**软件系统之间的接口方式**

**•  概述**

　　软件接口是实现一个系统跟另外系统进行信息交互的桥梁，在不同的系统之间，根据系统的关联程度的不同存在紧耦合和松耦合两种：紧耦合要求接口响应反应快，消息不能阻塞；松耦合对响应反应要求比较低。在目前应用中， Socket 、消息队列（Message Queue）、 WebService等都有相应的应用，但是应用中发现各通讯方式有自己固有的特征，“适合的才是最好的”，这是真理。

　　在接口和系统信息交互的过程中，两种模式使用得很普遍：同步调用和异步调用，同步调用要求接口发出请求消息后必须等待服务端系统的应答消息，接口阻塞直至超时；异步调用则发出请求消息后，接口可以从事其它处理，定时轮询服务端应答消息和消息或事件通知。同步方式简单，但是很容易造成接口阻塞，造成消息积压超时。

**•  技术实现**

**•  Socket 通讯**

　　Socket 通讯相对来说是很古老的通讯方式，也是最常用的通讯方式。 Socket 通讯有阻塞和非阻塞两种方式。在同步方式，采用阻塞编程比较简单，但是为了防止接口阻塞，我们需要设置 Socket 超时，因此可以使用 Socket 的 SELECT 模型（参考如下示例代码）：

　　ReceLen=0;

　　CurReceLen=0;

　　for(;;)

　　{

　　iResult=select(0,&fdread,NULL,NULL,&timeout);

　　if(iResult==0)

　　{

　　AfxMessageBox("接收应答消息超时!!!",MB\_OK|MB\_ICONERROR);

　　closesocket(Socket);

　　return FALSE;

　　}

　　CurReceLen = recv(Socket, oBuf+ReceLen, len, NO\_FLAG\_SET);

　　if((CurReceLen>0) && (CurReceLen != SOCKET\_ERROR))

　　{

　　oBuf[ReceLen+CurReceLen]='\0';

　　memcpy((char \*)&MsgLen,oBuf,sizeof(WORD32));

　　MsgLen=ntohl(MsgLen);

　　if(ReceLen+CurReceLen==MsgLen)

　　{

　　ReceLen+=CurReceLen;

　　break;

　　}

　　ReceLen+=CurReceLen;

　　}

　　}

　　在异步方式下，采用非阻塞方式实现比较方便，在非阻塞方式下可使用WSAAsyncSelect模型和 WSAEventSelect 模型： WSAAsyncSelect模型基于消息， WSAEventSelect 模型基于事件，下面的示例代码设置了 Socket 进行读写和关闭操作的消息：

　　status = WSAAsyncSelect(TempSocket, hWnd, WSA\_READ, FD\_READ | FD\_CLOSE | FD\_WRITE);

　　if (status == SOCKET\_ERROR)

　　{

　　WriteLogFile("Set stream socket module fail!!!IP(%s),Port(%d) and 　　error(%d)",GetIPAddr((PeerMap+node)->IPAddr),(PeerMap+node)->PeerPortNo,WSAGetLastError());

　　CloseSocket(TempSocket,\_\_LINE\_\_,\_\_FILE\_\_);

　　return FALSE;

　　}

无论使用阻塞方式或非阻塞方式编程，需要重点考虑的一个问题：粘包现象，即应用发送两个或以上的数据包，在 Socket 通讯层将数据包合并成一个发送出去，因此接收端收到数据包以后需要对数据包根据应用定义的长度进行拆分，否则导致应用层丢包。

应用方式可以由用户封转成DLL供使用方使用.

**•  消息队列(Message Queue)**

利用 MSMQ（Microsoft Message Queue），应用程序开发人员可以通过发送和接收消息方便地与应用程序进行快速可靠的通信。消息处理为您提供了有保障的消息传递和执行许多业务处理的可靠的防故障方法。 MSMQ与XML Web Services和.Net Remoting一样，是一种分布式开发技术。但是在使用XML Web Services或.Net Remoting组件时，Client端需要和Server端实时交换信息，Server需要保持联机。MSMQ则可以在Server离线的情况下工作，将Message临时保存在Client端的消息队列中，以后联机时再发送到Server端处理。 显然，MSMQ不适合于Client需要Server端及时响应的这种情况，MSMQ以异步的方式和Server端交互，不用担心等待Server端的长时间处理过程。 虽然XML Web Services和.Net Remoting都提供了[OneWay]属性来处理异步调用，用来解决Server端长方法调用长时间阻碍Client端。但是不能解决大量Client负载的问题，此时Server接受的请求快于处理请求。 一般情况下，[OneWay]属性不用于专门的消息服务中。

