解基层的网络传输协议。



异构语言系统数据通信原理图

图 1　异构语言数据通信流程

管道具有两个端点,有一端句柄的进程可以与有另一端句柄的进程进行通信。它可以用来发送一个数据流。虽然命名管道支持全双工通信,但为了保持稳定的数据通信,在实现时使用收管道和发管道来进行通信[2]。

通信底层为一个 DLL,用 VC 生成,其中封装 CNamedPipe 类,在此类中封装了用于数据传输的基本函数:bool CreatePipe() 用于命名管道的创建;bool ConnectPipe() 用于命名管道的连接;void SetPipeName(LPCTSTR szName,LPCTSTR szHost),用于设置管道名称;bool Send(LPCTSTR szMsg) 用于发送数据;监听线程函数 static void ListnerProc(LPVOID lpParameter) 和设置回调函数的函数 SetCallFunc()。

(1) 运行在两台终端上的进程采用命名管道进行通信时,其中一个进程作为服务器,另一个进程作为客户端。使用命名管道之前,首先要由服务器进程创建一个命名管道,并获得管道这一端的句柄。一旦管道被创建,就可以供其他进程使用它的名字与它进行连接。为了便于在网络上进行传输,创建命名管道时需要指定一个主机名和管道名,客户端可以采用以下格式完成:

\[host_name] \pipe \[pipe_name] \;也可以是:\. \pipe\pipe_name\(其中:. 表示本机)。而服务器端只能够以指定本机作为主机名,即只能使用 \. \pipe_name\格式。此外,需要注意的是:在同一主机上的管道名称是唯一的,一个命名管道一旦被创建,就不允许再创建相同名称的管道[3]。服务器进程通过函数 void SetPipeName() 设置命名管道的名称。在这里 szName 为要创建命名管道的名称;szHost 为主机 IP 地址。

(2) 服务器端创建管道:设置了管道路径和名称后,服务器进程通过 CreatePipe() 函数来创建命名管道,用 WIN API 函数 CreateNamedPipe() 来创建命名管道和打开已经存在的命名管道。其原型为:

```
HANDLE CreateNamedPipe(
  LPCTSTR lpName, //管道
  DWORD dwOpenMode, // 打开方式
  DWORD dwPipeMode, // 管道类型
  DWORD nMaxInstances, // 管道的最大数量
  DWORD nOutBufferSize, // 写缓冲区大小
  DWORD nInBufferSize, // 读缓冲区大小
  DWORD nDefaultTimeOut, // 最长的等待时间
  LPSECURITY_ATTRIBUTES lpSecurityAttributes // 安全属性
);
```

lpName 指出的管道名称,它包括网络名称和管道名称。这样可以保证管道可以被本地或者远程的进程使用。通过 dwOpenMode 可以指定管道的访问模式:PIPE_ACCESS_DUPLEX,允许数据双向传递;PIPE_ACCESS_INBOUND,数据从客户端传向服务器;PIPE_ACCESS_OUTBOUND,数据从服务器传向客户端。

(3) 客户端连接管道。一旦服务端创建管道并等待连接后,客户端通过 ConnectPipe() 函数来连接命名管道。由于命名管道单独构成一种独立的文件系统,对于管道两端的进程而言,就是一个文件,但它不是普通的文件,并且只存在与内存中。因而 Initialize() 调用 CreateFile() 来打开一个文件句柄用于与管道作新连接。

```
HANDLE CreateFile(
  LPCTSTR lpFileName, // 文件名
  DWORD dwDesiredAccess, // 存取方式
  DWORD dwShareMode, // 共享方式
  LPSECURITY – ATTRIBUTES lpSecurityAttributes, // 安全属性
  DWORD dwCreationDisposition, // 创建方式
  DWORD dwFlagsAndAttributes, // 文件属性
  HANDLE hTemplateFile // 临时文件句柄
);
```

其中,CreateFile 有很多用途,可以用来创建文件、管道、邮件槽、目录等,这里用 CreateFile 打开客户端命名管道;lp-

FileName 用于指明管道名称；dwDesiredAccess 用于表明使用方式，取下值之一：GENERIC – READ：打开一个只用于读的管道；GENERIC – WRITE：打开一个只用于写的管道；GENERIC – READ | GENERIC – WRITE：打开一个用于读和写的管道。命名管道创建之后，获得了命名管道的句柄，然后可以通过 WriteFil( ) 将缓存中的数据发送到对方进程。

（4）设置回调函数。当通信底层的监听线程接收到发送来的数据后，通过调用通信代理早先设置的回调函数来通知通信代理。首先在头文件中定义回调函数：static void (* funcCallBack) (LPCTSTR buf) ；其参数为收到字符串指针。然后通过函数 void SetCallBackFunc (void (* fCallBack) (LPCTSTR buf)) 设置回调函数，相关代码：funcCallBack = fCallBack。

（5）数据的发送和接收。命名管道建立之后，通信底层通过 Windows API 函数 WriteFile( ) 和 ReadFile( ) 来发送和接收数据。

发送数据函数为：bool CNamedPipe: : Send (LPCTSTR szMsg)

```
{
  DWORD dwSent;
  BOOL bOK = WriteFile ( m_hOutPipe, szMsg. , strlen ( szMsg) +1, &dwSent, NULL) ;
  if ( ! bOK || ( szMsg. length( ) +1) ! = dwSent)
    return false;
  return true;
}
```

通信底层通过监听线程来监听并接收数据。在函数 CreatePipe( ) 和 ConnectPipe( ) 中，创建或连接命名管道后，创建监听线程，相关代码为：

```
DWORD dwThreadId, dwThrdParam =1;
m _ hListner = CreateThread ( NULL, 0, ListnerProc, &dwThrdParam, 0, &dwThreadId) ;
if ( m_hListner = = NULL || m_hListner = = INVALID_HANDLE_VALUE)
  return false;
}
```

