互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

第 24 章　通道模型和绑定

从 SOA 的实现来看，客户与服务、服务于服务之间的消息传递是关键模块之一。在 WCF 的模型中，无论通信的双方处在完全不同的局域网内，还是恰好被寄宿在同一进程内，编程模型都将通信的另一端视为“远端”，都会为传输建立完整的通道和编码器。而绑定和地址，也正是终节点中和通信有关的两个要素。

24.1　WCF 通道模型

绑定的本质是一个配置好的通道栈，为了方便程序员专注于业务逻辑，WCF 提供了一系列常用的绑定。但是在学习绑定的使用之前，了解 WCF 通道模型有助于读者对消息的发送和接收有更直观地理解。本节将介绍 WCF 通道模型。

24.1.1　WCF 通道模型概述

正如本章前言所述的，在 WCF 编程模型中，无论交互的另一方的具体位置在哪里，WCF 都会为消息的发送和接收建立一套完整的消息管道，这个消息管道也被称为通道栈（channel stack）。通道栈中的每一个通道组件，都有机会对消息进行处理，而整个通道栈是可编辑并且可插入的，这就确保 WCF 的通道模型具有相当大的灵活性。另外，WCF 通道模型是完全和上层程序隔离的，任何一个服务/客户端都可以轻松地配置到不同的通道模型上去。

通道模型可以被划分为上下两个部分，上面部分的通道被称为协议通道，而下部分的通道被称为传输通道。一个通道栈可以拥有任意个协议通道，但一般只拥有一个传输通道。传输通道负责把消息进行编码并且发送到远端，编码时传输通道需要使用通道栈的编码器。图 24.1 展示了通道模型的结构。

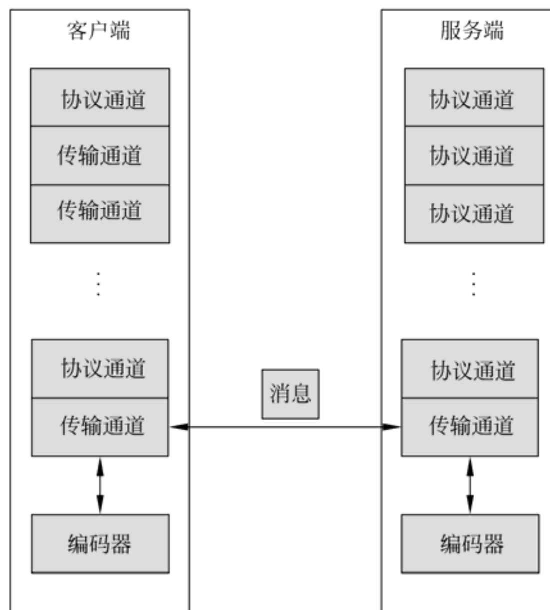


图 24.1　WCF 通道模型

一般而言，协议通道负责维护消息的非业务逻辑功能，这样的功能包括：事务、

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

日志、可靠消息、安全性等。程序员可自定义协议通道并且插入到通道栈中。在一个通道栈中，必须包含至少一个传输通道和编码器，传输通道负责消息的编码和发送。具体地，传输通道会尝试从 BindingContext 对象中查找编码器。如果没有找到则会使用默认的编码器，在完成消息的编码之后，传输通道负责把消息发送到远端，这里不同的传输通道将使用不同的传输协议，例如 HTTP、TCP、IPC 等。

24.1.2　消息交换模式和通道形状

站在消息传输层面，WCF 一共支持 6 种消息交互模式，分别为：数据报模式、请求-响应模式、双工模式、会话数据报模式、会话请求-响应模式和会话双工模式。一个通道可以支持其中一种或者几种交互模式，通道通过通道形状（channel shape）来实现具体的消息交互模式。

所谓的通道形状，指的是定义了发送、接收消息动作的 10 个接口，它们均继承自 IChannel 接口。这 10 个接口是：IInputChannel、IOutputChannel、IRequestChannel、IreplyChannel、IDuplexChannel、IInputSessionChannel、IOutputSessionChannel、IRequestSessionChannel、IReplySessionChannel 和 IDuplexSessionChannel。下面来介绍消息交互模式，以及和具体模式有关的通道形状。

24.1.3　数据报模式

数据报模式指的是发送端负责把消息发送给对方并且收到确认消息之后，就完成消息交互的方式。在这种模式下，发送方唯一能确定的就是消息发送成功，而对于消息是否最终到达服务的终节点、是否被成功处理、返回结果如何都一无所知。图 24.2 展示了数据报模式的传输机制。

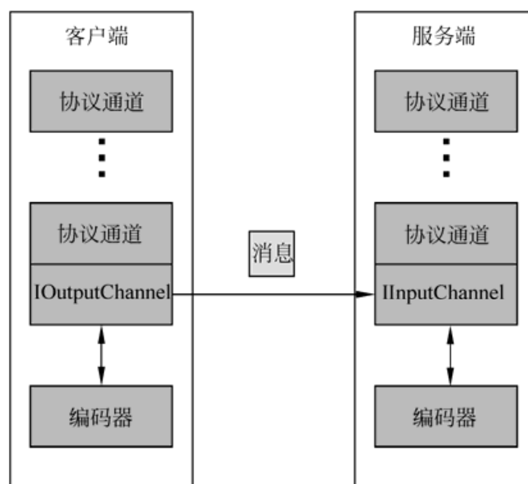
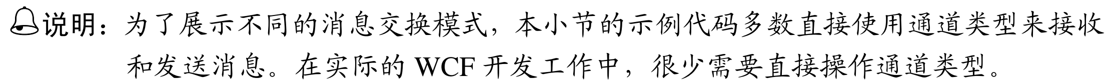


图 24.2　数据报模式传输机制

采用数据报模式的客户端通道实现 IOutputChannel 接口，而采用数据报模式的服务端通道则实现 IInputChannel 接口。在实际项目中，程序员直接使用数据报消息模式的机会并

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

不多。下面的代码展示了数据报模式的发送和接收。发送端的代码如示例代码 24-1 所示。



说明：为了展示不同的消息交换模式，本小节的示例代码多数直接使用通道类型来接收和发送消息。在实际的 WCF 开发工作中，很少需要直接操作通道类型。

示例代码 24-1

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Xml;
using System.Runtime.Serialization;
namespace WCF.Second
{
    class Output
    {
        static void Main(string[] args) //入口方法
        {
            try
            {
                //建立自定义的通道栈
                BindingElement[] bindingElements = new BindingElement[3];
                bindingElements[0] = new
                TextMessageEncodingBindingElement();//文本编码
                //OneWayBindingElement 可以使得传输通道支持数据报模式
                bindingElements[1] = new OneWayBindingElement();
                bindingElements[2] = new HttpTransportBindingElement();
                //Http 传输
                CustomBinding binding = new CustomBinding(bindingElements);
                using (Message message = Message.CreateMessage
                (binding.MessageVersion, "sendMessage","Message Body"))
                //创建消息
                {
                    //创建 ChannelFactory
                    IChannelFactory<IOutputChannel> factory = binding.
                    BuildChannelFactory<IOutputChannel>(new
                    BindingParameterCollection());
                    factory.Open();//打开 ChannelFactory
                    //这里创建 IOutputChannel
                    IOutputChannel outputChannel = factory.CreateChannel(new
                    EndpointAddress("http://localhost:9090/InputService"));
                    outputChannel.Open();//打开 IOutputChannel
                    outputChannel.Send(message); //发送消息
                    Console.WriteLine("已经成功发送消息！");
                    outputChannel.Close();//关闭 IOutputChannel
                    factory.Close();//关闭工厂
                }
            }
            catch (Exception ex) //捕捉异常
            {
                Console.WriteLine(ex.ToString());//输出异常便于调试
            }
            Finally
            {
                Console.Read();
            }
        }
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    }
}
```

而接收端的代码如示例代码 24-2 所示。

示例代码 24-2

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
namespace WCF.Second
{
    class Input
    {
        static void Main(string[] args)
        {
            try
            {
                //建立和发送端相同的通道栈
                BindingElement[] bindingElements = new BindingElement[3];
                bindingElements[0] = new
                TextMessageEncodingBindingElement();      //文本编码
                //OneWayBindingElement 可以使得传输通道支持数据报模式
                bindingElements[1] = new OneWayBindingElement();
                bindingElements[2] = new
                HttpTransportBindingElement();              //Http 传输
                CustomBinding binding = new CustomBinding(bindingElements);
                //建立 ChannelListner
                IChannelListener<IInputChannel> listener = binding.
                BuildChannelListener<IInputChannel>(new
                Uri("http://localhost:9090/InputService"), new
                BindingParameterCollection());
                listener.Open();//打开 ChannelListner
                IInputChannel inputChannel = listener.AcceptChannel();
                //创建 IInputChannel
                inputChannel.Open();//打开 IInputChannel
                Console.WriteLine("开始接收消息。。。");
                Message message = inputChannel.Receive();//接收并打印消息
                Console.WriteLine("接收到一条消息，action 为：{0}，body 为：{1}",
                message.Headers.Action, message.GetBody<String>());
                message.Close();                      //关闭消息
                inputChannel.Close();                 //关闭通道
                listener.Close();                     //关闭监听器
                Console.Read();
            }
            catch (Exception ex)                      //捕捉异常
            {
                Console.WriteLine(ex.ToString());//输出异常便于调试
            }
            Finally{
                Console.Read();
            }
        }
    }
}
```

编译上述代码，依次运行接收端和发送端，将得到图 24.3 所示的结果。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

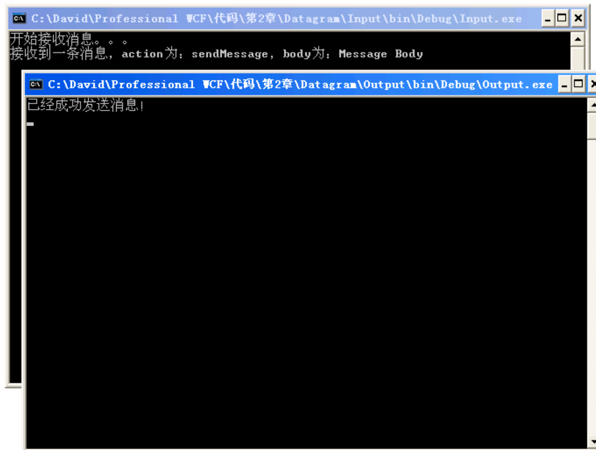


图 24.3　Datagram 执行结果

24.1.4　请求-响应模式

在请求-响应模式中，客户端发送一个消息并且接收一个返回消息来完成一次交互。请求-响应模式可以看作是一种特殊的双工模式。在该模式中，消息的发起端必然是客户端，并且从服务端返回的只有一条消息。客户端在发送出消息后会阻止当前线程并且等待服务端返回消息。这样的模式很适用于 HTTP 协议。图 24.4 展示了请求-响应模式的传输机制。

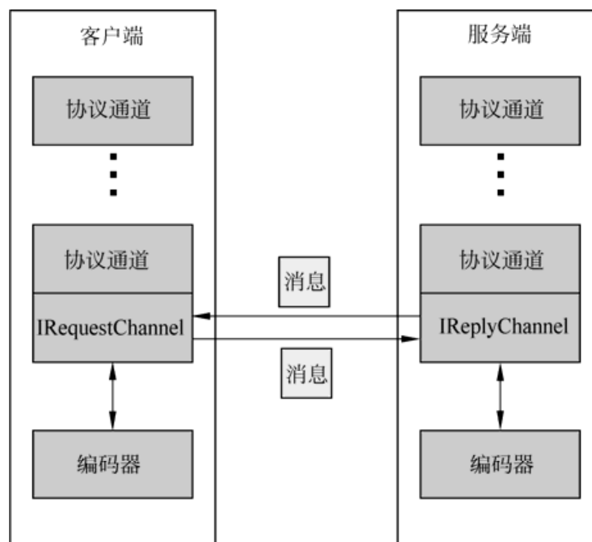


图 24.4　请求-响应模式传输机制

为了实现请求-响应模式，客户端通道需要实现 IRequestChannel 接口，而服务端则需要实现 IReplyChannel 接口。下面的代码展示了 IRequestChannel 和 IReplyChannel 的使用方法。发送端如示例代码 24-3 所示。

示例代码 24-3

```
using System;
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Xml;
using System.Runtime.Serialization;
namespace WCF.Second
{
    class Output
    {
        static void Main(string[] args) //入口方法
        {
            try
            {
                //建立自定义的通道栈
                BindingElement[] bindingElements = new BindingElement[2];
                bindingElements[0] = new
                TextMessageEncodingBindingElement();//文本编码
                bindingElements[1] = new
                HttpTransportBindingElement();//Http 传输
                CustomBinding binding = new CustomBinding(bindingElements);
                using (Message message = Message.CreateMessage
                (binding.MessageVersion, "sendMessage","Message Body"))
                {
                    //创建 ChannelFactory
                    IChannelFactory<IRequestChannel> factory =
                    binding.BuildChannelFactory<IRequestChannel>(new
                    BindingParameterCollection());
                    factory.Open();//打开 ChannelFactory
                    //这里创建 IRequestChannel
                    IRequestChannel requestChannel = factory.
                    CreateChannel(new EndpointAddress
                    ("http://localhost:9090/RequestReplyService"));
                    requestChannel.Open();        //打开通道
                    Message response = requestChannel.Request(message);
                    //发送消息
                    Console.WriteLine("已经成功发送消息! ");
                    //查看返回消息
                    Console.WriteLine("接收到一条返回消息，action 为: {0}，body
                    为: {1}", response.Headers.Action,
                    response.GetBody<String>());
                    requestChannel.Close();       //关闭通道
                    factory.Close();              //关闭工厂
                }
            }
            catch (Exception ex) {//捕捉异常
                Console.WriteLine(ex.ToString());//输出异常便于调试
            }
            finally {
                Console.Read();
            }
        }
    }
}
```

而相应的，接收端的代码如示例代码 24-4 所示。

示例代码 24-4

```
using System;
using System.ServiceModel;
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
using System.ServiceModel.Channels;
namespace WCF.Second
{
    class Input
    {
        static void Main(string[] args)
        {
            try
            {
                //建立和发送端相同的通道栈
                BindingElement[] bindingElements = new BindingElement[2];
                bindingElements[0] = new
                TextMessageEncodingBindingElement();//文本编码
                bindingElements[1] = new
                HttpTransportBindingElement();//Http 传输
                CustomBinding binding = new CustomBinding(bindingElements);
                //建立 ChannelListner
                IChannelListener<IReplyChannel> listener = binding.
                BuildChannelListener<IReplyChannel>(new Uri("http:
                //localhost:9090/RequestReplyService"), new
                BindingParameterCollection());
                listener.Open();      //打开监听
                //创建 IReplyChannel
                IReplyChannel replyChannel = listener.AcceptChannel();
                replyChannel.Open(); //打开通道
                Console.WriteLine("开始接收消息。。。");
                //接收并打印消息
                RequestContext requestContext = replyChannel.
                ReceiveRequest();
                Console.WriteLine("接收到一条消息，action 为：{0}，body 为：{1}",
                requestContext.RequestMessage.Headers.Action,
                requestContext.RequestMessage.GetBody<String>());
                //创建返回消息
                Message response = Message.CreateMessage
                (binding.MessageVersion, "response", "reponse body");
                //发送返回消息
                requestContext.Reply(response);
                //关闭
                requestContext.Close();
                replyChannel.Close(); //关闭通道
                listener.Close();     //关闭监听
                Console.Read();
            }
            catch (Exception ex) {//捕捉异常
                Console.WriteLine(ex.ToString());//输出异常便于调试
            }
            finally{
                Console.Read();
            }
        }
    }
}
```

编译上述代码，依次运行接收端和发送端，将得到图 24.5 所示的结果。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

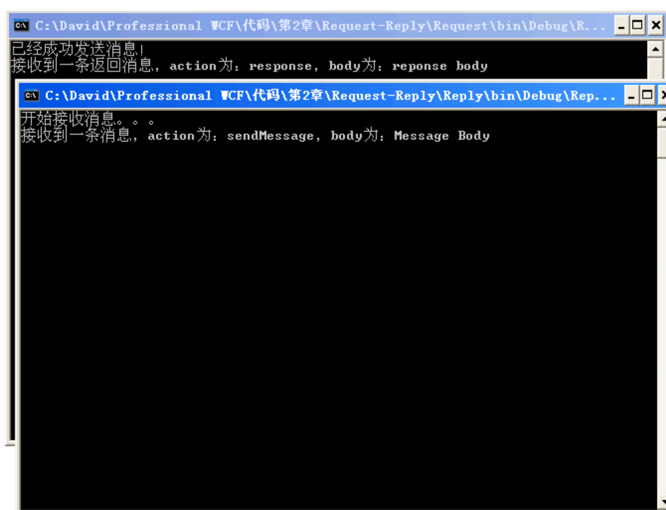


图 24.5　Request-Reply 执行结果

24.1.5　双工模式

在双工模式中，客户端和服务端都可以任意地向对方发送消息，而对方也可以以任意的次序来接收消息。在这种模式下，发送端和接收端的概念变得不再使用，取而代之的是通信的两个端点。图 24.6 展示了双工模式的传输机制。

