[老老实实学WCF] 第五篇 再探通信--ClientBase

老老实实学WCF
第五篇 再探通信--ClientBase

在上一篇中，我们抛开了服务引用和元数据交换，在客户端中手动添加了元数据代码，并利用通道工厂ChannelFactory<>类创建了通道，实现了和服务端的通信。然而，与服务端通信的编程模型不只一种，今天我们来学习利用另外一个服务类ClientBase<>来完成同样的工作，了解了这个类的使用方法，我们对服务引用中的关键部分就能够理解了。

ClientBase<>类也是一个泛型类，接受服务协定作为泛型参数，与ChannelFactory<>不同的是，这个类是一个基类，即抽象类，是不能实例化成对象直接使用的，我们需要自己写一个类来继承这个类，我们新写的类实例化出来就是客户端代理了，这个对象可以调用基类的一些受保护的方法来实现通信。
ClientBase<>为我们封装的很好，我们只需要写个新类来继承他就可以了，通信的很多部分他都替我们做好了，比如我们不用进行去创建通道和打开通道的操作，直接调用协定方法就可以了。这也是服务引用和其他元数据生成工具(如svcutil)使用这个类来构造客户端代理的原因。

我们一边动手一边学习

1. 建立客户端程序

> 我们依然不用服务引用，完全手写，这次还是建立一个控制台应用程序作为客户端。

2. 添加必要的引用

我们用到的ClientBase<>类在System.ServiceModel程序集中，因此，要添加这个程序集的引用，同时在program.cs中添加using语句

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

这一步应该很熟了，如果有疑问，返回前几篇温习一下。

3. 编写服务协定

因为没有使用服务引用，客户端现在没有任何元数据的信息，我们要来手写。首先把服务协定写进去。这个再熟悉不过了(写在Program类后面)：

```csharp
[ServiceContract]
public interface IHelloWCF
{
    [OperationContract]
    string HelloWCF();
}
```

4. 编写客户端代理类

上面已经提到，我们要自己写一个新类来继承ClientBase<>基类，这样这个新类就是代理类了，同时，为了能够用代理类直接调用服务协定的方法，我们还要让代理类实现服务协定的接口，注意，继承要写在前面，实现接口要写在后面。我们把这个类起名为HelloWCFClient。

```
[csharp]

1.  public class HelloWCFClient : ClientBase<IHelloWCF>, IHelloWCF
2.  {
3.
4.  }
```

提供给ClientBase的泛型参数应该是服务协定，这样他才能帮我们建立正确的通道，同时，代理类也实现服务协定接口。

ClientBase<>有许多的构造函数，接受不同种类的参数来创建代理类对象，其实这些参数都是元数据信息，刚才我们已经通过泛型参数传递给基类服务协定这个元数据了，现在基类还需要绑定和终结点地址这两个元数据才能正确创建连接，所以我们继承的新类应该把这个构造函数给覆载一下接受这两种元数据参数并传递给基类。
下面我们为新建立的代理类添加一个构造函数：

```
[csharp]

1.  public class HelloWCFClient : ClientBase<IHelloWCF>, IHelloWCF
2.  {
3.      public HelloWCFClient(System.ServiceModel.Channels.Binding binding, EndpointAddress remoteAd
4.          : base(binding, remoteAddress)
5.      {
6.
7.      }
8.  }
```

我们看到这个新建的构造函数什么也没做，只是接受了两个参数，一个是绑定，一个是终结点地址，然后直接调用基类(也就是ClientBase<>)的构造函数，把这个两个参数传递了上去。其实工作都是ClientBase<>做的，我们新建的类就是个传话的，要不然怎么叫代理呢，他什么活都不干。

既然我们实现了服务协定接口，当然要实现接口的方法了。下面我们把方法的实现写下来：

```csharp
public class HelloWCFClient : ClientBase<IHelloWCF>, IHelloWCF
{
    public HelloWCFClient(System.ServiceModel.Channels.Binding binding, EndpointAddress remoteAd
        : base(binding, remoteAddress)
    {

    }

    public string HelloWCF()
    {
        return base.Channel.HelloWCF();
    }
}
```

