

[老老实实学WCF] 第十篇 消息通信模式(下) 双工

老老实实学WCF

第十篇 消息通信模式(下) 双工

在前一篇的学习中，我们了解了单向和请求/应答这两种消息通信模式。我们知道可以通过配置操作协定的IsOneWay属性来改变模式。在这一篇中我们来研究双工这种消息通信模式。

在一定程度上说，双工模式并不是与前面两种模式相提并论的模式，双工模式的配置方法同前两者不同，而且双工模式也是基于前面两种模式之上的。

在双工模式下，服务端和客户端都可以独立地调用对方，谁都不用等待谁的答复，同样也不期待对方答复，因为如果期待答复，就变成请求/应答模式了。也就是说双方的调用都是单向调用，即我调你了，你不用回复我，你什么时候想回复我的时候呢你再调我，我就知道了，我是不会等着你的回复的。这样调用双方就会有很好的异步体验，我想调的时候就调，然后我就去干别的，什么时候调用完成了，你可以通过回调来通知我，我再决定下一步的动作，谁都不等谁。

因为在服务模型中，调用总是由客户端首先发起的，所以一般说的调用，都是客户端的行为，在双工模式下，服务器也可以调用客户端，我们就把它叫做回调，实际上这个调用和客户端的调用没什么本质区别，在回调的时候，服务端变成了客户端，客户端相当于在提供服务了，只不过总是客户端调用在前，我们就把服务端对客户端的调用叫做回调了。

1. 建立双工通信的条件

要建立双工通信，必须使用支持双工通信的绑定，比如wsHttpBinding是不支持的，必须采用wsDualHttpBinding才行，他会建立两条绑定来实现互相调用，因此我们首先要注意的是选择正确的绑定。

要建立双工通信，必须使用会话，即将ServiceContract的SessionMode配置为SessionMode.Required。

2. 如何配置双工通信

为了更好地理解这个问题，我们先考虑单工的情形，在单工模式下(单向和请求应答)，调用总是由客户端发起的，服务端可以回应也可以不回应。我们是怎么配置实现这个的呢？我们首先让两端共享服务协定接口，这样客户端才知道怎样调用服务端，然后把服务实现类写在服务端，这样服务端才能在收到请求的时候实例化这个类并执行服务的操作逻辑。也就是说，客户端要想调服务端，客户端必须拥有服务协定接口，而服务端必须拥有服务协定接口以及实现接口的服务类，在运行时服务端还要有服务类的实例。这些是必备条件。

再说回双工，双工让服务端可以调客户端，那么道理同单工，必备条件也要有才行，只是他们的地位互换了。也就是说服务端必须拥有协定接口，客户端必须拥有协定接口以及实现接口的类，在运行时客户端还要有类的实例。一般情况下我们管服务端的定义的协定接口叫做服务协定，协定接口实现类叫做服务类，这回换到客户端，我们管它叫回调接口，回调类，其实作用是一样的。

也就是说在定义上，我们需要在两边都定义两个协定接口，一个服务协定接口，一个回调协定接口，把服务协定接口的实现类写在服务端，把回调协定接口的实现类写在客户端。

通过配置服务协定的ServiceContract的CallbackContract属性来指定回调的时候使用的回调协定，考虑下面的代码：

```
[csharp]
1. [ServiceContract(SessionMode = SessionMode.Required,
2.     CallbackContract = typeof(IHelloWCFCallback))]
3. public interface IHelloWCF
4. {
5.     [OperationContract(IsOneWay = true)]
6.     void HelloWCF();
7. }
8.
9. public interface IHelloWCFCallback
10. {
11.     [OperationContract(IsOneWay = true)]
12.     void Callback(string msg);
13. }
```

我们定义了一个IHHelloWCFCallback的回调协定接口，这个接口的实现是写在客户端的，同时指定了IHHelloWCF服务协定接口的回调协定为该回调协定接口，这样服务协定就知道怎样进行回调了。