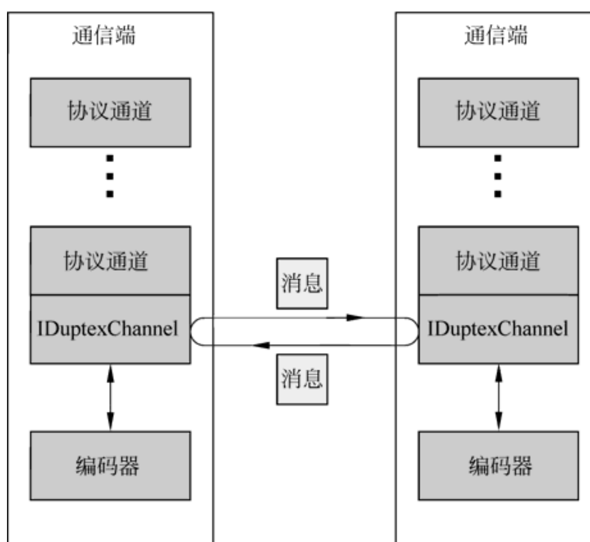


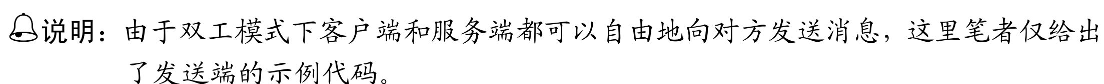
图 24.6　双工模式传输机制

为了实现双工模式，通信端需要实现 IDuplexChannel 接口。IDuplexChannel 接口实际同时实现了 IInputChannel 接口和 IOutputChannel 接口，并且使用 IInputChannel 和 IOutputChannel 来实现接收和发送消息。所以读者可以将其理解为一对 IInputChannel 接口和 IOutputChannel 接口的组合。其中，IInputChannel 负责接收消息，而 IOutputChannel 负责发送消息。示例代码 24-5 展示了 IDuplexChannel 的使用方式。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

示例代码 24-5

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
namespace WCF.Second
{
    class DuplexSender
    {
        static void Main(string[] args) //入口方法
        {
            try
            {
                NetTcpBinding binding = new NetTcpBinding();//创建绑定
                using (Message message = Message.CreateMessage
                (binding.MessageVersion, "sendMessage", "Message Body"))
                {
                    //创建 ChannelFactory
                    IChannelFactory<IDuplexChannel> factory = binding.
                    BuildChannelFactory<IDuplexChannel>(new
                    BindingParameterCollection());
                    factory.Open();                  //打开通道工厂
                    //这里创建 IRequestChannel
                    IDuplexChannel duplexChannel = factory.CreateChannel(new
                    EndpointAddress("net.tcp:
                    //localhost:9090/DuplexService/Point2"));
                    duplexChannel.Open();            //打开通道
                    duplexChannel.Send(message);     //发送消息
                    Console.WriteLine("已经成功发送消息！");
                    duplexChannel.Close();           //关闭通道
                    factory.Close();                 //关闭通道工厂
                }
            }
            catch (Exception ex) {//捕捉异常
                Console.WriteLine(ex.ToString());//输出异常便于调试
            }
            finally{
                Console.Read();
            }
        }
    }
}
```


说明：由于双工模式下客户端和服务端都可以自由地向对方发送消息，这里笔者仅给出了发送端的示例代码。

24.1.6 带会话的数据报模式、请求-响应模式和双工模式

前面读者已经介绍了 WCF 通道模型中 3 种基本的传输模式。WCF 提供了这 3 种基本传输模式的带会话模式，分别为带会话的数据报模式、带会话的请求-响应模式和带会话的双工模式。从通信层面来说，会话的概念类似于网络协议中的连接概念。带会话的通信模式类似于面向连接的网络协议，而不带会话的通信模型带相对于无连接的网络协议。一个典型的例子就是 TCP 协议和 UDP 协议。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

说明：在带会话的 3 个通信模式中使用的通道形状是：IInputSessionChannel、IOutputSessionChannel、IRequestSessionChannel、IReplySessionChannel 和 IDuplexSessionChannel。

在通道目标模型中，每个逻辑会话都表现为一个会话通道的实例。因此，由客户端创建并在服务端接收的每个新会话都与每一端的一个新会话通道相对应。图 24.7 展示了带会话的传输模式和不带会话的传输模式的区别。

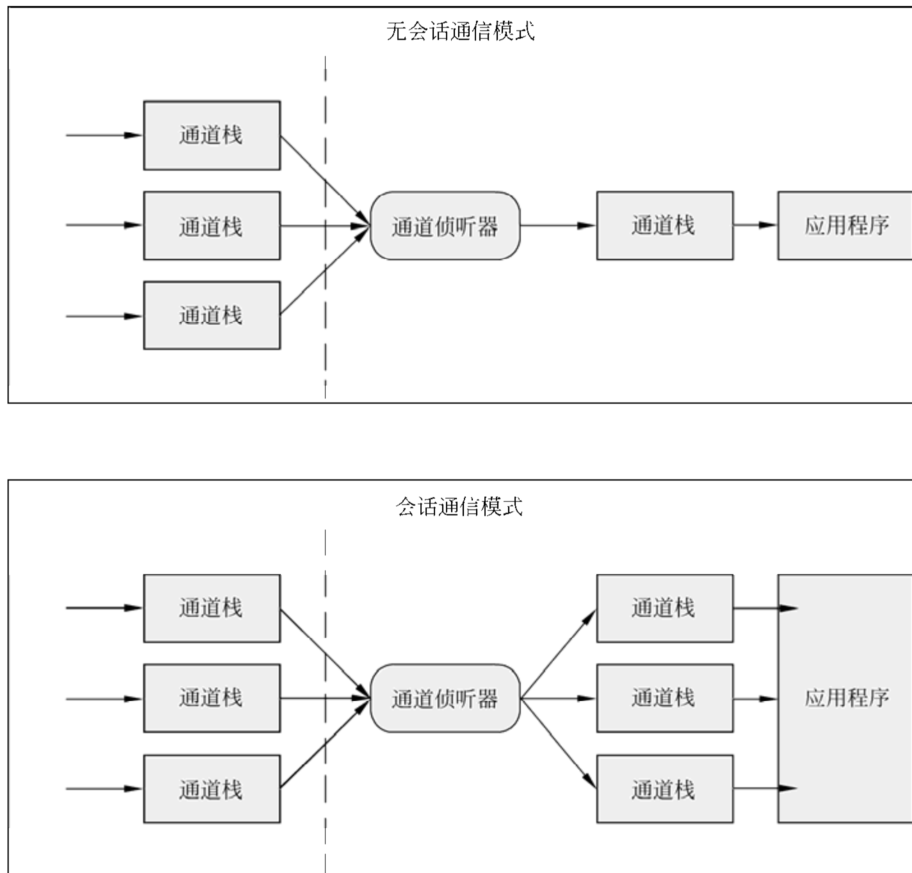


图 24.7　无会话/会话通信模式比较

24.1.7　通道形状的改变

在使用 CustomerBinding 时，读者可能已经发现，通信模式和通信协议是密切相关的。例如在使用 HTTP 协议进行消息传输时，受到协议的限制，数据报模式和双工模式并不能被使用。这个问题的解决方法是进行通道形状改变。通道形状的改变，指的是在传输通道上层添加特定的协议通道，来强制使用某种传输通道不支持的传输模式。在示例代码 24-1 和示例代码 24-2 中，读者已经使用了通道形状改变来演示数据报传输模式，关键代码如下所示。

```
BindingElement[] bindingElements = new BindingElement[3];
bindingElements[0] = new TextMessageEncodingBindingElement();
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
//OneWayBindingElement 可以使得传输通道支持数据报模式
bindingElements[1] = new OneWayBindingElement();
bindingElements[2] = new HttpTransportBindingElement();
```

这里添加了 OneWayBindingElement 协议通道来进行通道形状改变。在 WCF 中，一共有两种协议通道类型用以通道形状改变，分别为：OneWayBindingElement 和 CompositeDuplexBindingElement。前者把通道形状改变为数据报模式，而后者则把通道形状改变为双工模式。

24.1.8　通道形状和上层服务协议

如笔者在本节前文中演示的，通道栈会使用通道形状来实现不同的消息交互模式。举例来说，基于 UDP 的传输通道会实现 IInputChannel 和 IoutputChannel。这是因为其天生的数据报交互模式，而有些通道（比如基于 TCP 协议的通道）则会实现多种通道形状。WCF 会根据上层服务协议来自动选取需要的通道形状。表 24.1 列举了不同服务协议的设置下，WCF 会使用的通道形状。关于服务协议的概念，将在本书后续章节中介绍。

表 24.1　服务协议对应的通道形状

单　程	请求-应答	会　话	回　调	通 道 形 状
Any	Any	No	Yes	IDuplexChannel
Any	Any	No	Yes	IDuplexSessionChannel
Any	Any	Yes	Yes	IDuplexSessionChannel
Yes	Yes	No	No	IDuplexChannel
Yes	Yes	No	No	IRequestChannel
Yes	Yes	No	No	IDuplexSessionChannel
Yes	Yes	Yes	No	IDuplexSessionChannel
Yes	Yes	Yes	No	IRequestSessionChannel
Yes	No	No	No	IOutputChannel
Yes	No	No	No	IDuplexChannel
Yes	No	No	No	IDuplexSessionChannel
Yes	No	No	No	IRequestChannel
Yes	No	Yes	No	IOutputSessionChannel
Yes	No	Yes	No	IDuplexSessionChannel
Yes	No	Yes	No	IRequestSessionChannel
No	Yes	No	No	IRequestChannel
No	Yes	No	No	IDuplexChannel
No	Yes	No	No	IDuplexSessionChannel
No	Yes	Yes	No	IRequestSessionChannel
No	Yes	Yes	No	IDuplexSessionChannel

说明：不是所有通道都会实现所有通道形状的，有时候 WCF 会被迫使用其他的通道形状。举例来说，如果通道没有实现 IInputChannel 和 IOutputChannel，WCF 会尝试使用 IDuplexChannel 或者 IRequestChannel/IReplyChannel 来代替。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

24.1.9　通道管理器

在 WCF 中，实现了两类通道管理器，分别实现 IChannelListener<T>和 IChannel-Factory<T>接口。

IChannelListner<T>负责接收端的消息交互控制工作。这些通道管理器负责侦听消息、建立通道栈，并且提供通道栈顶层通道的引用。大多数情况下，程序员并不需要直接使用 IChannelListner<T>接口，而是直接使用 ServiceHost 类型，而在 ServiceHost 类型内部，仍然使用了 IChannelListner<T>来实现通道的管理。在本节前文中读者已经大致了解了 IChannelListner<T>的使用，其关键代码如下所示。

```
//建立 ChannelListner
ChannelListener<IReplyChannel> listener = binding.
BuildChannelListener<IReplyChannel>(new Uri("http://localhost:
9090/RequestReplyService"), new BindingParameterCollection());
listener.Open();//打开 ChannelListner
//创建 IReplyChannel
IReplyChannel replyChannel = listener.AcceptChannel();
replyChannel.Open();//打开 IReplyChannel
```

与 IChannelListner<T>对应的是 IChannelFactory<T>接口，这个管理器负责在发送端控制消息的发送。和 IChannelListner<T>一样，IChannelFactory<T>负责创建并管理通道栈。大多数程序员会使用 ClientBase<T>类型而代替直接使用 IChannelFactory<T>接口。使用 IChannelListner<T>的关键代码如下所示。

```
//创建 ChannelFactory
ChannelFactory<IRequestChannel> factory = binding.
BuildChannelFactory<IRequestChannel>(new BindingParameterCollection());
factory.Open();//打开 ChannelFactory
//这里创建 IRequestChannel
IRequestChannel requestChannel = factory.CreateChannel(new EndpointAddress
("http://localhost:9090/RequestReplyService"));
requestChannel.Open();//打开 IRequestChannel
```

两个通道管理器有一个明显的区别，那就是 IChannelFactory<T>负责关闭通道栈，而 IChannelListner<T>却不需要。也就是说，IChannelListner<T>可以独立于它的通道栈而关闭。

24.1.10　ICommunicationObject 接口和状态改变

一般来说，和通信有关的机制都会附带状态机，其状态转换与分配网络资源、生成或接收连接、关闭连接以及终止通信有关。在 WCF 中，ICommunicationObject 接口定义状态机制的统一模型。基本所有的通信组件，包括通道和通道管理器都实现了 Icommunication-Object 接口。ICommunicationObject 接口的定义如下所示：

```
public interface ICommunicationObject
{
    //事件
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    event EventHandler Closed;          //已关闭
    event EventHandler Closing;         //关闭中
    event EventHandler Faulted;         //错误
    event EventHandler Opened;          //已打开
    event EventHandler Opening;         //打开中
    //方法
    //这里皆为改变状态机状态的方法
    void Abort();
    IAsyncResult BeginClose(AsyncCallback callback, object state);
    IAsyncResult BeginClose(TimeSpan timeout, AsyncCallback callback,
    object state);
    IAsyncResult BeginOpen(AsyncCallback callback, object state);
    IAsyncResult BeginOpen(TimeSpan timeout, AsyncCallback callback, object
    state);
    void Close();
    void Close(TimeSpan timeout);
    void EndClose(IAsyncResult result);
    void EndOpen(IAsyncResult result);
    void Open();
    void Open(TimeSpan timeout);
    //属性
    CommunicationState State { get; }   //获取状态
}
```

从上面代码可以看出，ICommunicationObject 的目的集中在状态机的管理上，包括改变状态的方法、状态改变触发的时间以及当前状态的获取。在该状态机中，一共有 6 种可用状态，这些状态定义在 CommunicationState 枚举类型中。其定义如下所示。

```
public enum CommunicationState
{
    Created,     //已创建
    Opening,     //打开中
    Opened,      //已打开
    Closing,     //关闭中
    Closed,      //已关闭
    Faulted      //错误
}
```

在初始状态下，所有的状态机都维持在 Created 的状态下，随着系统的运行，ICommunicationObject 对象中的方法将被调用来改变状态机的状态。图 24.8 展示了状态机的状态改变。

注意：状态机的状态改变都是前向的。以图 24.8 而言，状态的改变都是从上至下的，这就意味着 WCF 的状态机不可逆。一旦某个状态机到达了 Closed 或者 Faulted 的状态，就无法再次回到 Created 状态，这意味着程序需要重新创建新的状态机。

ICommunicationObject 提供了 5 个状态改变事件，分别为 Opening、Opened、Faulted、Closing 和 Closed。灵活应用这些事件可以在状态改变时执行特定的状态。ClientBase<T>就是一个实现了 ICommunicationObject 接口的类型。以该类型为例，下面演示状态改变事件的使用。这里使用第一章给出的 HelloWorld 系统，同时修改客户端代码，如示例代码 24-6 所示。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

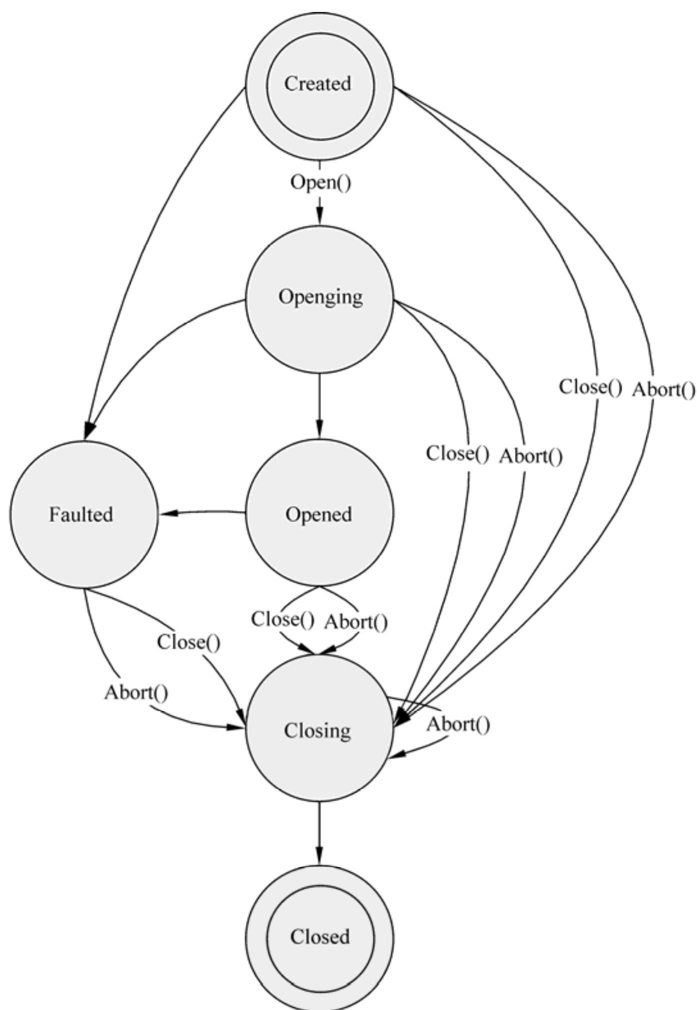


图 24.8 状态机的状态变化图

示例代码 24-6

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
namespace HelloWorldClient
{
    class Client
    {
        static void Main(string[] args) // 入口方法
        {
            using (HelloWorldProxy proxy = new HelloWorldProxy())
            {
                //获得状态机对象并添加处理事件
                ICommunicationObject communicationObject =
                (ICommunicationObject)proxy;
                communicationObject.Opening += new EventHandler(Opened);
                communicationObject.Closed += new EventHandler(Closed);
                //利用代理调用服务
                Console.WriteLine(proxy.HelloWorld("WCF"));
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
            }
            Console.Read();
        }
        static void Opened(object sender, EventArgs args)   //Opened 事件
        {
            Console.WriteLine("状态机开启！");
        }
        static void Closed(object sender, EventArgs args)   //Closed 事件
        {
            Console.WriteLine("状态机关闭！");
        }
    }
    [ServiceContract]
    interface Iservice                        //硬编码定义服务契约
    {

        [OperationContract]
        String HelloWorld(String name); //服务操作
    }
    class HelloWorldProxy : ClientBase<IService>, IService// 客户端代理类型
    {
        //硬编码定义绑定
        public static readonly Binding HelloWorldBinding = new
        NetNamedPipeBinding();
        //硬编码定义地址
        public static readonly EndpointAddress HelloWorldAddress=new
        EndpointAddress(new Uri("net.pipe://localhost/HelloWorld"));
        public HelloWorldProxy() : base(HelloWorldBinding,
        HelloWorldAddress) { // 构造方法}
        public String HelloWorld(String name)
        {
            return Channel.HelloWorld(name); //使用 Channel 属性对服务进行调用
        }
    }
}
```

