[老老实实学WCF] 第二篇 配置WCF

老老实实学WCF
第二篇 配置WCF

在上一篇中，我们在一个控制台应用程序中编写了一个简单的WCF服务并承载了它。先回顾一下服务端的代码：

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

using System.ServiceModel;
using System.ServiceModel.Description;

namespace HelloWCFService
{
    class Program
    {
        static void Main(string[] args)
        {
            Uri baseAddress = new Uri("http://localhost:8000/MyService");

            ServiceHost host = new ServiceHost(typeof(HelloWCFService), baseAddress);

            host.AddServiceEndpoint(typeof(IHelloWCFService), new WSHttpBinding(), "HelloWCFService");

            ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
            smb.HttpGetEnabled = true;
            host.Description.Behaviors.Add(smb);

            host.Open();

            Console.WriteLine("Service is Ready");
            Console.WriteLine("Press Any Key to Terminate...");
            Console.ReadLine();

            host.Close();

        }
    }

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

    public class HelloWCFService : IHelloWCFService
    {
        public string HelloWCF()
        {
            return "Hello WCF!";
        }
    }
}
```

所有的这些代码都写在program.cs中，干净清爽。

我们稍微审视一下这段程序会发现，我们用了很多的代码来定制服务的特性，例如基地址、终结点、绑定、行为等。这些都叫做配置。而真正对服务的本身的定义是很少的(主逻辑就是返回一个字符串而已)，因此我们不难看出，WCF的编程中配置占了很大的比重。

WCF的配置选项是很多的，我们这里只考虑最简单的情况。我们在定义和实现了服务协定后，至少应该做哪些配置才能让服务运行起来呢？

(1) 依据服务实现类配置一个服务(ServiceHost)。
(2) 指定一个基地址(如果终结点中指定了绝对地址，这步可以省略)。
(3) 建立一个终结点，并为其指定地址、绑定和服务协定。
(4) 建立一个元数据交换终结点。
(5) 为服务添加一个行为来启用元数据交换。

虽然在.Net 4.0下微软提供了简化配置，我们甚至可以一行配置都不做，但是为了搞清楚配置的基本原理，我们暂时不考虑简化配置的情况。

以下这些配置是我们必须要做的，我们从代码中可以看到以上几种配置相应语句：

建立基地址：

```
[csharp]
1. Uri baseAddress = new Uri("http://localhost:8000/MyService");
```

建立服务：

```
[csharp]
1. ServiceHost host = new ServiceHost(typeof(HelloWCFService), baseAddress);
```

建立终结点并指定地址、绑定和服务协定：

```
[csharp]
1. host.AddServiceEndpoint(typeof(IHelloWCFService), new WSHttpBinding(), "HelloWCFService");
```

添加元数据交换终结点并添加启用元数据交换行为

```
[csharp]
1. ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
2. smb.HttpGetEnabled = true;
3. host.Description.Behaviors.Add(smb);
```

看上去清楚明白，但是只是看上去很美，这样的配置方式存在弊端，例如基地址，如果当服务部署之后迁移了服务器，基地址发生变化，我们必须修改源程序并重新编译重新部署才能实现这个要求。对于其他的配置选项亦是如此。这对于产品环境是不能接受的。好在WCF提供针对这个问题的解决方案：配置文件。

我们把对服务的配置写在应用程序的配置文件中(IIS程序是web.config 其他程序是app.config)，当配置发生改变的时候我们就不用重新编译程序集了。

配置文件的写法很复杂，有很多选项，为了便于上手，我们先从跟本例相关的选项开始。

在配置文件中，根节是<configuration>，所有的配置元素都位于其中。对于WCF服务的配置部分，最外层的节是<system.serviceModel>，所以配置文件中至少先应该是这个样子：

