[老老实实学WCF] 第九篇 消息通信模式(上) 请求应答与单向

老老实实学WCF

第九篇 消息通信模式(上) 请求应答与单向

通过前两篇的学习，我们了解了服务模型的一些特性如会话和实例化，今天我们来进一步学习服务模型的另一个重要特性：消息通信模式。

WCF的服务端与客户端在通信时有三种模式：单向模式、请求/应答模式和双工模式。

如果选用了单向模式，调用方在向被调用方进行了调用后不期待任何回应，被调用方在执行完调用后不给调用方任何反馈。如客户端通过单向模式调用了一个服务端的操作后，就去干别的了，不会等待服务端给他任何响应，他也无从得知调用是否成功，甚至连发生了错误也全然不知。这种模式的特点是，客户端在调用操作后立即返回，从客户端角度看，用户操作的响应是非常快的，只是客户端无法得知调用结果。

如果选用了请求/应答模式，客户端向服务端发出调用后会一直等待服务端的回复，服务端在执行完操作后会把结果返回给客户端，即使服务操作签名返回值为void，服务端还是会返回一条空消息，告诉客户端调用完成了，客户端在接到返回后才会从调用方法返回继续进行下面的工作。这种模式的特点是客户端总是可以知道服务执行的情况，如果出错，错误也会返回，客户端对服务的执行监控的很好，但是由于在服务返回之前客户端会一直等待，所以如果服务端的服务执行时间比较长的话，客户端这边的用户响应就会很慢，如果客户端对服务的调用与用户界面在同一线程，在用户看来，应用程序就死在那里了。

如果选用了双工模式，客户端和服务端都可以单独的向对方发送消息调用，其实这种模式是在单向模式基础上进行的，两边的调用都是单向调用，但是两边都可以独立的进行，谁也不用等待谁，这种模式比较复杂一些，我们在下一篇再详细的研究。

1. 如何设置消息通信模式。

双工模式有其他的设置方式，单行模式和请求应答模式的设置位置是相同的，就是通过修改操作协定的OperationContract属性的IsOneWay属性来设置。如下面的代码将HelloWCF操作协定设置为了单向模式：

```
[csharp]

1.  [ServiceContract]
2.  public interface IHelloWCF
3.  {
4.      [OperationContract(IsOneWay=true)]
5.      void HelloWCF();
6.  }
```

如果不配置IsOneWay属性，那么他默认是False的，也就是说默认的消息通信模式是请求/应答模式，除非我们显式的指定为单向模式。
下面的代码将HelloWCF操作协定设置为了请求/应答模式：

```
[csharp]

1.  [ServiceContract]
2.  public interface IHelloWCF
3.  {
4.      [OperationContract(IsOneWay=false)]
5.      void HelloWCF();
6.  }
```

由于是默认值，IsOneWay属性不配置也是可以的。

注意，在单向模式下，返回值必须是void，并且不能使用任何Out或Ref的方式返回参数值，也就是说不能以任何手段返回任何值，这是基础结构所不允许的，这样做会导致服务端抛出异常。而在请求/应答模式下，这些都是可以的，即使没有返回值(返回值为void)，返回消息也会照样发送，只不过是个空消息。

2. 两种模式的例子

首先我们看一个请求/应答模式的例子，我用的还是前几篇中使用的IIS宿主服务的例子，如果你忘了，翻回去熟悉一下。
我们让服务端的HelloWCF在返回"Hello WCF!"字符串之前，先磨蹭一会，让他在线程上休眠一会儿。
HelloWCFService.CS的源代码如下：

```
[csharp]

using System;
using System.ServiceModel;

namespace LearnWCF
{
    [ServiceContract]
    public interface IHelloWCF
    {
        [OperationContract(IsOneWay=false)]
        string HelloWCF();
    }

    public class HelloWCFService : IHelloWCF
    {
        private int _Counter;
        public string HelloWCF()
        {
            System.Threading.Thread.Sleep(3000);
            return "Hello WCF!";
        }
    }
}
```

没什么变化，就是让他在线程上Sleep 3秒。
下面是Web.Config文件，也没什么变化：

```
[html]

<configuration>
  <system.serviceModel>
    <services>
      <service name="LearnWCF.HelloWCFService" behaviorConfiguration="metadataExchange">
```

```
        <endpoint address="" binding="wsHttpBinding" contract="LearnWCF.IHelloWCF"/>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="metadataExchange">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

下面是SVC文件，就一行代码，指示了这是个WCF服务，并指定了后台类型：

```
[html]