监听线程用 ReadFile( ) 来接收数据。ReadFile( ) 从命名管道中读取数据到自己的接收缓存中。命名管道是支持阻塞的，当没有数据到来时，就使监听线程挂起；如果对方进程终端或者网络故障，ReadFile( ) 则返回错误。相关代码为：

```
void ListenProcess( )
{
  DWORD dwRetVal = 0;
  for ( bool bContinue = true; bContinue && dwRetVal = = 0; Sleep( 0) )
  {
    char buf[ PIPE_BUF_SIZE] ;
    DWORD dwRead;
    BOOL bOK = ReadFile ( m_InPipe, buf, PIPE_BUF_SIZE, &dwRead, NULL) ;
    if ( dwRead = = 0)
      continue;
    if ( ! bOK)
      dwRetVal = 2;
    LPCTSTR szMsg = buf;
    funcCallBack( szMsg) ; ;
  }
}
```

这样通信底层就基本建立起来了。

3.2 通信代理的实现

虽然 VC 中可以支持命名管道，但是 PB 不支持命名管道，并且由于 PB 自身封装的严密性，它也不支持回调函数。为了便于 PB 程序能够并且便捷的使用通信底层，则需要通过一个通信代理来实现底层通信和应用系统的连接。由于在 VC 中创建管道很方便，该通信代理由 VC 来实现。

为了方便使用，通信代理 Agent 可以做成一个 OCX 控件。它位于通信底层与上层应用系统之间，提供数据收发、数据缓冲等服务。它实现四个接口：void ICreatePipe (BSTR bstr_Host , BSTR bstr_PipeName) 、void IConnectPipe (BSTR bstr_Host, BSTR bstr_PipeName) 、void ISend (BSTR szMsg) 、void IGet (BSTR * szMsg) [4] 和 ISetCallBackFunc (void (* fCallBack) (char * buf)) 。

服务器启动时，通过 ICreatePipe 创建管道，相关代码：

```
{ …
  SetPipeName( PipeName, Host) ; //指明要创建管道名称
  CreatePipe( ) ;
}
```

客户端通过 IConnectPipe 接口连接管道，相关代码：

```
{
  …
  SetPipeName( PipeName, Host) ;//指明要连接的管道名称
  ConnectPipe( ) ;
}
```

在底层铺设好通信管道后，通信双方可以通过 ISend 接

口将要发送的数据交给通信代理,再由通信代理通过管道传送到其他进程中去。通信代理将管道创建之后,同时在通信底层创建一个监听线程,负责监听从对方管道传送来的数据。

3.3 通信代理与应用程序的交互

通信底层可以与上层应用通过通信代理实现双层回调,即底层回调通信代理的函数,通信代理再次回调上层应用的函数;这样经过两次回调,底层可以把数据传送给上层应用。

通信代理在自身设置一个回调函数 static void OnBallBack(LPCTSTR szMsg);通过调用通信底层的 SetCallBackFunc()来设置回调函数;同时通信代理对上层应用也提供 ISetCallBackFunc 接口,以供上层应用设置自己的回调函数。当通信代理自己的回调函数被底层回调时,代理立刻回调上层应用的回调函数,从而将数据传递给上层应用。

通过 regsvr32 注册通信代理 OCX 控件,在 PB 中就调用 Agent 控件很方便。通过"create ole_control"按钮载入 Agent 控件后,按照 ole_agent. object. Func() 的格式就可以调用 Agent 提供的接口了。例如:PB 程序作服务客户端,用 ole_agent. object. ConnectPipe("192. 168. 100. 82","MyPipe") 连接运行在"192. 168. 100. 82"名"MyPipe"的为命名管道。

对于 PB 等不支持回调的开发工具,通信代理另外通过发送 WINDOWS 消息,触发上层应用自定义的消息响应函数,然后在该消息响应函数中调用 IGet 接口来取得数据。

在 PB 中,通过从 01 到 75 的 pbm_custom 这一套 EventID 来自定义消息。通过 WIN API SendMessage() 发送消息 WM_USER 就会触发 EventID 为 pbm_custom01 的自定义事件,发送消息 WM_USER + 66 就会触发 EventID 为 pbm_custom67 的自定义事件,依次类推。

通信代理的回调函数被触发后,立即执行 void NotifyMsg() 函数。该函数负责向上层 PB 应用发送消息,通知有数据收到。在 PB 应用中定义消息响应函数 OnReceiveData(),其 EventID 为 pbm_custom67。

4 结束语

本文提出了面向应用集成的一种基于通信底层和通信代理控件的异构语言应用系统间数据通信的模型及解决方案,并经过了实践验证。该方案把上层应用逻辑与底层通信分离开来,既能使上层应用系统关注其应用逻辑,又能高效、安全的实现跨进程的数据传输,并且提供统一的接口,使上层应用系统的通信更加方便,很好的解决了异构语言系统间集成的数据通信问题,对于提高企业中各部门的 IT 系统之间相互协作的效率有着积极意义。

参考文献

1 张金、易铭、陈卓林,基于组件技术的跨语言异构系统集成研究[J],计算机工程与科学 vol. 26, No. 8, 2004。
2 邢得奇、张有光、戴陇咸、马浩凯,进程间通信 COM 组件的设计和实现[J],计算机应用 vol. 23, June 2003。
3 求是科技, WINDOWS 网络编程[M],P77 – P104,人民邮电出版社,2003。
4 潘爱民, COM 原理与应用[M],清华大学出版社,2000。