同样在客户端要写出回调协定接口及其实现，如果我们使用了svcutil.exe或添加服务引用，系统会为我们自动填写回调协定接口，但是不会为我们写实现类，毕竟她不知道我们的客户端在接受回调的时候执行怎样的逻辑。我们得自己编写，一个回调协定接口的实现类看上去是这样的：

```
[csharp]
1. public class HelloWCFCallback : Services.IHelloWCFCallback
2. {
3.     public void Callback(string msg)
4.     {
5.         Console.WriteLine(msg);
6.     }
7. }
```

其实和实现服务协定没什么不同。

前面提过绑定需要支持双工，因此我们需要选择一个支持双工的绑定，我们采用wsDualHttpBinding。

```
[html]
1. <endpoint address="" binding="wsDualHttpBinding" contract="LearnWCF.IHelloWCF"/>
```

以上双工通信的配置就完成了，我们还需要添加一些代码来对双工的运行时进行实现。

3. 双工的运行时实现

现在准备工作已经完成，那么在运行时需要什么呢？需要通道和实例。回想单工通信，我们通过代理类建立了一个基于服务协定的通道到服务端，然后我们在这个通道上调用服务协定方法，在调用方法的时候，服务端实例上下文会生成并未我们new一个服务类的实例帮我们执行操作，当然这一步是服务端的系统自动完成的，我们不需要做特别的配置，直接调用就行了。

现在反过来服务端要调客户端了，服务端可没有代理类，那么通道何来呢？客户端这边也没有实例上下文和服务类实例，这边只是一个简单的控制台应用程序，我们怎么能指望客户端为我们自动生成这些呢？

所以我们得自己来。

首先在客户端我们要自己先实例化一个服务类的实例，然后用这个实例作为参数去创建一个实例上下文实例，这样运行时的服务实例就有了，然后把这个实例上下文对象作为参数传给代理类的构造函数来初始化代理类，在这里代理类帮了我们大忙，他会检测到服务端元数据中服务协定使用双工，他就会为我们准备好双工通道，当然前提是他会跟我们要实例上下文对象。这样我们通过代理类调用服务操作，双工通道就会建立了。看下面的代码(这是在客户端的Program.cs中)：

```
[csharp]
1. //建立回调服务对象
2. HelloWCFCallback callbackObject = new HelloWCFCallback();
3. //建立实例上下文对象
4. InstanceContext clientContext = new InstanceContext(callbackObject);
5. //用建立好的实例上下文对象初始化代理类对象
6. Services.HelloWCClient client = new Services.HelloWCClient(clientContext);
```

接下来轮到服务端，服务端既然受到的是一个支持双工的连接，他就可以在利用操作上下文对象来得到和打开回调的通道，回调通道使用回调协定声明的(正如通道使用服务协定声明一样)。然后再回调通道上调用回调操作就可以了：

```
[csharp]
1. string msg = "Hello From Service! Time" + DateTime.Now.ToLongTimeString();
2. //获得回调通道
```

```
3. IHelloWCFCallback callbackChannel = OperationContext.Current.GetCallbackChannel<IHelloWCFCallback>();  
4. //调用回调操作  
5. callbackChannel.Callback(msg);
```

这样，双工的运行时就实现了。如果你对上面提到的有些迷糊，赶紧翻回第四篇和第五篇温习一下有关通信的基础知识。

4. 双工通信实例

我们通过一个完整的例子来理解一下双工通信的过程。

我用IIS作为服务端宿主，客户端用一个控制台应用程序。我们来实现一个比较简单的双工通信，客户端先向服务端发起一个调用，然后去干别的，服务端等五秒后回调客户端的回调方法。

你可能对前几篇讲的知识印象模糊了，我们这次从头做一次。当然，温习一下前面几篇的内容是最好的。

(1) 建立SVC文件。

首先我们先建立IIS宿主，建立一个HelloWCFService.svc的文件保存在IIS应用程序的根路径下。我的IIS应用程序的路径是

```
[plain]
1. http://localhost/IISService
```