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

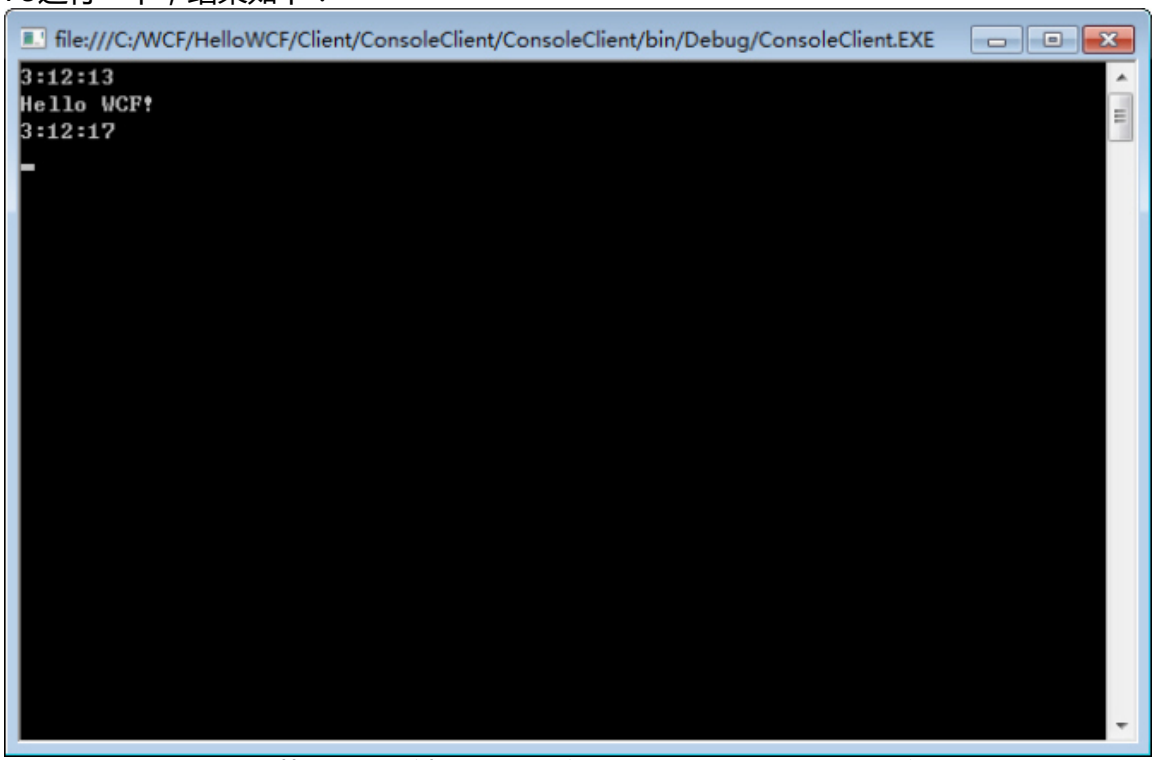
把SVC文件和Web.Config文件放在网站根文件夹下，CS文件放在App_Code文件夹下，启动IIS，服务就寄宿好了，如果你忘记了如何在IIS中寄宿，马上翻回第三篇熟悉一下。

用SVCUTIL.EXE或添加服务引用来生成客户端，为了能看出调用的时间，我们在调用前和调用后分别把时间输出来。Program.cs代码如下：

```
[csharp]

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.ServiceModel;

namespace ConsoleClient
{
    class Program
    {
        static void Main(string[] args)
```

```
13.         {
14.             Services.HelloWCFClient client = new Services.HelloWCFClient();
15.             Console.WriteLine(DateTime.Now.ToLongTimeString());
16.             Console.WriteLine(client.HelloWCF());
17.             Console.WriteLine(DateTime.Now.ToLongTimeString());
18.             Console.ReadLine();
19.         }
20.     }
21. }
```

F5运行一下，结果如下：



可以看到，整个调用花费了4秒钟，除了服务方法中Sleep了3秒，建立会话通讯什么的还用了1秒，在服务端方法Sleep的时候，客户端一直在等待。

接下来，我们再看单向模式的情况，我们修改一下服务协定的代码，让其采用单向模式，但是注意，此时不能有返回值了，必须设为void，服务方法中就是睡3秒，其他的什么也不做。