编译并运行改变后的 HelloWorld 系统，将得到如图 24.9 所示的结果。

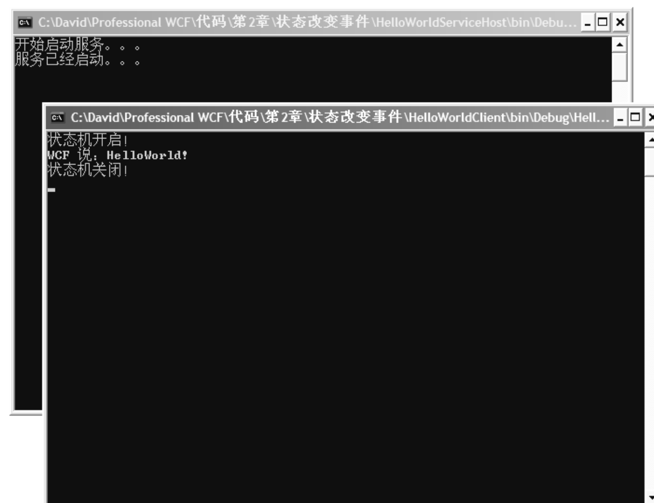


图 24.9　状态改变事件执行结果

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

24.2 标准绑定介绍

在了解了 WCF 的通道模型之后，读者能够更容易地理解绑定。绑定就是一个已经定制好的通道栈，本节将介绍 WCF 提供的常用绑定类型，以及它们的特性。在本书的后续章节中，会覆盖有关绑定扩展的内容。

24.2.1 绑定的基本概念

正如前一节中笔者介绍的，WCF 的通道模型具有极大的灵活性，程序员可以在协议通道、编码器、传输通道等各个方面进行设置。相应地，每次都需要设置一个完整的通道栈是一个较为繁琐的事情。从传输协议上来看，有 HTTP、TCP、UDP、P2P、IPC 和 MSMQ 等多种可选方案。从编码器上来看，有二进制编码、MTOM 等编码方式。再加上消息的安全策略、会话设置等，组合起来可达数千种组合方式。更为麻烦的是，并不是每一种组合方式都是可以运行的，这样的错误往往要等到实际运行时候才能被发现。为了使程序员摆脱这些复杂的设置，WCF 提出了绑定的概念。

绑定是一个定制好的通道栈，包含了协议通道、传输通道和编码器。而从其功能上来看，一个绑定集成了通信模式、可靠性、安全性、事务传播和互操作性等设置。一般来说，程序员很少自定义一个全新的绑定。即使不使用 WCF 提供的标准绑定，也可以在某一个标准绑定基础上进行修改，来达到自定义的目的。

24.2.2 标准绑定

在.NET Framework 3.5 中 WCF 一共提供了 12 种标准绑定，这些绑定基本能够覆盖用户所有的传输需求。表 24.2 列出了 12 个标准绑定的名字和简要说明。

表 24.2 12 种标准绑定简要说明

绑 定 名 称	简 要 介 绍	所需.NET Framework 版本
basicHttpBinding	基于 WS-I Basic Profile 1.1 的 Web 服务	3.0
wsHttpBinding	针对改进的 Web 服务的绑定，包括 WS-Security，WS-Transaction 等元素	3.0
wsDualHttpBinding	提供双工通信的 HTTP 绑定	3.0
webHttpBinding	支持 REST/POX 服务的绑定，使用 XML/JSON 序列化	3.0
netTcpBinding	使用 TCP 传输协议在跨主机的局域网内使用，支持可靠性、事务、安全等特性，并且该绑定被特别优化来支持 WCF 系统。但是，使用该绑定需要确保通信双方都基于 WCF 构建，这里并不符合 SOA 的原则	3.0
netNamedPipeBinding	支持和 netTcpBinding 大致相同的特性，但由于使用命名管道进行通信，所以通信不能跨越主机	3.0

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

续表

绑 定 名 称	简 要 介 绍	所需.NET Framework 版本
netMsmqBinding	使用微软消息队列（MSMQ）协议来进行异步脱机的消息交互。关于该绑定的交互方式，在本书的后续章节中有详细的介绍	3.0
netPeerTcpBinding	使用 P2P 协议在网格中进行消息交互	3.0
msmqIntegrationBinding	该绑定可以用来在 WCF 消息和 MSMQ 消息中进行转换	3.0
wsFederationHttpBinding	该绑定支持使用了联合安全机制的 Web 服务	3.0
ws2007HttpBinding	该绑定继承自 wsHttpBinding，其主要设计目的是为了支持 2007 年新制定的 WS 标准	3.5
ws2007FederationHttpBinding	该绑定继承自 wsFederationHttpBinding，和 ws2007HttpBinding 一样。其设计目的是为了支持 2007 年新制定的 WS 标准	3.5

24.2.3　设置绑定的方式

在 WCF 程序中，有两种方式来设置绑定，在代码中设置绑定和在配置文件中设置绑定。一般来说，在配置文件中设置更为常用，这是因为绑定的设置通常需要根据部署环境的改变而改变。在配置文件中更改设置非常方便，并且不需要重新编译整个系统。示例代码 24-7 展示了如何在配置文件中设置绑定。

示例代码 24-7

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />  //允许 debug
  </system.web>
  <system.serviceModel>
    <services>
      <service name="HelloWorldService.Service">
        //终节点定义
        <endpoint address ="HelloWorld"                    //地址
                  binding="netNamedPipeBinding"            //绑定
                  contract="HelloWorldService.IService">  //契约
        </endpoint>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

说明：示例代码 24-7 省略了服务中其他元素的设置，而只列举了终节点的设置。

在程序中设置绑定可以更加确保设置的安全性，但这样的做法显然缺乏灵活性。示例代码 24-8 展示了如何在程序中设置绑定。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

示例代码 24-8

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
namespace HelloWorldClient
{
    class Client
    {
        static void Main(string[] args) // 入口方法
        {
            //构造绑定
            NetNamedPipeBinding binding = new NetNamedPipeBinding();
            //构造终节点
            EndpointAddress address=new EndpointAddress(new Uri
            ("net.pipe://localhost/HelloWorld");
            using (HelloWorldProxy proxy = new
            HelloWorldProxy(binding,address))
            {
                Console.WriteLine(proxy.HelloWorld("WCF"));//利用代理调用服务
                Console.Read();
            }
        }
    }
}
```

24.2.4　如何选择绑定

如何选择绑定是编写 WCF 程序一个比较重要的话题。绑定的选择包含了很多因素，包括消息传输的可靠性，传输模式是否跨进程、主机、网络，传输模式的支持、安全性、性能等多个方面。而从本质上来看，绑定具有的这些特性源于其使用的网络协议和编码器。表 24.3 列出了基本绑定使用的网络协议、编码器以及其可交互性。

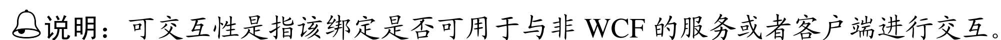

说明：可交互性是指该绑定是否可用于与非 WCF 的服务或者客户端进行交互。

表 24.3　标准绑定的特性

绑 定 名 称	网 络 协 议	编　码　器	可 交 互 性
basicHttpBinding	HTTP/HTTPS	Text，MTOM	可交互
wsHttpBinding	HTTP/HTTPS	Text，MTOM	可交互
wsDualHttpBinding	HTTP	Text，MTOM	可交互
webHttpBinding	HTTP/HTTPS	XML，JSON	可交互
netTcpBinding	TCP	二进制编码	不可交互
netNamedPipeBinding	IPC	二进制编码	不可交互
netMsmqBinding	MSMQ	二进制编码	不可交互
netPeerTcpBinding	P2P	二进制编码	不可交互
msmqIntergrationBinding	MSMQ	二进制编码	可交互
wsFederationHttpBinding	HTTP/HTTPS	Text，MTOM	可交互
ws2007HttpBinding	HTTP/HTTPS	Text，MTOM	可交互
ws2007FederationHttpBinding	HTTP/HTTPS	Text，MTOM	可交互

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

注意：细心的读者可能已经发现，绑定的命名和可交互性有一定的关联。所有以 net 作为前缀的标准绑定都是不可交互的，而对应的不以 net 为前缀的标准绑定都是可交互的。这是 WCF 的编码规范，读者在自定义绑定时，应该仍然遵守这样的命名规范，以保持代码的可读性

选择绑定是一个复杂的过程，没有万能的挑选公式可以套用。但是通常地，可以从是否需要交互特性、是否跨主机、是否需要脱机交互等几个方面着手。图 24.10 给出了一个粗略的选择方案以供读者参考。

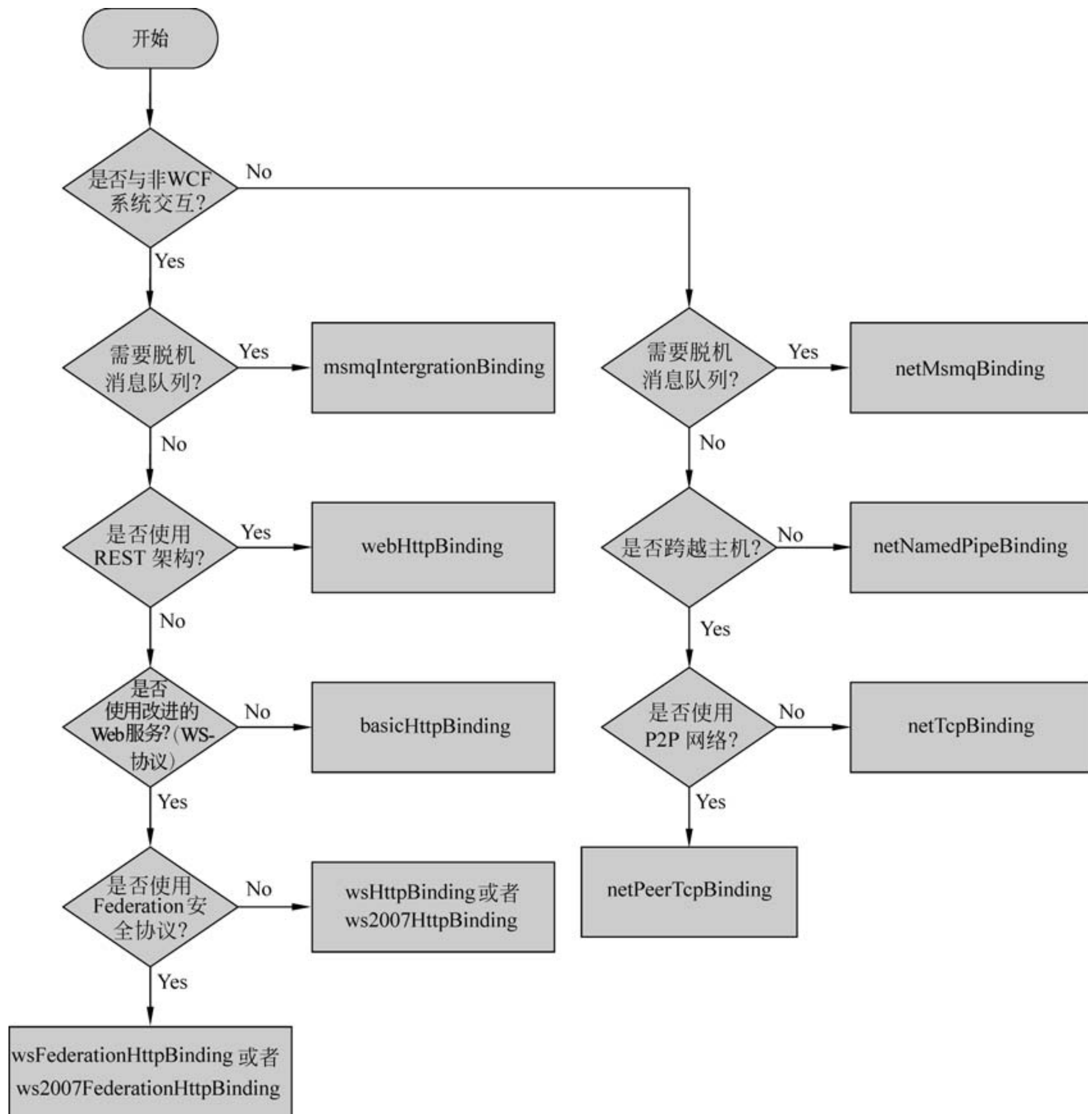


图 24.10　选择绑定的参考流程

除了功能之外，性能的要求也是挑选绑定的重要参考因素。通常情况下，在满足系统功能的前提下，程序员希望挑选性能最佳的绑定。一般来说，使用 IPC 的绑定性能要优于使用 TCP 的绑定，而使用 TCP 的绑定又优于使用 HTTP 的绑定。使用针对 WCF 优化的绑

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

定要优于能与非 WCF 系统交互的绑定。

24.3　本机 WCF-WCF 交互的绑定和地址

本机的交互在很多场景中都会有所使用，例如测试环境、超级主机上部署等。本机交互的特点是消息无须跨越防火墙和主机，并且也保证了交互双方的操作系统平台是一致的。本节将介绍本机 WCF-WCF 交互时绑定和地址的选择。

24.3.1　场景概述

很多时候公司内部多个系统都会被部署在同一台超级主机上，这样的做法既节省成本，又便于统一管理和备份。在这种情况下，系统与系统之间的访问不需要跨越主机。更进一步地，如果整个 SOA 系统都使用 WCF 架构，那就可以选择不具备兼容性的绑定，因为这些绑定往往具有更好的性能。

考虑在同一台主机上，WCF-WCF 的交互可以分为下列几种情况。

- 跨越主机进程交互。
- 跨越应用程序域但在同一进程内交互。
- 在同一应用程序域内交互。

以目前 WCF 的实现来说，应对这 3 种交互情况所选择的消息传输协议都是 IPC。截至到目前版本，WCF 并没有提供进程内的消息传输协议。虽然这样的协议可以进一步提高消息交互的性能，但是考虑到大多数情况下该协议并不适用，所以 WCF 开发小组最终放弃了开发进程内消息交互协议的想法。

24.3.2　IPC 基本概念

严格来说，IPC 是一种通信方法，而非通信协议。IPC 的全称是进程间通信（Inter-Process-Communication）。该概念源于 UNIX 系统，主要致力于解决进程间通信的问题。在 WIN32 系统中，IPC 的实现主要有如下几种方式。

1．使用剪贴板

剪贴板可以在进程间共享，并且这种机制容易理解，使用也方便。但是使用剪贴板的缺点也非常明显，剪贴板的使用非常频繁，所以这样的共享容易产生资源争用。同时，剪贴板的性能并不高。

2．使用匿名管道和命名管道

这是当前 Windows 平台中最为常用的 IPC 方式，创建一个负责在通信双方之间建立媒介的管道，通信双方就可以通过管道读写信息，达到交互的目的

管道是进程用来通信的共享内存区域。一个进程往管道中写入信息，而其他的进程可以从管道中读出信息。如其名，管道是进程间数据交流的通道。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

管道的类型有两种：匿名管道和命名管道。匿名管道是不命名的，它最初用于在本地系统中父进程与它启动的子进程之间的通信。命名管道更高级，它由一个名字来标识，以使客户端和服务端应用程序可以通过它进行彼此通信。而且，Win32 命名管道甚至可以在不同系统的进程间使用，这使它成为许多客户/服务器应用程序的理想之选。

就像水管连接两个地方并输送水一样，软件的管道连接两个进程并输送数据。一个一个管道一旦被建立，它就可以像文件一样被访问，并且可以使用许多与文件操作同样的函数。可以使用 CreateFile 函数获取一个已打开的管道的句柄，或者由另一个进程提供一个句柄。使用 WriteFile 函数向管道写入数据，之后这些数据可以被另外的进程用 ReadFile 函数读取。管道是系统对象，因此管道的句柄在不需要时必须使用 CloseHandle 函数关闭。

匿名管道只能单向传送数据，而命名管道可以双向传送。管道可以以比特流形式传送任意数量的数据。命名管道还可以将数据集合到称为消息的数据块中。命名管道甚至具有通过网络连接多进程的能力。但遗憾的是，Windows9X 不支持创建命名管道，它只能在 WindowsNT 内核的操作系统上创建。

当讨论管道时，通常涉及到两个进程：客户进程和服务进程。服务进程负责创建管道。客户进程连接到管道。服务进程可以创建一个管道的多个实例，以此支持多个客户进程。

3．使用邮件槽

广播式通信，在 WIN32 系统中提供的新方法，可以在不同主机间交换数据，实现了跨网络，单在 WIN9X 下只支持邮件槽客户。服务端必须运行在 Windows NT/2000/XP。

4．使用TCP/IP

它具备消息管道所有的功能，但遵守一套通信标准使的不同操作系统之上的应用程序之间可以互相通信。这种方式用于网络方面比较好，用于本地进程间交互性能较差。

5．使用COM/DCOM

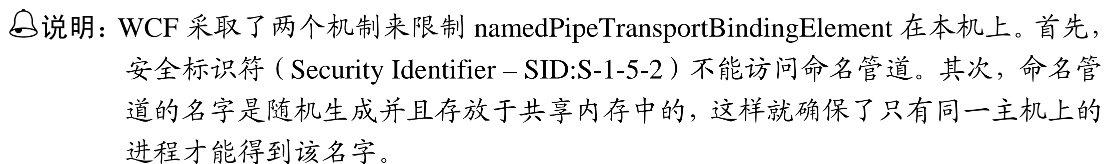
通过 COM 系统的代理存根方式进行进程间数据交换，但只能够表现在对接口函数的调用时传送数据，通过 DCOM 可以在不同主机间传送数据。

6．使用内存映射文件

Windows 操作在系统核心内存区域开辟一块内存，然后每个进程把这块内存映射到自己可以访问的虚内存地址中。对每个进程来说，似乎在操作各自的内存区域，而实际上所有的操作被映射到 Windows 核心的共享的内存区域。

24.3.3 使用 netNamedPipeBinding

在 WCF 提供的 12 种标准绑定中，适用于本机 WCF-WCF 交互性能最佳的绑定是 netNamedPipeBinding。顾名思义，netNamedPipeBinding 使用的是命名管道的方式实现 IPC 的。也正是由于这一点，netNamedPipeBinding 是 WCF 提供的 12 种标准绑定中性能最好的。值得读者注意的是，虽然 Windows 的命名管道机制允许跨主机的交互，但是 WCF 却对 netNamedPipeBinding 做了特殊的限制，使得通信的双方只能部署在同一台主机上。



说明：WCF 采取了两个机制来限制 namedPipeTransportBindingElement 在本机上。首先，安全标识符（Security Identifier – SID:S-1-5-2）不能访问命名管道。其次，命名管道的名字是随机生成并且存放于共享内存中的，这样就确保了只有同一主机上的进程才能得到该名字。

netNamedPipeBinding 提供了一系列可配置的属性，表 24.4 列出了这些属性的名称和简要说明。

表 24.4 netNamedPipeBinding的属性

属 性 名	描 述	默 认 值
closeTimeout	等待连接的超时时间	00:01:00
hostNameComparisonMode	指定将传入的消息调度到服务终节点时应该如何在 URI 比较中使用主机名	StrongWildCard
maxBufferPoolSize	传输使用缓冲池的最大值	524888
maxConnections	传输连接的最大数。注意这里发送连接和接收连接的数目是分开计算的	10
maxReceivedMessageSieze	接收消息大小的上限	65536
name	绑定名	
openTimeout	等待一个已经打开的连接完成操作的超时时间	00:01:00
readerQuotas	定义一个可被处理的消息的复杂度	N/A
receiveTimeout	等待一个接收操作完成的超时时间	00:01:00
security	设定绑定的安全设置	N/A
sendTimeout	等待一个发送操作完成的超时时间	00:01:00
transactionFlow	设置是否允许事务流	false
transactionProtocol	支持的事务类型，OleTransactions 或者 WSAtomic-Transactions	OleTransactions

24.3.4 netNamedPipeBinding 的地址和配置

如果使用 netNamedPipeBinding 绑定，那么终节点的地址将以如下的形式出现。