因此这个文件的地址就变成了：

```
[plain]
1. http://localhost/IISService/HelloWCFService.svc
```

这个文件的内容只有一行：

```
[html]
1. <%@ServiceHost language=c# Debug="true" Service="LearnWCF.HelloWCFService"%>
```

这行指令表示这是个WCF服务，服务的实现类是LearnWCF.HelloWCFService，注意这里命名空间要写全，名字你可以随意起。

(2) 建立服务代码文件。

代码文件是名为HelloWCFService.cs的文件，其实名字可以随意起，但是要保存在IIS应用程序根目录下的App_Code目录下(或者Bin目录也可以)。

代码文件内容如下：

```
[csharp]
1. using System;
2. using System.ServiceModel;
```

```
3.
4. namespace LearnWCF
5. {
6.     [ServiceContract(SessionMode = SessionMode.Required,
7.         CallbackContract = typeof(IHelloWCFCallback))]
8.     public interface IHelloWCF
9.     {
10.         [OperationContract(IsOneWay = true)]
11.         void HelloWCF();
12.     }
13.
14.     public interface IHelloWCFCallback
15.     {
16.         [OperationContract(IsOneWay = true)]
17.         void Callback(string msg);
18.     }
19.
20.     public class HelloWCFService : IHelloWCF
21.     {
22.         private int _Counter;
23.         public void HelloWCF()
24.         {
25.             System.Threading.Thread.Sleep(5000);
26.             string msg = "Hello From Service! Time" + DateTime.Now.ToLongTimeString();
27.             //获得回调通道
28.             IHelloWCFCallback callbackChannel = OperationContext.Current.GetCallbackChannel<IHelloWCFCallback>();
29.             //调用回调操作
30.             callbackChannel.Callback(msg);
31.         }
32.     }
33.
34. }
```

注意几个要点。服务协定IHelloWCF的ServiceContract属性的两个设置一个是SessionMode为Required，表示必须使用会话，另一个是指定了回调协定。同时我们能看到，把回调协定接口也定义了出来。服务操作HelloWCF在受到调用后先休眠5秒钟，然后从操作上下文获得回调通道，然后调用通道上的回调操作，把字符串Hello From Service和调用时间传递给了客户端。

(3) 编写配置文件。

编写Web.Config文件并保存在IIS应用程序的根目录下(跟svc文件放在一起)内容如下：

```
[html]
1. <configuration>
2.   <system.serviceModel>
3.     <services>
4.       <service name="LearnWCF.HelloWCFService" behaviorConfiguration="metadataExchange">
5.         <endpoint address="" binding="wsDualHttpBinding" contract="LearnWCF.IHelloWCF"/>
6.         <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
7.       </service>
8.     </services>
9.     <behaviors>
10.      <serviceBehaviors>
11.        <behavior name="metadataExchange">
12.          <serviceMetadata httpGetEnabled="true" />
13.        </behavior>
14.      </serviceBehaviors>
15.    </behaviors>
16.  </system.serviceModel>
17. </configuration>
```

注意看绑定，配置成了支持双工的wsDualHttpBinding。

服务端的部分就写好了。

(4) 建立客户端应用程序

建立一个控制台应用程序，命名为ConsoleClient。

(5) 添加服务引用

在引用上右击，选择添加服务引用，并在地址中输入，并点击前往

```
[plain]
1. http://localhost/IISService/HelloWCFService.svc
```