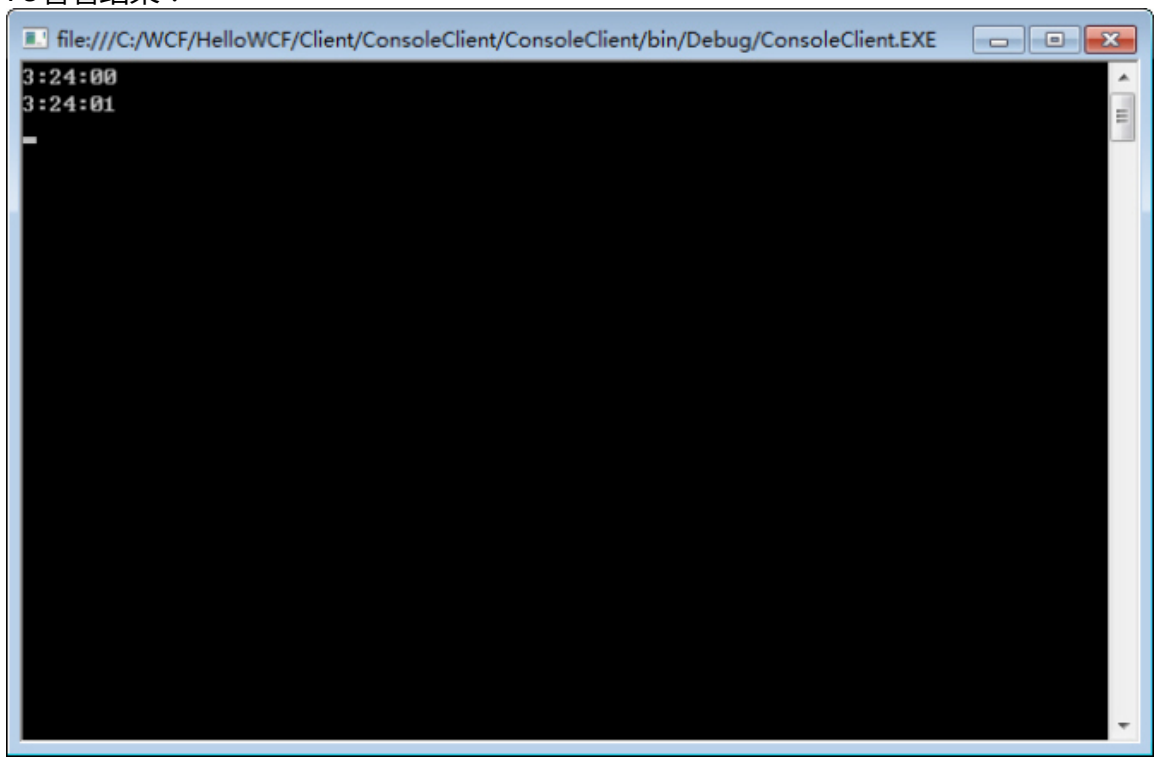
```csharp
using System;
using System.ServiceModel;

namespace LearnWCF
{
    [ServiceContract]
    public interface IHelloWCF
    {
        [OperationContract(IsOneWay=true)]
        void HelloWCF();
    }

    public class HelloWCFService : IHelloWCF
    {
        private int _Counter;
        public void HelloWCF()
        {
            System.Threading.Thread.Sleep(3000);
        }
    }
}
```

客户端需要重新下载一下元数据或更新一下服务引用，因为服务协定的内容变了，客户端Program.CS代码如下：

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.ServiceModel;

```

```
8.  namespace ConsoleClient
9.  {
10.     class Program
11.     {
12.         static void Main(string[] args)
13.         {
14.             Services.HelloWCFClient client = new Services.HelloWCFClient();
15.             Console.WriteLine(DateTime.Now.ToLongTimeString());
16.             client.HelloWCF();
17.             Console.WriteLine(DateTime.Now.ToLongTimeString());
18.             Console.ReadLine();
19.         }
20.     }
21. }
```

F5看看结果：



可以看到只用了1秒，客户端与服务端建立会话后把调用送出就立即返回了，没有等待服务端睡那三秒，当然此时的客户端也根本就不知道服务端在做什么。

注意，请求应答模式是需要会话支持的，必须使用支持会话的绑定，而且服务协定的SessionMode必须至少为Allowed，服务类的ServiceBehavior的InstanceContextMode必须是PerSession，我们在这里没有配置，因为他们是默认的，但是我们必须知道他们需要这样的配置才能支持请求/应答模式。

如果你在试验中遇到了莫名其妙的问题，尝试把客户端服务引用全部删掉重新添加服务引用，因为有的时候更新服务引用不总是那么好用。

3. 总结

通过这一篇的学习，我们了解了消息通讯的两种基本模式，在这个基础上还有更加复杂的双工通讯模式，我们在下一篇中详细研究。