```
net.pipe://[Hostname]:[Port]/[ServiceAddress]
```

地址中的 net.pipe 对应命名管道协议，不止 netNamedPipeBinding 绑定，任何使用了命名管道传输通道的绑定都使用 net.pipe 作为地址的协议部分。[Hostname]指的是主机，这里可以填写主机名、主机 IP。特别地，由于 netNamedPipeBinding 被限定在同一主机上的系统进行交互，所以[Hostname]也可以填写 localhost。[Port]指的是端口，在决定地址时不需确保使用的端口不和主机上的其他系统冲突。而[ServerAddress]则是一个可选项，笔者建议为了使得系统更加可读，[ServiceAddress]的命名应该和服务内容有关，令人一目了然地知道寄宿在该地址上的服务的作用。

这里借助在第一章给出的 HelloWorld 服务，来演示 netNamedPipeBinding 在配置文件中的配置方法。示例代码 24-9 展示了在服务端配置 netNamedPipeBinding。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

示例代码 24-9

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" /><!—允许 debug-->
  </system.web>
  <system.serviceModel>
    <services>
      <service name="HelloWorldService.Service">
        <host>
          <baseAddresses>
            <!--基地址-->
            <add baseAddress="net.pipe://localhost /"></add>
          </baseAddresses>
        </host>
        <!--使用 netNamedPipeBinding-->
        <endpoint address ="HelloWorldService"<!--地址-->
                  binding="netNamedPipeBinding"<!--绑定-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
      </service>
    </services>
    <bindings>
      <!--设置 netNamedPipeBinding 的属性-->
      <netNamedPipeBinding>
      <!--绑定属性设置-->
        <binding name="myBinding"<!--设置 name 属性-->
             hostNameComparisonMode="StrongWildcard"<!--设置
             hostNameComparisonMode 属性-->
             maxBufferSize="65536" <!--设置 maxBufferSize 属性-->
             maxConnections="10"<!--设置 maxConnections 属性-->
             maxReceivedMessageSize="65536"<!--设置 maxReceivedMessageSize
             属性-->
             receiveTimeout="00:01:00"<!--设置 receiveTimeout 属性-->
             transactionFlow="false"><!--设置 transactionFlow 属性-->
          <security mode="Transport">
          </security>
        </binding>
      </netNamedPipeBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

需要访问服务端的客户端（或者其他服务），也可以使用配置文件来配置 netNamedPipeBinding 的使用。示例代码 24-10 给出了客户端的配置示例。

示例代码 24-10

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <client>
        <!--使用 netNamedPipeBinding-->
        <endpoint address =" net.pipe://localhost /HelloWorldService" <!—
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
       地址-->
               binding="netNamedPipeBinding"<!--绑定-->
               contract="HelloWorldService.IService"><!--契约-->
       </endpoint>
    </client>
    <bindings>
      <!--设置 netNamedPipeBinding 的属性-->
      <netNamedPipeBinding>
        <!--绑定属性设置-->
        <binding name="myBinding"<!--设置 myBinding 属性-->
             hostNameComparisonMode="StrongWildcard"<!--设置
             hostNameComparisonMode 属性-->
             maxBufferSize="65536"<!--设置 maxBufferSize 属性-->
             maxConnections="10"<!--设置 maxConnections 属性-->
             maxReceivedMessageSize="65536"<!--设置 maxReceivedMessageSize
             属性-->
             receiveTimeout="00:01:00"<!--设置 receiveTimeout 属性-->
             transactionFlow="false"><!--设置 transactionFlow 属性-->
          <security mode="Transport">
          </security>
        </binding>
      </netNamedPipeBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

24.3.5　netNamedPipeBinding 特点总结

在本机 WCF-WCF 交互的情况下， netNamedPipeBinding 是最佳的标准绑定选择。其特点如下：

- 支持单程传输模式；
- 支持请求-响应传输模式；
- 支持双工传输模式；
- 性能在标准绑定中最佳；
- 支持 WS-事务协议；
- 支持传输层消息安全；
- 只限于 WCF 系统；
- 不支持跨主机交互。

24.4　跨主机 WCF-WCF 交互的绑定和地址

和本机交互不同，跨主机交互是指交互的双方在不同的主机之上。而在本节中，跨主机的交互特指在同一局域网内不同主机间的交互，这意味着交互双方不需要跨越任何防火墙。本节将介绍这种情况下绑定和地址的选择。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

24.4.1 场景概述

对于一些中、大型的 SOA 系统，最常见的情况仍然是多个服务部署在局域网内不同的主机之上，这时候限定本机交互的 IPC 就不再适用了。对于局域网内跨主机的 WCF-WCF 交互，采用 TCP 协议进行通信是较为合适的选择。首先 TCP 是面向连接的通信协议，其次其处于传输层，相对于应用层的协议来说效率更高。

24.4.2 TCP 协议概述

在网络通信中，分层的方法一般有两种，OSI 的 7 层协议和最常用的 5 层协议。TCP 协议并不符合 OSI 的 7 层协议，它属于 5 层协议的传输层。在实际使用中，以 TCP 作为传输层协议，并使用 IP 作为网络层协议是最普遍的做法。因此，TCP/IP 协议也成为了该模型的代名词。其结构如图 24.11 所示。

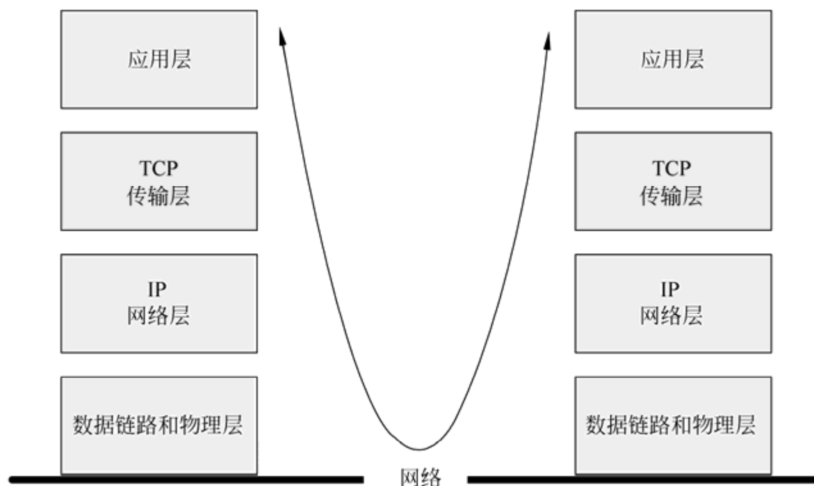


图 24.11　TCP/IP 网络模型

TCP 协议被称作一种端对端协议。这是因为它为两台计算机之间的连接起了重要作用：当一台计算机需要与另一台远程计算机连接时，TCP 协议会让它们建立一个连接、发送和接收资料以及终止连接。传输控制协议 TCP 协议利用重发技术和拥塞控制机制，向应用程序提供可靠的通信连接，使它能够自动适应网上的各种变化。

即使在 Internet 暂时出现堵塞的情况下，TCP 也能够保证通信的可靠。众所周知，Internet 是一个庞大的国际性网络。网络上的拥挤和空闲时间总是交替不定的，加上传送的距离也远近不同，所以传输资料所用时间也会变化不定。TCP 协议具有自动调整“超时值”的功能，能很好地适应 Internet 上各种各样的变化，确保传输数值的正确。因此，可以了解到：IP 协议只保证计算机能发送和接收分组资料，而 TCP 协议则可提供一个可靠的、可流控的、全双工的信息流传输服务。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

24.4.3　Net.Tcp 端口共享

WCF 提供了一种新的用于高性能通信的、基于 TCP 的网络协议，并且引入了一个新的系统组件 Net.TCP Port Sharing Service。该组件使得 net.tcp 端口可以在多个用户进程之间共享。

TCP/IP 通过为每一个应用程序协议分配一个唯一的 16 位端口号，从而使用端口号来区分应用程序。例如，HTTP 通信现在已经统一为使用 TCP 端口 80，SMTP 使用 TCP 端口 25，FTP 使用 TCP 端口 20 和 21。其他使用 TCP 作为传输协议的应用程序可以按习惯或遵循正式标准选择其他可用的端口号。

可使用端口号区分存在安全问题的应用程序。除少数几个已知的入口点之外，防火墙通常会配置为阻塞 TCP 通信。因此，部署使用非标准端口的应用程序经常会因为存在公司防火墙和个人防火墙而变得复杂或者甚至无法实现。通过已得到允许的标准已知端口进行通信的应用程序可减少外部攻击面。许多网络应用程序使用 HTTP 协议，这是因为大多数防火墙在默认情况下会配置为允许 TCP 端口 80 上的通信。

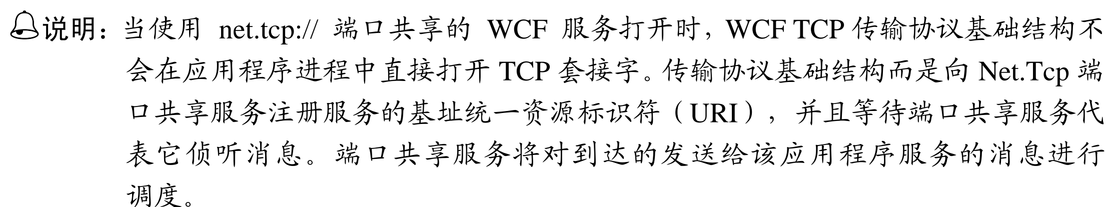
在 HTTP.SYS 模型中，许多不同的 HTTP 应用程序的通信中将多路复用到单个 TCP 端口。此模型已经成为 Windows 平台上的标准。这为防火墙管理员提供了一个公共控制点，同时可以让应用程序开发人员尽可能降低生成可利用网络的新应用程序的部署成本。

在多个 HTTP 应用程序之间共享端口的能力早已成为 Internet 信息服务（IIS）的一个功能。但是，只有在引入 IIS 6.0 附带的 HTTP.SYS（核心模式 HTTP 协议侦听器）之后，此基础结构才完全得到广泛使用。实际上，HTTP.SYS 允许任意用户进程共享专用于 HTTP 通信的 TCP 端口。此功能可以让许多 HTTP 应用程序在同一个物理计算机共存于不同的独立进程中，同时共享通过 TCP 端口 80 发送和接收通信所需要的网络基础结构。Net.Tcp 端口共享服务支持为 net.tcp 应用程序共享相同类型的端口。

WCF 中的端口共享结构有 3 个主要组件。

- 辅助进程：任何使用共享端口通过 net.tcp://通信的进程。
- WCF TCP 传输协议：实现 net.tcp:// 协议。
- Net.Tcp 端口共享服务允许许多辅助进程共享同一 TCP 端口。

Net.Tcp 端口共享服务是用户模式的 Windows 服务，可代表通过其连接的辅助进程接收 net.tcp://连接。当套接字连接到达时，端口共享服务将检查传入消息流以获取其目标地址。基于此地址，端口共享服务可以将数据流路由到最终处理它的应用程序。



说明：当使用 net.tcp:// 端口共享的 WCF 服务打开时，WCF TCP 传输协议基础结构不会在应用程序进程中直接打开 TCP 套接字。传输协议基础结构而是向 Net.Tcp 端口共享服务注册服务的基址统一资源标识符（URI），并且等待端口共享服务代表它侦听消息。端口共享服务将对到达的发送给该应用程序服务的消息进行调度。

24.4.4　使用 netTcpBinding

在 WCF 提供的绑定中，直接使用 TCP 协议进行消息交互的只有 netTcpBinding，并且

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

使用二进制编码来提高性能。在大多数局域网内 WCF-WCF 交互情况下，netTcpBinding 都是最为适用的标准绑定。这里笔者之所以强调局域网内，是因为跨网络的传输往往需要穿越防火墙，Tcp 协议交互无法直接穿越防火墙，这时候就需要使用 HTTP 协议或者 SOAP 协议。由于 TCP 协议的支持，netTcpBinding 支持传输安全。

和其他绑定一样，netTcpBinding 提供了可配置的属性，表 24.5 列出了这些属性的名称和简要说明。

表 24.5　netTcpBinding的属性

属 性 名	描 述	默 认 值
closeTimeout	等待连接的超时时间	00:01:00
hostNameComparisonMode	指定将传入的消息调度到服务终节点时应该如何在 URI 比较中使用主机名	StrongWildCard
listenBacklog	可挂起的最大排队连接请求数	10
maxBufferPoolSize	内存中用于对传入消息进行缓冲的最大字节数	524888
maxBufferSize	内存中用于对传入消息进行缓冲的最大字节数	65536
maxConnections	该值控制客户端上可存入池中以备后续重复使用的最大连接数，以及服务器上可挂起调度的最大连接数	10
name	绑定的名字	
openTimeout	在传输引发异常之前可用于打开连接的时间间隔	00:01:00
portSharingEnabled	该值指示是否为采用此绑定配置的连接启用 TCP 端口共享	false
readerQuotas	可由使用此绑定配置的终节点处理的 SOAP 消息的复杂性约束	N/A
receiveTimeout	在传输引发异常之前可用于完成读取操作的时间间隔	00:01:00
reliableSession	指示是否在通道终节点之间建立可靠会话	N/A
security	指定与采用此绑定配置的服务一起使用的安全类型	N/A
transactionFlow	确定是否启用事务流	false
transactionProtocol	服务在对事务进行流处理时使用的事务处理协议	OleTransaction

24.4.5　netTcpBinding 的地址和配置

如果使用 netTcpBinding 绑定，那么终节点的地址将以如下的形式出现。

```
net.tcp://[Hostname]:[Port]/[ServiceAddress]
```

地址中的 net.tcp 对应 TCP 协议，这里地址中的协议模式对应的是 TcpTransport-BindingElement。[Hostname]指的是主机，这里可以填写主机名、主机 IP， [Port]指的是端口，netTcpBinding 绑定使用的默认端口是 808 端口。

这里仍然借助在第 1 章给出的 HelloWorld 服务来演示 netTcpBinding 在配置文件中的配置方法。示例代码 24-11 展示了在服务端配置 netTcpBinding。

示例代码 24-11

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
<system.web>
  <compilation debug="true" /><!--允许 debug-->
</system.web>
<system.serviceModel>
  <services>
    <service name="HelloWorldService.Service">
      <host>
        <baseAddresses>
          <!--基地址-->
          <add baseAddress="net.tcp://ProfessionalWCFServer:808/ "></add>
        </baseAddresses>
      </host>
      <!--使用 netTcpBinding-->
      <endpoint address ="HelloWorldService" <!--地址-->
               binding="netTcpBinding" <!--绑定-->
               bindingConfiguration="myBinding"<!--绑定设置名-->
               contract="HelloWorldService.IService"><!--契约-->
      </endpoint>
    </service>
  </services>
  <bindings>
    <!--设置 netTcpBinding 的属性-->
    <netTcpBinding>
     <!--绑定属性设置-->
      <binding name="myBinding"<!--设置 myBinding 属性-->
              closeTimeout="00:01:00"<!--设置 closeTimeout 属性-->
              openTimeout="00:01:00"<!--设置 openTimeout 属性-->
              receiveTimeout="00:10:00"<!设置 receiveTimeout 属性-->
              sendTimeout="00:01:00"<!设置 sendTimeout 属性-->
              transactionFlow="false"<!设置 transactionFlow 属性-->
              transferMode="Buffered"<!设置 transferMode 属性-->
              transactionProtocol="OleTransactions"<!设置
              transactionProtocol 属性-->
              hostNameComparisonMode="StrongWildcard"<!设置
              hostNameComparisonMode 属性-->
              listenBacklog="10"<!设置 listenBacklog 属性-->
              maxBufferPoolSize="524288"<!设置 maxBufferPoolSize 属性-->
              maxBufferSize="65536"<!设置 maxBufferSize 属性-->
              maxConnections="10"<!设置 maxConnections 属性-->
              maxReceivedMessageSize="65536"><!设置
              maxReceivedMessageSize 属性-->
        <readerQuotas maxDepth="32"<!设置 SOAP 消息复杂性约束-->
                    maxStringContentLength="8192"
                    maxArrayLength="16384"
                    maxBytesPerRead="4096"
                    maxNameTableCharCount="16384" />
        <reliableSession ordered="true"<!设置可靠会话-->
                       inactivityTimeout="00:10:00"
                       enabled="false" />
        <security mode="Transport"><!设置安全模式-->
          <transport clientCredentialType="Windows" protectionLevel=
          "EncryptAndSign" />
        </security>
      </binding>
    </netTcpBinding>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    </bindings>
  </system.serviceModel>
</configuration>
```