1. 基本术语和概念（ Basic terms and concepts ）

“消息”是在两台计算机间传送的数据单位。消息可以非常简单，例如只包含文本字符串；也可以更复杂，可能包含嵌入对象。 消息被发送到队列中。“消息队列”是在消息的传输过程中保存消息的容器。消息队列管理器在将消息从它的源中继到它的目标时充当中间人。队列的主要目的是提供路由并保证消息的传递；如果发送消息时接收者不可用，消息队列会保留消息，直到可以成功地传递它。 “消息队列”是 Microsoft 的消息处理技术，它在任何安装了 Microsoft Windows 的计算机组合中，为任何应用程序提供消息处理和消息队列功能，无论这些计算机是否在同一个网络上或者是否同时联机。 “消息队列网络”是能够相互间来回发送消息的任何一组计算机。网络中的不同计算机在确保消息顺利处理的过程中扮演不同的角色。它们中有些提供路由信息以确定如何发送消息，有些保存整个网络的重要信息，而有些只是发送和接收消息。 “消息队列”安装期间，管理员确定哪些服务器可以互相通信，并设置特定服务器的特殊角色。构成此“消息队列”网络的计算机称为“站点”，它们之间通过“站点链接”相互连接。每个站点链接都有一个关联的“开销”，它由管理员确定，指示了经过此站点链接传递消息的频率。

“消息队列”管理员还在网络中设置一台或多台作为“路由服务器”的计算机。路由服务器查看各站点链接的开销，确定经过多个站点传递消息的最快和最有效的方法，以此决定如何传递消息。

2. 队列类型（ Queue Type ）

有两种主要的队列类型：由您或网络中的其他用户创建的队列和系统队列。 用户创建的队列可能是以下任何一种队列： “公共队列”在整个“消息队列”网络中复制，并且有可能由网络连接的所有站点访问。 “专用队列”不在整个网络中发布。相反，它们仅在所驻留的本地计算机上可用。专用队列只能由知道队列的完整路径名或标签的应用程序访问。 “管理队列”包含确认在给定“消息队列”网络中发送的消息回执的消息。指定希望MessageQueue 组件使用的管理队列（如果有的话）。 “响应队列”包含目标应用程序接收到消息时返回给发送应用程序的响应消息。指定希望 MessageQueue 组件使用的响应队列（如果有的话）。 系统生成的队列一般分为以下几类： “日记队列”可选地存储发送消息的副本和从队列中移除的消息副本。每个“消息队列”客户端上的单个日记队列存储从该计算机发送的消息副本。在服务器上为每个队列创建了一个单独的日记队列。此日记跟踪从该队列中移除的消息。 “死信队列”存储无法传递或已过期的消息的副本。如果过期或无法传递的消息是事务性消息，则被存储在一种特殊的死信队列中，称为“事务性死信队列”。死信存储在过期消息所在的计算机上。有关超时期限和过期消息的更多信息，请参见默认消息属性。 “报告队列”包含指示消息到达目标所经过的路由的消息，还可以包含测试消息。每台计算机上只能有一个报告队列。 “专用系统队列”是一系列存储系统执行消息处理操作所需的管理和通知消息的专用队列。 在应用程序中进行的大多数工作都涉及访问公共队列及其消息。但是，根据应用程序的日记记录、确认和其他特殊处理需要，在日常操作中很可能要使用几种不同的系统队列。

3. 同步和异步通信（ Synchronous VS. Asynchronous Communication ）

队列通信天生就是异步的，因为将消息发送到队列和从队列中接收消息是在不同的进程中完成的。另外，可以异步执行接收操作，因为要接收消息的人可以对任何给定的队列调用 BeginReceive 方法，然后立即继续其他任务而不用等待答复。这与人们所了解的“同步通信”截然不同。 在同步通信中，请求的发送方在执行其他任务前，必须等待来自预定接收方的响应。发送方等待的时间完全取决于接收方处理请求和发送响应所用的时间。

4. 同消息队列交互（ Interacting with Message Queues ）

消息处理和消息为基于服务器的应用程序组件之间的进程间通信提供了强大灵活的机制。同组件间的直接调用相比，它们具有若干优点，其中包括：

•稳定性 — 组件失败对消息的影响程度远远小于组件间的直接调用，因为消息存储在队列中并一直留在那里，直到被适当地处理。消息处理同事务处理相似，因为消息处理是有保证的。 •消息优先级 — 更紧急或更重要的消息可在相对不重要的消息之前接收，因此可以为关键的应用程序保证足够的响应时间。 •脱机能力 — 发送消息时，它们可被发送到临时队列中并一直留在那里，直到被成功地传递。当因任何原因对所需队列的访问不可用时，用户可以继续执行操作。同时，其他操作可以继续进行，如同消息已经得到了处理一样，这是因为网络连接恢复时消息传递是有保证的。 •事务性消息处理 — 将多个相关消息耦合为单个事务，确保消息按顺序传递、只传递一次并且可以从它们的目标队列中被成功地检索。如果出现任何错误，将取消整个事务。 •安全性 — MessageQueue 组件基于的消息队列技术使用 Windows 安全来保护访问控制，提供审核，并对组件发送和接收的消息进行加密和验证。