在下面的命名空间中为代理类指定一个新的命名空间Services。点确定

此时系统为我们自动添加了App.Config和代理类文件，点击解决方案浏览器上方的查看所有文件，逐层展开服务引用，最后打开reference.cs看看有什么变化
以下的代码是系统生成的代理类代码，不是我们输入的

```
[csharp]
1.  //-----
2.  // <auto-generated>
3.  //     此代码由工具生成。
4.  //     运行时版本:4.0.30319.261
5.  //
6.  //     对此文件的更改可能会导致不正确的行为，并且如果
7.  //     重新生成代码，这些更改将会丢失。
8.  // </auto-generated>
9.  //-----
10.
11. namespace ConsoleClient.Services {
12.
13.
14.     [System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "4.0.0.0")]
15.     [System.ServiceModel.ServiceContractAttribute(ConfigurationName="Services.IHelloWCF", Callback
16.     public interface IHelloWCF {
17.
18.         [System.ServiceModel.OperationContractAttribute(IsOneWay=true, Action="http://tempuri.org/HelloWCF")]
19.         void HelloWCF();
20.     }
21.
22.     [System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "4.0.0.0")]
23.     public interface IHelloWCFCallback {
24.
25.         [System.ServiceModel.OperationContractAttribute(IsOneWay=true, Action="http://tempuri.org/HelloWCFCallback")]
26.         void Callback(string msg);
27.     }
28.
29.     [System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "4.0.0.0")]
30.     public interface IHelloWCFChannel : ConsoleClient.Services.IHelloWCF, System.ServiceModel.ICommunicationChannel
31.     {
32.
33.         [System.Diagnostics.DebuggerStepThroughAttribute()]
34.         [System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "4.0.0.0")]
35.         public partial class HelloWCFClient : System.ServiceModel.DuplexClientBase<ConsoleClient.Services.IHelloWCFChannel>
```

```
36.
37.     public HelloWCFClient(System.ServiceModel.InstanceContext callbackInstance) :
38.         base(callbackInstance) {
39.     }
40.
41.     public HelloWCFClient(System.ServiceModel.InstanceContext callbackInstance, string endp
42.         base(callbackInstance, endpointConfigurationName) {
43.     }
44.
45.     public HelloWCFClient(System.ServiceModel.InstanceContext callbackInstance, string endp
46.         base(callbackInstance, endpointConfigurationName, remoteAddress) {
47.     }
48.
49.     public HelloWCFClient(System.ServiceModel.InstanceContext callbackInstance, string endp
50.         base(callbackInstance, endpointConfigurationName, remoteAddress) {
51.     }
52.
53.     public HelloWCFClient(System.ServiceModel.InstanceContext callbackInstance, System.Servi
54.         base(callbackInstance, binding, remoteAddress) {
55.     }
56.
57.     public void HelloWCF() {
58.         base.Channel.HelloWCF();
59.     }
60. }
61. }
```

我们可以看到，代理类为我们自动生成了回调协定IHelloWCFCallback，而且代理类HelloWCFClient的这些构造函数也不一样了，需要我们提供InstanceContext实例了，而且继承的类也不再是ClientBase<>，而是DuplexClientBase<>了，这就是代理发现我们用双工通信了，所以改变了代理类所继承的类为支持双工通信的基类。