别忘了你的处境，凡人！我们这是在客户端啊，怎么可能有服务协定呢？这个可以有，但是这个实现不是我们在做，而是要和服务端通信让服务端做，这里可以看到代理鲜明的特点了，代理类虽然实现了服务协定的方法，但是在方法中，他调用了基类(就是ClientBase<>)上的通道，并通过通道调用了了协定方法。此时，ClientBase<>已经为我们建立好与服务端的通道了，而且是用服务协定建立的，我们当然可以在通道上调用服务协定的方法。所以调用代理类对象的HelloWCF()的过程是代理类委托基类在已经建立好的服务协定通道上调用协定方法，并从服务端获得返回值，然后再返回给代理类对象的调用者。狗腿啊狗腿。

5. 编写程序主体

代理类已经写完了，我们现在开始写程序的主体，让我们来到Program的Main函数中。
还是有一些准备要做，还差两个元数据呢，对了，就是绑定和地址。和上一篇一样，我们先建立这个两个元数据的对象备用：

```
[csharp]
1.  WSHttpBinding binding = new WSHttpBinding();
2.  EndpointAddress remoteAddress = new EndpointAddress("http://localhost/IISService/HelloWCFService.
```

接下来就要建立我们的代理类对象了，new一个出来吧：

```
[csharp]
1.  HelloWCFClient client = new HelloWCFClient(binding, remoteAddress);
```

把我们刚刚建立的两个元数据对象作为参数传递进去。

接下来就可以调用服务协定方法了：

```
[csharp]
1.  string result = client.HelloWCF();
```

别忘了关闭通道，ClientBase<>很贴心的为我们准备了这个方法，不用再做强制类型转换什么的了(见前一篇)。

```
[csharp]
1.  client.Close();
```

最后再添加输出语句来看结果：

```
[csharp]
1.  Console.WriteLine(result);
2.  Console.ReadLine();
```

F5一下，熟悉的结果是不是出现了？

以下是Program.cs的全部代码：

```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)
        {
            WSHttpBinding binding = new WSHttpBinding();
            EndpointAddress remoteAddress = new EndpointAddress("http://localhost/IISService/Hel

            HelloWCFClient client = new HelloWCFClient(binding, remoteAddress);

            string result = client.HelloWCF();

            client.Close();

            Console.WriteLine(result);
            Console.ReadLine();
        }
    }

    [ServiceContract]
    public interface IHelloWCF
    {
        [OperationContract]
        string HelloWCF();
    }

```

```
35.     public class HelloWCFClient : ClientBase<IHelloWCF>, IHelloWCF
36.     {
37.         public HelloWCFClient(System.ServiceModel.Channels.Binding binding, EndpointAddress remo
38.             : base(binding, remoteAddress)
39.         {
40.
41.         }
42.
43.         public string HelloWCF()
44.         {
45.             return base.Channel.HelloWCF();
46.         }
47.     }
48. }
```

6. 再展开一点点

这样看上去已经挺成功了，我们现在再打开服务引用的reference.cs代码，看看是不是大部分都看得懂了？监管有些地方他写的可能有些复杂，比如描述协定的属性参数啊，代理类更多的构造函数啊，但是核心的就是我们刚刚写的部分，那一大堆wsdl什么的其实都不是核心，不信的话你把他们都删掉，就留一个reference.cs看看还好不好用？那一堆东西也不是没用就是现在自己看起来还蛮复杂的，我们到后面一点点学习。

我们仔细看服务引用的reference.cs代码，有一样东西是我们有而他没有的，那就是对终结点地址和绑定的建立，而且在使用服务引用的时候我们也没有提供之两样东西，直接掉服务协定方法就行了(见第一篇)，那么服务引用是从哪里找到这两个关键的元数据元素呢？