5. 在 .Net 环境下编写简单的 Message Queue 程序

（1）先安装Message Queuing Services

通过Control Panel，“Add/Remove Programs” – “Add/Remove Windows Components”步骤安装MSMQ。 MSMQ可以安装为工作组模式或域模式。如果安装程序没有找到一台运行提供目录服务的消息队列的服务器，则只可以安装为工作组模式，此计算机上的“消息队列”只支持创建专用队列和创建与其他运行“消息队列”的计算机的直接连接。

（2）配置MSMQ

打开Computer Management – Message Queuing，在Private Queues下创建MSMQDemo队列

（3）编写代码－简单演示MSMQ对象

MessageQueue 类是“消息队列”周围的包装。MessageQueue 类提供对“消息队列”队列的引用。可以在 MessageQueue 构造函数中指定一个连接到现有资源的路径，或者可在服务器上创建新队列。在调用 Send、Peek 或 Receive 之前，必须将 MessageQueue 类的新实例与某个现有队列关联。 MessageQueue 支持两种类型的消息检索：同步和异步。同步的 Peek 和 Receive 方法使进程线程用指定的间隔时间等待新消息到达队列。异步的 BeginPeek 和 BeginReceive 方法允许主应用程序任务在消息到达队列之前，在单独的线程中继续执行。这些方法通过使用回调对象和状态对象进行工作，以便在线程之间进行信息通讯。

// Send Message

private void btnSendMessage\_Click(object sender, System.EventArgs e)

{

// Open queue

System.Messaging.MessageQueue queue = new System.Messaging.MessageQueue(".\\Private$\\MSMQDemo");

// Create message

System.Messaging.Message message = new System.Messaging.Message();

message.Body = txtMessage.Text.Trim();

message.Formatter = new System.Messaging.XmlMessageFormatter(new Type[] {typeof(string)});

// Put message into queue

queue.Send(message);

}

// Receive Message

private void btnReceiveMessage\_Click(object sender, System.EventArgs e)

{

// Open queue

System.Messaging.MessageQueue queue = new System.Messaging.MessageQueue(".\\Private$\\MSMQDemo");

// Receive message, 同步的Receive方法阻塞当前执行线程，直到一个message可以得到

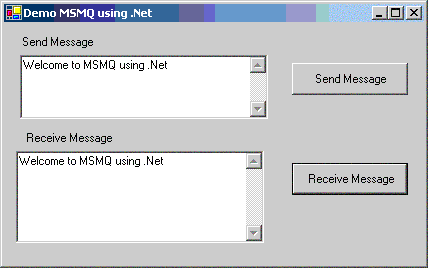
System.Messaging.Message message = queue.Receive();

message.Formatter = new System.Messaging.XmlMessageFormatter(new Type[] {typeof(string)});

txtReceiveMessage.Text = message.Body.ToString();

}

Demo界面：



[gigi_miao](http://tech.163.com/)

**•  WebService**

　　SOAP 作为一种协议，同服务端 WebService 进行通讯。微软提供了 SOAP 协议的 SDK ，我使用的是微软的 SOAP Toolkit3.0 ，这是基于 COM 的一套组件，因此具有 COM 的特征，在调用参数的处理， windows 和 unix 顺序恰好相反，下面的代码实现了调用一个 Web Service ：

　　if(!m\_bFlatType)

　　{

　　for(i=paramNum,j=0;i>j;i--,j++)

　　{

　　VARIANTARG argTemp;

　　VariantInit(&argTemp);

　　argTemp=va[i-1];

　　va[i-1]=va[j];

　　va[j]=argTemp;

　　}

　　}

　　params.cArgs = paramNum;

　　params.rgvarg = va;

　　params.cNamedArgs = 0;

　　params.rgdispidNamedArgs = NULL;

　　hr = SoapConnect.pSoapClient[index]->Invoke(dispidFn, IID\_NULL, LOCALE\_SYSTEM\_DEFAULT, 　　DISPATCH\_METHOD, &params,

　　&result, 0, 0);

　　if(FAILED(hr))

　　{

　　HandleHResult(\_T("Invoke of "+strService+" method failed."), hr);

　　VariantClear(&result);

　　for(i=0;i<MAX\_PARAM\_NUM;i++) VariantClear(&va[i]);

　　SysFreeString(bstrServiceName);

　　CoUninitialize();

　　return FALSE;

　　}

**•  总结**

　　在三种通讯方式中，各有优缺点，但是主要还在于服务端采用什么技术方案来实现，接口必须对应采用相应的通讯模式。