```
[html]
1. <?xml version="1.0" encoding="utf-8" ?>
2. <configuration>
3.   <system.serviceModel>
4.
5.   </system.serviceModel>
6. </configuration>
```

现在我们准备开始配置一个服务，服务配置元素标签为<services></services>，是<system.serviceModel>的子节，在一个宿主上可以承载许多服务，每一个服务用<service></service>来配置，它是<services>的子节。在配置<service>前，我们还要先添加一个基地址配置，基地址用<baseaddress>描述，他是<host>的子节，<host>是<service>的子节。

晕了么...慢慢来。
先把<services>节加上，这里可以容纳许多服务，注意这个是带s的

```
[html]
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>

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

在<services>的怀抱中，我们添加一个<service>，这是我们要配置的服务本体，注意这个是不带s的

```
[html]
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service>

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

在<service>中，添加一个基地址，先添加一个<host>再添加一个<baseaddress>

```
[html]
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service>
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/MyService"/>
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

到这里，基地址的部分已经完成，对应代码中的位置你找到了么？
服务的配置还差一点，我们在代码中为服务指定了实现类型，在配置文件中如何指定呢？就用<service>标签的name属性，指定的时候后要用完全限定名(带着命名空间)

```
[html]
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="HelloWCFService.HelloWCFService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/MyService"/>
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

我这个例子举的不好了，命名空间和实现类型是一个名字，要注意区分。

到这里，服务也配置完了，对应代码的位置翻上去找一下。

接下来配置终结点，终结点用<endpoint>元素表示，正如代码实现中的参数，在配置中也需要一一指定地址、绑定和服务协定接口，分别用address、binding和contract属性来表示，当然<endpoint>也是<service>的子节，毕竟他是属于服务的嘛。

```
[html]
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="HelloWCFService.HelloWCFService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/MyService"/>
          </baseAddresses>
        </host>
        <endpoint address="HelloWCFService" binding="wsHttpBinding" contract="HelloWCFService.IHelloWCFS
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

这里用了相对地址"HelloWCFService"，他会和基地址组合在一起(排在后面)成为终结点的地址，这里也可以指定为空字符串""，此时基地址(即服务地址)就是终结点的地址，还可以指定一个绝对地址，那样他会覆盖基地址，基地址对这个终结点来说就不起作用了，这里可以看出，终结点的地址是独立的，可以是基地址的子地址，也可以独立使用另一个地址，他们之间没有必然的链接。
这里的contract 同<service>里面一样，也要使用完全限定名称(带上命名空间)。
注意，在使用IIS承载的时候，必须使用相对地址，也就是终结点必须是基地址的子地址，这是由IIS的部署结构决定的。

到这里终结点也配置完成，对应代码的位置找到了吗？

接下来是最后一步(或者说两步)，配置元数据交换终结点并开启元数据交换行为。这个过程，代码中用了三行，实际上代码这三行仅仅是添加了元数据交换行为，并没有配置元数据交换终结点，运行时系统为我们自动添加了终结点。这两点缺一不可，虽然系统会为我们添加，我们还是要知道这个配置的写法。这个很重要。

开启元数据交换从原理上应该是两件事，第一是服务允许元数据交换，第二是服务提供元数据交换方式，第一条就是添加元数据交换行为，表示服务允许这样的请求，第二条就是服务告诉你如何请求，客户端是通过一个固定的终结点去请求的，这个终结点的地址、绑定和协定都是固定的，我们不能更改，这个是框架的约定，我们只能按要求配置。

首先，第一条，允许元数据交换，这个是通过为服务添加一个行为来实现的，行为是用来描述服务的特性的，不同的服务可能有相同的行为，行为并不是归某服务独有的，因此行为被定义为<system.serviceModel>节的子节，可以被不同的服务引用，他的定义有些像服务，外面是带s的，里面是不带s的，毕竟可能有许多的行为定义嘛。
先定义个行为：

```
[html]
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="HelloWCFService.HelloWCFService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/MyService"/>
          </baseAddresses>
        </host>
        <endpoint address="HelloWCFService" binding="wsHttpBinding" contract="HelloWCFService.IHelloWCFS
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="metaExchange">
          <serviceMetadata httpGetEnabled="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
```

```
22.  </configuration>
```