同样的，也可以使用配置文件来配置 netTcpBinding 的使用。示例代码 24-12 给出了客户端的配置示例。

示例代码 24-12

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <client>
        <!--使用 netTcpBinding-->
        <endpoint address ="net.tcp://ProfessionalWCFServer:808/
        HelloWorldService" <!--地址-->
                  binding="netTcpBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
    </client>
    <bindings>
      <!--设置 netTcpBinding 的属性-->
      <netTcpBinding>
        <!--绑定属性设置-->
        <binding name="myBinding"<!--设置 myBinding 属性-->
                 closeTimeout="00:01:00"<!--设置 closeTimeout 属性-->
                 openTimeout="00:01:00"<!--设置 openTimeout 属性-->
                 receiveTimeout="00:10:00"<!设置 receiveTimeout 属性-->
                 sendTimeout="00:01:00"<!设置 sendTimeout 属性-->
                 transactionFlow="false"<!设置 transactionFlow 属性-->
                 transferMode="Buffered"<!设置 transferMode 属性-->
                 transactionProtocol="OleTransactions"<!设置
                 transactionProtocol 属性-->
                 hostNameComparisonMode="StrongWildcard"<!设置
                 hostNameComparisonMode 属性-->
                 listenBacklog="10"<!设置 listenBacklog 属性-->
                 maxBufferPoolSize="524288"<!设置 maxBufferPoolSize 属性-->
                 maxBufferSize="65536"<!设置 maxBufferSize 属性-->
                 maxConnections="10"<!设置 maxConnections 属性-->
                 maxReceivedMessageSize="65536"><!设置
                 maxReceivedMessageSize 属性-->
          <readerQuotas maxDepth="32"<!设置 SOAP 消息复杂性约束-->
                        maxStringContentLength="8192"
                        maxArrayLength="16384"
                        maxBytesPerRead="4096"
                        maxNameTableCharCount="16384" />
          <reliableSession ordered="true"<!设置可靠会话-->
                           inactivityTimeout="00:10:00"
                           enabled="false" />
          <security mode="Transport"><!设置安全模式-->
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
          <transport clientCredentialType="Windows" protectionLevel=
          "EncryptAndSign" />
        </security>
      </binding>
    </netTcpBinding>
  </bindings>
 </system.serviceModel>
</configuration>
```

24.4.6　netTcpBinding 特点总结

在跨主机的 WCF-WCF 交互的情况下，netTcpBinding 是最佳的标准绑定选择。其特点如下：

- 支持单程传输模式。
- 支持请求-响应传输模式。
- 支持双工传输模式。
- 性能在标准绑定中较佳，但次于 netNamedPipeBinding。
- 支持 WS-事务协议。
- 支持传输层消息安全。
- 只限于 WCF 系统。
- 支持跨主机交互。
- 支持可靠会话传输。

24.5　与 WS-I Basic Web 服务进行交互的绑定和地址

WS-I Basic Web 服务是指基本 Web 服务。由于 Web 服务的使用非常普遍，并且它是平台无关的，所以与基本 Web 服务进行交互的情况非常常见。本节将介绍在于基本 Web 服务交互的情况下，绑定和地址的选择。

24.5.1　场景概述

Web Service 是一种网络服务，通过通用的规范，Web Service 技术允许使用者访问网络上每一个 Web Service 所提供的服务。在网络快速发展的今天，这种基于网络的分布式服务已经被广泛地应用。由于 Web 服务有较好的跨平台性，并且发展的较为成熟，其在 SOA 系统中占据了举足轻重的作用。正因为此，与 Web 服务进行交互也成为了 WCF 系统中较为常见的一种需求。

成熟的 Web 服务技术框架很多，包括基于 Java 的 AXIS 组件、Weblogic 容器，基于.NET 的 ASP.NET（ASMX 资源），以及后来的 WSE 开发包。这些组件有些只支持基本的 Web 服务协议（WS-I），而有些则支持改进的 Web 服务（包括一系列 WS-*协议）。本小节将讨论与基本 Web 服务交互的场景。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

24.5.2　SOAP 协议概述

Web 服务的消息传递都是基于 SOAP 协议进行的，而 SOAP 协议本身，又处于 HTTP/HTTPS 之上，所以 SOAP 协议可以轻易地穿越防火墙。同时借助 XML 的跨平台特性，SOAP 协议和具体的实现平台完全无关。SOAP 协议的全称是简单对象访问协议（Simple Object Access Protocol）。SOAP 致力于以 XML 形式提供了一个简单、轻量的用于在分散或分布环境中交换结构化和类型信息的机制。SOAP 只规范对象访问的方式，而不限制具体实现的技术环境。这意味着 SOAP 协议是一种跨技术平台的协议，一个.NET 客户端程序完全可以按照 SOAP 协议，访问一个基于 J2EE 技术体系架构的 Web Service。SOAP 访问仍然基于 HTTP 协议，同时其内容又以 XML 形式表示。

SOAP 规范由 4 部分组成：SOAP 信封（SOAP envelop）、SOAP 编码规则（SOAP encoding rules）、SOAP RPC 表示（SOAP RPC representation）、SOAP 绑定（SOAP binding）。笔者这里并不打算全面展开 SOAP 的全部细节，一个实际的例子可能更能够给予读者一个直观的认识。

在 Web 服务端，打算对外提供一个公共方法以供客户端调用，而客户端则需要提供这个方法需要的参数，并且最终得到返回值。假设这个方法被声明在 MyService.asmx 文件中，并且逻辑定义如下所示。

```
String GetString(int par1,int par2)//示例方法
{
    int temp=part1+part2;
    return temp.ToString();
}
```

当客户端试图使用这个 We 服务方法时，就需要向服务器端发出这样一个 HTTP 请求。

```
POST url/MyService.asmx HTTP/1.1
Host: [host]
Content-Type: text/xml;charset=utf-8
Content-Length:[length]
SOAPAction:"http://www.book.com/webservices/GetString"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <GetString xmlns="http://www.book.com/webservices">
            <par1>1</par1><!--参数 1-->
            <par2>2</par2><!--参数 2-->
        </<GetString>
    </soap:Body>
</soap:Envelope>
```

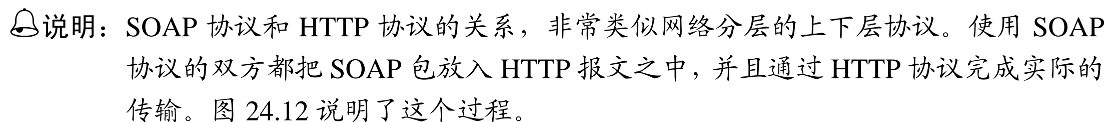
而等到服务器端接收到这样的请求后，就可以进行相应的逻辑处理，并且返回结果。根据 SOAP 协议，HTTP 响应类似下面的形式。

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Content-Length:[length]
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
             xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
             xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <GetString xmlns="http://www.book.com/webservices">
         <GetStringResult>3</GetStringResult><!--返回参数-->
      </<GetString>
   </soap:Body>
</soap:Envelope>
```

这样，客户端就实际得到了服务器端的处理结果。也就是说，客户端已经实际得到了 Web 服务提供的服务。



说明：SOAP 协议和 HTTP 协议的关系，非常类似网络分层的上下层协议。使用 SOAP 协议的双方都把 SOAP 包放入 HTTP 报文之中，并且通过 HTTP 协议完成实际的传输。图 24.12 说明了这个过程。

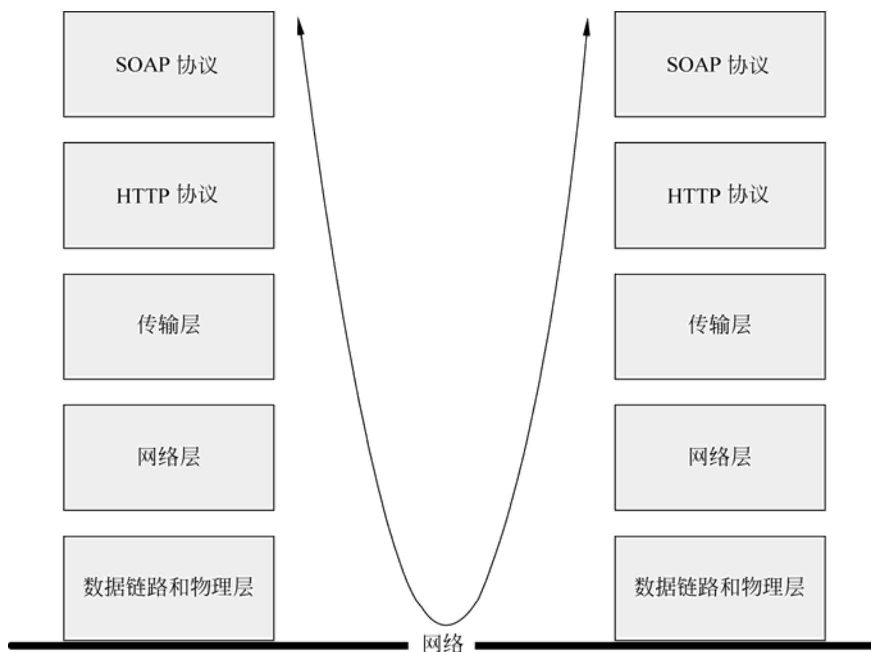


图 24.12　SOAP 协议位置

24.5.3　使用 basicHttpBinding

basicHttpBinding 绑定直接支持 WS-I Basic Web 服务，其内容包括了 SOAP1.1，WSDL1.1 和消息安全协议。basicHttpBinding 提供了和其他系统的兼容性，但是它却不支持改进的 Web 服务协议。

和其他绑定一样，basicHttpBinding 提供了可配置的属性。表 24.6 列出了这些属性的名称和简要说明。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

表 24.6　basicHttpBinding的属性

属　性　名	描　　述	默　认　值
bypassProxyOnLocal	在访问本地终节点时绕过代理	false
closeTimeout	等待连接的超时时间	00:01:00
hostNameComparisonMode	指定将传入的消息调度到服务终节点时应该如何在 URI 比较中使用主机名	StrongWildCard
maxBufferPoolSize	内存中用于对传入消息进行缓冲的最大字节数	524888
maxBufferSize	内存中用于对传入消息进行缓冲的最大字节数	65536
maxReceivedMessageSize	定义在采用此绑定配置的通道上可以接收的消息的最大消息大小（字节），包括消息标头。如果消息对于接收方而言太大，则发送方将收到 SOAP 错误。接收方将删除该消息，并在跟踪日志中创建事件项	65536
messageEncoding	定义用于对 SOAP 消息进行编码的编码器，可选项包括 Text 和 Mtom	Text
name	绑定的名字	
openTimeout	在传输引发异常之前可用于打开连接的时间间隔	00:01:00
proxyAddress	代理地址	N/A
readerQuotas	可由使用此绑定配置的终节点处理的 SOAP 消息的复杂性约束	N/A
security	指定与采用此绑定配置的服务一起使用的安全类型	N/A
sendTimeout	指定为完成发送操作提供的时间间隔	00:01:00
textEncoding	设置要用来在绑定上发出消息的字符集编码，可选项包括：BigEndianUnicode、Unicode 和 UTF8	utf-8
transferMode	一个有效的 TransferMode 值，指定为请求或响应对消息进行缓冲处理还是流式处理	Buffered
userDefaultWebProxy	指定是否应在可用时使用系统的自动配置 HTTP 代理	true

24.5.4　basicHttpBinding 的地址和配置

由于 Web 服务最终是基于 HTTP/HTTPS 协议的，所以使用 basicHttpBinding 绑定的地址设定也使用 HTTP/HTTPS 作为地址协议部分。其具体模式如下所示。

```
http://[Hostname]:[Port]/[ServiceAddress]
https://[Hostname]:[Port]/[ServiceAddress]
```

上面的地址形式对于任何使用 HttpTransportBindingElement 都是适用的。默认情况下，使用 HTTP 协议的端口号是 80，而使用 HTTPS 协议的端口号是 443。示例代码 24-13 展示了如何在服务端配置 basicHttpBinding。

示例代码 24-13

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" /> <!--允许 debug-->
  </system.web>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
  <system.serviceModel>
    <services>
      <service name="HelloWorldService.Service">
        <host>
          <baseAddresses>
            <!--基地址-->
            <add baseAddress="http://ProfessionalWCFServer:80/"></add>
          </baseAddresses>
        </host>
        <!--使用 basicHttpBinding-->
        <endpoint address ="HelloWorldService" <!--地址-->
                  binding="basicHttpBinding" <!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
      </service>
    </services>
    <bindings>
      <!--设置 basicHttpBinding 的属性-->
      <basicHttpBinding>
        <binding name="myBinding"<!--设置 name 属性-->
               hostNameComparisonMode="StrongWildcard"
               <!--设置 hostNameComparisonMode 属性-->
               receiveTimeout="00:10:00"<!--设置 receiveTimeout 属性-->
               sendTimeout="00:10:00"<!--设置 sendTimeout 属性-->
               openTimeout="00:10:00"<!--设置 openTimeout 属性-->
               closeTimeout="00:10:00"<!--设置 closeTimeout 属性-->
               maxMessageSize="65536"<!--设置 maxMessageSize 属性-->
               maxBufferSize="65536"<!--设置 maxBufferSize 属性-->
               maxBufferPoolSize="524288"<!--设置 maxBufferPoolSize 属性-->
               transferMode="Buffered"<!--设置 transferMode 属性-->
               messageEncoding="Text"<!--设置 messageEncoding 属性-->
               textEncoding="utf-8"<!--设置 textEncoding 属性-->
               bypassProxyOnLocal="false"<!--设置 bypassProxyOnLocal 属性-->
               useDefaultWebProxy="true" ><!--设置 useDefaultWebProxy 属性-->
          <security mode="None" />
        </binding>
      </basicHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

对应的，在客户端的配置如示例代码 24-14 所示。

示例代码 24-14

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" /> <!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <client>
        <!--使用 basicHttpBinding-->
        <endpoint address ="http://ProfessionalWCFServer:80/
        HelloWorldService" <!--地址-->
                  binding="basicHttpBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
                contract="HelloWorldService.IService"><!--契约-->
       </endpoint>
    </client>
    <bindings>
      <!--设置 basicHttpBinding 的属性-->
      <basicHttpBinding>
        <binding name="myBinding"<!--设置 name 属性-->
               hostNameComparisonMode="StrongWildcard"<!--设置
               hostNameComparisonMode 属性-->
               receiveTimeout="00:10:00"<!--设置 receiveTimeout 属性-->
               sendTimeout="00:10:00"<!--设置 sendTimeout 属性-->
               openTimeout="00:10:00"<!--设置 openTimeout 属性-->
               closeTimeout="00:10:00"<!--设置 closeTimeout 属性-->
               maxMessageSize="65536"<!--设置 maxMessageSize 属性-->
               maxBufferSize="65536"<!--设置 maxBufferSize 属性-->
               maxBufferPoolSize="524288"<!--设置 maxBufferPoolSize 属性-->
               transferMode="Buffered"<!--设置 transferMode 属性-->
               messageEncoding="Text"<!--设置 messageEncoding 属性-->
               textEncoding="utf-8"<!--设置 textEncoding 属性-->
               bypassProxyOnLocal="false"<!--设置 bypassProxyOnLocal 属性-->
               useDefaultWebProxy="true" ><!--设置 useDefaultWebProxy 属性-->
          <security mode="None" />
        </binding>
      </basicHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

24.5.5 basicHttpBinding 特点总结

在与基本 Web 服务系统交互的情况下，basicHttpBinding 是最佳的标准绑定选择。其特点如下：

- 支持单程传输模式。
- 支持请求-响应传输模式。
- 不支持双工传输模式。
- 性能在标准绑定中一般。
- 不支持 WS-事务协议。
- 支持传输层消息安全。
- 提供与基本 Web 服务的交互性。
- 支持跨主机交互。
- 支持可靠会话传输。

24.6 与改进 Web 服务进行交互的绑定和地址

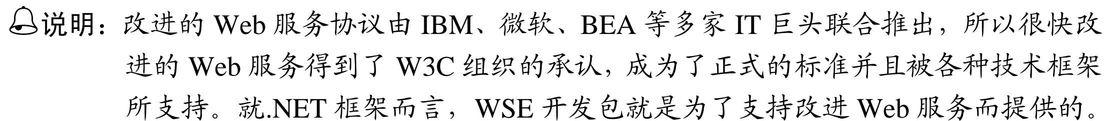
与基本 Web 服务相比，改进 Web 服务新增了安全、身份验证、加密、事务控制、可靠传输等多项内容，使得 Web 服务功能更加强大。在改进 Web 服务交互的场景中，绑定

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

和地址的选择都有所不同。本节将介绍与改进 Web 服务交互时绑定和地址的选择。

24.6.1　场景概述

Web 服务自身标准在经历了一段时间发展之后，提出了协议的改进版本，也就是俗称的改进的 Web 服务。改进的 Web 服务新增一系列 WS-*协议，从而在消息的过程中对安全、事务、可靠传输等方面进行支持。



说明：改进的 Web 服务协议由 IBM、微软、BEA 等多家 IT 巨头联合推出，所以很快改进的 Web 服务得到了 W3C 组织的承认，成为了正式的标准并且被各种技术框架所支持。就.NET 框架而言，WSE 开发包就是为了支持改进 Web 服务而提供的。