好，关掉它，我们继续完善客户端

(6) 编写客户端代码

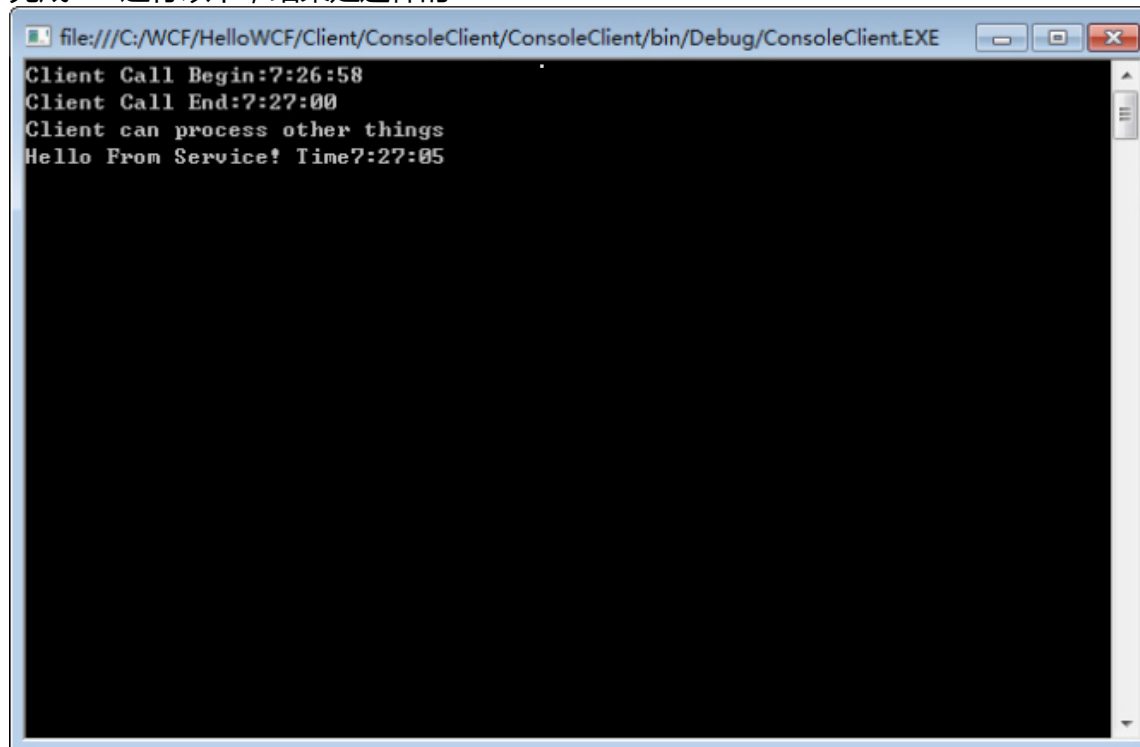
我们还没有在客户端实现回调协定，首先要实现他，还要为运行时添加调用代码，Program.cs的代码如下：

[csharp]

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5.
6. using System.ServiceModel;
7.
8. namespace ConsoleClient
9. {
10.     class Program
11.     {
12.         static void Main(string[] args)
13.         {
14.             //建立回调服务对象
15.             HelloWCFCallback callbackObject = new HelloWCFCallback();
16.             //建立实例上下文对象
17.             InstanceContext clientContext = new InstanceContext(callbackObject);
18.             //用建立好的实例上下文对象初始化代理类对象
19.             Services.HelloWCFClient client = new Services.HelloWCFClient(clientContext);
20.             Console.WriteLine("Client Call Begin:"+DateTime.Now.ToLongTimeString());
21.             client.HelloWCF();
22.             Console.WriteLine("Client Call End:"+DateTime.Now.ToLongTimeString());
23.             Console.WriteLine("Client can process other things");
24.             Console.ReadLine();
25.         }
26.     }
27.
28.     public class HelloWCFCallback : Services.IHelloWCFCallback
29.     {
30.         public void Callback(string msg)
31.         {
32.             Console.WriteLine(msg);
33.         }
34.     }
35. }
```

首先在下面我们实现了回调协定接口，逻辑就是把服务端传过来的参数输出。然后我们建立服务类的对象以及实例上下文对象，接着构造代理类对象。接下来就是调用服务操作了。我们在这里记录了时间便于观察顺序。注意看，这里客户端没有等待服务端的任何返回，也没有任何的输出的动作。

完成 F5 运行以下，结果是这样的：



```
file:///C:/WCF/HelloWCF/Client/ConsoleClient/ConsoleClient/bin/Debug/ConsoleClient.EXE
Client Call Begin:7:26:58
Client Call End:7:27:00
Client can process other things
Hello From Service! Time7:27:05
```

我们参照源代码可以看出，客户端就调用了服务端一次，耗时2秒，然后客户端就可以去做别的了。在客户端发起调用5秒以后，服务端向客户端执行了一次调用，也就是客户端调用的服务操作休眠了5秒以后执行了回调。最后一条输出，是服务端主动调用的客户端的回调服务方法输出的。

5. 总结