就在配置文件里，我们做的客户端还没有配置文件，我们可以把这两个元素都放在配置文件里，这样就可以避免硬编码了。

为我们的控制台应用程序添加一个应用程序配置文件。方法是右键点击项目->添加->新建项->应用程序配置文件。保持默认名称app.config。
打开来看，里面没写什么实际的内容：

```
[html]

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

我们对配置服务应该不陌生，如果忘记了，翻回到第二篇去回顾一下。
首先还是要添加<System.ServiceModel>节，无论是服务端还是客户端，只要是WCF的服务配置都要在这个节里面：

```
[html]

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

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

在这里我们要配置的是客户端，所以我们不添加<Services>节了，而改成<Client>：

```
[html]

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>

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

然后我们在<Client>中添加一个终结点，这个是客户端的终结点，我们前面曾经提过，通信实际上发生在两个终结点间，客户端也有个终结点，然而请求总是从客户端首先发起，所以终结点地址应该填写为服务端终结点的地址让客户端来寻址，但是服务协定要客户端本地是有的，所以这个要写本地定义的那个协定的完全限定名：

```
[html]

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint binding="wsHttpBinding" address="http://localhost/IISService/HelloWCFService.svc"
    </client>
  </system.serviceModel>
</configuration>
```

如果你忘记了我们在第三篇中配置的IIS服务的内容，可能稍有迷惑，这里的地址看上去是个服务地址而不是终结点地址，这是因为我们在IIS中把终结点地址设置为了空字符串，此时服务地址就是终结点地址了。注意看后面的contract，他的完全限定名的命名空间是客户端程序的命名空间ConsoleClient，这也表示这个类型是定义在本地的，不要搞错了。

保存，配置已经写完了，你如果看服务引用为我们生成的配置会看到一堆东西，实际上核心的就是这些。

既然我们已经在配置中声明了绑定和终结点地址，在代码中就不再需要了。
首先我们修改一下代理类，为他提供一个没有参数的构造函数，否则在new他的时候他会非管我们要两个参数。

```csharp
public HelloWCFClient()
    : base()
{

}
```

还是什么也不做，直接调基类的无参构造函数。

然后修改一下Main函数，去掉终结点对象和地址对象的声明，并用无参构造函数来new 代理类的实例：

```csharp
static void Main(string[] args)
{
    HelloWCFClient client = new HelloWCFClient();

    string result = client.HelloWCF();

    client.Close();

    Console.WriteLine(result);
    Console.ReadLine();
}
```

F5运行一下，结果如一吧！
到这里已经和服务引用的感觉基本一样了，我们已经写出了服务引用的核心部分。
以下是修改后的Program.cs完整代码：

```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)
        {
            HelloWCFClient client = new HelloWCFClient();

            string result = client.HelloWCF();

            client.Close();

            Console.WriteLine(result);
            Console.ReadLine();
        }
    }

    [ServiceContract]
    public interface IHelloWCF
    {
        [OperationContract]
        string HelloWCF();
    }

    public class HelloWCFClient : ClientBase<IHelloWCF>, IHelloWCF
    {
        public HelloWCFClient()
            : base()
        {

        }

        public HelloWCFClient(System.ServiceModel.Channels.Binding binding, EndpointAddress remo
            : base(binding, remoteAddress)
        {
```

```
43.
44.         }
45.
46.         public string HelloWCF()
47.         {
48.             return base.Channel.HelloWCF();
49.         }
50.     }
51. }
```

7. 总结

通过本篇的学习，我们熟悉了另一种与服务端通信的办法，即通过ClientBase<>派生代理类的方法，这种方法实际上就是服务引用使用的方法，可以说我们已经在最简单级别上手写了一个服务引用的实现。

使用ChannelFactory<>和ClientBase<>都可以实现与服务端的通信，这是类库支持的最高层次的通讯方法了，其实还有更底层的通道通信方法，我们现在就不再深入了。而选择这两种方法完全取决于开发人员，有人喜欢工厂，有人喜欢代理类。既然我们都已经掌握了，就随意选择吧。

至此，我们对通信有了一个基本的了解，只能算一个初探吧。然而我相信这一次的小小深入会对我们今后的学习带来大大的帮助的。