由于改进 Web 服务协议的普遍应用，势必然 WCF 设计小组需要考虑与改进 Web 服务系统交互的能力。而其结果是 WCF 可以完全支持与改进 Web 服务进行消息交互，并且支持一系列的 WS-*协议。

24.6.2　改进 Web 服务协议概述

和其他所有的协议发展一样，改进 Web 服务协议并没有颠覆基本的 Web 服务，而是在其基础上添加了一系列的 WS-*协议，来支持功能更加丰富的 SOAP 消息交互。所以从其实现上来说，改进 Web 服务能够有效地兼容基本 Web 服务。

WS-*的协议众多，但其中最为核心的 3 个内容，分别为 WS-Security、WS-Atomic 事务流和 WS-ReliableMessaging。这里将简要介绍这个协议的内容。

1．WS-Security安全协议

WS-Security 的制定目标是使用应用程序构建安全的 SOAP 消息交换。WS-Security 很灵活，它被设计成用来构建多种安全性模型（包括 PKI、Kerberos 和 SSL）的基础。WS-Security 特别为多安全性令牌、多信任域、多签名格式和多加密技术提供支持。

WS-Security 通过利用现有标准和规范来实现安全性，这样就不必在 WS-Security 中定义一个完整的安全性解决方案了。业界已经解决了许多此类问题。例如 Kerberos 和 X.509 用于身份验证；X.509 还使用现有的 PKI 进行密钥管理；XML 加密和 XML 签名描述了 XML 消息内容的加密和签名方法；XML 标准描述了为签名和加密而准备 XML 的方法。WS-Security 在现有规范中添加了一个框架，用于将这些机制嵌入到 SOAP 消息中。这是以一种与传输无关的方式完成的。

（1）身份验证

从身份验证的内容来说，WS-Security 提供了 3 种方法来实现，分别为：

- 用户名/密码。
- 通过 X.509 证书的 PKI。
- Kerberos。

用户名/密码的验证方法如下列消息范例所示。

```
<xs:element name="UsernameToken"> <!--用户名/密码的验证方法-->
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="David"/>
            <xs:element ref="DavidPwd" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="Id" type="xs:ID"/>
    </xs:complexType>
</xs:element>
```

（2）签名

签名后的消息几乎无法被篡改。消息签名不能禁止外部各方查看消息内容。使用签名，SOAP 消息的接收方可以知道已签名的元素在路由中未发生改变。只要可能，就应当使用 XML 签名对消息进行签名。XML 签名已经处理了许多难以描述的问题。WS-Security 只是简单解释了如何使用签名来证明消息没有被更改。在 SOAP 消息中，签名和所需的额外数据添加了大量额外信息。下面是一个签名的消息示例。

```
<Signature Id="MySignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-
    c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/
    xmldsig#dsa-sha1"/>
    <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
        20010315"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    </Reference>
  </SignedInfo>
  <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue><!--签名-->
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <p>省略信息</p>
        <Q>省略信息</Q>
        <G>省略信息</G>
        <Y>省略信息</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
</Signature>
```

（3）加密

有时，仅仅证明消息发送方的身份和表明消息未经更改仍然不够。如果通过签名的纯文本来发送信用卡号或银行账号，攻击者实际上可以验证没有其他攻击者更改过消息的内容。与处理消息签名一样，WS-Security 为此也提供了相应规范，采用了现有标准，并且能够很好地完成加密工作。实际上，它们并入了 XML 加密。

加密数据时，可以选择使用对称加密或不对称加密。对称加密需要一个共享密钥。也就是说，加密消息与解密消息使用的是同一个密钥。如果同时控制两个端点并且可以信任使用密钥的用户和应用程序，则可以使用对称加密。对称加密在密钥分发上存在一点问题。在某个时间点上，密钥需要发送给接收方。

如果需要使用简单的分布式密钥来发送数据，则可以使用不对称加密。X.509 证书允

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

许使用不对称加密。接收数据的端点可以公布它的证书，并允许任何人使用公钥来加密信息。只有接收方知道私钥。因此，只有接收方可以得到加密的数据并将其重新转换为可读内容。

那么，加密后的消息是什么形式呢？如果使用的是 Triple-DES，则发送方和接收方都必须以某种安全的方式交换密钥。对称密钥可以隐藏在 Kerberos 票据内，或被取出交换。

2．WS-Atomic事务流

为了在普通的 SOAP 消息交互中加入事务处理的能力，改进 Web 服务协议中定义了 WS-Atomic 事务流协议，该协议建立在 WS-Coordination 协议之上。WS-Coordination 协议定义了激活、注册服务。这里暂且不讨论 WS-Coordination 协议的细节，有兴趣的读者可以参考 W3C 的协议文档。

WS-Atomic 事务流协议主要定义了两个事务协议，如下所示。

（1）完成协议（Completion）

完成协议被用来启动提交/回滚流程。基于注册协议的每一个事务参与者，事务协调者会先处理易失两段提交，再处理持久两段提交，其结果最后被发送到事务发起者。

事务协调者接收下列所示的消息。

- 提交（Commit）：接收到提交通知时，协调者认为事务参与方已经完成了事务并且准备提交。
- 回滚（Rollback）：接收到回滚通知时，协调者认为事务参与方中止了事务并且准备回滚。

而事务发起者接收下列消息。

- 已提交（Committed）：当收到已提交通知后，事务发起者认为事务协调者决定提交事务。
- 已中止（Aborted）：当收到已中止通知后，事务发起者认为事务协调者决定中止事务。图 24.13 展示了 WS-Atomic 完成协议的流程。

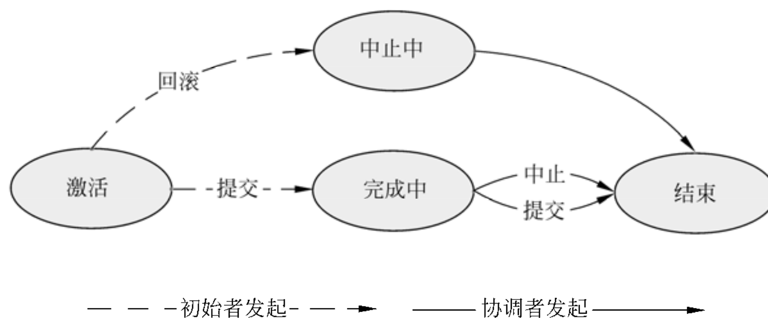


图 24.13　完成协议流程

（2）两段提交协议

两段提交协议是特别针对分布式事务提出的，两段提交协议分为两个类型，分别为易失两段提交协议和持久两段提交协议。接收到提交通知后，协调者开始所有注册了易失两段提交协议的参与者的提交准备，所有参与者都必须及时给出回复。在所有易失二段提交协议参与者的提交准备结束后，协调者将开始所有注册了持久两段提交协议的提交准备。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

在两段提交协议中，事务参与者接收如下所示的消息。

- 准备（Prepare）：当接收到准备通知后，参与者将进入第一阶段并且进行投票，是提交该事务还是中止该事务。如果参与者已经对该事务进行投票，那它必须再次投票并且保证投票结果和上次一致。如果参与者对该事务没有了解，则它必须投中止票。
- 回滚（Rollback）：当接收到回滚通知后，参与者将中止并丢弃事务。该消息既可以出现在提交的第一阶段，也可以出现在第二阶段。一旦发出该消息，协调者将丢弃该事务。
- 提交（Commit）：当收到提交通知后，参与者将提交事务。该消息必定在第二阶段发出，并且必须在参与者投出提交票之后。如果参与者不了解该事务，必须向协调者发出已提交（Committed）通知。

事务协调者接收下列消息：

- 准备完毕（Prepared）：当收到准备完毕通知后，协调者知道参与者已经准备好并投票提交事务。
- 只读（ReadOnly）：当收到只读通知后，协调者知道参与者投出了提交票，并且已经丢弃事务。该参与者并不希望参与到两段提交的第二阶段。
- 中止（Aborted）：当收到中止通知后，协调者知道参与者已经中止并丢且了事务。
- 已提交（Commited）：当收到已提交通知后，协调者知道参与者已经提交事务，这时参与者可以安全地丢弃事务了。
- 重现（Replay）：当收到重现通知后，协调者将视参与者刚刚从一个可恢复的错误中恢复。协调者将重新发送上一个协议通知。

图 24.14 展示了 WS-Atomic 两段提交协议的流程。

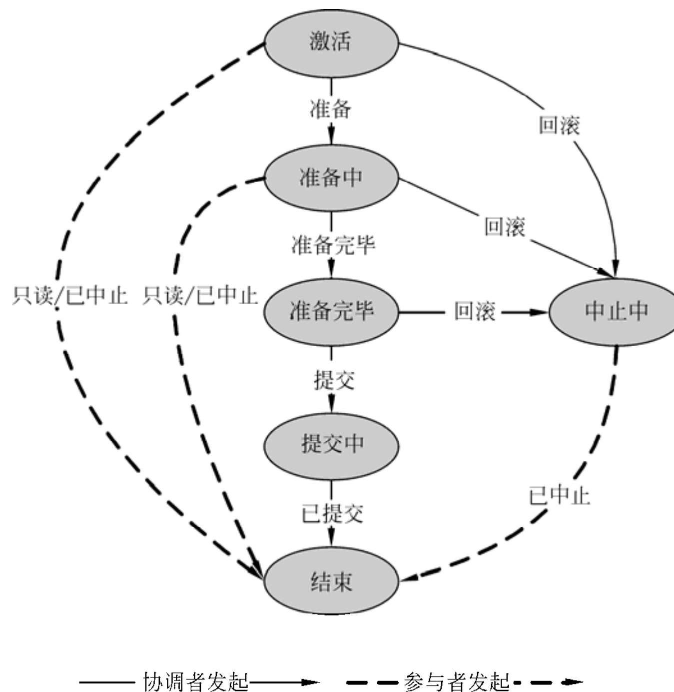


图 24.14　WS-Atomic 两段提交协议流程

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

3．WS-ReliableMessaging协议

WS-ReliableMessaging 协议指定的目的是保证消息的可靠传输。这里的可靠传输精确概念定义如下所示。

- 最多一次：如果没有发生任何错误，那一条消息最多只会被发送一次。
- 最少一次：每一条消息必须最少被发送一次。
- 有序：所有的消息必须被有序地发送。

WS-ReliableMessaging 的设计和 TCP 协议非常地类似，这里笔者仅列举一个实际传输示例来简要地介绍 WS-ReliableMessaging 协议。传输示例如图 24.15 所示。

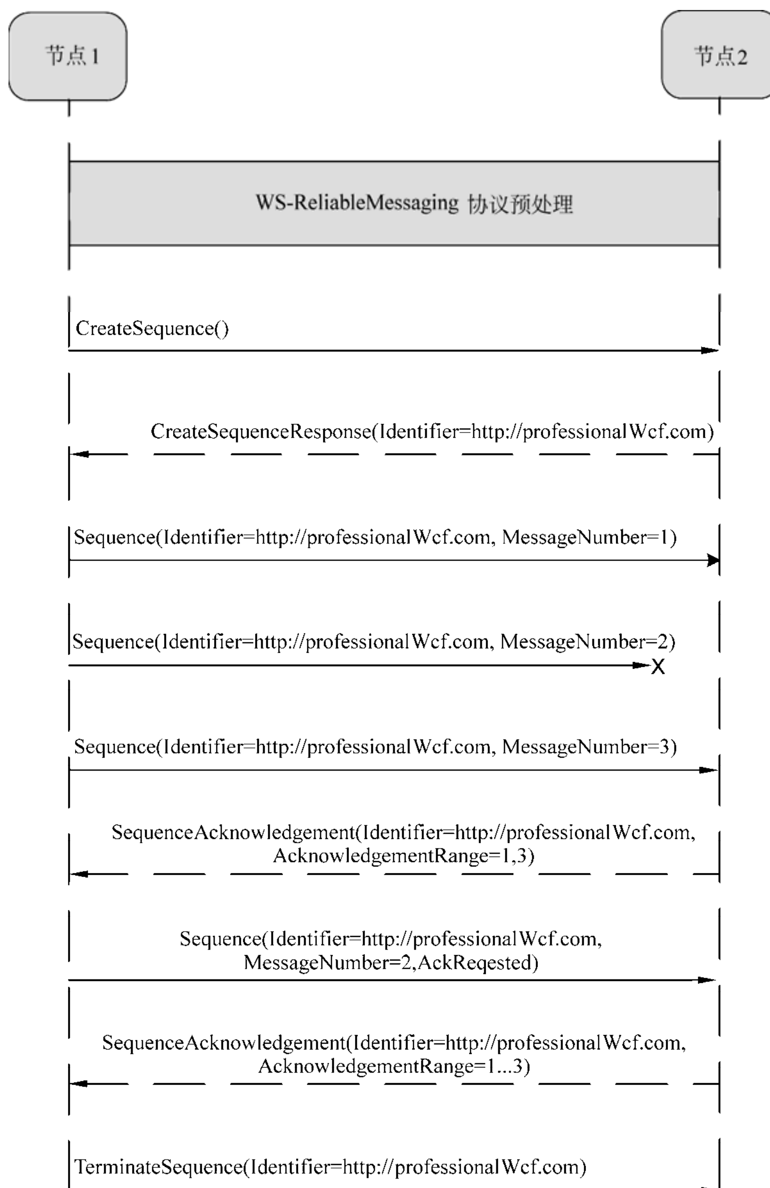


图 24.15　WS-ReliableMessaging 传输示例

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

在图 24.15 的示例中，节点 1 和节点 2 的消息传输经历了下述过程。

- 协议预处理，策略交换，节点信息交互，连接建立。
- 可靠消息源发出建立可靠消息传输请求。
- 可靠消息目的地建立消息传输，并返回一个 GUID。
- 可靠消息源开始依次发送消息，从 1 号到 3 号。
- 在 3 号消息发送时，发送端包含了<LastMessage>以通知发送完毕。
- 由于可靠消息发送目的地没有收到 2 号消息，所以只发送 1 号和 3 号消息的 ACK。
- 可靠消息发送源重新发送 2 号消息，并且指定要求 ACK。
- 可靠消息目的地收到 2 号消息并发送 1、2、3 号消息的 ACK。
- 可靠消息源收到 ACK，并且发送关闭可靠消息传输的命令。
- 可靠消息目的地关闭可靠消息传输。

24.6.3 使用 wsHttpBinding

WCF 对改进的 Web 服务提供了全面的支持，就绑定来说，包括 wsHttpBinding 在内的多个标准绑定都支持改进的 Web 服务。wsHttpBinding 提供了和其他技术平台进行交互的能力，而其使用的通信协议正是改进的 SOAP 协议以及一系列 WS-*协议。表 24.7 列出了 wsHttpBinding 所支持的 WS-*协议。

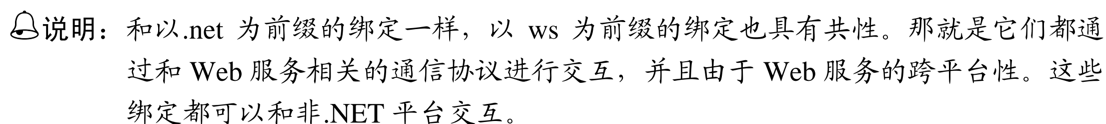

说明：和以.net 为前缀的绑定一样，以 ws 为前缀的绑定也具有共性。那就是它们都通过和 Web 服务相关的通信协议进行交互，并且由于 Web 服务的跨平台性。这些绑定都可以和非.NET 平台交互。

表 24.7 wsHttpBinding支持的改进Web服务协议

协　议	说　明
SOAP1.2	简单对象访问协议 1.2 版本
WS-Addressing 2005/08	为以同步和/或异步方式传输的 SOAP 消息提供了一种统一的寻址方法
WSS Message Security 1.0	支持使用诸如 PKI、Kerberos、SSL 等机制保证 Web 服务安全性
WSS Message Security UsernameToken Profile1.1	支持使用用户名/密码机制保证安全
WSS SOAP Message Security X509 Token Profile1.1	支持使用 X.509 证书
WS-SecureConversation	安全策略断言
WS-ReliableMessage	可靠消息传输
WS-Coordination	协调分布式系统动作的协议
WS-Atomic Transactions	分布式事务协议
WS-Addressing	为以同步和/或异步方式传输的 SOAP 消息提供了一种统一的寻址方法

wsHttpBinding 的可配置属性如表 24.8 所示。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

表 24.8　wsHttpBinding的属性

属 性 名	描　　述	默 认 值
bypassProxyOnLocal	在访问本地终节点时绕过代理	false
closeTimeout	等待连接的超时时间	00:01:00
hostNameComparisonMode	指定将传入的消息调度到服务终节点时应该如何在 URI 比较中使用主机名	StrongWildCard
maxBufferPoolSize	内存中用于对传入消息进行缓冲的最大字节数	524888
maxReceivedMessageSize	定义在采用此绑定配置的通道上可以接收的消息的最大消息大小（字节），包括消息标头。如果消息对于接收方而言太大，则发送方将收到 SOAP 错误。接收方将删除该消息，并在跟踪日志中创建事件项	65536
messageEncoding	定义用于对 SOAP 消息进行编码的编码器，可选项包括 Text 和 Mtom	Text
name	绑定的名字	
openTimeout	在传输引发异常之前可用于打开连接的时间间隔	00:01:00
proxyAddress	代理地址	N/A
readerQuotas	可由使用此绑定配置的终节点处理的 SOAP 消息的复杂性约束	N/A
receiveTimeout	在传输引发异常之前可用于完成读取操作的时间间隔	00:01:00
reliableSession	获取一个对象，当使用系统提供的一个绑定时，该对象可提供对可用的可靠会话绑定元素属性的便捷访问	false
security	指定与采用此绑定配置的服务一起使用的安全类型	N/A
sendTimeout	指定为完成发送操作提供的时间间隔。	00:01:00
textEncoding	设置要用来在绑定上发出消息的字符集编码，可选项包括 BigEndianUnicode、Unicode 和 UTF8。	utf-8
transactionFlow	指示此绑定是否应支持流动 WS-Transactions	false
userDefaultWebProxy	指定是否应在可用时使用系统的自动配置 HTTP 代理	true