因为存在服务行为和终结点行为之分，所有<behaviors>和<behavior>之间又套了一个<serviceBehaviors>，表示这个是服务行为，我们为行为制定了名字，好让<service>可以引用，也可以不指定，那么所有服务都会应用。交换元数据的行为有固定的标签描述，就是<serviceMetadata>，对着代码看，很熟悉吧。

建立了行为以后，要让我们刚才建立的服务去引用他，用<service>的behaviorConfiguration属性：

```
[html]

1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <system.serviceModel>
4.      <services>
5.        <service name="HelloWCFService.HelloWCFService"behaviorConfiguration="metaExchange">
6.          <host>
7.            <baseAddresses>
8.              <add baseAddress="http://localhost:8000/MyService"/>
9.            </baseAddresses>
10.         </host>
11.         <endpoint address="HelloWCFService" binding="wsHttpBinding" contract="HelloWCFService.IHelloWCFS
12.       </service>
13.     </services>
14.     <behaviors>
15.       <serviceBehaviors>
16.         <behavior name="metaExchange">
17.           <serviceMetadata httpGetEnabled="true"/>
18.         </behavior>
19.       </serviceBehaviors>
20.     </behaviors>
21.   </system.serviceModel>
22. </configuration>
```

接下来第二步，建立元数据交换终结点，建立的位置和我们刚才建立的终结点位置相同，但是属性是固定的，大小写都不能写错。

```
[html]

1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <system.serviceModel>
4.      <services>
5.        <service name="HelloWCFService.HelloWCFService" behaviorConfiguration="metaExchange">
6.          <host>
7.            <baseAddresses>
8.              <add baseAddress="http://localhost:8000/MyService"/>
9.            </baseAddresses>
10.         </host>
11.         <endpoint address="HelloWCFService" binding="wsHttpBinding" contract="HelloWCFService.IHelloWCFS
12.         <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
13.       </service>
14.     </services>
15.     <behaviors>
16.       <serviceBehaviors>
17.         <behavior name="metaExchange">
18.           <serviceMetadata httpGetEnabled="true"/>
19.         </behavior>
20.       </serviceBehaviors>
21.     </behaviors>
22.   </system.serviceModel>
23. </configuration>
```

到这里，配置文件就写完了。我们把它放到程序里面去，打开上一篇中建立的服务端程序，为程序添加一个配置文件
右键点击项目->添加->新建项->应用程序配置文件，名字系统会起好(app.config)。把上面的配置写进去，保存。
既然我们已经有了配置文件，就不需要(也不应该)在代码中配置了。代码中的配置会覆盖掉配置文件中的配置。所以我们要对代码修改一下。
main函数中只留下这么几行：

```csharp
ServiceHost host = new ServiceHost(typeof(HelloWCFService));

host.Open();

Console.WriteLine("Service is Ready");
Console.WriteLine("Press Any Key to Terminate...");
Console.ReadLine();

host.Close();
```

其中，建立SeviceHost 那行被修改了，去掉了baseAddress的参数，但是我们仍需要告诉host 我们要寄存的服务类的类型。
F5运行起来。
然后在浏览器中访问一下服务试试

```html
http://localhost:8000/MyService
```

是不是和昨天的结果一样呢。
(CSDN的传图好象挂了哩...)

总结一下今天的学习。

> 我们使用配置文件的方法完成了对WCF服务的配置，从中接触到了服务、终结点和行为的配置方法。配置文件的元素还有许多，像绑定、安全性等等特性。在今后学到的时候再慢慢展开，配置文件的每一个元素都应该力求背着写下来，一行一行的写，在写的过程中体会，而不是四处复制和粘贴，这样才能对配置文件的写法有深刻的印象。

相关资源

> 徐长龙老师播讲的《跟我一起从零开始学WCF系列课程》
> http://msdn.microsoft.com/zh-cn/hh148206 (http://msdn.microsoft.com/zh-cn/hh148206)
>
> MSDN技术资源库中的WCF参考
> http://msdn.microsoft.com/zh-cn/library/dd456779.aspx
> (http://msdn.microsoft.com/zh-cn/library/dd456779.aspx)