24.6.4　wsHttpBinding 的地址和配置

改进的 Web 服务仍然基于 HTTP/HTTPS 协议进行消息交互，所以和 basicHttpBinding 一样，wsHttpBinding 使用 HTTP/HTTPS 作为地址协议部分。其具体模式如下所示。

```
http://[Hostname]:[Port]/[ServiceAddress]
https://[Hostname]:[Port]/[ServiceAddress]
```

示例代码 24-15 展示了如何在服务端配置 wsHttpBinding。

示例代码 24-15

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <services>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
      <service name="HelloWorldService.Service">
        <host>
          <baseAddresses>
            <!--基地址-->
            <add baseAddress="http://ProfessionalWCFServer:80/"></add>
          </baseAddresses>
        </host>
        <!--使用 wsHttpBinding-->
        <endpoint address ="HelloWorldService" <!--地址-->
                  binding="wsHttpBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
      </service>
    </services>
    <bindings>
      <!--设置 wsHttpBinding 的属性-->
      <wsHttpBinding>
        <binding name="myBinding"<!--设置 name 属性-->
            closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
            openTimeout="00:00:20"<!--设置 openTimeout 属性-->
            receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
            sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
            bypassProxyOnLocal="false"<!--设置 bypassProxyOnLocal 属性-->
            transactionFlow="false"<!--设置 transactionFlow 属性-->
            hostNameComparisonMode="WeakWildcard"<!--设置
            hostNameComparisonMode 属性-->
            maxMessageSize="1000"<!--设置 maxMessageSize 属性-->
            messageEncoding="Mtom"<!--设置 messageEncoding 属性-->
            proxyAddress="http://ProfessionalWCF/proxy"<!--设置
            proxyAddress 属性-->
            textEncoding="utf-16"<!--设置 textEncoding 属性-->
            useDefaultWebProxy="false"><!--设置 useDefaultWebProxy 属性-->
          <reliableSession ordered="false"<!--设置可靠会话-->
              inactivityTimeout="00:02:00"
              enabled="true" />
          <security mode="Transport"><!--安全设置-->
            <transport clientCredentialType="Digest"
              proxyCredentialType="None"
              realm="someRealm" />
            <message clientCredentialType="Windows"
              negotiateServiceCredential="false"
              algorithmSuite="Aes128"
              defaultProtectionLevel="None" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

示例代码 24-16 展示了如何在客户端配置 wsHttpBinding。

示例代码 24-16

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <client>
        <!--使用 basicHttpBinding-->
        <endpoint address ="http://ProfessionalWCFServer:80/
        HelloWorldService" <!--地址-->
                  binding="wsHttpBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
    </client>
    <bindings>
      <!--设置 wsHttpBinding 的属性-->
      <wsHttpBinding>
        <binding name="myBinding"<!--设置 name 属性-->
            closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
            openTimeout="00:00:20"<!--设置 openTimeout 属性-->
            receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
            sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
            bypassProxyOnLocal="false"<!--设置 bypassProxyOnLocal 属性-->
            transactionFlow="false"<!--设置 transactionFlow 属性-->
            hostNameComparisonMode="WeakWildcard"<!--设置
            hostNameComparisonMode 属性-->
            maxMessageSize="1000"<!--设置 maxMessageSize 属性-->
            messageEncoding="Mtom"<!--设置 messageEncoding 属性-->
            proxyAddress="http://ProfessionalWCF/proxy"<!--设置
            proxyAddress 属性-->
            textEncoding="utf-16"<!--设置 textEncoding 属性-->
            useDefaultWebProxy="false"><!--设置 useDefaultWebProxy 属性-->
          <reliableSession ordered="false"<!--设置可靠会话-->
              inactivityTimeout="00:02:00"
              enabled="true" />
          <security mode="Transport"><!--安全设置-->
            <transport clientCredentialType="Digest"
               proxyCredentialType="None"
               realm="someRealm" />
            <message clientCredentialType="Windows"
               negotiateServiceCredential="false"
               algorithmSuite="Aes128"
               defaultProtectionLevel="None" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

24.6.5　wsHttpBinding 特点总结

WCF 提供了如下所示的多个支持改进 Web 服务的绑定，wsHttpBasic 是其中较为常用的一个，其比较显著的一个特点在于不支持双工模式，这也是它和 wsDualHttpBinding 最大的区别。

- 支持单程传输模式。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- 支持请求-响应传输模式。
- 不支持双工传输模式。
- 性能在标准绑定中一般。
- 支持 WS-事务协议。
- 支持传输层消息安全。
- 提供与改进 Web 服务的交互性。
- 支持跨主机交互。
- 支持可靠会话传输。

24.6.6 使用 wsDualHttpBinding

使用 wsHttpBinding 基本能够满足和改进 Web 服务系统进行交互的需求，但 wsHttpBinding 却不支持双工传输模式。如果希望支持和改进 Web 服务系统进行双工交互，就需要选择 wsDualHttpBinding，wsDualHttpBinding 支持双工传输模式，但却不支持传输层的消息安全。除了这两点外，wsDualHttpBinding 和 wsHttpBinding 完全一致。表 24.9 列出了 wsDualHttpBinding 的属性。

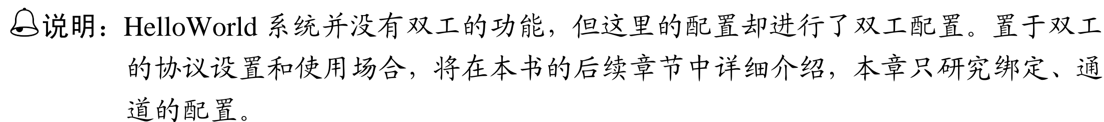
表 24.9 wsDualHttpBinding的属性

属 性 名	描 述	默 认 值
bypassProxyOnLocal	在访问本地终节点时绕过代理	false
closeTimeout	等待连接的超时时间	00:01:00
hostNameComparisonMode	指定将传入的消息调度到服务终节点时应该如何在 URI 比较中使用主机名	StrongWildCard
maxBufferPoolSize	内存中用于对传入消息进行缓冲的最大字节数	524888
maxReceivedMessageSize	定义在采用此绑定配置的通道上可以接收的消息的最大消息大小（字节），包括消息标头。如果消息对于接收方而言太大，则发送方将收到 SOAP 错误。接收方将删除该消息，并在跟踪日志中创建事件项	65536
messageEncoding	定义用于对 SOAP 消息进行编码的编码器，可选项包括 Text 和 Mtom	Text
name	绑定的名字	
openTimeout	在传输引发异常之前可用于打开连接的时间间隔	00:01:00
proxyAddress	代理地址	N/A
readerQuotas	可由使用此绑定配置的终节点处理的 SOAP 消息的复杂性约束	N/A
receiveTimeout	在传输引发异常之前可用于完成读取操作的时间间隔	00:01:00
reliableSession	获取一个对象，当使用系统提供的一个绑定时，该对象可提供对可用的可靠会话绑定元素属性的便捷访问	false
security	指定与采用此绑定配置的服务一起使用的安全类型	N/A
sendTimeout	指定为完成发送操作提供的时间间隔	00:01:00
textEncoding	设置要用来在绑定上发出消息的字符集编码，可选项包括：BigEndianUnicode、Unicode 和 UTF8	utf-8
transactionFlow	指示此绑定是否应支持流动 WS-Transactions	false
userDefaultWebProxy	指定是否应在可用时使用系统的自动配置 HTTP 代理	true

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

24.6.7　wsDualHttpBinding 的地址和配置

wsDualHttpBinding 的地址形式和 wsHttpBinding 完全一致，这里不再重复列出。示例代码 24-17 展示了如何在服务端配置 wsDualHttpBinding。


说明：HelloWorld 系统并没有双工的功能，但这里的配置却进行了双工配置。置于双工的协议设置和使用场合，将在本书的后续章节中详细介绍，本章只研究绑定、通道的配置。

示例代码 24-17

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <services>
      <service name="HelloWorldService.Service">
        <host>
          <baseAddresses>
            <!--基地址-->
            <add baseAddress="http://ProfessionalWCFServer:80/"></add>
          </baseAddresses>
        </host>
        <!--使用 wsDualHttpBinding-->
        <endpoint address ="HelloWorldService" <!--地址-->
                  binding="wsDualHttpBinding" <!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
      </service>
    </services>
    <bindings>
      <!--设置 wsDualHttpBinding 的属性-->
      <wsDualHttpBinding>
        <binding name="myBinding"<!--设置 name 属性-->
            closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
            openTimeout="00:00:20"<!--设置 openTimeout 属性-->
            receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
            sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
            bypassProxyOnLocal="false"<!--设置 bypassProxyOnLocal 属性-->
            transactionFlow="true"<!--设置 transactionFlow 属性-->
            hostNameComparisonMode="WeakWildcard"<!--设置
            hostNameComparisonMode 属性-->
            maxReceivedMessageSize="1000"<!--设置 maxReceivedMessageSize
            属性-->
            messageEncoding="Mtom"<!--设置 messageEncoding 属性-->
            proxyAddress="http://ProfessionalWCF/proxy"<!--设置
            proxyAddress 属性-->
            textEncoding="utf-16"<!--设置 textEncoding 属性-->
            useDefaultWebProxy="false"><!--设置 useDefaultWebProxy 属性-->
          <reliableSession ordered="false"<!--设置可靠会话-->
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
            inactivityTimeout="00:02:00" />
          <security mode="None"><!--安全设置-->
            <message clientCredentialType="None"
               negotiateServiceCredential="false"
               algorithmSuite="Aes128" />
          </security>
        </binding>
      </wsDualHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

示例代码 24-18 展示了如何在客户端配置 wsDualHttpBinding。读者需要注意，由于是双工模式交互，这里的客户端配置和本章前面介绍的客户端绑定略有不同，需要在绑定配置中指定客户端的基地址。

示例代码 24-18

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <client>
        <!--使用 basicHttpBinding-->
        <endpoint address ="http://ProfessionalWCFServer:80/
        HelloWorldService" <!--地址-->
                  binding="wsDualHttpBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
    </client>
    <bindings>
      <!--设置 wsDualHttpBinding 的属性-->
      <wsDualHttpBinding>
        <binding name="myBinding"<!--设置 name 属性-->
            closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
            openTimeout="00:00:20"<!--设置 openTimeout 属性-->
            receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
            sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
            bypassProxyOnLocal="false"<!--设置 bypassProxyOnLocal 属性-->
            transactionFlow="true"<!--设置 transactionFlow 属性-->
            hostNameComparisonMode="WeakWildcard"<!--设置
            hostNameComparisonMode 属性-->
            maxReceivedMessageSize="1000"<!--设置 maxReceivedMessageSize
            属性-->
            messageEncoding="Mtom"<!--设置 messageEncoding 属性-->
            proxyAddress="http://ProfessionalWCF/proxy"<!--设置
            proxyAddress 属性-->
            textEncoding="utf-16"<!--设置 textEncoding 属性-->
            useDefaultWebProxy="false"><!--设置 useDefaultWebProxy 属性-->
          <reliableSession ordered="false"<!--设置可靠会话-->
            inactivityTimeout="00:02:00" />
          <security mode="None"><!--安全设置-->
            <message clientCredentialType="None"
               negotiateServiceCredential="false"
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
                algorithmSuite="Aes128" />
          </security>
        </binding>
      </wsDualHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

24.6.8 wsDualHttpBinding 特点总结

正如笔者在前文中所述，wsDualHttpBinding 和 wsHttpBinding 的区别在于两方面：前者支持双工模式，而后者支持传输层的消息安全。wsDualHttpBinding 的特点总结如下：

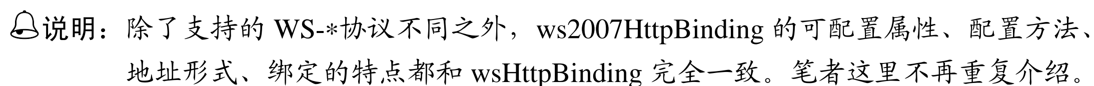
- 支持单程传输模式。
- 支持请求-响应传输模式。
- 支持双工传输模式。
- 性能在标准绑定中一般。
- 支持 WS-事务协议。
- 不支持传输层消息安全。
- 提供与改进 Web 服务的交互性。
- 支持跨主机交互。
- 支持可靠会话传输。

24.6.9 使用 ws2007HttpBinding

.NET Framework 3.5 版本推出了 wsHttpBinding 的更新版本：ws2007HttpBinding。ws2007HttpBinding 继承自 wsHttpBinding，与旧版本相比，ws2007HttpBinding 支持最新的 WS-*协议。表 24.10 列出了 ws2007HttpBinding 所支持的最新的 WS-*协议。

表 24.10 ws2007HttpBinding所支持的WS-*协议

协 议	说 明
WS-SecureConversation v1.3	WS-Security 的扩展，为多消息交互提供安全上下文
WS-Trust v1.3	WS-Security 的扩展，用于管理信任连接
WS-SecurityPolicy v1.2	安全策略断言 v1.2
WS-ReliableMessage v1.1	可靠消息传输 v1.1
WS-Atomic Transactions v1.1	分布式事务协议 v1.1
WS Services Coordination v1.1	协调分布式系统动作协议 v1.1



说明：除了支持的 WS-*协议不同之外，ws2007HttpBinding 的可配置属性、配置方法、地址形式、绑定的特点都和 wsHttpBinding 完全一致。笔者这里不再重复介绍。

24.7 使用脱机模式进行消息交互的绑定和地址

在有些场景下，消息交互的一方无法确定另一方是否可访问。而在大多数交互方式中，

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

如果交互的一方处于脱机状态下，则交互将失败。脱机模式则是指交互双方无须同时在线的一种交互方式，本节将介绍使用脱机模式进行消息交互时绑定和地址的选择。

24.7.1　场景概述

如果按照交互的实时性来分类，通信协议可以分为两大类：联机式交互和脱机式交互。笔者在本节前文中所介绍的绑定，都属于联机式交互。联机式交互是指通信的双方都必须在同一时刻处于运行状态，并且都是可访问的。而相对的，脱机式交互则没有这方面的要求，脱机式交互允许客户端在服务端不可访问的情况下正常运行，存储所有的消息包，并且在服务端可访问之后再发送消息。最基本的，一个脱机式交互协议需要在本地缓存数据，异步地尝试连接服务端并进行发送，并且在服务端不可访问的情况下对数据进行持久化处理。

在 WCF 中，脱机式交互模式通过 MSMQ（微软消息队列）来实现。在标准绑定中，有两个是支持 MSMQ 的，分别为 netMsmqBinding 和 msmqIntegrationBinding。前者使用 MSMQ 协议在 WCF 系统之间进行脱机式交互，而后者则允许 WCF 系统和一个基于 MSMQ 协议的系统进行交互。

24.7.2　MSMQ 协议概述

MSMQ（微软消息队列，Microsoft Message Queue）是在多个不同的应用之间实现相互通信的一种异步传输模式。相互通信的应用可以分布于同一台机器上，也可以分布于相连的网络空间中的任一位置。它的实现原理是：消息的发送者把自己想要发送的信息放入一个容器中（称之为 Message），然后把它保存至一个系统公用空间的消息队列（Message Queue）中；本地或者是异地的消息接收程序再从该队列中取出发给它的消息进行处理。

在消息传递机制中，有两个比较重要的概念。一个是消息，一个是队列。消息是由通信的双方所需要传递的信息，它可以是各式各样的媒体，如文本、声音、图像等。消息最终的理解方式，为消息传递的双方事先商定。这样做的好处是，一是相当于对数据进行了简单的加密，二则采用自己定义的格式可以节省通信的传递量。消息可以含有发送和接收者的标识，这样只有指定的用户才能看到只传递给他的信息和返回是否操作成功的回执。消息也可以含有时间戳，以便于接收方对某些与时间相关的应用进行处理。消息还可以含有到期时间，它表明如果在指定时间内消息还未到达则作废，这主要应用于时间性关联较为紧密的应用。

消息队列是发送和接收消息的公用存储空间，它可以存在于内存中或者是物理文件中。消息可以以两种方式发送，即快递方式（express）和可恢复模式（recoverable）。它们的区别在于，快递方式为了消息的快速传递，把消息放置于内存中，而不放于物理磁盘上，以获取较高的处理能力；可恢复模式在传送过程的每一步骤中，都把消息写入物理磁盘中，以得到较好的故障恢复能力。消息队列可以放置在发送方、接收方所在的机器上，也可以单独放置在另外一台机器上。

正是由于消息队列在放置方式上的灵活性，形成了消息传送机制的可靠性。当保存消息队列的机器发生故障而重新启动以后，以可恢复模式发送的消息可以恢复到故障发生之

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

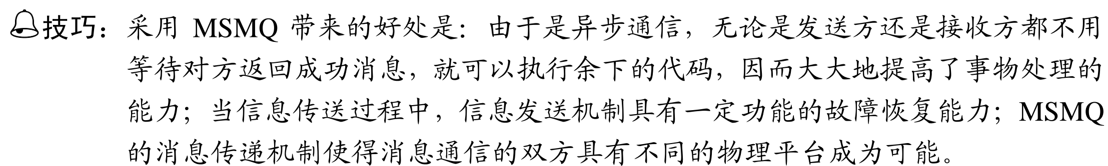
前的状态，而以快递方式发送的消息则丢失了。另一方面，采用消息传递机制，发送方不必要再担心接收方是否启动、是否发生故障等非必要因素，只要消息成功发送出去，就可以认为处理完成。而实际上对方可能未曾开机，或者实际完成交易时可能已经是第二天了。

采用 MSMQ 服务，用户可以创建下面几种队列。

- 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）。
- 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）。

系统本身也可以生成队列，系统可以生成下面几种队列。

- 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。
- 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。


技巧：采用 MSMQ 带来的好处是：由于是异步通信，无论是发送方还是接收方都不用等待对方返回成功消息，就可以执行余下的代码，因而大大地提高了事物处理的能力；当信息传送过程中，信息发送机制具有一定功能的故障恢复能力；MSMQ 的消息传递机制使得消息通信的双方具有不同的物理平台成为可能。

24.7.3 使用 netMsmqBinding

如果选择脱机式交互模式，那 netMsmqBinding 是较为常用的一种绑定，netMsmqBinding 使用 MSMQ 进行脱机式消息交互。本质上来说，使用该绑定的客户端发送消息到某特定队列之中，而服务端则从该特定队列中读取消息，从而实现了脱机的交互。和其他以 net 为前缀的绑定一样，netMsmqBinding 只支持在 WCF 系统之间进行交互。同时，由于 MSMQ 的交互特点，netMsmqBinding 只支持单程（OneWay）的服务访问。也就是说，所有使用 netMsmqBinding 的服务契约都必须定义 IsOneWay 属性，并且设置为 true。

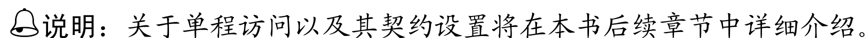

说明：关于单程访问以及其契约设置将在本书后续章节中详细介绍。

表 24.11 列出了 netMsmqBinding 的可配置属性。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

表 24.11 netMsmqBinding的属性

属 性 名	描 述	默 认 值
closeTimeout	等待连接的超时时间	00:01:00
customDeadLetterQueue	应用程序死信队列的位置。死信指的是那些已经过期或者发送失败的消息	N/A
deadLetterQueue	死信队列的类型，可选项有 None、System 和 Custom	None
Durable	设置该队列是持久的还是易失的	true
exactlyOnce	配置是否确保只发送一次	true
maxBufferPoolSize	内存中用于对传入消息进行缓冲的最大字节数	524888
maxReceivedMessageSize	定义在采用此绑定配置的通道上可以接收的消息的最大消息大小（字节），包括消息标头	65536
maxRetryCycles	尝试发送消息的次数，一旦超过，则认为发送失败	2
queueTransferProtocol	设置队列传输协议，可选择项包括 Native、Srmp 和 SrmpSecure。Native 指的是 MSMQ 协议，Srmp 指的是 SOAP 可靠消息传输协议	Native
name	绑定的名字	N/A
openTimeout	在传输引发异常之前可用于打开连接的时间间隔	00:01:00
readerQuotas	可由使用此绑定配置的终节点处理的消息的复杂性约束	N/A
receiveErrorHandling	设置如何处理坏死消息，可选项包括：Drop、Fault、Move 和 Reject	Fault
receiveTimeout	在传输引发异常之前可用于完成读取操作的时间间隔	00:10:00
retryCycleDelay	在两次尝试发送之间的等待时间	00:10:00
security	指定与采用此绑定配置的服务一起使用的安全类型	N/A
sendTimeout	指定为完成发送操作提供的时间间隔	00:01:00
timeToLive	消息生存的时间，一旦超时，则消息将被视为过期并放入死信队列中	01：00：00
useActiveDirectory	设置绑定是否需要通过 ActiveDirectory 来分析机器名	false
useMsmqTracing	设置是否是要跟踪 MSMQ 队列，一旦设置为 true，每当消息到达队列或者离开队列，一个记录消息会被发送到报告队列上	false
useSourceJournal	设置是否需要保存消息副本到日志队列上	false

24.7.4 netMsmqBinding 的地址和配置

当使用 netMsmqBinding 时，终节点的地址将以如下的形式出现。

```
net.msmq://[Hostname]/public(private)/[QueueName]
```

netMsmqBinding 的地址稍微有点特别，首先其地址协议部分使用的是 net.msmq，紧跟着是主机名，主机名之后却没有端口号，这是因为 netMsmqBinding 默认使用 1801 端口，并且不可配置。在主机名之后是一个特殊部分，用以说明该绑定使用的队列是公用的还是私用的，对应的选项是 public 和 private。如果忽略该设置，则 netMsmqBinding 会默认使用公用队列。最后是使用的服务名，在 netMsmqBinding 中也是队列名。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

示例代码 24-19 展示了 netMsmqBinding 的服务端配置。

示例代码 24-19

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <client>
      <!--使用 netMsmqBinding-->
      <endpoint address ="net.msmq://ProfessionalWCFServer/private/
      HelloWorldClient"<!--地址-->
                binding="netMsmqBinding"<!--绑定-->
                bindingConfiguration="myBinding"<!--绑定设置名-->
                contract="HelloWorldService.IClientService"<!--契约-->
                name="NetMsmqHelloWorldClient"><!--终结点名-->
      </endpoint>
    </client>
    <services>
      <service name="HelloWorldService.Service">
        <!--使用 netMsmqBinding-->
        <endpoint address ="net.msmq://ProfessionalWCFServer/private/
        HelloWorldServer" <!--地址-->
                  binding="netMsmqBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IServerService"><!--契约-->
        </endpoint>
      </service>
    </services>
    <bindings>
      <!--设置 netMsmqBinding 的属性-->
      <netMsmqBinding>
        <binding name="myBinding"<!--设置 name 属性-->
                 closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
                 openTimeout="00:00:20"<!--设置 openTimeout 属性-->
                 receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
                 sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
                 deadLetterQueue="net.msmq://ProfessionalWCFServer/
                 DeadLetter"<!--设置 deadLetterQueue 属性-->
                 durable="true"<!--设置 durable 属性-->
                 exactlyOnce="true"<!--设置 exactlyOnce 属性-->
                 maxMessageSize="1000"<!--设置 maxMessageSize 属性-->
                 maxRetries="11"<!--设置 maxRetries 属性-->
                 maxRetryCycles="12"<!--设置 maxRetryCycles 属性-->
                 poisonMessageHandling="Disabled"<!--设置
                 poisonMessageHandling 属性-->
                 rejectAfterLastRetry="false"<!--设置 rejectAfterLastRetry
                 属性-->
                 retryCycleDelay="00:05:55"<!--设置 retryCycleDelay 属性-->
                 timeToLive="00:11:11"<!--设置 timeToLive 属性-->
                 sourceJournal="true"<!--设置 sourceJournal 属性-->
                 useMsmqTracing="true"<!--设置 useMsmqTracing 属性-->
                 useActiveDirectory="true"><!--设置 useActiveDirectory
                 属性-->
          <security>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
            <message clientCredentialType="Windows" />
          </security>
        </binding>
      </netMsmqBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

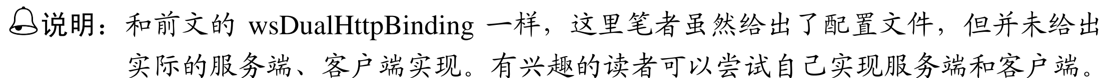
示例代码 24-20 展示了 netMsmqBinding 的客户端配置。

示例代码 24-20

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <client>
        <!--使用 netMsmqBinding-->
        <endpoint address ="net.msmq://ProfessionalWCFServer/private/
        HelloWorldServer" <!--地址-->
                  binding="netMsmqBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IServerService"><!--契约-->
        </endpoint>
    </client>
    <services>
      <service name="HelloWorldService.Client">
        <!--使用 netMsmqBinding-->
        <endpoint address ="net.msmq://ProfessionalWCFServer/private/
        HelloWorldClient"<!--地址-->
                  binding="netMsmqBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IClientService"><!--契约-->
        </endpoint>
      </service>
    </services>
    <bindings>
      <!--设置 netMsmqBinding 的属性-->
      <netMsmqBinding>
        <binding name="myBinding"<!--设置 name 属性-->
                 closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
                 openTimeout="00:00:20"<!--设置 openTimeout 属性-->
                 receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
                 sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
                 deadLetterQueue="net.msmq://ProfessionalWCFServer/
                 DeadLetter"<!--设置 deadLetterQueue 属性-->
                 durable="true"<!--设置 durable 属性-->
                 exactlyOnce="true"<!--设置 exactlyOnce 属性-->
                 maxMessageSize="1000"<!--设置 maxMessageSize 属性-->
                 maxRetries="11"<!--设置 maxRetries 属性-->
                 maxRetryCycles="12"<!--设置 maxRetryCycles 属性-->
                 poisonMessageHandling="Disabled"<!--设置
                 poisonMessageHandling 属性-->
                 rejectAfterLastRetry="false"<!--设置 rejectAfterLastRetry
                 属性-->
                 retryCycleDelay="00:05:55"<!--设置 retryCycleDelay 属性-->
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
                timeToLive="00:11:11"<!--设置 timeToLive 属性-->
                sourceJournal="true"<!--设置 sourceJournal 属性-->
                useMsmqTracing="true"<!--设置 useMsmqTracing 属性-->
                useActiveDirectory="true"><!--设置 useActiveDirectory
                属性-->
          <security>
            <message clientCredentialType="Windows" />
          </security>
        </binding>
      </netMsmqBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```



说明：和前文的 wsDualHttpBinding 一样，这里笔者虽然给出了配置文件，但并未给出实际的服务端、客户端实现。有兴趣的读者可以尝试自己实现服务端和客户端。

24.7.5　netMsmqBinding 特点总结

由于属于脱机式交互，netMsmqBinding 和一般联机式的绑定有很大的差别。其特点如下：

- 支持单程传输模式。
- 不支持请求-响应传输模式。
- 不支持双工传输模式。
- 性能在标准绑定中较好。
- 支持 WS-事务协议。
- 支持传输层消息安全。
- 支持跨主机交互。
- 仅限于 WCF 系统交互。
- 不支持可靠会话传输。

24.7.6　使用 msmqIntegrationBinding

和 netMsmqBinding 不同，msmqIntegrationBinding 被用于和现成的 MSMQ 应用系统进行交互。msmqIntegrationBinding 的作用是在 MSMQ 消息和 WCF 消息之间进行映射，这是通过 MSmqMessage<T>类型实现的。这里的泛型参数就代表了 MSMQ 消息类型。表 24.12 列出了 msmqIntegrationBinding 的可配置属性。

表 24.12　msmqIntegrationBinding的属性

属　性　名	描　　述	默　认　值
closeTimeout	等待连接的超时时间	00:01:00
customDeadLetterQueue	应用程序死信队列的位置。死信指的是那些已经过期或者发送失败的消息	N/A
deadLetterQueue	死信队列的类型，可选项有 None、System 和 Custom	None
Durable	设置该队列是持久的还是易失的	true

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

续表

属　性　名	描　　述	默　认　值
exactlyOnce	配置是否确保只发送一次	true
maxReceivedMessageSize	定义在采用此绑定配置的通道上可以接收的消息的最大消息大小（字节）	65536
maxRetryCycles	尝试发送消息的次数，一旦超过，则认为发送失败	2
name	绑定的名字	N/A
openTimeout	在传输引发异常之前可用于打开连接的时间间隔	00:01:00
readerQuotas	可由使用此绑定配置的终节点处理的消息的复杂性约束	N/A
receiveErrorHandling	设置如何处理坏死消息，可选项包括：Drop，Fault，Move 和 Reject	Fault
receiveTimeout	在传输引发异常之前可用于完成读取操作的时间间隔	00:10:00
retryCycleDelay	在两次尝试发送之间的等待时间	00:10:00
security	指定与采用此绑定配置的服务一起使用的安全类型	N/A
sendTimeout	指定为完成发送操作提供的时间间隔	00:01:00
serializationFormat	消息体使用的序列化方式。可选项包括 XML、Binary、ActiveX、ByteArray 和 Stream	XML
timeToLive	消息生存的时间，一旦超时，则消息将被视为过期并放入死信队列中	01:00:00
useMsmqTracing	设置是否是要跟踪 MSMQ 队列。一旦设置为 true，每当消息到达队列或者离开队列，一个记录消息会被发送到报告队列上	false
useSourceJournal	设置是否需要保存消息副本到日志队列上	false

24.7.7　msmqIntegrationBinding 的地址和配置

当使用 msmqIntegrationBinding 时，终节点的地址将以如下的形式出现。

```
msmq.formatname://{msmq format name}
```

和 netMsmqBinding 一样，msmqIntegrationBinding 的地址不能配置端口，并且使用默认 MSMQ 端口：1801。示例代码 24-21 展示了 msmqIntegrationBinding 的服务端配置。

示例代码 24-21

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <services>
      <service name="HelloWorldService.Service">
        <!--使用 msmqIntegrationBinding-->
        <endpoint address ="net.formatname:DIRECT=OS:.\private$\Service"
        <!--地址-->
                binding="msmqIntegrationBinding"<!--绑定-->
                bindingConfiguration="myBinding"<!--绑定设置名-->
                contract="HelloWorldService.IService"><!--契约-->
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
        </endpoint>
      </service>
    </services>
    <bindings>
      <!--设置 msmqIntegrationBinding 的属性-->
      <msmqIntegrationBinding>
        <binding name="myBinding"<!--设置 name 属性-->
                 closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
                 openTimeout="00:00:20"<!--设置 openTimeout 属性-->
                 receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
                 sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
                 deadLetterQueue="net.msmq://ProfessionalWCFServer/
                 DeadLetter"<!--设置 deadLetterQueue 属性-->
                 durable="true"<!--设置 durable 属性-->
                 exactlyOnce="true"<!--设置 exactlyOnce 属性-->
                 maxReceivedMessageSize="1000"<!--设置
                 maxReceivedMessageSize 属性-->
                 maxImmediateRetries="11"<!--设置 maxImmediateRetries 属性-->
                 maxRetryCycles="12"<!--设置 maxRetryCycles 属性-->
                 poisonMessageHandling="Disabled"<!--设置
                 poisonMessageHandling 属性-->
                 rejectAfterLastRetry="false"<!--设置 rejectAfterLastRetry
                 属性-->
                 retryCycleDelay="00:05:55"<!--设置 retryCycleDelay 属性-->
                 timeToLive="00:11:11"<!--设置 timeToLive 属性-->
                 useSourceJournal="true"<!--设置 useSourceJournal 属性-->
                 useMsmqTracing="true"<!--设置 useMsmqTracing 属性-->
                 serializationFormat="Binary"><!--设置 serializationFormat
                 属性-->
          <security mode="None" />
        </binding>
      </msmqIntegrationBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

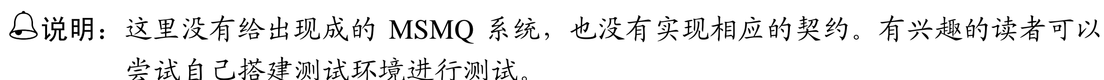
示例代码 24-22 展示了 msmqIntegrationBinding 的客户端配置。

示例代码 24-22

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>    <compilation debug="true" /><!--允许 debug-->
  </system.web>
  <system.serviceModel>
    <client>
        <!--使用 msmqIntegrationBinding-->
        <endpoint address ="net.formatname:DIRECT=OS:.\private$\Service"
        <!--地址-->
                  binding="msmqIntegrationBinding"<!--绑定-->
                  bindingConfiguration="myBinding"<!--绑定设置名-->
                  contract="HelloWorldService.IService"><!--契约-->
        </endpoint>
    </client>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    <bindings>
      <!--设置 msmqIntegrationBinding 的属性-->
      <msmqIntegrationBinding>
        <binding name="myBinding"<!--设置 name 属性-->
                 closeTimeout="00:00:10"<!--设置 closeTimeout 属性-->
                 openTimeout="00:00:20"<!--设置 openTimeout 属性-->
                 receiveTimeout="00:00:30"<!--设置 receiveTimeout 属性-->
                 sendTimeout="00:00:40"<!--设置 sendTimeout 属性-->
                 deadLetterQueue="net.msmq://ProfessionalWCFServer/
                 DeadLetter"<!--设置 deadLetterQueue 属性-->
                 durable="true"<!--设置 durable 属性-->
                 exactlyOnce="true"<!--设置 exactlyOnce 属性-->
                 maxReceivedMessageSize="1000"<!--设置
                 maxReceivedMessageSize 属性-->
                 maxImmediateRetries="11"<!--设置 maxImmediateRetries 属性-->
                 maxRetryCycles="12"<!--设置 maxRetryCycles 属性-->
                 poisonMessageHandling="Disabled"<!--设置
                 poisonMessageHandling 属性-->
                 rejectAfterLastRetry="false"<!--设置 rejectAfterLastRetry
                 属性-->
                 retryCycleDelay="00:05:55"<!--设置 retryCycleDelay 属性-->
                 timeToLive="00:11:11"<!--设置 timeToLive 属性-->
                 useSourceJournal="true"<!--设置 useSourceJournal 属性-->
                 useMsmqTracing="true"<!--设置 useMsmqTracing 属性-->
                 serializationFormat="Binary"><!--设置 serializationFormat
                 属性-->
          <security mode="None" />
        </binding>
      </msmqIntegrationBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```



说明：这里没有给出现成的 MSMQ 系统，也没有实现相应的契约。有兴趣的读者可以尝试自己搭建测试环境进行测试。

24.7.8 msmqIntegrationBinding 特点总结

msmqIntegrationBinding 的最大功能在于集成现有的 MSMQ 系统，其具体特点如下所示。

- 支持单程传输模式；
- 不支持请求-响应传输模式；
- 不支持双工传输模式；
- 性能在标准绑定中较好；
- 不支持 WS-事务协议；
- 支持传输层消息安全；
- 支持和 MSMQ 系统机交互；
- 支持跨主机交互；

互动出版网
China-Pub.com
专业图书网上第一专营店
www.china-pub.com
独家提供样章

- 不支持可靠会话传输。

24.8　小　　结

本章详细地介绍了 WCF 中的通道模型和绑定。作为终节点的三要素之一，绑定在 WCF 中起到了基石的作用。可以说，正是因为绑定和通道模型的工作，才使得 WCF 的编程工作相对其他分布式模型而言轻松很多。而基本上 WCF 中所有的功能，又最终依靠通道模型和绑定来得以实现。

在实际的工作中，使用到绑定和通道模型的可能性较小。但了解其机制，对于扩展 WCF 系统、调试 WCF 程序都有很大的帮助。笔者建议读者在使用 WCF 前，花一些时间来了解 WCF 的通道和绑定实现。